PEMROGRAMAN DASAR

dengan C, C++, dan Python



Irfan Subakti

Profitina

profitina.com

Pemrograman Dasar

dengan C, C++, dan Python

Irfan Subakti



Hak cipta © 2026 Irfan Subakti. Hak cipta dilindungi undang-undang.

Diterbitkan oleh Profitina · profitina.com

Edisi pertama, September 2026.

Setiap program dalam buku ini sudah dikompilasi dan dijalankan sebelum keluarannya dicetak. Kode sumber lengkapnya ada di folder Code yang menyertai buku ini. Program-program tersebut disediakan untuk keperluan belajar-mengajar; Irfan Subakti dan penerbit tidak bertanggung jawab atas penggunaannya di luar keperluan itu.

Daftar Isi

Kata Pengantar ..... **xxv**
Untuk siapa buku ini ditulis xxv
Satu gagasan, beberapa bahasa xxv
Isi buku ini xxvi
Cara membacanya xxvii
Tentang penulis xxvii
Ucapan terima kasih xxvii

BAB 1 Komputer, C, dan Program Pertama Anda **1**
1.1 Struk yang Ditulis Tangan 1
1.2 Apa yang Dikerjakan Komputer 2
1.2.1 Perangkat keras dan perangkat lunak 2
1.2.2 Program dan bahasa pemrograman 2
1.3 C dan Python: Dua Bahasa, Satu Cara Berpikir 3
1.3.1 Mengapa C 3
1.3.2 Mengapa juga Python 3
1.3.3 Dikompilasi dan diinterpretasi 4
1.4 Menyiapkan Meja Kerja 5
1.4.1 Windows 10 dan 11 5
1.4.2 macOS 6
1.4.3 Linux (Ubuntu, Debian, dan kerabatnya) 6
1.4.4 Visual Studio Code dan dua ekstensinya 6
1.4.5 Folder kode bab ini 7
1.5 Program Pertama Anda, Baris demi Baris 7
1.6 Dari Teks Sumber menjadi Program yang Berjalan 8
1.6.1 Apa yang sebenarnya dikerjakan `gcc` 8
1.6.2 Ketika ada yang salah: galat dan peringatan 8
1.7 Variabel dan Memori 9
1.7.1 Kotak yang bernama 9
1.7.2 Membaca masukan dengan `scanf` 10
1.7.3 Empat tipe dasar 11
1.8 Aritmetika 13
1.8.1 Lima operator dan satu jebakan 13
1.8.2 Presedensi: siapa lebih dahulu 14
1.8.3 Aritmetika yang sama di Python 14
1.9 Keputusan dengan `if` 15
1.10 Kasir Profitina Laundry v0.1 16
1.11 Membuatnya Lebih Baik: Berat Desimal dan Tarif Minimum 17
1.12 Memakainya Sungguhan 19
1.13 Kesalahan yang Sering Terjadi, dan Cara Menghindarinya 19
1.14 Ringkasan Bab 20
1.15 Soal Latihan 21
1.16 Penyelesaian Soal Latihan 21
Daftar Pustaka 23

BAB 2 Pengembangan Program Terstruktur: Algoritma, Pseudokode, dan Kontrol **25**
2.1 Suatu Petang di Meja Kasir 25
2.2 Algoritma: Aksi dalam Urutan yang Benar 25
2.3 Pseudokode 26
2.4 Tiga Struktur Sudah Cukup 27

2.4.1 Runtunan, seleksi, dan perulangan ... 27
2.4.2 Membaca diagram alir ... 28
2.4.3 Blok ... 29
2.5 Seleksi ... 29
2.5.1 `if` dan `if`...`else` ... 29
2.5.2 Rantai keputusan ... 29
2.5.3 Operator kondisional ... 31
2.6 Perulangan yang Dikendalikan Pencacah ... 32
2.6.1 Perulangan `while` ... 32
2.6.2 Menghitung sampai lima ... 32
2.7 Penghalusan Bertahap dari Atas ke Bawah: Perulangan yang Dikendalikan Sentinel ... 33
2.7.1 Masalah yang lebih sulit ... 33
2.7.2 Puncak dan penghalusan pertama ... 34
2.7.3 Penghalusan kedua ... 34
2.7.4 Programnya ... 35
2.7.5 Memeriksa rancangan dengan tangan: tabel penelusuran ... 36
2.7.6 *Cast* ... 37
2.8 Struktur Kontrol Bersarang ... 38
2.9 Operator Penugasan dan Operator Increment ... 40
2.9.1 Penugasan gabungan ... 40
2.9.2 Increment dan decrement ... 41
2.10 Kasir Profitina Laundry v0.2 ... 42
2.11 Membuatnya Lebih Baik: Menjaga dari Masukan yang Mustahil ... 45
2.12 Memakainya Sungguhan ... 45
2.13 Kesalahan yang Sering Terjadi, dan Cara Menghindarinya ... 46
2.14 Ringkasan Bab ... 47
2.15 Soal Latihan ... 47
2.16 Penyelesaian Soal Latihan ... 48
Daftar Pustaka ... 49

BAB 3 Kontrol Program: for, do...while, switch, dan Operator Logika ... 50
3.1 Mesin Cuci Baru dan Laporan yang Lebih Rapi ... 50
3.2 Pernyataan `for` ... 51
3.2.1 Perulangan berhitung dalam satu baris ... 51
3.2.2 `for` dan `range` di Python ... 52
3.3 Pameran `for`: Bunga Majemuk dan Lebar Kolom ... 53
3.3.1 Rumusnya ... 53
3.3.2 Programnya ... 53
3.3.3 Lebar kolom ... 54
3.3.4 Membaca tabel seperti seorang pemilik usaha ... 55
3.4 Pernyataan `do`...`while` ... 55
3.4.1 Paling sedikit sekali ... 55
3.4.2 Tanya sampai jawabannya sah ... 56
3.5 Pernyataan `switch` ... 57
3.5.1 Satu keputusan, banyak kasus ... 57
3.5.2 Jatuh terus ke bawah ... 59
3.5.3 `switch` atau rantai `else if`? ... 59
3.5.4 `match` di Python ... 59
3.6 `break` dan `continue` ... 60
3.7 Operator Logika ... 61
3.7.1 `&&`, `||`, dan `!` ... 61
3.7.2 Evaluasi hubung-singkat ... 62
3.7.3 Hukum De Morgan ... 63
3.7.4 Dari aturan usaha menjadi kondisi ... 63
3.8 Memilih Struktur yang Tepat ... 64

3.9 Kasir Profitina Laundry v0.3 65
3.10 Membuatnya Lebih Baik: Daftar Harga untuk Dinding 68
3.11 Memakainya Sungguhan 69
3.12 Kesalahan yang Sering Terjadi, dan Cara Menghindarinya 70
3.13 Ringkasan Bab 71
3.14 Soal Latihan 71
3.15 Penyelesaian Soal Latihan 72
Daftar Pustaka 72

BAB 4 Evaluasi dan Pembahasan I: Lima Soal Diselesaikan Langkah demi Langkah 74
4.1 Posisi Kita Sekarang 74
4.2 Cara Menghadapi Soal Pemrograman 75
4.2.1 Tujuh langkah 75
4.2.2 Susunan setiap soal, dan letak kodenya 75
4.3 Soal 4.1: Luas Bangun Datar 76
4.3.1 Soalnya 76
4.3.2 Memahaminya dengan tangan 76
4.3.3 Merancangnya di atas kertas 76
4.3.4 Solusi C 77
4.3.5 Solusi Python 78
4.3.6 Solusi keliru yang menggoda 78
4.4 Soal 4.2: Permainan Tebak Angka 79
4.4.1 Soalnya 79
4.4.2 Memahaminya dengan tangan 80
4.4.3 Merancangnya di atas kertas 80
4.4.4 Solusi C 80
4.4.5 Solusi Python 81
4.4.6 Solusi keliru yang menggoda 82
4.5 Soal 4.3: 100 Pangkat n 83
4.5.1 Soalnya 83
4.5.2 Memahaminya dengan tangan 83
4.5.3 Merancangnya di atas kertas 84
4.5.4 Solusi C 84
4.5.5 Solusi Python 85
4.5.6 Solusi keliru yang menggoda 86
4.6 Soal 4.4: Batu, Kertas, Gunting 87
4.6.1 Soalnya 87
4.6.2 Memahaminya dengan tangan 87
4.6.3 Merancangnya di atas kertas 88
4.6.4 Solusi C 88
4.6.5 Solusi Python 90
4.6.6 Solusi keliru yang menggoda 90
4.7 Soal 4.5: Pertarungan Goblin 91
4.7.1 Soalnya 91
4.7.2 Memahaminya dengan tangan 92
4.7.3 Merancangnya di atas kertas 92
4.7.4 Solusi C 94
4.7.5 Solusi Python 95
4.7.6 Solusi keliru yang menggoda 96
4.8 Soal-soal Berdampingan 97
4.9 Memakainya Sungguhan 98
4.10 Kesalahan yang Sering Terjadi, dan Cara Menghindarinya 98
4.11 Ringkasan Bab 99
4.12 Soal Latihan 99
4.13 Penyelesaian Soal Latihan 100

Daftar Pustaka ... 101

BAB 5 Fungsi: Bagi, Beri Nama, dan Taklukkan ... 102

5.1 Kasir Mbak Rina Sudah Terlalu Panjang ... 102
5.2 Gagasannya dengan Kata-Kata Sederhana ... 103
5.2.1 Fungsi adalah sepotong pekerjaan yang diberi nama ... 103
5.2.2 Apa yang terjadi selama pemanggilan ... 103
5.2.3 Bagi dan taklukkan: mengapa potongan kecil menang ... 104
5.3 Fungsi Pertama Anda ... 105
5.3.1 Satu aturan, ditulis sekali ... 105
5.3.2 Prototipe, fungsi `void`, dan pass-by-value ... 106
5.3.3 Cara pandang Python: nama harus ada saat dipanggil ... 108
5.4 Meminjam Fungsi: Pustaka Standar ... 108
5.5 Cakupan dan Masa Hidup: Tempat Nama Tinggal ... 111
5.6 Keacakan yang Dapat Diulang ... 113
5.7 Rekursi: Fungsi yang Memanggil Dirinya Sendiri ... 116
5.7.1 Kasus dasar dan masalah yang lebih kecil ... 116
5.7.2 Rekursi yang berguna: rupiah bertitik ... 117
5.7.3 Harga rekursi: Fibonacci ... 118
5.7.4 Rekursi atau perulangan? ... 121
5.8 Membangun Kasir Versi 4 ... 121
5.8.1 Satu berkas aturan, satu berkas cerita ... 121
5.8.2 Mengompilasi beberapa berkas ... 126
5.9 Menjalankannya ... 127
5.10 Mengujinya ... 127
5.11 Membuatnya Lebih Baik ... 129
5.12 Memakainya Sungguhan ... 130
5.13 Kesalahan yang Sering Terjadi, dan Cara Menghindarinya ... 130
5.14 Ringkasan Bab ... 131
5.15 Soal Latihan ... 131
5.16 Penyelesaian Soal Latihan ... 132
Daftar Pustaka ... 137

BAB 6 Array: Banyak Nilai, Satu Nama ... 138

6.1 Catatan di Penghujung Hari ... 138
6.2 Gagasannya dengan Kata-Kata Sederhana ... 139
6.2.1 Array adalah deretan kotak bernomor ... 139
6.2.2 Mengapa indeks dimulai dari 0 ... 139
6.2.3 Tidak ada yang menjaga pagar ... 140
6.2.4 List milik Python ... 140
6.3 Mendeklarasikan, Mengisi, dan Menelusuri Array ... 141
6.4 Array sebagai Penghitung ... 143
6.5 Meneruskan Array ke Fungsi ... 145
6.6 Pengurutan: Bubble Sort ... 148
6.7 Pencarian: Linear dan Biner ... 150
6.8 Array Dua Dimensi ... 154
6.9 Membangun Kasir Versi 5 ... 157
6.10 Menjalankannya ... 163
6.11 Mengujinya ... 164
6.12 Membuatnya Lebih Baik ... 166
6.13 Memakainya Sungguhan ... 167
6.14 Kesalahan yang Sering Terjadi, dan Cara Menghindarinya ... 168
6.15 Ringkasan Bab ... 168
6.16 Soal Latihan ... 169

6.17 Penyelesaian Soal Latihan 169
Daftar Pustaka 172

BAB 7 Pointer: Alamat sebagai Nilai 173
7.1 Satu Panggilan, Tiga Jawaban 173
7.2 Gagasannya dengan Kata-Kata Sederhana 174
7.2.1 Setiap variabel punya alamat 174
7.2.2 Mengikuti panah 174
7.2.3 Tipe sebuah pointer, dan pointer yang tidak menunjuk apa-apa 175
7.2.4 Pointer yang sudah Anda pakai 176
7.3 Pointer Pertama Anda 176
7.4 Pass-by-Reference: Mengubah Variabel Pemanggil 178
7.5 Parameter Keluaran: Satu Panggilan, Beberapa Jawaban 180
7.6 Array dan Pointer: Dua Notasi, Satu Memori 183
7.6.1 Nama array adalah alamat elemen pertamanya 183
7.6.2 Aritmetika pointer 183
7.6.3 Berjalan dengan pointer 184
7.6.4 Mengapa pointer, bukan hanya indeks? 186
7.7 `const` dan Pointer 186
7.8 Pointer ke Fungsi: Perilaku sebagai Parameter 187
7.9 Membangun Kasir Versi 6 190
7.10 Menjalankannya 194
7.11 Mengujinya 195
7.12 Membuatnya Lebih Baik: Keselamatan Pointer 197
7.13 Memakainya Sungguhan 198
7.14 Kesalahan yang Sering Terjadi, dan Cara Menghindarinya 199
7.15 Ringkasan Bab 199
7.16 Soal Latihan 200
7.17 Penyelesaian Soal Latihan 201
Daftar Pustaka 204

BAB 8 Evaluasi dan Pembahasan II: Fungsi, Array, dan Pointer, Diselesaikan Langkah demi Langkah 205
8.1 Titik Berangkat Kita 205
8.2 Cara Menghadapi Sebuah Masalah 206
8.2.1 Delapan langkah 206
8.2.2 Susunan setiap masalah, dan letak kodenya 207
8.3 Soal 8.1: Fungsi-Fungsi Statistik 207
8.3.1 Masalahnya 207
8.3.2 Memahaminya dengan tangan 207
8.3.3 Merancangnya di atas kertas 208
8.3.4 Penyelesaian dalam C 209
8.3.5 Penyelesaian dalam Python 210
8.3.6 Solusi salah yang menggoda 211
8.4 Soal 8.2: Fibonacci dengan Dua Cara 214
8.4.1 Masalahnya 214
8.4.2 Memahaminya dengan tangan 214
8.4.3 Merancangnya di atas kertas 214
8.4.4 Penyelesaian dalam C 215
8.4.5 Penyelesaian dalam Python 217
8.4.6 Mengukur perbedaannya 219
8.4.7 Solusi salah yang menggoda 219
8.5 Soal 8.3: Palindrom 221
8.5.1 Masalahnya 221
8.5.2 Memahaminya dengan tangan 221

8.5.3 Merancangnya di atas kertas 222
8.5.4 Penyelesaian dalam C 222
8.5.5 Penyelesaian dalam Python 224
8.5.6 Solusi salah yang menggoda 226
8.6 Soal 8.4: Determinan Matriks 3 × 3 227
8.6.1 Masalahnya 227
8.6.2 Memahaminya dengan tangan 228
8.6.3 Merancangnya di atas kertas 228
8.6.4 Penyelesaian dalam C 229
8.6.5 Penyelesaian dalam Python 230
8.6.6 Solusi salah yang menggoda 231
8.7 Soal 8.5: Mencari Sebuah Nota 232
8.7.1 Masalahnya 232
8.7.2 Memahaminya dengan tangan, dan rancangannya 233
8.7.3 Penyelesaiannya 233
8.7.4 Solusi salah yang menggoda 234
8.8 Soal 8.6: Menggabungkan Dua Sif 235
8.8.1 Masalahnya 235
8.8.2 Memahaminya dengan tangan, dan rancangannya 235
8.8.3 Penyelesaiannya 235
8.8.4 Solusi salah yang menggoda 237
8.9 Keenam Masalah Berdampingan 238
8.10 Kesalahan yang Sering Terjadi, dan Cara Menghindarinya 238
8.11 Ringkasan Bab 239
8.12 Soal Latihan 239
8.13 Penyelesaian Soal Latihan 240
Daftar Pustaka 241

BAB 9 Karakter, String, dan I/O Terformat 242
9.1 Posisi Kita Sekarang: Nama-nama di Buku Mbak Rina 242
9.2 Gagasannya dengan Kata-kata Sederhana: Teks Adalah Bilangan 243
9.2.1 Karakter adalah bilangan kecil 243
9.2.2 String adalah larik yang diakhiri nol 243
9.2.3 Program yang sama dalam Python 246
9.3 Satu Karakter pada Satu Waktu: `<ctype.h>` 247
9.4 Pustaka String dan Sisi Tajamnya 251
9.5 Keluaran Terformat: Seluruh Tombol Pengatur `printf` 254
9.6 Struk yang Kolomnya Rata 257
9.7 Membaca Teks secara Aman 262
9.8 Membangun Kasir v0.7 263
9.9 Menjalankan dan Mengujinya 274
9.10 Membuatnya Lebih Baik 276
9.11 Memakainya secara Nyata 276
9.12 Kesalahan yang Sering Terjadi, dan Cara Menghindarinya 277
9.13 Ringkasan Bab 277
9.14 Soal Latihan 278
9.15 Penyelesaian Soal Latihan 278
Daftar Pustaka 281

BAB 10 Struktur, Enum, dan Linked List Pertama 283
10.1 Posisi Kita Sekarang: Tiga Larik yang Harus Sepakat 283
10.2 Gagasannya dengan Kata-kata Sederhana: Satu Rekaman, Satu Nama 284
10.2.1 Mendeklarasikan struktur 284
10.2.2 Rupa struktur di memori 284
10.2.3 Program pertama 285

10.3 Struktur dan Fungsi 289
10.4 `enum` dan `typedef`: Nama Menggantikan Angka 294
10.5 Memori Sesuai Permintaan: `malloc` dan `free` 297
10.6 Struktur yang Menunjuk Dirinya Sendiri: Linked List 298
10.6.1 Gagasannya 298
10.6.2 Menyusuri senarai 300
10.6.3 Menambah simpul 300
10.6.4 Menghapus simpul 300
10.6.5 Membebaskan seluruh senarai 301
10.6.6 Programnya 301
10.7 Membangun Kasir v0.8 308
10.8 Menjalankan dan Mengujinya 322
10.9 Membuatnya Lebih Baik 323
10.10 Memakainya secara Nyata 324
10.11 Kesalahan yang Sering Terjadi, dan Cara Menghindarinya 324
10.12 Ringkasan Bab 325
10.13 Soal Latihan 325
10.14 Penyelesaian Soal Latihan 326
Daftar Pustaka 330

BAB 11 Pemrosesan Berkas: Data yang Bertahan Saat Listrik Padam 331
11.1 Posisi Kita Sekarang: Pukul Empat, dan Listrik Padam 331
11.2 Gagasannya dengan Kata-kata Sederhana: Berkas dan Aliran 332
11.2.1 Aliran dan pointer `FILE` 332
11.2.2 Mode 332
11.3 Menulis dan Membaca Berkas Teks 333
11.4 Jebakan `feof` 335
11.5 Membaca Data Nyata: Berkas CSV Baris demi Baris 337
11.6 Mode Tambah: Cara Log Bertumbuh 342
11.7 Berkas Biner dan Akses Acak 344
11.8 Menyimpan Tanpa Kehilangan: Tulis, Tutup, Ganti Nama 348
11.9 Membangun Kasir v0.9 349
11.9.1 Satu program, beberapa berkas 349
11.9.2 Modul penyimpanan 351
11.9.3 Program utama 354
11.10 Menjalankan dan Mengujinya 358
11.11 Membuatnya Lebih Baik 359
11.12 Memakainya secara Nyata 360
11.13 Kesalahan yang Sering Terjadi, dan Cara Menghindarinya 360
11.14 Ringkasan Bab 361
11.15 Soal Latihan 361
11.16 Penyelesaian Soal Latihan 362
Daftar Pustaka 364

BAB 12 Evaluasi dan Pembahasan III: Senarai, Pohon, dan Seekor Kuda Catur 366
12.1 Posisi Kita Sekarang 366
12.2 Cara Menghadapi Soal Struktur Data 366
12.2.1 Langkah-langkahnya 366
12.2.2 Letak kodenya 367
12.3 Soal 12.1: Urutkan Linked List, Lalu Sisipkan di Akhir 367
12.3.1 Soalnya 367
12.3.2 Memahaminya secara manual 368
12.3.3 Merancangnya di atas kertas 368
12.3.4 Penyelesaian dalam C 369

12.3.5 Penyelesaian dalam Python 372
12.3.6 Menjalankan dan mengujinya 373
12.3.7 Penyelesaian keliru yang menggoda 374
12.3.8 Periksa diri 376
12.4 Soal 12.2: Pohon yang Ditumbuhkan dari Berkas 376
12.4.1 Soalnya 376
12.4.2 Memahaminya secara manual 376
12.4.3 Merancangnya di atas kertas 377
12.4.4 Penyelesaian dalam C 378
12.4.5 Penyelesaian dalam Python 383
12.4.6 Menjalankan dan mengujinya 386
12.4.7 Penyelesaian keliru yang menggoda 388
12.4.8 Periksa diri 388
12.5 Soal 12.3: Lintasan Terpendek Kuda Catur 388
12.5.1 Soalnya 388
12.5.2 Memahaminya secara manual 389
12.5.3 Merancangnya di atas kertas 390
12.5.4 Penyelesaian dalam C 391
12.5.5 Penyelesaian dalam Python 395
12.5.6 Menjalankan dan mengujinya 398
12.5.7 Penyelesaian keliru yang menggoda 399
12.5.8 Periksa diri 400
12.6 Soal-soal Berdampingan 400
12.7 Memakainya secara Nyata 401
12.8 Kesalahan yang Sering Terjadi, dan Cara Menghindarinya 401
12.9 Ringkasan Bab 402
12.10 Soal Latihan 402
12.11 Penyelesaian Soal Latihan 403
Daftar Pustaka 407

BAB 13 C++ sebagai C yang Lebih Baik 408
13.1 Struk Bu Rina 408
13.2 Gagasannya dalam Kata-Kata Sederhana 409
13.2.1 Satu keluarga, dua dialek 409
13.2.2 Arti "C yang lebih baik" 409
13.2.3 Mengompilasi program C++ 410
13.3 Stream: `cout` dan `cin` 410
13.4 `std::string`: Teks yang Selalu Muat 412
13.5 `std::vector`: Array yang Bisa Bertambah Panjang 414
13.6 Referensi: Mengubah Variabel Pemanggil Tanpa Pointer 416
13.7 Overloading dan Argumen Bawaan 419
13.8 `new` dan `delete` 421
13.9 Membangunnya: Kasir v1.0 423
13.10 Menjalankannya 429
13.11 Mengujinya 430
13.12 Membuatnya Lebih Baik: Menolak Baris Buruk dengan Sopan 432
13.13 Memakainya Sungguhan 433
13.14 Kesalahan yang Sering Terjadi, dan Cara Menghindarinya 434
13.15 Ringkasan Bab 434
13.16 Soal Latihan 435
13.17 Penyelesaian Soal Latihan 435
Daftar Pustaka 437

BAB 14 Kelas dan Abstraksi Data 438
14.1 Cucian Gratis, Tanpa Sengaja 438

14.2 Gagasannya dalam Kata-Kata Sederhana 439
14.2.1 Abstraksi data 439
14.2.2 Invarian 439
14.2.3 Dari `struct` ke `class` 440
14.3 Masalahnya, Diukur 440
14.4 Membangun Kelas `Order` 442
14.4.1 Header: antarmukanya 443
14.4.2 Implementasinya 444
14.4.3 Memakai kelasnya 446
14.4.4 Kelas yang sama dalam Python 448
14.4.5 Dua pertanyaan yang sering diajukan pemula 450
14.4.6 Menelusuri pesanan yang ditolak langkah demi langkah 450
14.5 Konstruktor dan Destruktor: Lahir Sah, Dibereskan Otomatis 451
14.6 Sekilas tentang Pewarisan 453
14.7 Membangunnya: Toko 458
14.7.1 Kelas `Shop` 458
14.7.2 Program di sekelilingnya 460
14.8 Menjalankannya 463
14.9 Mengujinya 464
14.10 Membuatnya Lebih Baik 465
14.11 Memakainya Sungguhan 465
14.12 Kesalahan yang Sering Terjadi, dan Cara Menghindarinya 466
14.13 Ringkasan Bab 466
14.14 Soal Latihan 467
14.15 Penyelesaian Soal Latihan 468
Daftar Pustaka 468

BAB 15 Overloading Operator dan Studi Kasus Utuh 470
15.1 Struk yang Jumlahnya Cocok 470
15.2 Gagasannya dalam Kata-Kata Sederhana 471
15.2.1 Operator adalah fungsi dengan nama khusus 471
15.2.2 Aturannya 471
15.3 Kelas Uang: `Rupiah` 472
15.3.1 Kelasnya 472
15.3.2 Kelasnya beraksi 475
15.3.3 Menelusuri satu ekspresi dengan tangan 476
15.3.4 Kelas yang sama dalam Python 477
15.4 Contoh Kedua: Pecahan dan Algoritma Euclid 479
15.5 Studi Kasus Utuh: Profitina Laundry 485
15.5.1 Datanya 485
15.5.2 Aturannya 486
15.5.3 Programnya 486
15.5.4 Mengikuti dua pesanan melewati `take` 492
15.6 Menjalankannya 497
15.7 Mengujinya 499
15.8 Membuatnya Lebih Baik 499
15.9 Memakainya Sungguhan 500
15.10 Kesalahan yang Sering Terjadi, dan Cara Menghindarinya 501
15.11 Ringkasan Bab 501
15.12 Soal Latihan 502
15.13 Penyelesaian Soal Latihan 502
Daftar Pustaka 503

BAB 16 Evaluasi dan Pembahasan IV: Kelas, Overloading, dan Seluruh Buku, Diselesaikan Langkah demi Langkah 505
16.1 Di Mana Kita Berhenti 505
16.2 Cara Menghadapi Sebuah Soal 506
16.2.1 Tujuh langkah 506
16.2.2 Susunan setiap soal, dan letak kodenya 506
16.3 Soal 16.1: Bidak Catur 506
16.3.1 Soalnya 506
16.3.2 Memahaminya dengan tangan 507
16.3.3 Merancangnya di atas kertas 508
16.3.4 Penyelesaian C++ 508
16.3.5 Penyelesaian Python 512
16.3.6 Penyelesaian keliru yang menggoda 513
16.4 Soal 16.2: Overloading `add` 514
16.4.1 Soalnya 514
16.4.2 Memahaminya dengan tangan 515
16.4.3 Merancangnya di atas kertas 515
16.4.4 Penyelesaian C++ 515
16.4.5 Penyelesaian Python 516
16.4.6 Penyelesaian keliru yang menggoda 516
16.5 Soal 16.3: Mencari Anagram 517
16.5.1 Soalnya 517
16.5.2 Memahaminya dengan tangan 518
16.5.3 Merancangnya di atas kertas 518
16.5.4 Penyelesaian C++ 519
16.5.5 Penyelesaian Python 521
16.5.6 Penyelesaian keliru yang menggoda 522
16.6 Soal 16.4: Perkalian Matriks 523
16.6.1 Soalnya 523
16.6.2 Memahaminya dengan tangan 523
16.6.3 Merancangnya di atas kertas 524
16.6.4 Penyelesaian C++ 525
16.6.5 Penyelesaian Python 527
16.6.6 Penyelesaian keliru yang menggoda 528
16.7 Soal 16.5: Kapan Bisa Diambil? 530
16.7.1 Soalnya 530
16.7.2 Memahaminya dengan tangan 530
16.7.3 Merancangnya di atas kertas 530
16.7.4 Penyelesaian C++ 531
16.7.5 Penyelesaian Python 533
16.7.6 Penyelesaian keliru yang menggoda 535
16.8 Kelima Soal Berdampingan 536
16.9 Memakainya Sungguhan 537
16.10 Kesalahan yang Sering Terjadi, dan Cara Menghindarinya 537
16.11 Ringkasan Bab 538
16.12 Soal Latihan 538
16.13 Penyelesaian Soal Latihan 539
Daftar Pustaka 540

Daftar Gambar

Gambar 1.1 Dua jalur dari teks sumber menjadi keluaran: program C melewati praprosesor, kompilator, dan *linker* satu kali lalu menjadi berkas *executable*; program Python dibaca dan dijalankan oleh interpreter setiap kali. 4

Gambar 1.2 Tiga variabel sesudah `add_two` dijalankan: `scanf` menulis 45 dan 72 ke alamat `integer1` dan `integer2`, lalu penugasan membaca keduanya dan menulis 117 ke `sum`. 10

Gambar 2.1 Tiga struktur kontrol: runtunan, seleksi, dan perulangan; masing-masing punya tepat satu jalan masuk dan satu jalan keluar. 28

Gambar 2.2 Rata-rata terkendali sentinel: baca sebelum perulangan, uji di puncak, baca lagi di akhir badan, dan bagi hanya bila ada paling sedikit satu pesanan. 35

Gambar 3.1 Anatomi pernyataan `for`: inisialisasi (1) dijalankan sekali; lalu kondisi (2), badan (3), dan *increment* (4) berulang sampai kondisinya salah. 51

Gambar 3.2 `while` menguji sebelum badan, sehingga badan mungkin tak pernah dijalankan; `do`…`while` menguji sesudah badan, sehingga badan selalu dijalankan paling sedikit sekali. 56

Gambar 4.1 Aturan batu, kertas, gunting sebagai siklus: setiap benda mengalahkan benda berikutnya menurut arah anak panah. 87

Gambar 4.2 Satu giliran pertarungan: pilih kerusakan, pukul goblin hidup yang pertama, biarkan goblin yang hidup membalas, cetak satu baris, dan putuskan apakah pertarungan berlanjut. 93

Gambar 5.1 Pemanggilan `bill_for(7000, 3)`: argumen disalin ke parameter, badan fungsi berjalan, dan `return` mengirim satu nilai kembali ke tempat pemanggilan, tempat `main` melanjutkan kerjanya. 104

Gambar 5.2 Tumpukan panggilan pada titik terdalam `factorial(4)`: setiap pemanggilan memiliki bingkai sendiri dengan `n` sendiri, dan setiap bingkai di bawah puncak menunggu jawaban bingkai di atasnya. 117

Gambar 5.3 Setiap pemanggilan oleh `fib(5)`: lima belas pemanggilan untuk menghitung satu bilangan, dan pemanggilan berwarna kuning mengulang pekerjaan yang sudah dilakukan di tempat lain dalam pohon. 118

Gambar 6.1 Array `bill` dengan delapan elemen: satu nama, indeks 0 sampai 7, dan elemen-elemen berdampingan di memori, berjarak delapan bita di komputer yang menyimpan `long` dalam delapan bita. 139

Gambar 6.2 Bubble sort pada enam tagihan: setelah setiap putaran satu nilai lagi (berarsir) berada di tempat akhirnya di kanan; putaran 4 tidak menukar apa pun, sehingga pengurutan berhenti lebih awal. 149

Gambar 6.3 Tiga baris pertama tabel mingguan, seperti yang kita bayangkan dan seperti letaknya di memori: baris demi baris, sehingga `kg[r][c]` terletak $3r + c$ elemen sesudah awal (empat bita per `int`). 156

Gambar 7.1 Tiga variabel di memori: `p` menyimpan alamat `kg`, digambar sebagai panah; `*p` mengikuti panah itu, sehingga `*p = 7` mengubah `kg`. 175

Gambar 7.2 Kiri: `swap_copies` bekerja pada salinan, dan `main` tetap memegang 9 dan 4. Kanan: `swap` menerima alamat variabel milik `main` dan menukar nilai-nilai di sana. 179

Gambar 7.3 Aritmetika pointer: `p + k` menunjuk `k` elemen lebih jauh, berapa pun ukuran elemennya, sehingga `bill[i]`, `*(bill + i)`, dan `p[i]` menamai kotak yang sama. 183

Gambar 8.1 Kerja kedua cara, seperti dihitung oleh `fibonacci.c`: langkah perulangan tumbuh seperti n, pemanggilan rekursif seperti 1.618^n (skala logaritmik). 217

Gambar 8.2 Pemindaian dua pointer pada `kasurrusak`: huruf berwarna kuning sedang dibandingkan, yang abu-abu sudah selesai; setelah lima pasangan yang sama, kedua pointer bersilang dan kata itu palindrom. 221

Gambar 8.3 Ekspansi sepanjang baris pertama: setiap entri baris pertama (kuning) mengalikan determinan 2×2 dari sel-sel yang tersisa ketika baris dan kolomnya dihapus (hijau kebiruan), dengan tanda $+, -, +$. 228

Gambar 9.1 Dua string di memori: `name` menampung empat huruf dan penutup dalam tepat lima byte, sedangkan `buffer` memesan enam belas byte dan string hanya memakai enam. Bilangan abu-abu di bawah `name` adalah kode ASCII-nya. 244
Gambar 9.2 Bagian-bagian spesifikasi konversi `%-9.2f`: *flag* `-` meratakan kiri, lebar 9 adalah banyak karakter minimum, presisi 2 memberi angka di belakang titik desimal, dan tipe `f` menyatakan bahwa yang dicetak adalah `double`. 255
Gambar 10.1 Tata letak `struct Order` di memori: empat anggota di posisi 0, 4, 28, dan 32, dan dua byte *padding* sesudah `customer` agar `tenths` dimulai di kelipatan empat. 285
Gambar 10.2 *Singly linked list* berisi pesanan: (a) `head` menunjuk simpul pertama dan setiap `next` menunjuk simpul berikutnya, yang terakhir menunjuk `NULL`; (b) menambah di akhir mengubah `next` simpul terakhir dan pointer `tail`; (c) menghapus simpul di tengah menjembatani simpul itu dengan satu penugasan. 299
Gambar 11.1 Berkas biner berisi empat rekaman 32 byte: menurut Persamaan (11.1), rekaman 2 dimulai di byte $2 \cdot 32 = 64$, dan `fseek` langsung ke sana. 344
Gambar 11.2 Dua cara menyimpan: (a) menimpa berkas di tempat membuatnya kosong atau setengah tertulis selama penyimpanan; (b) menulis berkas baru lalu mengganti namanya menimpa berkas lama menyisakan, pada setiap saat, satu berkas yang utuh. 348
Gambar 12.1 Menumbuhkan pohon dari `tree5.txt`: setelah setiap perintah, simpul baru (kuning) menjadi simpul saat ini, dan perintah `u...ur` lebih dulu naik satu tingkat per `u` darinya. 377
Gambar 12.2 Banyak langkah kuda yang diperlukan dari petak awal (7,0) ke setiap petak; tujuan (0,7) memerlukan 6, dan petak-petak kuning menunjukkan satu lintasan terpendek. 390
Gambar 13.1 Tiga cara mengirim `points`: parameter nilai menerima salinan, parameter pointer menerima alamat, dan parameter referensi adalah nama lain bagi variabel milik pemanggil sendiri. 416
Gambar 14.1 Kelas `Order`: antarmuka *public* mengelilingi data *private*; kode dari luar hanya dapat memanggil fungsi *public*, dan konstruktor memeriksa setiap nilai. 442
Gambar 14.2 Siklus hidup sebuah pesanan: `advance()` memajukannya dua kali, `pickUp()` menyerahkannya, dan setiap panggilan lain ditolak tanpa mengubah keadaannya. 446
Gambar 15.1 Apa yang dilakukan kompilator terhadap `wash + iron` untuk operan berupa kelas: ia mencari `operator+(wash, iron)` non-anggota atau `wash.operator+(iron)` anggota, lalu memanggil yang ditemukannya. 471
Gambar 15.2 Studi kasus: tiga berkas data dimuat ke dalam `Shop`, setiap pesanan diperiksa dan diubah menjadi `Receipt`, lalu struk-struk dicetak dan diringkas dalam laporan yang ditulis ke layar sekaligus ke `report.txt`. 485
Gambar 16.1 Petak-petak yang dapat dicapai setiap bidak dari (3, 3) di papan kosong, beserta syarat yang menentukannya. 507
Gambar 16.2 Kelima jendela sepanjang 4 dalam `aabaabaa`: empat di antaranya anagram dari `aaba`; jendela terakhir dimulai di $n - m = 4$. 518
Gambar 16.3 Kilogram harian per layanan dikali harga per kilogram menghasilkan pendapatan harian; $C00$ memadukan baris 0 dari A dengan kolom 0 dari B. 524

Daftar Tabel

Tabel 1.1 Perkalian yang sama pada tiga tingkat bahasa; baris bahasa mesin dan rakitan adalah contoh khas untuk prosesor x86-64. 3
Tabel 1.2 Meja kerja buku ini; semua alat gratis dan berjalan di Windows, macOS, dan Linux. 5
Tabel 1.3 Presedensi operator yang sudah kita kenal; bila ragu, tambahkan tanda kurung. 14
Tabel 2.1 Penelusuran **Pseudokode 2.6** untuk masukan 4, 7, 3, 0: sentinel dibaca di akhir iterasi ketiga dan tidak pernah dijumlahkan. 37
Tabel 2.2 Operator penugasan gabungan di C; Python memiliki operator yang sama, dengan `//=` untuk pembagian bulat. 40

Tabel 3.1 Lebar kolom dan presisi pada `printf`; spesifikasi format Python di dalam f-string memakai angka yang sama, dengan `>` untuk rata kanan dan `<` untuk rata kiri. 54
Tabel 3.2 Tabel kebenaran operator logika AND (`&&`), OR, dan NOT (`!`); di C sebuah kondisi bernilai salah bila 0 dan benar bila selain itu. 62
Tabel 3.3 Memilih struktur kontrol: pertanyaan yang diajukan dan pernyataan yang menjawabnya. 64
Tabel 4.1 Pangkat-pangkat 100 terhadap batas tipe bilangan bulat C: `int` berhenti sesudah $n = 4$, `long long` sesudah $n = 9$. 84
Tabel 4.2 Kesembilan kombinasi diperiksa terhadap Persamaan (4.3): sisa 0 berarti seri, 1 berarti pemain 1 menang, 2 berarti pemain 2 menang. 88
Tabel 4.3 Kelima soal bab ini, strukturnya, dan jebakan setiap solusi keliru yang menggoda. 98
Tabel 5.1 Header standar yang paling sering dipakai buku ini; header memuat prototipe, pustaka memuat fungsi yang sudah dikompilasi. 109
Tabel 5.2 Cakupan dan masa hidup empat jenis variabel dalam C. 111
Tabel 7.1 Letak `const` dalam deklarasi pointer menentukan apa yang dilindungi: variabel yang ditunjuk, pointernya, atau keduanya. 187
Tabel 8.1 Enam masalah dalam bab ini dan alat dari Bab 5 sampai 7 yang paling banyak dilatih masing-masing. 206
Tabel 8.2 Simpangan baku kesepuluh nilai yang dihitung dengan tangan: simpangan-simpangannya berjumlah nol, dan kuadratnya berjumlah 166,40. 208
Tabel 8.3 Waktu jalan cara rekursif dan iteratif yang terukur; waktu rekursif tumbuh sekitar sebelas kali dari $n = 35$ ke $n = 40$, seperti diramalkan $1.6185 \approx 11$. 219
Tabel 8.4 Keenam masalah berdampingan: biaya setiap penyelesaian, jebakan solusi salah yang menggoda, dan masukan uji yang menyingkap jebakan itu. 238
Tabel 9.1 Fungsi `<ctype.h>` yang paling sering dipakai. Karakter yang bukan huruf melewati `toupper` dan `tolower` tanpa berubah. 247
Tabel 9.2 Fungsi sehari-hari `<string.h>` (dan `snprintf` dari `<stdio.h>`), masing-masing dengan aturan keamanannya. 251
Tabel 10.1 Kerja operasi-operasi umum pada n pesanan, untuk larik dan untuk *linked list* yang menyimpan pointer ke simpul pertama dan terakhirnya. 300
Tabel 11.1 Mode-mode `fopen`. Menambahkan `b` memberi mode yang sama untuk aliran biner. 333
Tabel 12.1 Ketiga soal dibandingkan: struktur, operasi kunci, gagasan yang membuatnya berhasil, biaya, dan kesalahan yang harus dihindari. 401
Tabel 13.1 Tabel migrasi: pekerjaan yang sama dalam C, dalam C++, dan dalam Python. 410
Tabel 14.1 Tiga penentu akses dan siapa yang boleh memakai anggota di masing-masing bagian. 440
Tabel 15.1 Operator-operator yang di-*overload* dalam bab ini, bentuk lazimnya di C++, dan padanannya di Python. 472
Tabel 16.1 Kelima soal bab ini berdampingan. 537

Daftar Kode

Kode 1.1 `hello.c`: program C pertama mencetak dua baris teks. 7
Kode 1.2 `hello.py`: dua baris yang sama dalam Python. 8
Kode 1.3 `add_two.c`: membaca dua bilangan bulat lalu menjumlahkannya. 9
Kode 1.4 `add_two.py`: program yang sama dalam Python. 11
Kode 1.5 `types.c`: sebuah `int`, sebuah `double`, dan sebuah `char`, serta jangkauan `int`. 11
Kode 1.6 `types.py`: nilai-nilai yang sama dalam Python, yang tidak punya tipe `char` dan tidak membatasi `int`. 12
Kode 1.7 `arithmetic.c`: lima operator, pembagian bilangan bulat, dan presedensi. 13
Kode 1.8 `arithmetic.py`: hasil yang sama di Python, yang memerlukan `//`, `int(...)`, dan `math.fmod` untuk meniru C. 14

Kode 1.9 `compare.c`: enam operator pembanding, masing-masing menjaga satu `printf`. 15
Kode 1.10 `compare.py`: perbandingan yang sama dalam Python. 15
Kode 1.11 `receipt.c`: kasir v0.1 mencetak struk untuk layanan cuci, kering, dan lipat. 16
Kode 1.12 `receipt.py`: kasir v0.1 dalam Python. 17
Kode 1.13 `receipt_min.c`: kasir v0.1 yang diperbaiki, dengan berat desimal dan tarif minimum. 17
Kode 1.14 `receipt_min.py`: kasir yang diperbaiki dalam Python. 19
Kode 1.15 `wrong_average.c`: `1 / 2` bernilai 0, sehingga separuh dari 7 kg menjadi 0.00 kg. 20
Kode 1.16 `change.c`: kembalian dipecah menjadi lembaran dengan `/` dan `%`. 22
Kode 2.1 `discount.c`: rantai `if`…`else if`…`else`, sebuah `if` terpisah, dan operator kondisional. 30
Kode 2.2 `discount.py`: `if`, `elif`, dan `else`, serta ekspresi kondisional Python. 31
Kode 2.3 `average_counter.c`: perulangan `while` yang dikendalikan pencacah. 32
Kode 2.4 `average_counter.py`: perulangan terkendali pencacah yang sama dalam Python. 33
Kode 2.5 `average_sentinel.c`: perulangan terkendali sentinel, sebuah *cast*, dan pengaman terhadap pembagian dengan nol. 36
Kode 2.6 `average_sentinel.py`: algoritma yang sama dalam Python. 37
Kode 2.7 `survey.c`: `if`…`else` yang bersarang di dalam `while`. 38
Kode 2.8 `survey.py`: struktur bersarang yang sama dalam Python, dengan indentasi saja yang menunjukkan bahwa `if` menjadi milik `while`. 39
Kode 2.9 `increment.c`: penugasan gabungan, serta *increment* prefiks lawan postfiks. 41
Kode 2.10 `increment.py`: hasil yang sama di Python, yang memiliki penugasan gabungan tetapi tidak memiliki `++`. 41
Kode 2.11 `cashier_v02.c`: kasir v0.2 melayani antrean pelanggan sampai kasir ditutup. 43
Kode 2.12 `cashier_v02.py`: kasir v0.2 dalam Python. 44
Kode 2.13 `heavy_count.c`: menghitung pesanan yang lebih berat dari 5 kg, dengan berat negatif ditolak. 48
Kode 3.1 `for_basics.c`: menghitung naik, menghitung mundur, menghitung berlangkah, dan jumlah berjalan. 52
Kode 3.2 `for_basics.py`: perulangan yang sama dengan `for` dan `range`. 52
Kode 3.3 `savings.c`: setoran yang tumbuh dengan bunga majemuk, dicetak sebagai tabel yang rata. 54
Kode 3.4 `savings.py`: tabel yang sama dalam Python. 55
Kode 3.5 `validate.c`: `do`…`while` terus bertanya sampai beratnya berada dalam rentang. 56
Kode 3.6 `validate.py`: bentuk Python untuk "tanya sampai sah" adalah `while True` dengan `break`. 57
Kode 3.7 `service_count.c`: `switch` dengan kasus bersama, `break`, dan `default`. 58
Kode 3.8 `service_count.py`: `match`, dengan `case "W" | "w"` mendaftar dua alternatif dan `case _` menangkap sisanya. 59
Kode 3.9 `break_continue.c`: `break` mengakhiri perulangan di 5; `continue` hanya melewati 5. 60
Kode 3.10 `logic.c`: tabel kebenaran `&&`, `||`, dan `!`, serta pembagian yang dilindungi evaluasi hubung-singkat. 62
Kode 3.11 `logic.py`: `and`, `or`, dan `not`, dengan perlindungan hubung-singkat yang sama. 63
Kode 3.12 `cashier_v03.c`: kasir v0.3 dengan `switch`, berat yang divalidasi, dan laporan harian berformat. 65
Kode 3.13 `cashier_v03.py`: kasir v0.3 dalam Python, dengan `match` dan ungkapan `while True`. 67
Kode 3.14 `price_table.c`: perulangan `for` bersarang mencetak daftar harga dengan kolom-kolom yang rata. 68
Kode 4.1 `geometry.c`: luas empat bangun datar, dengan panjang negatif ditolak. 77
Kode 4.2 `geometry.py`: luas yang sama dalam Python. 78
Kode 4.3 `geometry_wrong.c`: `1 / 2` dievaluasi lebih dahulu, sebagai pembagian bulat. 78
Kode 4.4 `guess.c`: permainan tebak angka sebagai perulangan `while` baca-lebih-dahulu. 80
Kode 4.5 `guess.py`: perulangan baca-lebih-dahulu yang sama dalam Python. 81
Kode 4.6 `guess_wrong.c`: badannya menilai setiap tebakan, termasuk tebakan yang menang. 82
Kode 4.7 `power100.c`: 100 pangkat *n*, dicetak digit demi digit, tanpa pernah meluap. 84

Kode 4.8 `power100.py`: perulangan pencetak digit yang sama dalam Python. 85
Kode 4.9 `power100_wrong.c`: `pow` mengembalikan `double`, yang hanya memiliki sekitar 16 angka penting. 86
Kode 4.10 `rps.c`: masukan tervalidasi, nama dicetak dengan `switch`, dan pemenang ditemukan dengan satu sisa pembagian. 88
Kode 4.11 `rps.py`: permainan yang sama dengan validasi `while True` dan `match`. 90
Kode 4.12 `rps_wrong.c`: tanpa `+ 3`, `%` di C memberi −2 untuk batu melawan gunting. 90
Kode 4.13 `battle.c`: pertarungan sebagai satu perulangan `while` atas lima variabel keadaan, dicetak sebagai tabel. 94
Kode 4.14 `battle.py`: pertarungan yang sama dalam Python. 95
Kode 4.15 `battle_wrong.c`: tanpa pemeriksaan "masih hidup?", goblin yang mati terus bertarung dan HP menjadi negatif. 96
Kode 5.1 `first_function.c`: satu aturan tagihan, didefinisikan sekali di atas `main` dan dipanggil empat kali. 105
Kode 5.2 `first_function.py`: aturan yang sama sebagai fungsi Python. 105
Kode 5.3 `prototypes.c`: prototipe di atas, `main` lebih dulu, fungsi `void`, fungsi yang dibangun dari fungsi lain, dan peragaan pass-by-value. 106
Kode 5.4 `prototypes.py`: program yang sama dalam Python, tempat fungsi harus ada ketika dipanggil, bukan ketika dibaca. 108
Kode 5.5 `rounding.c`: `floor`, `round`, dan `ceil` dari `<math.h>`, serta aturan penagihan laundry sebagai fungsi. 109
Kode 5.6 `rounding.py`: tabel yang sama dalam Python, dengan fungsi kecil yang membulatkan setengah seperti C. 110
Kode 5.7 `scope.c`: penghitung global, variabel lokal, blok yang menyembunyikan nama, dan penghitung nota `static`. 111
Kode 5.8 `scope.py`: program yang sama dalam Python, dengan deklarasi `global` dan variabel modul yang berperan sebagai `static` milik C. 112
Kode 5.9 `lucky_draw.c`: pembangkit bilangan acak yang dapat diulang dengan keadaan `static` tersembunyi, dadu yang dibuat darinya, dan uji keadilan. 114
Kode 5.10 `lucky_draw.py`: pembangkit yang sama dalam Python, dengan perputaran 32 bit yang dinyatakan secara eksplisit. 115
Kode 5.11 `recursion.c`: faktorial, rupiah bertitik, dan Fibonacci dengan penghitung pemanggilan. 118
Kode 5.12 `recursion.py`: tiga rekursi yang sama dalam Python. 119
Kode 5.13 `laundry.h`: janji-janji modul aturan bisnis, beserta konstanta dan include guard. 121
Kode 5.14 `laundry.c`: aturan bisnis Profitina Laundry, satu fungsi untuk setiap aturan. 122
Kode 5.15 `cashier.c`: kasir versi 4, yang `main`-nya bercerita dan menyerahkan pekerjaan kepada fungsi-fungsi bernama. 123
Kode 5.16 `laundry.py`: aturan bisnis sebagai modul Python. 125
Kode 5.17 `cashier.py`: kasir versi 4 dalam Python, yang mengimpor aturan dari `laundry.py`. 125
Kode 5.18 `test_laundry.c`: dua belas pemeriksaan aturan bisnis terhadap jawaban hitungan tangan. 127
Kode 5.19 `test_laundry.py`: pemeriksaan yang sama dalam Python, ditambah satu untuk teks rupiah. 128
Kode 5.20 `ex5_gcd.c`: algoritma Euclid secara rekursif, dan kelipatan persekutuan terkecil yang dibangun darinya. 133
Kode 5.21 `ex5_gcd.py`: hal yang sama dalam Python. 133
Kode 5.22 `ex5_digits.c`: jumlah digit secara rekursif dan dengan perulangan, serta digit pemeriksa. 134
Kode 5.23 `ex5_digits.py`: hal yang sama dalam Python. 134
Kode 5.24 `ex5_hanoi.c`: Menara Hanoi, masalah klasik yang mudah secara rekursif dan sulit dengan perulangan. 135
Kode 5.25 `ex5_hanoi.py`: hal yang sama dalam Python. 136
Kode 6.1 `day_bills.c`: delapan tagihan dalam satu array, dicetak, dijumlahkan, dicari yang terbesar, dan dihitung. 141

Kode 6.2 `day_bills.py`: penelusuran yang sama atas list Python. 142
Kode 6.3 `service_count.c`: array penghitung yang diindeks dengan kode layanan, dan sebatang bintang untuk masing-masing. 144
Kode 6.4 `service_count.py`: penghitung yang sama dalam Python. 144
Kode 6.5 `array_functions.c`: fungsi yang menerima array beserta ukurannya; `const` berjanji tidak mengubahnya, dan `add_to_all` mengubah array milik pemanggil. 145
Kode 6.6 `array_functions.py`: fungsi-fungsi yang sama atas list Python. 147
Kode 6.7 `sort_search.c`: bubble sort dengan jejak setiap putaran yang dicetak, pencarian linear, dan pencarian biner dengan jejak setiap tengokan yang dicetak. 151
Kode 6.8 `sort_search.py`: pengurutan dan pencarian yang sama dalam Python. 152
Kode 6.9 `week_table.c`: kilogram per hari dan layanan dalam array dua dimensi, dengan total baris, total kolom, dan hari tersibuk. 155
Kode 6.10 `week_table.py`: tabel yang sama sebagai list berisi list di Python. 156
Kode 6.11 `bills.h`: janji-janji modul untuk array tagihan. 157
Kode 6.12 `bills.c`: total, indeks maksimum, penyalinan, pengurutan dari terbesar ke terkecil, dan median. 158
Kode 6.13 `day_report.c`: kasir versi 5, yang menyimpan hari itu dalam array paralel dan mencetak laporan tutup toko. 159
Kode 6.14 `bills.py`: alat-alat yang sama untuk list Python. 161
Kode 6.15 `day_report.py`: kasir versi 5 dalam Python. 162
Kode 6.16 `test_bills.c`: sembilan pemeriksaan alat array, termasuk array kosong dan janji bahwa `median` tidak mengubah masukannya. 165
Kode 6.17 `test_bills.py`: pemeriksaan yang sama dalam Python. 165
Kode 6.18 `ex6_reverse.c`: membalik array di tempat dengan dua indeks. 169
Kode 6.19 `ex6_reverse.py`: hal yang sama dalam Python (Python yang idiomatis cukup menulis `a.reverse()`). 170
Kode 6.20 `ex6_insertion.c`: insertion sort dengan jejak setelah setiap penyisipan. 171
Kode 6.21 `ex6_insertion.py`: hal yang sama dalam Python. 171
Kode 7.1 `first_pointer.c`: mengambil alamat, mengikutinya, menulis melaluinya, mengarahkan ulang pointer, menyalinnya, dan `NULL`. 176
Kode 7.2 `first_pointer.py`: list satu elemen sebagai kotak yang dapat dipakai bersama dua nama, yaitu bagian pointer yang dimiliki Python. 177
Kode 7.3 `swap.c`: menukar salinan tidak mengubah apa pun di `main`; menukar melalui pointer mengubah variabel milik `main`. Tiga langkah banding-tukar lalu mengurutkan tiga bilangan. 178
Kode 7.4 `swap.py`: Python tidak dapat menukar dua variabel milik pemanggilnya; Python menukar dengan penugasan tuple di pihak pemanggil, atau mengubah list yang dipakai bersama. 179
Kode 7.5 `out_params.c`: `split_weight` memberikan dua jawaban melalui pointer; `day_stats` memberikan tiga jawaban dan mengembalikan apakah jawaban itu sah. 181
Kode 7.6 `out_params.py`: cara Python: kembalikan tuple, atau `None` bila tidak ada jawaban yang sah. 182
Kode 7.7 `pointer_walk.c`: tiga nama untuk satu elemen, penelusuran dengan pointer, selisih pointer, indeks negatif dari pointer yang sudah dimajukan, dan kata yang dibalik dengan dua pointer. 184
Kode 7.8 `pointer_walk.py`: penelusuran yang sama dengan indeks, cara Python menyatakan apa yang dikerjakan C dengan pointer. 185
Kode 7.9 `sort_by.c`: satu bubble sort yang bertanya kepada sebuah fungsi mana dari dua nilai yang lebih dulu, tiga fungsi semacam itu dalam sebuah array, dan `qsort` dari pustaka dengan fungsi pembandingnya. 187
Kode 7.10 `sort_by.py`: fungsi yang diteruskan sebagai nilai di Python, dan `sorted` dari pustaka. 190
Kode 7.11 `stats.h`: janji-janji modul statistik. 190
Kode 7.12 `stats.c`: `split_weight`, `day_stats` dengan empat jawaban dan penelusuran pointer, serta fungsi pembanding untuk `qsort`. 191
Kode 7.13 `closing.c`: kasir versi 6: ringkasan tutup toko dari satu pemanggilan `day_stats`. 192
Kode 7.14 `stats.py`: modul statistik dalam Python, yang mengembalikan tuple. 193

Kode 7.15 `closing.py`: kasir versi 6 dalam Python. 193
Kode 7.16 `test_stats.c`: empat belas pemeriksaan, termasuk pembulatan di batas kilogram, hari tanpa pesanan, pointer null, dan janji bahwa pemanggilan yang ditolak membiarkan jawaban tak tersentuh. 195
Kode 7.17 `test_stats.py`: pemeriksaan yang sama dalam Python. 196
Kode 7.18 `ex7_strlen.c`: panjang string dengan selisih pointer, dan pencacahan sebuah karakter dengan penelusuran pointer. 201
Kode 7.19 `ex7_strlen.py`: hitungan yang sama dalam Python, yang string-nya tidak membawa nol penutup. 201
Kode 7.20 `ex7_change.c`: kembalian dalam lembaran uang, setiap pemanggilan mengurangi sisa jumlah milik pemanggil melalui pointer. 202
Kode 7.21 `ex7_change.py`: hal yang sama dengan tuple yang dikembalikan. 202
Kode 8.1 `stats.c`: kerangka yang diberikan dengan kelima badan fungsi yang dilengkapi; `INT_MIN` dan `INT_MAX` memulai pencarian, cast datang sebelum pembagian, dan simpangan memakai `sqrt` 209
Kode 8.2 `stats.py`: kelima fungsi yang sama dalam Python, tempat `/` selalu menghasilkan pecahan. 210
Kode 8.3 `wrong_stats.c`: solusi salah yang menggoda, yang memulai minimum dari 0 dan membagi dua bilangan bulat. 212
Kode 8.4 `wrong_stats.py`: dua kesalahan yang sama dalam Python, tempat pembagian bilangan bulat ditulis `//`. 212
Kode 8.5 `fibonacci.c`: cara iteratif dengan jumlah melalui parameter keluaran, cara rekursif dengan penghitung pemanggilan, dan tabel kerja untuk *n* yang lebih besar. 215
Kode 8.6 `fibonacci.py`: kedua cara yang sama dalam Python; cara iteratif mengembalikan tuple. 217
Kode 8.7 `wrong_fibonacci.c`: solusi salah yang menggoda, yang memulai barisan dari 1, 1. 219
Kode 8.8 `wrong_fibonacci.py`: kesalahan yang sama dalam Python. 220
Kode 8.9 `palindrome.c`: tiga cara memutuskan palindrom yang harus sepakat: dua pointer, rekursi pada dua pointer, dan XOR atas pasangan-pasangan. 222
Kode 8.10 `palindrome.py`: ketiga cara yang sama dalam Python, dengan indeks. 224
Kode 8.11 `wrong_palindrome.c`: solusi salah yang menggoda, yang indeks pasangannya `n - i` alih-alih `n - 1 - i`. 226
Kode 8.12 `wrong_palindrome.py`: kesalahan yang sama dalam Python, dengan karakter nol eksplisit untuk meniru C. 226
Kode 8.13 `determinant.c`: membaca matriks 3 × 3 dengan perulangan bersarang dan menghitung setiap determinan dengan ekspansi sepanjang baris pertama. 229
Kode 8.14 `determinant.py`: determinan yang sama dalam Python, dengan matriks yang dibangun dari slice. 230
Kode 8.15 `wrong_determinant.c`: solusi salah yang menggoda, yang menjumlahkan suku tengah alih-alih mengurangkannya. 231
Kode 8.16 `wrong_determinant.py`: kesalahan yang sama dalam Python. 231
Kode 8.17 `find_receipt.c`: pencarian biner rekursif atas nomor nota yang terurut, mencetak setiap pemanggilan dengan indentasi sesuai kedalamannya. 233
Kode 8.18 `find_receipt.py`: rekursi yang sama dalam Python; `f"{'':{2 * depth}}"` mencetak indentasinya. 234
Kode 8.19 `merge_shifts.c`: menggabungkan dua array terurut dengan tiga pointer yang berjalan, termasuk kasus sif sore yang kosong. 236
Kode 8.20 `merge_shifts.py`: penggabungan yang sama dalam Python dengan dua indeks; `extend` menyalin sisa list yang belum habis. 236
Kode 9.1 `string_basics.c`: string adalah larik `char` yang diakhiri `'\0'`; `sizeof` menghitung byte, `strlen` menghitung karakter. 244
Kode 9.2 `string_basics.py`: baris-baris yang sama dalam Python, yang stringnya mengetahui panjangnya dan tidak dapat diubah di tempat. 246
Kode 9.3 `char_classes.c`: menggolongkan karakter, menyimpan angka-angka nomor telepon, dan merapikan nama. 247
Kode 9.4 `char_classes.py`: tiga tugas yang sama dengan metode string Python. 250

Kode 9.5 `string_library.c`: menyalin, menyambung, memformat berbatas, membandingkan, mencari, dan memotong string. 251
Kode 9.6 `string_library.py`: padanan fungsi-fungsi `<string.h>` dalam Python. 253
Kode 9.7 `format_dial.c`: *flag*, lebar, presisi, dan tipe `printf`, satu per satu. 255
Kode 9.8 `format_dial.py`: spesifikasi yang sama dengan operator `%` Python. 256
Kode 9.9 `receipt.c`: struk yang ditata dengan lebar ruas, pemusatan `%*s`, dan pemisah ribuan. 257
Kode 9.10 `receipt.py`: struk yang sama dalam Python. 260
Kode 9.11 `cashier_v7.c`: kasir Profitina Laundry v0.7, penerjemah perintah dengan nama yang rapi dan pencarian dari sebagian nama. 264
Kode 9.12 `cashier_v7.py`: kasir yang sama dalam Python. 270
Kode 9.13 `word_stats.c`: menghitung baris, kata, dan huruf, kata terpanjang, dan huruf yang paling sering muncul. 279
Kode 9.14 `word_stats.py`: statistik yang sama dalam Python. 280
Kode 10.1 `order_struct.c`: mendeklarasikan, menginisialisasi, menyalin, dan membandingkan struktur, serta mengukur tata letaknya. 285
Kode 10.2 `order_struct.py`: *data class* untuk pesanan, dan `ctypes` untuk mengukur tata letak C. 287
Kode 10.3 `order_functions.c`: struktur yang diberikan secara nilai dan lewat pointer, fungsi yang mengembalikan pointer ke dalam larik, dan `qsort` dengan fungsi pembanding. 289
Kode 10.4 `order_functions.py`: fungsi-fungsi yang sama dalam Python, tempat setiap argumen adalah rujukan. 292
Kode 10.5 `service_enum.c`: enumerasi untuk layanan dan status, serta tabel layanan yang dibangun dengan *designated initializer* 294
Kode 10.6 `service_enum.py`: `IntEnum` dan kamus sebagai tabel layanan. 296
Kode 10.7 `order_list.c`: *linked list* berisi pesanan dengan `push_front`, `append`, `find`, `remove_id`, dan `free_all`. 301
Kode 10.8 `order_list.py`: *linked list* yang sama dalam Python, dibangun dari objek `Node`. 305
Kode 10.9 `cashier_v8.c`: kasir Profitina Laundry v0.8, dengan satu struktur per pesanan dan pesanan di toko disimpan sebagai *linked list*. 308
Kode 10.10 `cashier_v8.py`: kasir v0.8 dalam Python. 317
Kode 10.11 `list_reverse.c`: membalik *linked list* di tempat. 327
Kode 10.12 `list_reverse.py`: hal yang sama dalam Python. 328
Kode 11.1 `write_read_text.c`: menulis tiga pesanan ke berkas teks dan membacanya kembali. 333
Kode 11.2 `write_read_text.py`: berkas yang sama ditulis dan dibaca dalam Python. 335
Kode 11.3 `feof_trap.c`: perulangan `while (!feof(fp))` berjalan satu kali terlalu banyak. 335
Kode 11.4 `feof_trap.py`: tiruan perulangan C dalam Python. 336
Kode 11.5 `csv_report.c`: membaca berkas CSV baris demi baris, melewati baris rusak dengan pesan, dan menjumlahkan per layanan dan per hari. 338
Kode 11.6 `csv_report.py`: laporan yang sama dalam Python. 341
Kode 11.7 `activity_log.c`: menambahkan satu entri bernomor ke berkas log pada setiap kali dijalankan. 342
Kode 11.8 `activity_log.py`: log yang sama dalam Python. 343
Kode 11.9 `binary_records.c`: rekaman berukuran tetap yang ditulis dengan `fwrite`, dicapai dengan `fseek`, dan diubah di tempat. 345
Kode 11.10 `binary_records.py`: rekaman yang sama dengan modul `struct` Python. 346
Kode 11.11 `shop.h`: tipe-tipe dan fungsi-fungsi modul toko, dilindungi *include guard*. 349
Kode 11.12 `storage.h`: memuat dan menyimpan toko. 351
Kode 11.13 `storage.c`: pemuat CSV, yang memeriksa setiap ruas setiap baris, dan penyimpan, yang menulis berkas sementara lalu mengganti namanya. 351
Kode 11.14 `cashier_v9.c`: kasir v0.9, yang memuat pesanan saat mulai dan menyimpan setelah setiap perubahan. 354
Kode 11.15 `storage.py`: memuat dan menyimpan dalam Python. 355
Kode 11.16 `cashier_v9.py`: kasir v0.9 dalam Python. 356
Kode 11.17 `find_customer.c`: mencari di berkas CSV baris demi baris. 362

Kode 11.18 `find_customer.py`: pencarian yang sama dalam Python. 363
Kode 12.1 `list_sort.c`: *bubble sort* pada *linked list* dengan menukar data, lalu penyisipan di akhir. 369
Kode 12.2 `list_sort.py`: penyelesaian yang sama dalam Python. 372
Kode 12.3 `wrong_list_sort.c`: satu lintasan pertukaran tetangga, yang bukan pengurutan. 374
Kode 12.4 `wrong_list_sort.py`: kesalahan yang sama dalam Python. 375
Kode 12.5 `tree_from_file.c`: menumbuhkan pohon biner dari berkas perintah, mencetaknya dalam tiga urutan dan menyamping, lalu membebaskannya. 378
Kode 12.6 `tree_from_file.py`: pembangun pohon yang sama dalam Python. 383
Kode 12.7 `knight.c`: *breadth-first search* dengan antrean *linked list* untuk langkah kuda paling sedikit. 391
Kode 12.8 `knight.py`: penelusuran yang sama dalam Python. 395
Kode 12.9 `knight_dfs.c`: *depth-first search* melaporkan lintasan pertama yang ditemukannya, bukan yang terpendek. 399
Kode 12.10 `knight_dfs.py`: kesalahan yang sama dalam Python. 399
Kode 12.11 `list_insertion_sort.c`: *insertion sort* dengan menyambung ulang simpul. 403
Kode 12.12 `list_insertion_sort.py`: hal yang sama dalam Python. 405
Kode 13.1 `hello_laundry.cpp`: membaca nama dan berat dengan `cin` serta mencetak dengan `cout` 410
Kode 13.2 `hello_laundry.py`: program yang sama dalam Python. 412
Kode 13.3 `strings_demo.cpp`: operasi sehari-hari `std::string` 413
Kode 13.4 `strings_demo.py`: operasi yang sama dengan `str` milik Python. 414
Kode 13.5 `vector_demo.cpp`: berat pesanan sehari dalam sebuah `std::vector`. 415
Kode 13.6 `vector_demo.py`: `list` milik Python melakukan apa yang dilakukan `std::vector` 416
Kode 13.7 `references.cpp`: parameter nilai, pointer, dan referensi, serta fungsi dengan dua hasil. 417
Kode 13.8 `references.py`: apa yang dilakukan Python sebagai ganti referensi. 418
Kode 13.9 `overload_default.cpp`: tiga overload `printLine` dan satu argumen bawaan pada `charge`. 419
Kode 13.10 `overload_default.py`: argumen bawaan dalam Python, dan tiga nama sebagai ganti overload. 421
Kode 13.11 `new_delete.cpp`: `new[]` dengan `delete[]`, `new` dengan `delete`, dan sebuah `vector` yang tidak memerlukan keduanya. 421
Kode 13.12 `new_delete.py`: Python membebaskan memori dengan sendirinya. 422
Kode 13.13 `cashier.cpp`: kasir Profitina Laundry v1.0 dalam C++. 423
Kode 13.14 `cashier.py`: kasir yang sama dalam Python. 427
Kode 13.15 `tests/run_tests.py`: mengompilasi, menjalankan, dan membandingkan setiap program dalam satu bab. 430
Kode 14.1 `struct_problem.cpp`: struktur yang terbuka menerima nilai apa pun di kolom mana pun. 440
Kode 14.2 `struct_problem.py`: dataclass Python sama terbukanya. 441
Kode 14.3 `order.h`: antarmuka kelas `Order`. 443
Kode 14.4 `order.cpp`: implementasi kelas `Order`. 444
Kode 14.5 `order_main.cpp`: membuat, menolak, dan memajukan pesanan. 446
Kode 14.6 `order.py`: kelas `Order` dalam Python. 448
Kode 14.7 `order_main.py`: memakai `Order` versi Python; keluarannya persis sama dengan `order_main.cpp` 449
Kode 14.8 `raii_demo.cpp`: konstruktor dan destruktor mengumumkan dirinya. 451
Kode 14.9 `raii_demo.py`: jaminan yang sama dengan pernyataan `with` milik Python. 452
Kode 14.10 `services.cpp`: kelas dasar `Service` dan tiga jenis layanan. 453
Kode 14.11 `services.py`: hierarki yang sama dalam Python. 456
Kode 14.12 `shop.h`: antarmuka kelas `Shop` 458
Kode 14.13 `shop.cpp`: implementasi kelas `Shop` 458
Kode 14.14 `shop_main.cpp`: membaca perintah dan meneruskannya ke toko. 460

Kode 14.15 `shop.py`: kelas `Shop` dalam Python. 461
Kode 14.16 `shop_main.py`: perulangan perintah dalam Python. 462
Kode 15.1 `rupiah.h`: jumlah uang dalam rupiah utuh, beserta operator-operatornya. 472
Kode 15.2 `rupiah_demo.cpp`: setiap operator `Rupiah` dipakai. 475
Kode 15.3 `rupiah.py`: kelas `Rupiah` dalam Python. 477
Kode 15.4 `rupiah_demo.py`: demo Python mencetak persis apa yang dicetak `rupiah_demo.cpp`. 478
Kode 15.5 `fraction.cpp`: kelas `Fraction` yang dibangun di atas FPB Euclid, dengan operator aritmetika, perbandingan, dan keluaran. 480
Kode 15.6 `fraction.py`: kelas `Fraction` dalam Python. 483
Kode 15.7 `laundry.cpp`: program Profitina Laundry yang lengkap dalam C++. 486
Kode 15.8 `laundry.py`: program Profitina Laundry yang lengkap dalam Python. 493
Kode 16.1 `chess.cpp`: `Piece` dan `Pawn` yang diberikan, serta keempat bidak tambahan. 508
Kode 16.2 `chess.py`: bidak-bidak catur dalam Python; keluarannya identik. 512
Kode 16.3 `wrong_rook.cpp`: syarat benteng dengan `||` dan syarat gajah tanpa $dx \neq 0$. 513
Kode 16.4 `add_overload.cpp`: `add` untuk dua sampai lima bilangan bulat, untuk dua `double`, dan sebuah template variadik. 515
Kode 16.5 `add_overload.py`: satu fungsi Python untuk berapa pun argumen. 516
Kode 16.6 `wrong_add.cpp`: `add` berargumen bawaan di samping yang diberikan; pemanggilan dengan dua argumen menjadi ambigu. 517
Kode 16.7 `anagram.cpp`: pencarian anagram dengan `std::is_permutation` dan dengan hitungan bergeser. 519
Kode 16.8 `anagram.py`: kedua metode yang sama dalam Python. 521
Kode 16.9 `wrong_anagram.py`: perulangan berhenti satu jendela terlalu awal. 522
Kode 16.10 `matrix.cpp`: perkalian matriks dengan pemeriksaan ukuran. 525
Kode 16.11 `matrix.py`: perkalian matriks dalam Python. 527
Kode 16.12 `wrong_matrix.cpp`: indeks B tertukar, diuji dengan matriks identitas dan dengan matriks biasa. 528
Kode 16.13 `pickup_time.cpp`: kelas `Time` dengan operator-operatornya, dan waktu pengambilan lima pesanan. 531
Kode 16.14 `pickup_time.py`: kelas `Time` dalam Python. 533
Kode 16.15 `wrong_time.py`: jam dan menit disimpan terpisah, tanpa simpanan (*carry*). 535

Daftar Pseudokode

Pseudokode 2.1 Pseudokode untuk program struk dari Bab 1. 26
Pseudokode 2.2 Pseudokode diskon: rantai tiga kasus, lalu keputusan terpisah untuk anggota. 29
Pseudokode 2.3 Perulangan terkendali pencacah: tepat lima berat dibaca. 32
Pseudokode 2.4 Puncak: satu kalimat yang menyatakan seluruh tugas. 34
Pseudokode 2.5 Penghalusan pertama: inisialisasi, proses, selesaikan. 34
Pseudokode 2.6 Penghalusan kedua: perulangan terkendali sentinel dengan pengaman terhadap pembagian dengan nol. 34
Pseudokode 2.7 Pseudokode survei: sebuah seleksi yang bersarang di dalam perulangan terkendali pencacah, diikuti seleksi kedua. 38
Pseudokode 2.8 Rancangan kasir v0.2: perulangan sentinel yang membungkus sebuah seleksi, dan rata-rata yang dilindungi di akhir. 42
Pseudokode 2.9 Masukan berat yang dihaluskan: tanya lagi sampai beratnya masuk akal. 45
Pseudokode 4.1 Rancangan Soal 4.1: runtunan pembacaan, satu pengaman, dan empat rumus. 76
Pseudokode 4.2 Rancangan Soal 4.2: perulangan baca-lebih-dahulu yang sentinelnya adalah jawaban itu sendiri. 80

Pseudokode 4.3 Rancangan Soal 4.3: cetak digit-digit $102n$ alih-alih menghitung bilangannya. 84
Pseudokode 4.4 Rancangan Soal 4.4: dua perulangan validasi, sebuah `switch` untuk nama-nama, dan aturan Persamaan (4.3). 88
Pseudokode 4.5 Rancangan Soal 4.5: lima variabel keadaan dan satu perulangan, satu iterasi per giliran. 93
Pseudokode 6.1 Bubble sort dengan keluar lebih awal: setiap putaran membawa nilai terbesar yang tersisa ke ujung bagian yang belum terurut. 149
Pseudokode 8.1 Fibonacci iteratif: dua tetangga dalam barisan bergerak maju n kali, dan jumlahnya mengumpulkan setiap nilai yang pernah dipegang `prev`. 214
Pseudokode 12.1 *Bubble sort* pada *singly linked list*, dengan menukar bilangan simpul-simpul yang bertetangga. 369
Pseudokode 12.2 Membangun pohon biner dari perintah navigasi, dengan pointer orang tua di setiap simpul. 377
Pseudokode 12.3 *Breadth-first search* untuk langkah kuda paling sedikit, dengan antrean berupa *linked list* 390
Pseudokode 16.1 Gerakan yang dipakai bersama semua bidak; MOVE-CONDITION berbeda untuk setiap jenis bidak. 508
Pseudokode 16.2 Mencari anagram dengan menguji setiap jendela memakai pemeriksaan permutasi. 519
Pseudokode 16.3 Perkalian matriks dengan pemeriksaan ukuran lebih dulu. 524
Pseudokode 16.4 Menghitung waktu pengambilan dari waktu penitipan dan layanannya. 530

Daftar Persamaan

(1.1) 13
(2.1) 30
(2.2) 35
(3.1) 53
(3.2) 63
(4.1) 76
(4.2) 83
(4.3) 88
(5.1) 109
(5.2) 113
(5.3) 116
(5.4) 118
(5.5) 118
(6.1) 139
(6.2) 142
(6.3) 150
(6.4) 154
(6.5) 155
(7.1) 175
(7.2) 180
(7.3) 183
(7.4) 183
(7.5) 189
(8.1) 207
(8.2) 208
(8.3) 214

(8.4) 228
(8.5) 228
(8.6) 235
(9.1) 244
(9.2) 255
(10.1)....285
(11.1)....344
(12.1)....368
(12.2)....389
(13.1)....412
(13.2)....420
(14.1)....440
(14.2)....446
(15.1)....474
(15.2)....479
(15.3)....480
(15.4)....480
(15.5)....486
(16.1)....507
(16.2)....515
(16.3)....518
(16.4)....524
(16.5)....524
(16.6)....530

Kata Pengantar

Setiap program yang pernah Anda pakai dulunya adalah sederet instruksi kecil yang tepat, yang dituliskan seseorang: ambil bilangan ini, bandingkan dengan yang itu, ulangi sampai tumpukannya habis, cetak hasilnya. Komputer memang cepat dan tidak kenal lelah, tetapi ia hanya mengerjakan apa yang dikatakan instruksi itu — tidak lebih, tidak kurang, dan tidak pernah sekadar apa yang Anda maksudkan. Belajar menulis instruksi seperti itu dengan jernih, sehingga benar setiap kali dan tetap mudah dibaca bulan depan, itulah pemrograman, dan itulah seluruh pokok bahasan buku ini. Pegang kalimat itu, dan tidak ada yang terasa sewenang-wenang di halaman-halaman berikut.

Buku ini tumbuh dari mata kuliah Pemrograman Dasar (EF234101) yang diajarkan di Departemen Teknik Informatika, Institut Teknologi Sepuluh Nopember (ITS), Surabaya, dan telah disusun ulang sehingga dapat berdiri sepenuhnya sendiri. Keenam belas babnya mengikuti irama satu semester enam belas minggu, tetapi Anda tidak memerlukan kelas, dosen, maupun dokumen lain untuk belajar darinya.

Untuk siapa buku ini ditulis

Anda dianggap belum tahu apa-apa. Bukan “belum tahu apa-apa tentang C” — benar-benar belum tahu apa-apa. Jika Anda belum pernah menulis sebaris kode, belum pernah memasang kompilator, belum pernah membuka terminal, Andalah pembaca yang dibayangkan saat buku ini ditulis. Setiap istilah diperkenalkan dengan kata-kata sederhana sebelum dipakai, setiap program ditelusuri di kertas sebelum dijalankan, dan setiap program dalam buku ini sudah dijalankan sebelum keluarannya dicetak di sini.

Prinsip itu membuat buku ini tebal. Buku yang lebih tipis akan menganggap Anda sudah menguasai bagian-bagian yang sulit, dan buku semacam itu tidak berguna bagi orang yang belum menguasainya.

Satu gagasan, beberapa bahasa

Pemrograman adalah cara berpikir, bukan fitur milik satu bahasa, dan buku ini memperlakukannya demikian. Bahasa yang diajarkan pertama adalah **C**, karena C hampir tidak menyembunyikan apa pun: Anda melihat variabel yang tinggal di memori, alamat yang menunjuk ke sana, dan saat persis sebuah berkas ditulis ke disk. Sesudah dua belas bab C, buku ini beralih ke **C++** dan menunjukkan apa yang ditambahkan kelas, *reference*, dan operator yang di-*overload* kepada C yang sudah Anda kuasai. Dan setiap program, di kedua bagian, diberikan sekali lagi, lengkap dan siap dijalankan, dalam **Python**, mengerjakan hal yang sama dan mencetak keluaran yang sama, baris demi baris. Melihat satu gagasan ditulis dengan dua cara adalah jalan tercepat untuk memahami gagasan itu sendiri, bukan sekadar kebiasaan satu perkakas; dan itu memperlihatkan apa yang diam-diam dikerjakan Python untuk Anda — sehingga bahasa apa pun yang dipakai di tempat belajar atau bekerja Anda kelak, Anda sudah tahu apa yang ada di baliknya.

Isi buku ini

Bab 1 sampai 3 membangun landasan: apa yang dikerjakan komputer dan bagaimana program C menjadi program yang berjalan, lengkap dengan pemasangan VS Code, kompilator C, dan Python di Windows, macOS, dan Linux; program-program pertama Anda, variabel di memori, aritmetika, dan keputusan dengan `if`; lalu algoritma dan pseudokode, tiga struktur kontrol yang cukup untuk program apa pun, perulangan yang dikendalikan pencacah atau nilai *sentinel*, dan penghalusan bertahap dari atas ke bawah; lalu `for`, `do`…`while`, `switch`, `break` dan `continue`, operator logika, serta cara memilih struktur yang tepat. Bab 5 sampai 7 memberi program bentuknya: fungsi, pustaka standar, lingkup dan masa hidup, bilangan acak yang dapat diulang, dan rekursi; *array*, dipakai sebagai pencacah, dikirim ke fungsi, diurutkan dengan *bubble sort*, dicari secara linear dan dengan membagi dua, serta disusun dalam dua dimensi; dan *pointer*, dengan *pass-by-reference*, parameter keluaran, kekerabatan erat *array* dan *pointer*, `const`, serta *pointer* ke fungsi. Bab 9 sampai 11 membuat program mampu menangani data nyata: karakter, *string*, `<ctype.h>`, pustaka *string* beserta sisi tajamnya, seluruh kemampuan `printf`, dan masukan yang aman; struktur, `enum` dan `typedef`, memori sesuai kebutuhan dengan `malloc` dan `free`, serta *linked list* pertama; dan berkas, teks maupun biner, jebakan `feof`, membaca CSV baris demi baris, mode *append*, akses acak, dan menyimpan tanpa kehilangan. Bab 13 sampai 15 melangkah ke C++: *stream*, `std::string`, `std::vector`, *reference*, *overloading*, serta `new` dan `delete`; kelas, dengan konstruktor dan destruktor yang membuat setiap objek lahir dalam keadaan sah dan dibereskan secara otomatis, serta sekilas tentang pewarisan; dan *operator overloading*, dengan kelas uang dan kelas pecahan, diakhiri sebuah studi kasus utuh.

Bab 4, 8, 12, dan 16 adalah bab **Evaluasi dan Pembahasan**. Masing-masing menutup seperempat buku dengan menyelesaikan soal-soal utuh secara lengkap — setiap soal dinyatakan ulang, dirancang di kertas, diprogram, diuji, dijalankan, dan dijelaskan — dengan penyelesaian keliru yang menggoda diperlihatkan lalu dibongkar, dan varian "periksa diri Anda" di akhir setiap soal. Soal-soalnya merentang dari luas bangun datar, permainan tebak angka, dan batu-gunting-kertas, melalui statistik, Fibonacci dengan dua cara, palindrom, dan determinan matriks 3 × 3, sampai mengurutkan *linked list*, menumbuhkan *tree* dari berkas, dan jalur terpendek kuda catur, dan akhirnya bidak catur sebagai hierarki kelas, anagram, dan perkalian matriks.

Di sepanjang buku, satu usaha kecil tumbuh bab demi bab: **Profitina Laundry**, sebuah *laundry* di jalan yang ramai di Surabaya, milik Pak Hendra dengan Mbak Rina di meja depan. Kisahnya dimulai dari struk yang ditulis tangan dan orang yang lelah mengalikan berat dengan harga di kepalanya. Program kasirnya lalu tumbuh satu versi per bab: struk pertama dengan tarif minimum, perulangan yang melayani pelanggan sampai jam tutup, daftar harga untuk dinding, fungsi yang menyimpan setiap harga tepat di satu tempat, tagihan sehari dalam *array* yang diurutkan dan dicari, nama pada struk, pesanan sebagai rekaman dalam *linked list*, pekerjaan sehari yang selamat dari listrik padam, dan akhirnya toko utuh yang ditulis dalam C++ dengan kelas dan tipe uang yang tidak mungkin keliru. Setiap bab juga memiliki subbab *Memakainya Sungguhan* yang menunjukkan bagaimana gagasan yang sama terbawa ke usaha kecil lain dan masalah sehari-hari.

Setiap program dalam buku ini sudah dijalankan sebelum keluarannya dicetak. Program-program itu dikumpulkan, bab demi bab, dalam folder `Code` yang menyertai buku ini, dengan versi `c` (atau `cpp`) dan `python` berdampingan di setiap bab, beserta pengujiannya, dan README yang menjelaskan cara membangun serta menjalankan setiap program di terminal dan di VS Code, pada Windows, macOS, dan Linux. Setiap kode dalam buku ini adalah berkas di folder itu, karakter demi karakter.

Cara membacanya

Dengan editor terbuka dan kompilator siap. Ketikkan contohnya, jangan hanya menyalinnya; saat mengetik itulah kesalahan terjadi, dan dari kesalahan itulah pelajaran didapat. Sebelum menjalankan sebuah program, telusurilah di kertas — variabel di dalam kotaknya, nilai yang dipegangnya sesudah setiap baris — dan tebak apa yang akan dicetaknya; lalu jalankan dan bandingkan. Setelah sebuah contoh berjalan, rusakkan dengan sengaja lalu baca apa yang dikatakan kompilator atau seperti apa kegagalannya. Semua soal latihan di akhir tiap bab dilengkapi penyelesaian lengkap — gunakan sesudah usaha Anda sendiri, bukan sebagai penggantinya.

Gambar, tabel, kode, pseudokode, dan persamaan yang bernomor semuanya adalah tautan hidup: pada edisi elektronik, klik “Kode 5.3” atau “[4]” di bagian mana pun dan Anda langsung dibawa ke sana. Daftar Isi serta daftar-daftar di bagian depan buku bekerja dengan cara yang sama.

Tentang penulis

Irfan Subakti menggambarkan dirinya dalam satu baris — *Father | Educator | Entrepreneur | Bridging Tech & Nature: CS Lecturer × Crypto/Digital Business × AI/ML Specialist × Fruit Farmer | Storyteller* — ayah, pendidik, wirausahawan, penjembatan teknologi dan alam: dosen ilmu komputer, pelaku bisnis kripto dan digital, spesialis AI/ML, petani buah, dan pencerita. Setiap bagian dari baris itu ikut masuk ke dalam buku ini. Ia mengajar di Departemen Teknik Informatika ITS sejak 1998. Ia membimbing dan menguji tugas akhir mahasiswa sarjana, mengajar MLOps dalam program IBM SkillsBuild, dan membangun produk: ia mendirikan Profitina, platform visibilitas AI, dan Profitina Laundry, contoh berjalan buku ini, dipinjam dari keluarga studi kasusnya. Jauh dari papan ketik, ia merawat kebun buah dan durian keluarga — di sanalah ia belajar bahwa perangkat lunak yang baik, seperti kebun yang baik, ditumbuhkan dengan sabar, satu bagian yang terawat demi satu bagian. Jiwa penceritanya adalah alasan buku ini memilih kisah ketimbang daftar, setiap kali sebuah kisah bisa mengantarkan maknanya.

Ucapan terima kasih

Kepada para mahasiswa yang mempelajari versi-versi awal materi ini dan mengajukan pertanyaan yang menyingkapkan di mana sebuah penjelasan masih tipis, serta kepada rekan-rekan di ITS yang bersama-sama mengajarkan dan menyempurnakan mata kuliah ini dari tahun ke tahun. Standar buku ini juga berutang pada buku-buku teks klasik pemrograman C dan C++ serta dokumentasi resmi C, C++, dan Python, yang dikutip di setiap tempat penggunaannya.

Surabaya, September 2026



Irfan Subakti · yifana@gmail.com

Profitina

profitina.com

Bab 1

Komputer, C, dan Program Pertama Anda

Setiap program yang pernah Anda pakai, dari aplikasi kasir di minimarket sampai sistem operasi ponsel Anda, berawal dari beberapa baris teks biasa yang diketik seseorang, dikompilasi, lalu dijalankan. Bab ini membawa Anda dari nol sampai ke titik itu. Anda akan mempelajari apa yang sebenarnya dikerjakan komputer terhadap sebuah program, mengapa buku ini mengajarkan C dan Python berdampingan, dan bagaimana menyiapkan meja kerja yang gratis dan lengkap di Windows, macOS, atau Linux: Visual Studio Code, kompilator GCC, dan Python 3.14. Setelah itu Anda akan menulis, mengompilasi, dan menjalankan program pertama Anda serta memahami setiap barisnya: mencetak teks, menyimpan bilangan di dalam variabel, membaca apa yang diketik pengguna, berhitung dengan lima operator aritmetika, dan mengambil keputusan pertama dengan `if`. Bab ini ditutup dengan versi 0.1 program kasir Profitina Laundry, sebuah *laundry* kecil di lingkungan kampus yang kisahnya menemani seluruh buku, beserta versi perbaikannya yang menangani berat desimal dan tarif minimum. Di akhir bab Anda akan mampu menulis, menjalankan, dan menjelaskan program-program kecil dalam kedua bahasa, dan Anda akan mengenal segelintir kesalahan yang menjebak hampir setiap pemula.

1.1 Struk yang Ditulis Tangan

Profitina Laundry adalah *laundry* kecil di jalan yang ramai dekat sebuah universitas. Mahasiswa, pekerja kantor, dan keluarga datang membawa pakaian dalam kantong plastik; Mbak Rina, yang menjaga meja depan, menimbang setiap kantong dengan timbangan digital, menulis berat dan harganya di struk kertas, lalu memberi tahu pelanggan kapan cucian bisa diambil. Sebagian besar pelanggan memilih layanan sehari-hari, *cuci, kering, dan lipat*, dengan tarif Rp8.000 per kilogram. Sebagian lagi terburu-buru dan membayar Rp12.000 per kilogram untuk layanan ekspres satu hari; yang lain hanya perlu disetrika, Rp6.000 per kilogram.

Pada pagi yang sepi, cara ini berjalan baik. Pada hari Senin sesudah libur panjang, dengan sepuluh orang mengantre dan kantong menumpuk di meja, cara ini tidak lagi berjalan. Mbak Rina mengalikan 4,3 dengan 8.000 di kepalanya dan menulis Rp34.000, padahal seharusnya Rp34.400. Ia lupa bahwa kantong di bawah tiga kilogram ditagih sebagai tiga kilogram, aturan yang dibuat pemilik karena kantong kecil pun memakai mesin cuci sama lamanya dengan kantong penuh. Di akhir hari ia menjumlahkan empat puluh struk dengan kalkulator, dan totalnya tidak cocok dengan uang di laci. Tidak ada yang curang; hitungannya saja yang terlalu banyak bagi orang yang lelah dan terburu-buru.

Pekerjaan seperti inilah yang dikerjakan komputer lebih baik daripada manusia. Komputer tidak lelah, tidak lupa aturan begitu aturan itu dituliskan, dan mengalikan 4,3 dengan 8.000 dengan benar setiap kali. Akan tetapi komputer tidak berbuat apa-apa dengan sendirinya. Seseorang harus menuliskan, dalam bahasa yang dapat diikuti komputer, apa itu struk, bagaimana harga dihitung, dan apa yang terjadi pada kantong di bawah tiga kilogram. Prosedur tertulis itu disebut **program**, dan kegiatan menulisnya disebut **pemrograman**.

Buku ini mengajari Anda menulis program semacam itu dari awal sekali. Kita akan membangun kasir Profitina Laundry langkah demi langkah: di bab ini ia mencetak satu struk; di Bab 2 ia melayani seluruh antrean pelanggan; di Bab 3 ia memeriksa apa yang diketik kasir dan mencetak laporan harian; dan di bab-bab berikutnya ia akan mengingat pelanggan, mengurutkan pesanan, menyimpan semuanya ke berkas, dan akhirnya menjadi program C++ dengan kelas. Di sepanjang jalan Anda juga akan menjumpai masalah sehari-hari yang sama sekali tidak berhubungan dengan *laundry*, karena beberapa gagasan yang sama menyelesaikan semuanya.

1.2 Apa yang Dikerjakan Komputer

1.2.1 Perangkat keras dan perangkat lunak

Komputer adalah mesin yang mengerjakan dua hal dengan sangat cepat: menyimpan data, dan menjalankan instruksi yang mengubah data itu. Bagian fisik mesin disebut **perangkat keras** (*hardware*): prosesor, memori, media penyimpanan, layar, papan ketik, kartu jaringan. Instruksinya disebut **perangkat lunak** (*software*). Laptop tanpa perangkat lunak hanyalah kotak logam dan kaca yang menyala tanpa berbuat apa pun; laptop yang sama dengan program *spreadsheet* menjadi kalkulator, dan dengan program kasir menjadi mesin kasir.

Empat bagian perangkat keras penting untuk pemrograman, dan ada gunanya membayangkannya sejak awal:

- **Prosesor** (CPU, *central processing unit*) menjalankan instruksi satu demi satu, miliaran instruksi setiap detik. Setiap instruksi sangat kecil: menjumlahkan dua bilangan, membandingkan dua bilangan, menyalin bilangan dari satu tempat ke tempat lain, melompat ke instruksi lain.
- **Memori utama** (RAM, *random-access memory*) menampung program yang sedang berjalan dan data yang diolahnya. Memori adalah deretan panjang sel bernomor yang disebut *byte*; nomor sebuah sel adalah **alamatnya**. Memori cepat, tetapi melupakan segalanya saat listrik padam.
- **Media penyimpanan** (SSD atau *hard disk*) menyimpan berkas ketika listrik mati: program Anda, foto Anda, catatan *laundry*. Ia jauh lebih lambat daripada memori.
- **Perangkat masukan dan keluaran** menghubungkan mesin dengan manusia: papan ketik dan tetikus memasukkan data, layar dan pencetak mengeluarkan hasil.

Segala sesuatu di dalam mesin disimpan sebagai **bit**, sakelar yang mati (0) atau hidup (1). Delapan bit membentuk satu **byte**, yang dapat menyimpan 256 pola berbeda, cukup untuk satu huruf atau satu bilangan kecil. Bilangan yang lebih besar memerlukan beberapa byte; satu bilangan bulat di C biasanya memakai empat byte, yaitu 32 bit. Anda jarang akan berpikir dalam bit di buku ini, tetapi hal itu menjelaskan fakta yang mengejutkan setiap pemula: bilangan di dalam komputer punya batas. Bilangan bulat empat byte tidak dapat melebihi 2.147.483.647, dan Bab 4 menyelesaikan sebuah soal yang seluruhnya berkisar pada batas itu.

1.2.2 Program dan bahasa pemrograman

Prosesor hanya memahami **bahasa mesinnya** sendiri: instruksi bernomor yang ditulis sebagai pola bit. Beberapa byte seperti `01 d8` bisa berarti “tambahkan bilangan di register B ke bilangan di register A” pada satu jenis prosesor dan sesuatu yang sama sekali lain pada prosesor jenis lain. Tidak ada lagi yang menulis program seperti itu. Satu tingkat di atasnya adalah **bahasa rakitan** (*assembly language*), yang memberi setiap instruksi mesin nama pendek seperti `add` atau `mov`; ia dapat dibaca, tetapi tetap satu baris untuk setiap instruksi kecil, dan berbeda untuk setiap jenis prosesor.

Bahasa tingkat tinggi seperti C, C++, Java, atau Python memungkinkan kita menulis dalam istilah masalahnya, bukan istilah mesinnya. Kita menulis `total = weight * price;`, lalu sebuah program penerjemah mengubah baris itu menjadi puluhan instruksi mesin yang diperlukan. **Tabel 1.1** membandingkan ketiga tingkat pada satu tugas kecil.

Tabel 1.1 Perkalian yang sama pada tiga tingkat bahasa; baris bahasa mesin dan rakitan adalah contoh khas untuk prosesor x86-64.

Tingkat	Wujud perkalian berat dan harga	Siapa yang membacanya
Bahasa mesin	`8b 45 f8 0f af 45 fc 89 45 f4`	hanya prosesor
Bahasa rakitan	`mov eax, [weight]` / `imul eax, [price]` / `mov [total], eax`	para spesialis
Bahasa tingkat tinggi (C)	`total = weight * price;`	setiap pemrogram

Program tingkat tinggi punya dua keunggulan lagi. Ia **portabel**: program C yang sama dapat diterjemahkan untuk laptop Windows, MacBook dengan prosesor Apple, dan server Linux, lalu berjalan di ketiganya. Ia juga **mudah dibaca**: orang lain, atau Anda sendiri enam bulan kemudian, dapat memahami apa yang dilakukan `total = weight * price;` tanpa tahu apa pun tentang prosesornya.

1.3 C dan Python: Dua Bahasa, Satu Cara Berpikir

1.3.1 Mengapa C

C diciptakan oleh Dennis Ritchie di Bell Labs antara 1969 dan 1973 sebagai bahasa untuk menulis ulang sistem operasi UNIX [1]. Uraian klasiknya adalah buku Kernighan dan Ritchie [2], dan bahasa ini sekarang didefinisikan oleh sebuah standar internasional; buku ini memakai versi yang disebut C17, yang diterbitkan sebagai ISO/IEC 9899:2018 [3]. Lebih dari lima puluh tahun kemudian C masih ada di mana-mana: inti Linux, Windows, dan macOS, basis data yang menyimpan catatan dunia, interpreter yang menjalankan Python itu sendiri, serta komputer-komputer kecil di dalam mesin cuci, mobil, dan ponsel sebagian besar ditulis dalam C. Dalam indeks TIOBE bulan September 2026, yang memeringkat bahasa menurut seberapa sering dibicarakan dan dicari di web, C berada di peringkat kedua dengan nilai 10,28%, hanya di bawah Python dengan 17,76% [4], [5].

Untuk pelajaran pemrograman yang pertama, C punya keutamaan khusus: ia menyembunyikan sangat sedikit. Saat menyimpan bilangan, Anda memilih berapa byte yang dipakainya. Saat sebuah fungsi perlu mengubah variabel Anda, Anda memberikan alamatnya. Saat program Anda membaca di luar sebuah larik, tidak ada yang menghentikannya. Hal ini membuat C kurang nyaman dibandingkan Python, tetapi berarti bahwa di akhir buku ini Anda akan tahu apa yang dikerjakan setiap bahasa lain di balik layar.

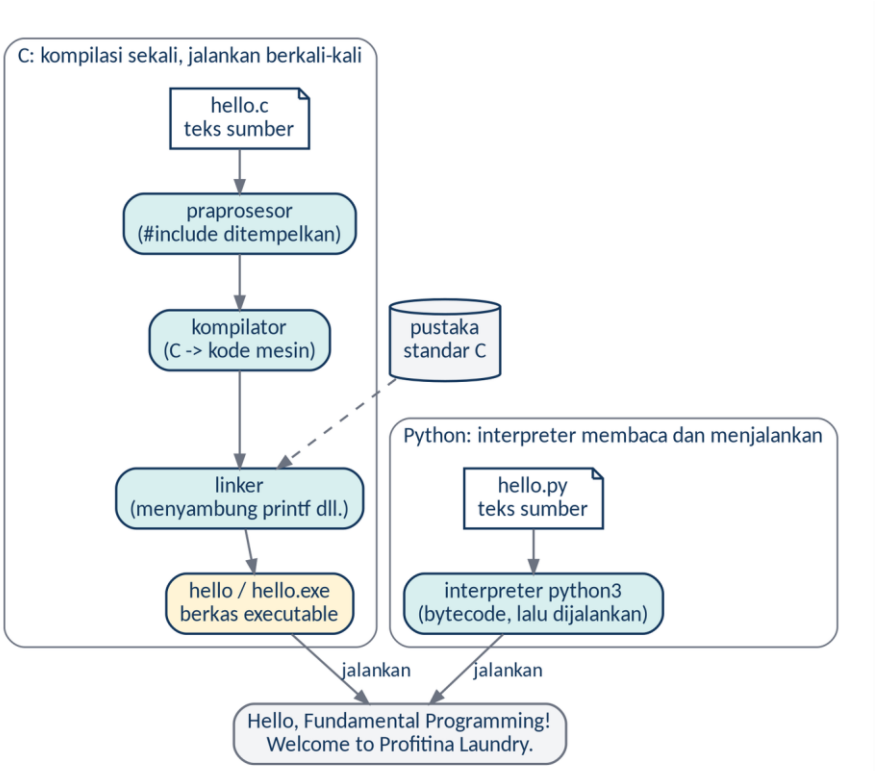
1.3.2 Mengapa juga Python

Python diciptakan oleh Guido van Rossum dan pertama kali dirilis pada 1991; pada 2026 ia adalah bahasa yang paling banyak dipakai di dunia [4], bahasa pertama di banyak sekolah, dan alat sehari-hari untuk analisis data dan kecerdasan buatan. Ia juga bahasa yang sangat berbeda dari C. Python mengurus pembukuan untuk Anda: bilangan tumbuh sebesar yang diperlukan, daftar memanjang saat Anda menambahkan isinya, dan memori dilepaskan secara otomatis.

Karena itu buku ini menyajikan setiap program dua kali, dalam C dan dalam Python, dan kedua versi selalu mencetak keluaran yang persis sama, karakter demi karakter. Membacanya berdampingan mengajarkan dua hal sekaligus. Pertama, *gagasan-gagasan* pemrograman (variabel, keputusan, perulangan, fungsi) sama dalam setiap bahasa; hanya ejaannya yang berubah. Kedua, *perbedaannya* mendidik: ketika Python diam-diam mengerjakan sesuatu yang di C harus Anda kerjakan sendiri, Anda belajar apa sesuatu itu. Versi Python saat ini adalah 3.14; rilis pemeliharaan terbarunya ketika buku ini ditulis adalah Python 3.14.7, yang dirilis pada 5 Agustus 2026 [6].

1.3.3 Dikompilasi dan diinterpretasi

Kedua bahasa juga berbeda dalam *cara* program berpindah dari teks menjadi tindakan, dan **Gambar 1.1** memperlihatkan kedua jalurnya. Program C **dikompilasi**: sebuah penerjemah yang disebut **kompilator** (*compiler*) membaca seluruh berkas sumber satu kali dan menulis berkas baru, **berkas *executable***, yang berisi bahasa mesin. Anda lalu menjalankan berkas itu sesering yang Anda mau, dan kompilator tidak lagi terlibat. Python **diinterpretasi**: program `python3`, sang **interpreter**, membaca berkas sumber Anda setiap kali program dijalankan, menerjemahkannya ke bentuk antara yang disebut *bytecode*, lalu langsung menjalankannya.



Gambar 1.1 Dua jalur dari teks sumber menjadi keluaran: program C melewati praprosesor, kompilator, dan *linker* satu kali lalu menjadi berkas *executable*; program Python dibaca dan dijalankan oleh interpreter setiap kali.

Kompilasi menambah satu langkah, tetapi imbalannya sepadan: kompilator memeriksa seluruh program sebelum program itu dijalankan, dan berkas *executable* yang jadi berjalan cepat. Perulangan yang menjumlahkan jutaan bilangan biasanya berjalan puluhan kali lebih cepat sebagai C terkompilasi daripada sebagai Python biasa, dan itulah sebabnya bagian-bagian Python yang harus cepat justru ditulis dalam C. Untuk program-program kecil di buku ini, keduanya sama-sama seketika.

1.4 Menyiapkan Meja Kerja

Seorang pemrogram memerlukan tiga alat: **editor** untuk menulis teks sumber, **kompilator** atau **interpreter** untuk mengubahnya menjadi tindakan, dan **terminal**, jendela teks tempat Anda mengetik perintah. Semua yang ada di subbab ini gratis, dan langkah-langkahnya sama untuk laptop baru maupun lama. **Tabel 1.2** mendaftar apa saja yang kita pasang.

Tabel 1.2 Meja kerja buku ini; semua alat gratis dan berjalan di Windows, macOS, dan Linux.

Alat	Versi (September 2026)	Fungsinya
Visual Studio Code	1.139 [7]	editor: mewarnai kode, menjalankannya, membantu menemukan kesalahan
GCC (Windows, Linux)	rilis terbaru 16.2 [8]; versi 9 ke atas mana pun dapat dipakai	kompilator C, perintah `gcc`
Apple clang (macOS)	ikut dalam Xcode Command Line Tools	kompilator C yang juga menjawab nama `gcc`
Python	3.14.7 [6]	interpreter, perintah `python3` (`python` atau `py` di Windows)
Ekstensi VS Code	*C/C++* dan *Python*, keduanya dari Microsoft	pewarnaan, pemeriksaan kesalahan, dan *debugging* di dalam editor

Setiap program C di buku ini dikompilasi dengan perintah yang sama, apa pun sistem operasinya:

```
gcc -std=c17 -Wall -Wextra -O2 hello.c -o hello
```

Bacalah dari kiri ke kanan: jalankan `gcc`, ikuti standar C17 (`-std=c17`), tampilkan semua peringatan yang lazim ditambah beberapa peringatan ekstra (`-Wall -Wextra`), optimalkan kode mesinnya (`-O2`), terjemahkan `hello.c`, dan tulis hasilnya ke berkas bernama `hello` (`-o hello`; di Windows berkasnya menjadi `hello.exe`). Program yang memakai pustaka matematika, yang pertama kali kita perlukan di Bab 3, menambahkan `-lm` di akhir perintah.

1.4.1 Windows 10 dan 11

Windows tidak memiliki kompilator C sendiri yang bekerja dari terminal, jadi kita mengambil GCC dari **MSYS2**, kumpulan alat yang dipakai para pemrogram Linux dan terawat dengan baik; jalur ini juga yang diuraikan dokumentasi VS Code [9], [10].

1. Unduh pemasangnya dari `https://www.msys2.org` lalu jalankan, dengan folder `C:\msys64`.
2. Dari menu Start, buka **MSYS2 UCRT64**. Namanya penting: UCRT64 adalah varian modern, dan paket yang dipasang dari terminal MSYS2 lain tidak akan cocok. Ketik perintah di bawah, tekan Enter untuk menerima semua anggota grupnya, lalu jawab `Y`.
   ```
   pacman -S --needed base-devel mingw-w64-ucrt-x86_64-toolchain
   ```
3. Beri tahu Windows di mana kompilatornya berada. Buka **Settings**, cari *environment variables*, pilih **Edit environment variables for your account**, pilih **Path**, klik **Edit**, lalu **New**, dan tambahkan `C:\msys64\ucrt64\bin`. Tutup semua terminal yang sedang terbuka sesudahnya: terminal hanya membaca Path saat mulai.
4. Untuk Python, pasang **Python install manager**, dari Microsoft Store atau dari `https://www.python.org/downloads/`. Sejak Python 3.14 inilah cara yang dianjurkan di Windows; pemasang lengkap yang lama masih ada tetapi sudah usang (*deprecated*) [11]. Lalu buka PowerShell dan ketik `py install 3.14`.
5. Pasang **Visual Studio Code** dari `https://code.visualstudio.com`, dengan tetap mencentang pilihan *Add to PATH*.

Sekarang buka jendela PowerShell yang *baru* dan periksa ketiganya:

```
gcc --version
python --version
code --version
```

Baris pertama semestinya menyebut GCC (MSYS2 menyediakan GCC 15 atau 16), baris kedua Python 3.14.7 atau rilis 3.14 yang lebih baru, dan baris ketiga versi VS Code, 1.139 atau lebih baru.

1.4.2 macOS

Apple menyediakan kompilator C gratis, **clang**, di dalam Xcode Command Line Tools; Anda tidak memerlukan aplikasi Xcode yang lengkap [12].

1. Buka **Terminal** (Applications → Utilities) dan ketik `xcode-select --install`, lalu klik *Install* pada jendela yang muncul.
2. Unduh pemasang macOS untuk Python 3.14 dari `https://www.python.org/downloads/` lalu jalankan. Pemasang ini menyediakan perintah `python3`. Jangan menghapus atau mengubah Python lama yang mungkin sudah ada di macOS.
3. Unduh **Visual Studio Code** dari `https://code.visualstudio.com`, ekstrak, lalu seret ke folder Applications. Jalankan, tekan Cmd+Shift+P, dan jalankan perintah *Shell Command: Install 'code' command in PATH*.

Periksa hasilnya di jendela Terminal baru dengan `gcc --version`, `python3 --version`, dan `code --version`. Di Mac, `gcc --version` menjawab "Apple clang": nama `gcc` dipertahankan agar perintah seperti milik kita tetap bekerja, dan clang menerima opsi yang sama.

1.4.3 Linux (Ubuntu, Debian, dan kerabatnya)

Linux adalah kandang GCC. Di terminal, ketik:

```
sudo apt update
sudo apt install build-essential gdb python3
```

Paket `build-essential` membawa `gcc`, pustaka standar C, dan alat-alat di sekitarnya. Ubuntu 26.04 LTS, yang dirilis pada April 2026, menyertakan GCC 15.2 dan Python 3.14 sebagai bawaan [13]; pada Ubuntu 24.04 yang lebih lama Anda mendapat Python 3.12, yang tetap menjalankan setiap program Python di buku ini, karena fitur terbaru yang kita pakai, pernyataan `match` di Bab 3, hadir sejak Python 3.10. VS Code tersedia sebagai paket `.deb` dari `https://code.visualstudio.com` atau melalui `sudo snap install code --classic`. Di Fedora, perintah padanannya adalah `sudo dnf install gcc gdb python3`.

1.4.4 Visual Studio Code dan dua ekstensinya

Jalankan VS Code, klik ikon **Extensions** di sebelah kiri (empat kotak kecil), lalu pasang dua ekstensi terbitan Microsoft: **C/C++**, yang mewarnai kode C, memeriksanya selagi Anda mengetik, dan memungkinkan *debugging*, serta **Python**, yang melakukan hal yang sama untuk Python dan sekaligus memasang *language server* Pylance [9], [14]. Kemudian pilih **File → Open Folder** dan buka folder tempat Anda menyimpan program, misalnya folder `Code_new/Chapter01` dari kode buku ini.

VS Code punya terminal di dalamnya: **View → Terminal** membuka satu terminal di bagian bawah jendela, langsung di dalam folder yang Anda buka. Di situlah kita mengetik setiap perintah di buku ini. Anda juga dapat menjalankan berkas Python yang sedang terbuka dengan tombol segitiga di kanan atas; untuk C, terminal lebih sederhana dan mengajarkan apa yang sesungguhnya terjadi, jadi kita memakainya di sepanjang buku.

> **WAWASAN KUNCI:** Simpan satu folder untuk setiap bab dan bukalah folder itu, bukan satu berkas saja, di VS Code. Terminal lalu dimulai di tempat yang benar, perintah-perintah di buku bekerja persis seperti tercetak, dan ekstensinya tahu berkas mana yang saling berkaitan.

1.4.5 Folder kode bab ini

Semua program bab ini ada di folder `Chapter01` pada kode buku. Folder `c` berisi program C, folder `python` berisi program Python dengan nama yang sama, dan folder `tests` berisi contoh masukan beserta keluaran persis yang harus dicetak setiap program; `README.md` mendaftar perintah untuk Windows, macOS, dan Linux. Mengetik sendiri programnya mengajarkan lebih banyak daripada menyalinnya, tetapi ketika sesuatu menolak bekerja, membandingkan berkas Anda dengan berkas buku adalah cara tercepat untuk menemukan perbedaannya.

1.5 Program Pertama Anda, Baris demi Baris

Menurut tradisi, program pertama mencetak sebuah salam. Program kita menyapa pembaca dan *laundry*-nya. **Kode 1.1** adalah seluruh program dalam C.

Kode 1.1 `hello.c`: program C pertama mencetak dua baris teks.

```c
/* hello.c - program pertama: mencetak dua baris teks */
#include <stdio.h>

int main(void)
{
    printf("Hello, Fundamental Programming!\n");
    printf("Welcome to Profitina Laundry.\n");
    return 0;   /* 0 memberi tahu sistem operasi bahwa semua lancar */
}
```

Enam gagasan tersembunyi dalam sembilan baris ini, dan semuanya kembali di setiap program C yang akan Anda tulis.

- Teks di antara `/*` dan `*/` adalah **komentar**. Kompilator mengabaikannya sama sekali; komentar ditulis untuk manusia. Baris pertama menyebut berkas apa ini, dan komentar sesudah `return 0;` menjelaskan baris yang mungkin membuat pemula bertanya-tanya. C juga menerima komentar yang diawali `//` dan berlangsung sampai akhir baris.
- `#include <stdio.h>` adalah instruksi bagi **praprosesor**, tahap pertama penerjemahan. Ia menempelkan berkas `stdio.h` (“*standard input and output*”), yang memberi tahu kompilator apa itu `printf` dan bagaimana ia harus dipanggil. Tanpa baris ini kompilator tidak mengenal `printf`.
- `int main(void)` memulai **fungsi** `main`. Setiap program C mulai berjalan pada baris pertama `main`, di mana pun letaknya di dalam berkas. Kata `int` menyatakan bahwa `main` mengembalikan sebuah bilangan bulat ketika selesai, dan `void` menyatakan bahwa ia tidak menerima apa pun.
- Kurung kurawal `{` dan `}` melingkupi **badan** `main`: pernyataan-pernyataan yang dijalankan satu per satu dari atas ke bawah.
- `printf("...\n");` mencetak teks di antara tanda petik ganda. Dua karakter `\n` adalah **urutan escape** (*escape sequence*) yang berarti “baris baru”: kursor melompat ke awal baris berikutnya. Setiap pernyataan diakhiri titik koma, seperti setiap kalimat diakhiri titik.
- `return 0;` mengakhiri `main` dan melaporkan bilangan 0 kepada sistem operasi, yang menurut kesepakatan berarti “semua berjalan baik”. Bilangan lain berarti ada yang salah, dan skrip yang menjalankan program secara otomatis memeriksa bilangan ini.

Kode 1.2 adalah program yang sama dalam Python.

Kode 1.2 `hello.py`: dua baris yang sama dalam Python.

```
# hello.py - program pertama: mencetak dua baris teks
print("Hello, Fundamental Programming!")
print("Welcome to Profitina Laundry.")
```

Versi Python lebih pendek karena Python mengerjakan beberapa hal untuk Anda. Tidak ada `#include`, karena `print` selalu tersedia. Tidak ada `main`: Python cukup menjalankan berkas dari baris pertama sampai terakhir. Tidak ada kurung kurawal dan titik koma, karena di Python akhir baris mengakhiri pernyataan. Dan `print` menambahkan baris baru dengan sendirinya. Komentar di Python diawali `#`.

Untuk menjalankan program C, buka terminal di folder `c`, kompilasi, lalu jalankan berkas *executable*-nya. Di Linux dan macOS Anda mengetik `./hello`, dengan `./` berarti "di folder ini"; di Windows PowerShell Anda mengetik `.\hello.exe`:

```
gcc -std=c17 -Wall -Wextra -O2 hello.c -o hello
./hello
```

Untuk menjalankan program Python, buka terminal di folder `python` dan ketik `python3 hello.py` (di Windows `python hello.py` atau `py hello.py`). Keduanya mencetak dua baris yang sama:

```
Hello, Fundamental Programming!
Welcome to Profitina Laundry.
```

Jika dua baris ini muncul, meja kerja Anda berfungsi, dan Anda telah menulis, menerjemahkan, dan menjalankan sebuah program. Semua hal lain di buku ini dibangun di atas momen ini.

1.6 Dari Teks Sumber menjadi Program yang Berjalan

1.6.1 Apa yang sebenarnya dikerjakan `gcc`

Satu perintah `gcc` menyembunyikan empat langkah, dan pesan kesalahan yang akan Anda jumpai menyebut nama-nama langkah itu, jadi ada gunanya melihatnya sekali. **Gambar 1.1** telah menampilkannya sebagai kotak-kotak. **Praprosesor** menangani baris-baris yang diawali `#`: ia menempelkan isi `stdio.h` dan membuang komentar. **Kompilator** menerjemahkan teks C yang dihasilkan menjadi bahasa mesin untuk prosesor Anda; di sinilah sebagian besar kesalahan ditemukan. **Assembler** mengubah bahasa mesin itu menjadi *object file*. Akhirnya **linker** menggabungkan *object file* Anda dengan potongan-potongan **pustaka standar C** yang dipakai program Anda, misalnya kode `printf`, yang tidak Anda tulis tetapi harus ada di dalam berkas *executable*. Hasilnya satu berkas yang dapat dimuat sistem operasi ke memori lalu dijalankan.

1.6.2 Ketika ada yang salah: galat dan peringatan

Semua orang pernah salah ketik, dan kompilator adalah yang pertama menyadarinya. Misalkan kita lupa titik koma di akhir baris `printf` yang pertama. GCC menolak menerjemahkan berkas itu dan berkata:

```
hello.c: In function 'main':
hello.c:6:48: error: expected ';' before 'printf'
    6 |     printf("Hello, Fundamental Programming!\n")
      |                                                ^
      |                                                ;
    7 |     printf("Welcome to Profitina Laundry.\n");
      |     ~~~~~~
```

Bacalah pesan galat (*error*) dari atas. `hello.c:6:48` berarti "berkas `hello.c`, baris 6, kolom 48". Kata `error` berarti tidak ada berkas *executable* yang dihasilkan. Kompilator bahkan menunjukkan

tempat karakter yang hilang. Perhatikan bahwa ia mengeluh tentang tempat *sebelum* `printf` di baris 7: galat sering dilaporkan sedikit sesudah kesalahan yang sebenarnya, karena kompilator baru menyadari masalah ketika kata berikutnya tidak cocok. Jadi bila sebuah pesan menunjuk baris yang tampak sempurna, periksalah baris di atasnya.

Kesalahan serupa di Python, kurung tutup yang hilang, dilaporkan ketika interpreter membaca berkas, sebelum satu baris pun dijalankan:

```
  File "hello.py", line 1
    print("Hello, Fundamental Programming!"
          ^
SyntaxError: '(' was never closed
```

Peringatan (*warning*) berbeda dari galat: program tetap diterjemahkan, tetapi kompilator melihat sesuatu yang sah menurut C namun hampir pasti bukan yang Anda maksud. Opsi `-Wall -Wextra` menyalakan peringatan-peringatan yang berguna, dan buku ini memperlakukan setiap peringatan sebagai galat yang menyamar. Semua program di buku ini terkompilasi tanpa satu peringatan pun; program-program itu juga dijalankan dengan *sanitizer* GCC (`-fsanitize=address,undefined`), yang menghentikan program begitu ia membaca memori yang bukan miliknya atau menghitung sesuatu yang dibiarkan tak terdefinisi oleh standar C.

> **PERINGATAN:** Jangan pernah mengabaikan peringatan karena "programnya tetap jalan". Program yang berjalan belum tentu program yang benar. Di Subbab 1.13 Anda akan melihat `&` yang terlupa yang hanya menghasilkan peringatan, lalu program yang mencetak sampah.

Ada jenis kesalahan ketiga yang tidak dapat ditemukan kompilator maupun interpreter: **kesalahan logika**. Programnya sah dan berjalan, tetapi menghitung hal yang salah, misalnya karena Anda mengetik `+` padahal maksudnya `*`. Hanya pengujian yang menemukan kesalahan logika, dan itulah sebabnya setiap program di buku ini dijalankan pada contoh-contoh yang jawaban benarnya sudah kita ketahui lebih dahulu.

1.7 Variabel dan Memori

1.7.1 Kotak yang bernama

Program kasir harus mengingat banyak hal: berat kantong, harga per kilogram, totalnya. **Variabel** adalah tempat bernama di memori yang menyimpan satu nilai. **Kode 1.3** membaca dua bilangan bulat dari papan ketik dan mencetak jumlahnya.

Kode 1.3 `add_two.c`: membaca dua bilangan bulat lalu menjumlahkannya.

```c
/* add_two.c - membaca dua bilangan bulat dan mencetak jumlahnya */
#include <stdio.h>

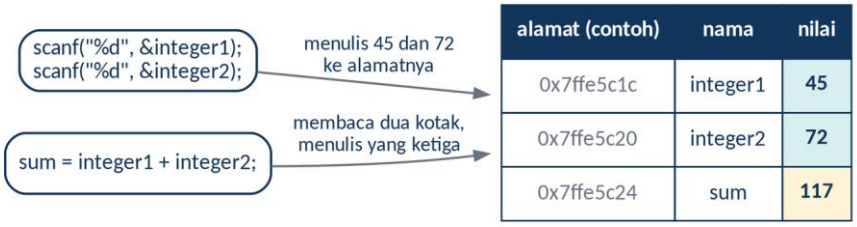
int main(void)
{
    int integer1;   /* bilangan pertama yang diketik pengguna */
    int integer2;   /* bilangan kedua yang diketik pengguna */
    int sum;        /* jumlah keduanya */

    printf("Enter first integer: ");
    if (scanf("%d", &integer1) != 1) return 1;   /* & berarti "alamat dari" */
    printf("Enter second integer: ");
    if (scanf("%d", &integer2) != 1) return 1;

    sum = integer1 + integer2;
    printf("Sum is %d\n", sum);
    return 0;
}
```

Baris `int integer1;` adalah sebuah **deklarasi**. Ia meminta sebuah kotak di memori yang cukup besar untuk satu `int`, yaitu bilangan bulat (*integer*), dan memberi kotak itu nama `integer1`. Nama boleh berisi huruf, angka, dan garis bawah, tidak boleh diawali angka, dan membedakan huruf besar-kecil: `total` dan `Total` adalah dua variabel yang berbeda. Pilih nama yang menyatakan isi kotaknya; `weightKg` lebih baik daripada `w`.

Tiga fakta tentang setiap variabel itu penting, dan **Gambar 1.2** menampilkannya sesudah program dijalankan: **namanya**, yang dipakai program; **nilainya**, isi kotak saat itu; dan **alamatnya**, nomor sel memori tempat kotak itu dimulai. Alamat di gambar hanyalah contoh; alamatnya berbeda di setiap komputer, bahkan di setiap kali program dijalankan.



Gambar 1.2 Tiga variabel sesudah `add_two` dijalankan: `scanf` menulis 45 dan 72 ke alamat `integer1` dan `integer2`, lalu penugasan membaca keduanya dan menulis 117 ke `sum`.

1.7.2 Membaca masukan dengan `scanf`

`printf` menampilkan sebuah nilai; `scanf` melakukan kebalikannya dan memasukkan apa yang diketik pengguna ke sebuah variabel. Dalam `scanf("%d", &integer1)`, **format** `"%d"` berarti "baca sebuah bilangan bulat yang ditulis dalam desimal", dan `&integer1` adalah **alamat** variabel itu, dibaca "alamat dari `integer1`". `scanf` memerlukan alamat karena ia harus *menulis* ke dalam kotak, dan sebuah fungsi hanya dapat menulis ke kotak Anda bila Anda memberi tahu di mana kotak itu berada. `printf`, yang hanya membaca nilainya, menerima `integer1` tanpa `&`.

`scanf` juga melaporkan berapa nilai yang berhasil dibacanya. Jika pengguna mengetik `abc` alih-alih bilangan, ia tidak membaca apa pun dan melaporkan 0. Program kita memeriksanya dengan `if (scanf(...) != 1) return 1;`, yang berarti "jika yang terbaca tidak tepat satu nilai, hentikan program dan laporkan kegagalan". Anda akan berkenalan dengan `if` secara layak di Subbab 1.9; untuk sementara bacalah baris itu sebagai pengaman. Selain merupakan kebiasaan yang baik, pemeriksaan ini diperlukan agar kompilasi bersih: di banyak sistem Linux GCC memberi peringatan bila hasil `scanf` diabaikan.

Baris `sum = integer1 + integer2;` adalah sebuah **penugasan** (*assignment*). Ruas kanan dihitung lebih dahulu dengan nilai-nilai di kedua kotak, lalu hasilnya disimpan ke kotak yang disebut di ruas kiri. Tanda `=` di C tidak berarti "sama dengan"; artinya "menjadi". Pernyataan seperti `count = count + 1;`, yang mustahil sebagai matematika, sangat masuk akal sebagai instruksi: ambil nilai `count`, tambah satu, lalu simpan hasilnya kembali ke `count`.

Berikut satu kali jalan dengan pengguna mengetik 45 dan 72:

```
Enter first integer: 45
Enter second integer: 72
Sum is 117
```

Sebelum `scanf` yang pertama, program mencetak sebuah **prompt**, pesan pendek yang memberi tahu pengguna apa yang harus diketik. Program tanpa prompt menunggu dalam diam, dan pengguna tidak tahu bahwa program mengharapkan sesuatu. Perhatikan bahwa prompt itu tidak diakhiri `\n`, sehingga kursor menunggu di baris yang sama.

Kode 1.4 adalah versi Python-nya.

Kode 1.4 `add_two.py`: program yang sama dalam Python.

```python
# add_two.py - membaca dua bilangan bulat dan mencetak jumlahnya
integer1 = int(input("Enter first integer: "))    # teks ketikan -> bilangan bulat
integer2 = int(input("Enter second integer: "))

total = integer1 + integer2    # "sum" sudah menjadi nama fungsi Python
print(f"Sum is {total}")
```

Python tidak mengenal deklarasi: sebuah variabel muncul begitu sesuatu ditugaskan kepadanya untuk pertama kali. Fungsi `input` mencetak prompt-nya, menunggu pengguna menekan Enter, lalu mengembalikan apa yang diketik sebagai *teks*. Teks `"45"` bukanlah bilangan, jadi `int(...)` mengubahnya menjadi bilangan bulat 45. Jika pengguna mengetik sesuatu yang bukan bilangan bulat, `int` menghentikan program dengan pesan galat bernama `ValueError`; Bab 9 menunjukkan cara menangani masukan semacam itu dengan anggun dalam kedua bahasa. Hasilnya kita namai `total`, bukan `sum`, karena `sum` sudah menjadi nama fungsi bawaan Python.

Teks di antara `f"` dan `"` adalah sebuah **f-string**: Python mengganti setiap `{...}` di dalamnya dengan nilai ekspresi di antara kurung kurawal. `f"Sum is {total}"` menjadi `Sum is 117`. Perannya sama dengan `%d` pada `printf`.

1.7.3 Empat tipe dasar

Setiap variabel C memiliki **tipe**, yang menetapkan berapa byte yang dipakai kotaknya dan jenis nilai apa yang dapat ditampungnya. Bab ini memakai empat:

- `int`, bilangan bulat, biasanya empat byte, dari −2.147.483.648 sampai 2.147.483.647;
- `double`, bilangan dengan bagian pecahan seperti 4,3, delapan byte, dengan kira-kira 15 sampai 16 angka penting;
- `char`, satu karakter seperti `'A'`, satu byte, disimpan sebagai bilangan kecil;
- dan, di beberapa tempat, `long long`, bilangan bulat delapan byte yang mencapai sekitar $9{,}2 \times 10^{18}$; Bab 4 memerlukannya.

Kode 1.5 memperlihatkan tiga yang pertama dan batas-batas `int`.

Kode 1.5 `types.c`: sebuah `int`, sebuah `double`, dan sebuah `char`, serta jangkauan `int`.

```c
/* types.c - empat tipe dasar dan cara printf menampilkannya */
#include <stdio.h>
#include <limits.h>

int main(void)
{
    int kilos = 5;           /* bilangan bulat */
    double price = 8000.0;   /* bilangan yang punya bagian pecahan */
    char grade = 'A';        /* satu karakter, disimpan sebagai bilangan kecil */

    printf("kilos = %d\n", kilos);
    printf("price = %.2f\n", price);
    printf("bill  = %.2f\n", kilos * price);   /* int kali double = double */
    printf("grade = %c, stored as the number %d\n", grade, grade);
    printf("largest int  = %d\n", INT_MAX);
    printf("smallest int = %d\n", INT_MIN);
    return 0;
}
```

```
kilos = 5
price = 8000.00
bill  = 40000.00
grade = A, stored as the number 65
largest int  = 2147483647
smallest int = -2147483648
```

Setiap tipe punya **spesifikasi konversi** sendiri pada `printf`: `%d` untuk `int`, `%f` untuk `double` (`%.2f` berarti "dengan dua angka di belakang titik desimal"), dan `%c` untuk `char`. Ketika `grade` dicetak dengan `%d`, kita melihat bilangan 65 yang sesungguhnya disimpan komputer untuk huruf A; penomoran ini disebut ASCII. Konstanta `INT_MAX` dan `INT_MIN` berasal dari *header* `limits.h`. Pada ekspresi `kilos * price`, sebuah `int` dikalikan dengan `double`; C lebih dahulu mengubah `int` menjadi `double`, sehingga hasilnya memiliki bagian pecahan. Perhatikan juga bahwa program mencetak pecahan dengan titik, bukan koma: C dan Python selalu memakai titik desimal, seperti bahasa Inggris.

Kode 1.6 memberikan nilai-nilai yang sama dalam Python.

Kode 1.6 `types.py`: nilai-nilai yang sama dalam Python, yang tidak punya tipe `char` dan tidak membatasi `int`.

```python
# types.py - nilai yang sama di Python
kilos = 5            # int: bilangan bulat (int Python tidak pernah meluap)
price = 8000.0       # float: bilangan 64-bit yang sama dengan double di C
grade = "A"          # string sepanjang satu; Python tidak punya tipe char

print(f"kilos = {kilos}")
print(f"price = {price:.2f}")
print(f"bill  = {kilos * price:.2f}")
print(f"grade = {grade}, stored as the number {ord(grade)}")
print(f"largest int  = {2**31 - 1}")     # batas int 32-bit di C
print(f"smallest int = {-2**31}")
```

Python punya lebih sedikit tipe, dan perilakunya berbeda. `int` di Python tidak punya batas atas: `2**100` dihitung dengan tepat. `float` di Python persis sama dengan bilangan delapan byte `double` di C, dan itulah sebabnya kedua program di buku ini selalu sepakat soal pecahan. Python juga tidak punya tipe karakter tersendiri: satu huruf hanyalah string sepanjang satu, dan `ord` memberikan nomornya. Untuk mencetak batas `int` di C, program Python menghitungnya: `2**31 - 1` berarti 2 pangkat 31, dikurangi 1.

1.8 Aritmetika

1.8.1 Lima operator dan satu jebakan

C memiliki lima operator aritmetika: `+` (penjumlahan), `-` (pengurangan), `*` (perkalian), `/` (pembagian), dan `%` (sisa pembagian, sering disebut *modulus*). Tiga yang pertama berperilaku seperti di sekolah. Pembagian memerlukan kehati-hatian. Bila kedua ruas `/` adalah bilangan bulat, C melakukan **pembagian bilangan bulat** (*integer division*): ia menghitung bagian bulat hasil bagi dan membuang pecahannya. Jadi `7 / 3` bernilai 2, bukan 2,333. Operator `%` memberikan apa yang dibuang oleh pembagian bulat: `7 % 3` bernilai 1, karena 7 adalah dua kali tiga dengan sisa satu. Kedua operator selalu cocok satu sama lain:

$$a = (a/b) \cdot b + a \,\%\, b \tag{1.1}$$

Untuk $a = 7$ dan $b = 3$, Persamaan (1.1) terbaca $7 = 2 \cdot 3 + 1$. Begitulah kasir menghitung kembalian, jam mengubah menit menjadi jam, dan permainan memutuskan bahwa "setiap giliran ketiga" telah tiba: `turn % 3 == 0` benar tepat pada giliran 3, 6, 9, dan seterusnya. **Kode 1.7** mencoba semua ini.

Kode 1.7 `arithmetic.c`: lima operator, pembagian bilangan bulat, dan presedensi.

```c
/* arithmetic.c - lima operator aritmetika dan presedensinya */
#include <stdio.h>

int main(void)
{
    printf("7 + 3 = %d\n", 7 + 3);
    printf("7 - 3 = %d\n", 7 - 3);
    printf("7 * 3 = %d\n", 7 * 3);
    printf("7 / 3 = %d\n", 7 / 3);        /* pembagian bilangan bulat */
    printf("7 %% 3 = %d\n", 7 % 3);       /* sisa pembagian */
    printf("-7 / 2 = %d\n", -7 / 2);      /* C memotong ke arah nol */
    printf("-7 %% 2 = %d\n", -7 % 2);
    printf("7.0 / 2 = %.2f\n", 7.0 / 2);  /* satu double membuatnya tepat */
    printf("2 + 3 * 4 = %d\n", 2 + 3 * 4);
    printf("(2 + 3) * 4 = %d\n", (2 + 3) * 4);

    int a = 80, b = 90, c = 85;
    printf("a + b + c / 3 = %d\n", a + b + c / 3);
    printf("(a + b + c) / 3 = %d\n", (a + b + c) / 3);
    return 0;
}
```

```
7 + 3 = 10
7 - 3 = 4
7 * 3 = 21
7 / 3 = 2
7 % 3 = 1
-7 / 2 = -3
-7 % 2 = -1
7.0 / 2 = 3.50
2 + 3 * 4 = 14
(2 + 3) * 4 = 20
a + b + c / 3 = 198
(a + b + c) / 3 = 85
```

Bacalah keluarannya baris demi baris. `7 / 3` menghasilkan 2 dan `7 % 3` menghasilkan 1, seperti dijanjikan Persamaan (1.1). Untuk bilangan negatif C memotong pecahan *ke arah nol*: −3,5 menjadi −3, dan sisanya lalu harus −1 agar Persamaan (1.1) tetap berlaku: $-7 = (-3) \cdot 2 + (-1)$. Begitu salah satu ruas `/` berupa `double`, seperti pada `7.0 / 2`, pembagian mempertahankan

pecahannya dan menghasilkan 3,5. Untuk mencetak tanda persen dengan `printf` Anda harus menuliskannya dua kali, `%%`, karena satu `%` memulai spesifikasi konversi.

1.8.2 Presedensi: siapa lebih dahulu

Bila sebuah ekspresi berisi beberapa operator, C tidak sekadar bekerja dari kiri ke kanan. Seperti aritmetika sekolah, ia punya aturan **presedensi** (*precedence*), yang diringkas di **Tabel 1.3**. Perkalian, pembagian, dan sisa pembagian mengikat lebih kuat daripada penjumlahan dan pengurangan, sehingga `2 + 3 * 4` bernilai 14, bukan 20. Tanda kurung mengalahkan aturan itu: `(2 + 3) * 4` bernilai 20.

Tabel 1.3 Presedensi operator yang sudah kita kenal; bila ragu, tambahkan tanda kurung.

Operator	Arti	Urutan
`( )`	tanda kurung	pertama; pasangan terdalam lebih dahulu
`* / %`	perkalian, pembagian, sisa pembagian	berikutnya; dari kiri ke kanan bila muncul beberapa
`+ -`	penjumlahan, pengurangan	berikutnya; dari kiri ke kanan
`=`	penugasan	terakhir; nilai di kanan masuk ke variabel di kiri

Dua baris terakhir keluaran menampilkan *bug* presedensi yang paling terkenal. Untuk merata-ratakan tiga nilai 80, 90, dan 85, seorang pemula menulis `a + b + c / 3`. C menghitung `c / 3` lebih dahulu (28, dengan pembagian bulat), lalu menambahkan 80 dan 90, dan mencetak 198, yang bahkan tidak terletak di antara ketiga nilai. Bentuk yang benar, `(a + b + c) / 3`, menghasilkan 85. Tanda kurung tidak memakan biaya apa pun, dan sekaligus memberi tahu pembaca apa maksud Anda.

1.8.3 Aritmetika yang sama di Python

Kode 1.8 mencetak baris-baris yang persis sama dalam Python, dan dengan begitu menunjukkan di mana kedua bahasa berpisah jalan.

Kode 1.8 `arithmetic.py`: hasil yang sama di Python, yang memerlukan `//`, `int(...)`, dan `math.fmod` untuk meniru C.

```python
# arithmetic.py - operator yang sama di Python, dicetak seperti program C
import math

print(f"7 + 3 = {7 + 3}")
print(f"7 - 3 = {7 - 3}")
print(f"7 * 3 = {7 * 3}")
print(f"7 / 3 = {7 // 3}")              # // adalah pembagian bilangan bulat
print(f"7 % 3 = {7 % 3}")               # sisa pembagian
print(f"-7 / 2 = {int(-7 / 2)}")        # potong ke arah nol, seperti C
print(f"-7 % 2 = {int(math.fmod(-7, 2))}")   # sisa ala C; -7 % 2 menghasilkan 1
print(f"7.0 / 2 = {7.0 / 2:.2f}")       # / selalu menghasilkan pecahan
print(f"2 + 3 * 4 = {2 + 3 * 4}")
print(f"(2 + 3) * 4 = {(2 + 3) * 4}")

a, b, c = 80, 90, 85
print(f"a + b + c / 3 = {a + b + c // 3}")
print(f"(a + b + c) / 3 = {(a + b + c) // 3}")
```

Di Python, `/` *selalu* menghasilkan pecahan: `7 / 3` bernilai 2.3333333333333335. Pembagian bilangan bulat punya operator sendiri, `//`. Untuk bilangan negatif Python membulatkan *ke bawah*, ke arah minus tak hingga, bukan ke arah nol: `-7 // 2` bernilai −4 dan `-7 % 2` bernilai 1 di Python, sedangkan C memberi −3 dan −1. Kedua pilihan memenuhi Persamaan (1.1); keduanya hanya sepakat pada hasil bagi yang berbeda. Untuk mencetak jawaban versi C, program Python

memakai `int(-7 / 2)`, yang memotong ke arah nol, dan `math.fmod`, yang menghitung sisa dengan cara C [15], [16]. Untuk bilangan positif, yang hampir selalu terjadi di *laundry*, kedua bahasa sepakat.

> **PERINGATAN:** Rumus satu baris yang menggoda, `half = 1 / 2 * kilos;`, menghasilkan 0 di C untuk nilai `kilos` berapa pun, karena `1 / 2` adalah pembagian bulat dan dihitung lebih dahulu. Tulislah `kilos / 2.0` atau `0.5 * kilos`. Subbab 1.13 menunjukkan kekeliruan ini saat dijalankan.

1.9 Keputusan dengan `if`

Program menjadi menarik ketika ia dapat memilih. Pernyataan `if` menjalankan sebuah pernyataan hanya bila sebuah **kondisi** bernilai benar. Kondisi dibangun dengan enam operator pembanding `==` (sama dengan), `!=` (tidak sama dengan), `<`, `>`, `<=`, dan `>=`. **Kode 1.9** membandingkan dua bilangan dengan segala cara.

Kode 1.9 `compare.c`: enam operator pembanding, masing-masing menjaga satu `printf`.

```c
/* compare.c - enam operator pembanding dengan if */
#include <stdio.h>

int main(void)
{
    int num1, num2;

    printf("Enter two integers: ");
    if (scanf("%d %d", &num1, &num2) != 2) return 1;

    if (num1 == num2)
        printf("%d is equal to %d\n", num1, num2);
    if (num1 != num2)
        printf("%d is not equal to %d\n", num1, num2);
    if (num1 < num2)
        printf("%d is less than %d\n", num1, num2);
    if (num1 > num2)
        printf("%d is greater than %d\n", num1, num2);
    if (num1 <= num2)
        printf("%d is less than or equal to %d\n", num1, num2);
    if (num1 >= num2)
        printf("%d is greater than or equal to %d\n", num1, num2);
    return 0;
}
```

```
Enter two integers: 3 7
3 is not equal to 7
3 is less than 7
3 is less than or equal to 7
```

Format `"%d %d"` membaca dua bilangan dari satu baris, dan `scanf` melaporkan 2 bila keduanya terbaca. Setiap `if` menguji kondisinya; bila kondisi benar, pernyataan di bawahnya dijalankan, dan bila salah, pernyataan itu dilewati. Dari enam kondisi, tiga benar untuk 3 dan 7, dan tepat ketiga baris itulah yang dicetak. Indentasi baris `printf` hanya untuk manusia; C akan menerima semuanya dalam satu baris, tetapi indentasi memperlihatkan sekilas pernyataan mana milik `if` yang mana.

Di C, sebuah kondisi hanyalah bilangan: 0 berarti salah dan nilai lain berarti benar. Perbandingan seperti `3 < 7` menghasilkan `int` 1, dan `3 > 7` menghasilkan 0.

Kode 1.10 `compare.py`: perbandingan yang sama dalam Python.

```python
# compare.py - enam operator pembanding dengan if
words = input("Enter two integers: ").split()   # "3 7" -> ["3", "7"]
num1 = int(words[0])
num2 = int(words[1])

if num1 == num2:
    print(f"{num1} is equal to {num2}")
if num1 != num2:
    print(f"{num1} is not equal to {num2}")
if num1 < num2:
    print(f"{num1} is less than {num2}")
if num1 > num2:
    print(f"{num1} is greater than {num2}")
if num1 <= num2:
    print(f"{num1} is less than or equal to {num2}")
if num1 >= num2:
    print(f"{num1} is greater than or equal to {num2}")
```

Kode 1.10 menunjukkan bentuk Python-nya. Kondisi diikuti titik dua, dan pernyataan milik `if` *diindentasi*: di Python, indentasi bukan hiasan melainkan tata bahasa, dan empat spasi per tingkat adalah kesepakatan universal. Untuk membaca dua bilangan dari satu baris, program memanggil `split` pada teks yang diketik, yang memotong `"3 7"` menjadi daftar `["3", "7"]`; `words[0]` adalah elemen pertamanya dan `words[1]` elemen keduanya, dan masing-masing diubah dengan `int`.

> **PERINGATAN:** `=` menugaskan dan `==` membandingkan. Di C, `if (x = 5)` itu sah: ia menyimpan 5 ke `x` lalu menguji 5 itu, yang bernilai benar, sehingga pernyataannya selalu dijalankan. GCC memperingatkannya dengan `-Wall`; Python menolaknya mentah-mentah. Bacalah `==` sebagai "sama dengan" dan `=` sebagai "menjadi", dan keduanya tak akan tertukar lagi.

1.10 Kasir Profitina Laundry v0.1

Kini kita punya semua bagian untuk versi pertama kasir *laundry*: variabel untuk berat, harga, dan total; `scanf` untuk membaca berat; satu perkalian; dan `printf` untuk struknya. **Kode 1.11** merangkai semuanya.

Kode 1.11 `receipt.c`: kasir v0.1 mencetak struk untuk layanan cuci, kering, dan lipat.

```c
/* receipt.c - kasir Profitina Laundry v0.1: satu struk */
#include <stdio.h>

int main(void)
{
    int weightKg;              /* berat cucian, kilogram bulat */
    int pricePerKg = 8000;     /* cuci, kering & lipat: rupiah per kilogram */
    int total;                 /* jumlah yang harus dibayar, dalam rupiah */

    printf("=== Profitina Laundry - cashier v0.1 ===\n");
    printf("Weight (kg): ");
    if (scanf("%d", &weightKg) != 1) return 1;

    total = weightKg * pricePerKg;

    printf("\n--- RECEIPT ---\n");
    printf("Service  : Wash, dry & fold\n");
    printf("Weight   : %d kg\n", weightKg);
    printf("Price/kg : Rp %d\n", pricePerKg);
    printf("TOTAL    : Rp %d\n", total);
    return 0;
}
```

Setiap variabel membawa komentar yang menyebut satuannya: kilogram, rupiah per kilogram, rupiah. Uang disimpan sebagai bilangan bulat rupiah, tidak pernah sebagai pecahan, karena jumlah rupiah tidak memiliki sen dan bilangan bulat itu tepat. Baris `printf("\n--- RECEIPT ---\n");` diawali `\n` untuk menyisakan satu baris kosong di antara masukan dan struk. Berikut pelanggan dengan lima kilogram:

```
=== Profitina Laundry - cashier v0.1 ===
Weight (kg): 5

--- RECEIPT ---
Service  : Wash, dry & fold
Weight   : 5 kg
Price/kg : Rp 8000
TOTAL    : Rp 40000
```

Kode 1.12 adalah versi Python-nya. `print()` tanpa isi di antara tanda kurung mencetak baris kosong, pasangan `\n` di awal teks pada versi C.

Kode 1.12 `receipt.py`: kasir v0.1 dalam Python.

```python
# receipt.py - kasir Profitina Laundry v0.1: satu struk
price_per_kg = 8000    # cuci, kering & lipat: rupiah per kilogram

print("=== Profitina Laundry - cashier v0.1 ===")
weight_kg = int(input("Weight (kg): "))

total = weight_kg * price_per_kg

print()
print("--- RECEIPT ---")
print("Service  : Wash, dry & fold")
print(f"Weight   : {weight_kg} kg")
print(f"Price/kg : Rp {price_per_kg}")
print(f"TOTAL    : Rp {total}")
```

Meskipun kecil, program ini sudah mengikuti pola hampir setiap program yang akan Anda tulis: **masukan** (membaca berat), **pemrosesan** (menghitung total), dan **keluaran** (mencetak struk). Saat merencanakan program baru, ajukan tiga pertanyaan ini lebih dahulu: apa yang masuk, apa yang harus dihitung, dan apa yang keluar?

1.11 Membuatnya Lebih Baik: Berat Desimal dan Tarif Minimum

Kasir v0.1 punya dua masalah yang akan disadari Mbak Rina pada pagi pertama memakainya. Timbangan menunjukkan 4,3 kg, tetapi program hanya menerima kilogram bulat. Dan aturan toko, bahwa kantong di bawah tiga kilogram ditagih sebagai tiga kilogram, belum ada. **Kode 1.13** memperbaiki keduanya.

Kode 1.13 `receipt_min.c`: kasir v0.1 yang diperbaiki, dengan berat desimal dan tarif minimum.

```c
/* receipt_min.c - kasir v0.1 diperbaiki: berat desimal, minimum 3 kg */
#include <stdio.h>

int main(void)
{
    double weight;              /* kilogram, misalnya 4.3 */
    double charged;             /* berat yang ditagihkan */
    int pricePerKg = 8000;
    int total;

    printf("=== Profitina Laundry - cashier v0.1 (improved) ===\n");
    printf("Weight (kg): ");
    if (scanf("%lf", &weight) != 1) return 1;   /* %lf membaca sebuah double */

    charged = weight;
    if (weight < 3.0) {
        charged = 3.0;          /* tarif minimum toko */
    }
    total = (int)(charged * pricePerKg + 0.5);   /* bulatkan ke rupiah utuh */

    printf("\n--- RECEIPT ---\n");
    printf("Weight   : %.1f kg\n", weight);
    if (weight < 3.0) {
        printf("Minimum  : 3.0 kg charged\n");
    }
    printf("Price/kg : Rp %d\n", pricePerKg);
    printf("TOTAL    : Rp %d\n", total);
    return 0;
}
```

Tiga perubahan layak diperhatikan lebih dekat. Pertama, berat kini bertipe `double`, dibaca dengan `%lf` (“*long float*”; `scanf` memerlukan `l` untuk `double`, sedangkan `printf` menerima `%f` untuk keduanya) dan dicetak dengan `%.1f`, satu angka di belakang titik, persis seperti tampilan timbangan. Kedua, tarif minimum adalah sebuah keputusan: kita mulai dengan menagih berat sebenarnya dan, *jika* berat itu kurang dari 3.0, menggantinya dengan 3.0. Kurung kurawal di sekeliling badan `if` menjelaskan pernyataan mana yang menjadi miliknya; bila badan berisi lebih dari satu pernyataan kurung itu wajib, dan banyak pemrogram selalu memakainya. Ketiga, total harus berupa rupiah bulat, tetapi `charged * pricePerKg` bertipe `double`. Menambahkan 0.5 lalu mengubahnya menjadi `int` dengan `(int)` **membulatkan ke bilangan bulat terdekat**: konversi itu memotong pecahan, sehingga 34.400,3 + 0,5 menjadi 34.400 dan 34.400,7 + 0,5 menjadi 34.401. Notasi `(int)` disebut **cast**; Bab 2 memakainya ke arah sebaliknya.

Dua kali jalan memperlihatkan kedua jalur di dalam program:

```
=== Profitina Laundry - cashier v0.1 (improved) ===
Weight (kg): 4.3

--- RECEIPT ---
Weight   : 4.3 kg
Price/kg : Rp 8000
TOTAL    : Rp 34400
=== Profitina Laundry - cashier v0.1 (improved) ===
Weight (kg): 2.2

--- RECEIPT ---
Weight   : 2.2 kg
Minimum  : 3.0 kg charged
Price/kg : Rp 8000
TOTAL    : Rp 24000
```

Kantong pertama lebih berat dari tiga kilogram dan membayar berat sebenarnya: 4,3 × 8.000 = 34.400. Kantong kedua hanya 2,2 kg, sehingga struk menyebut bahwa tarif minimum diterapkan

dan pelanggan membayar 3 kg, Rp24.000. Perhatikan bahwa berat diketik dengan titik, `4.3`, karena begitulah `scanf` dan `float` membaca pecahan. **Kode 1.14** adalah versi Python-nya; `float` mengubah teks ketikan menjadi bilangan pecahan, dan format `{weight:.1f}` di dalam f-string sepadan dengan `%.1f`.

Kode 1.14 `receipt_min.py`: kasir yang diperbaiki dalam Python.

```
# receipt_min.py - kasir v0.1 diperbaiki: berat desimal, minimum 3 kg
price_per_kg = 8000

print("=== Profitina Laundry - cashier v0.1 (improved) ===")
weight = float(input("Weight (kg): "))    # float() menerima 4.3

charged = weight
if weight < 3.0:
    charged = 3.0          # tarif minimum toko
total = int(charged * price_per_kg + 0.5)   # bulatkan ke rupiah utuh

print()
print("--- RECEIPT ---")
print(f"Weight   : {weight:.1f} kg")
if weight < 3.0:
    print("Minimum  : 3.0 kg charged")
print(f"Price/kg : Rp {price_per_kg}")
print(f"TOTAL    : Rp {total}")
```

Mengapa kita menguji dengan 4,3 dan 2,2, bukan dengan 5? Karena uji yang baik dipilih untuk menjalani setiap jalur program: satu berat di atas minimum dan satu di bawahnya. Uji ketiga, tepat 3,0 kg, memeriksa tepi tempat kedua jalur bertemu; kita serahkan kepada Soal Latihan 1.4.

1.12 Memakainya Sungguhan

Program struk ini kecil, tetapi bentuknya cocok untuk banyak sekali usaha kecil. Tempat fotokopi di dekat kampus menagih per halaman, depot isi ulang air per galon, tempat parkir per jam, penjual beras per kilogram, penjahit per potong. Dalam setiap kasus program membaca satu besaran, mengalikannya dengan harga, mungkin menerapkan minimum, dan mencetak struk. Untuk menyesuaikan `receipt_min` dengan usaha Anda sendiri, ubah tiga hal: nama layanan di struk, harga per satuan, dan batas minimumnya (atau buang minimum itu bila tidak ada). Pertahankan satuan di komentar setiap variabel, dan pertahankan uang dalam rupiah bulat.

Dua peringatan sebelum program seperti ini dipakai di meja kasir sungguhan. Pertama, ujilah pada kasus-kasus yang Anda hafal di luar kepala: kantong terkecil, terbesar, satu tepat di batas minimum, dan satu dengan berat desimal. Tulis struk yang diharapkan di kertas *sebelum* menjalankan program, lalu bandingkan. Kedua, program hanya sebenar aturannya. Bila toko mengubah harga, program harus ikut berubah, dan itulah sebabnya Bab 11 memindahkan harga dari dalam program ke sebuah berkas yang dapat diedit pemilik. Untuk sementara, komentar di bagian atas yang menyebut harga dan tanggal berlakunya adalah kebiasaan yang baik.

1.13 Kesalahan yang Sering Terjadi, dan Cara Menghindarinya

- **Lupa titik koma.** Setiap pernyataan C diakhiri `;`. Kompilator biasanya melaporkan galatnya di baris *berikutnya*; lihatlah satu baris ke atas.
- **Lupa `&` pada `scanf`.** `scanf("%d", kilos);` memberikan nilai `kilos`, sebuah bilangan acak, di tempat `scanf` mengharapkan alamat. GCC dengan `-Wall` berkata `format '%d' expects`

`argument of type 'int *', but argument 2 has type 'int'`; bila tetap dijalankan, program biasanya mogok. `printf` menerima nilai, `scanf` menerima alamat.

- **Spesifikasi konversi tertukar.** Mencetak `double` dengan `%d`, atau membaca `double` dengan `%f` alih-alih `%lf`, menghasilkan omong kosong. GCC memperingatkan keduanya; bacalah peringatannya.
- **Kejutan presedensi.** `a + b + c / 3` hanya membagi `c`. Pakailah tanda kurung setiap kali sebuah ekspresi mencampur `+` atau `-` dengan `*`, `/`, atau `%`.
- **`=` alih-alih `==`.** `if (x = 5)` menugaskan dan selalu benar. Kompilasi dengan `-Wall` dan jangan pernah mengabaikan peringatannya.
- **Menjalankan berkas *executable* lama.** Sesudah mengedit `hello.c`, Anda harus mengompilasi ulang; menjalankan `./hello` tanpa kompilasi menjalankan versi lama. Bila sebuah perubahan tampak tak berpengaruh, kompilasi dulu.
- **Indentasi campur aduk di Python.** Python menghitung spasi. Biarkan VS Code menyisipkan empat spasi per tingkat dan jangan pernah mencampur tab dengan spasi.
- **Pembagian bilangan bulat padahal yang dimaksud pecahan.** Kekeliruan ini layak mendapat program sendiri, **Kode 1.15**, karena ia lolos kompilasi tanpa satu peringatan pun.

Kode 1.15 `wrong_average.c`: `1 / 2` bernilai 0, sehingga separuh dari 7 kg menjadi 0.00 kg.

```c
/* wrong_average.c - kekeliruan klasik yang lolos kompilasi tanpa peringatan */
#include <stdio.h>

int main(void)
{
    int kilos = 7;
    double half = 1 / 2 * kilos;          /* 1 / 2 bernilai 0 lebih dulu */
    printf("half of %d kg = %.2f kg\n", kilos, half);
    printf("half of %d kg = %.2f kg\n", kilos, kilos / 2.0);
    return 0;
}
```

```
half of 7 kg = 0.00 kg
half of 7 kg = 3.50 kg
```

Baris pertama salah dan baris kedua benar. Versi Python-nya, `wrong_average.py` di folder kode, membuat kekeliruan yang sama dengan `1 // 2` dan mencetak dua baris yang sama. Obatnya adalah menjadikan salah satu ruas pembagian sebuah `double`, seperti pada `kilos / 2.0`, atau mengalikan dengan `0.5`.

1.14 Ringkasan Bab

- Komputer menyimpan data di memori dan menjalankan instruksi dengan prosesornya; perangkat keras adalah mesinnya, perangkat lunak instruksinya. Memori adalah deretan byte bernomor, dan nomor sebuah byte adalah alamatnya.
- Bahasa tingkat tinggi seperti C dan Python memungkinkan kita menulis dalam istilah masalahnya; kompilator (C) atau interpreter (Python) menerjemahkan program bagi prosesor. C dikompilasi sekali menjadi berkas *executable*; Python dibaca dan dijalankan interpreter setiap kali (**Gambar 1.1**).
- Meja kerja buku ini adalah VS Code 1.139 dengan ekstensi C/C++ dan Python, GCC (MSYS2 di Windows, `build-essential` di Linux) atau Apple clang (Xcode Command Line Tools di macOS), serta Python 3.14. Setiap program C dikompilasi dengan `gcc -std=c17 -Wall -Wextra -O2`.

- Program C dimulai di `main`; `#include <stdio.h>` memperkenalkan `printf` dan `scanf`; pernyataan diakhiri `;`; `return 0;` melaporkan keberhasilan. Python menjalankan berkas dari atas ke bawah, dan indentasi adalah bagian dari tata bahasanya.
- Variabel memiliki nama, tipe, nilai, dan alamat (**Gambar 1.2**). `scanf` memerlukan alamat (`&`) karena ia menulis ke dalam variabel; `input` di Python mengembalikan teks yang diubah oleh `int` atau `float`.
- Pembagian bilangan bulat membuang pecahan dan `%` memberikan sisanya (Persamaan (1.1)); di Python `/` selalu mempertahankan pecahan dan `//` adalah pembagian bulat, yang membulatkan ke bawah alih-alih ke arah nol. Presedensi mendahulukan `*`, `/`, `%` sebelum `+`, `-` (**Tabel 1.3**); tanda kurung mengalahkannya.
- `if` menjalankan sebuah pernyataan hanya bila kondisinya benar; kondisi memakai `==`, `!=`, `<`, `>`, `<=`, `>=`. `=` menugaskan, `==` membandingkan.
- Kasir v0.1 membaca berat dan mencetak struk; versi perbaikannya membaca berat desimal, menerapkan minimum tiga kilogram, dan membulatkan total ke rupiah utuh.

1.15 Soal Latihan

1. Tulislah program `about_me` dalam C dan Python yang mencetak nama Anda di baris pertama, kota Anda di baris kedua, dan makanan favorit Anda di baris ketiga, dengan tiga pernyataan `printf` (atau `print`). Lalu cetak tiga baris yang sama dengan *satu* `printf` saja.
2. Tanpa menjalankannya, tentukan apa yang dicetak `printf("%d\n", 17 / 5 * 5 + 17 % 5);`. Jelaskan, dengan Persamaan (1.1), mengapa ekspresi itu mengembalikan 17 untuk sembarang bilangan bulat positif pengganti 17 dan 5.
3. Hitung dengan tangan, lalu periksa dengan program: (a) `10 - 4 - 3`; (b) `10 - (4 - 3)`; (c) `2 * 3 % 4`; (d) `-9 / 4` dan `-9 % 4` di C; (e) `-9 // 4` dan `-9 % 4` di Python.
4. Jalankan `receipt_min` dengan berat 3.0, 0.4, dan 12.75. Tulis dahulu struk yang diharapkan. Mana di antara ketiganya yang menguji tepi di antara kedua jalur, dan apa yang semestinya dicetak?
5. *Laundry* menambahkan biaya ekspres: setiap pesanan membayar Rp5.000 di atas tagihan berat. Ubahlah `receipt.c` dan `receipt.py` agar struk menampilkan baris `Surcharge: Rp 5000` dan total yang baru. Baris mana yang berubah, dan mana yang tetap?
6. Seorang pelanggan membayar dengan uang kertas dan kasir harus memberi kembalian dengan lembaran sesedikit mungkin: lembaran Rp50.000, 20.000, 10.000, 5.000, 2.000, dan 1.000, dan sisanya berupa koin. Tulislah program `change` yang membaca tagihan dan jumlah uang yang dibayar, lalu mencetak berapa lembar dari setiap jenis yang harus diberikan. Bila uang yang dibayar kurang, program mencetak berapa kekurangannya. Pakailah hanya `/`, `%`, dan `if`.

1.16 Penyelesaian Soal Latihan

1. Di C, tiga pernyataan seperti `printf("Name: Dewi\n");` dan satu pernyataan gabungan `printf("Name: Dewi\nCity: Surabaya\nFood: rawon\n");` menghasilkan tiga baris yang sama, karena setiap `\n` memulai baris baru. Di Python, `print("Name: Dewi")` tiga kali, atau satu `print("Name: Dewi\nCity: Surabaya\nFood: rawon")`.

2. `17 / 5` bernilai 3; `3 * 5` bernilai 15 (operator `/`, `*`, dan `%` memiliki presedensi yang sama dan diterapkan dari kiri ke kanan); `17 % 5` bernilai 2; `15 + 2` bernilai 17. Program mencetak 17. Secara umum `a / b * b` adalah kelipatan `b` terbesar yang tidak melebihi `a` (untuk bilangan

positif), dan `a % b` adalah sisanya, sehingga jumlah keduanya kembali menjadi `a`. Itulah Persamaan (1.1) dibaca dari kanan ke kiri.

3. (a) Pengurangan berjalan dari kiri ke kanan: `(10 - 4) - 3` bernilai 3. (b) Tanda kurung lebih dahulu: `10 - 1` bernilai 9. (c) `*` dan `%` sama presedensinya dan berjalan dari kiri ke kanan: `6 % 4` bernilai 2. (d) C memotong ke arah nol: `-9 / 4` bernilai −2 dan `-9 % 4` bernilai −1, karena $-9 = (-2) \cdot 4 + (-1)$. (e) Python membulatkan ke bawah: `-9 // 4` bernilai −3 dan `-9 % 4` bernilai 3, karena $-9 = (-3) \cdot 4 + 3$.

4. 3,0 kg adalah tepinya: kondisi `weight < 3.0` bernilai salah, sehingga baris minimum tidak dicetak dan totalnya 3,0 × 8.000 = Rp24.000, sama dengan yang dibayar kantong yang lebih kecil. 0,4 kg berada di bawah minimum: struk menampilkan `Weight  : 0.4 kg`, baris `Minimum  : 3.0 kg charged`, dan Rp24.000. 12,75 kg membayar 12,75 × 8.000 = Rp102.000; beratnya dicetak sebagai `12.8 kg`, karena `%.1f` membulatkan hanya untuk tampilan, sedangkan total memakai 12,75 secara utuh.

5. Tambahkan variabel `int surcharge = 5000;`, ubah total menjadi `total = weightKg * pricePerKg + surcharge;`, lalu tambahkan `printf("Surcharge: Rp %d\n", surcharge);` sebelum baris total. Masukan, harga, dan baris struk lainnya tidak berubah. Di Python tiga perubahan yang sama: `surcharge = 5000`, `total = weight_kg * price_per_kg + surcharge`, dan `print(f"Surcharge: Rp {surcharge}")`.

6. Setiap jenis lembaran ditangani dengan cara yang sama: bagi dengan nilainya untuk mendapatkan jumlah lembar, lalu simpan sisanya untuk lembaran yang lebih kecil. **Kode 1.16** adalah program C-nya; `change.py` di folder kode adalah versi Python-nya, dengan `//` menggantikan `/`.

Kode 1.16 `change.c`: kembalian dipecah menjadi lembaran dengan `/` dan `%`.

```c
/* change.c - memecah uang kembalian menjadi lembaran rupiah */
#include <stdio.h>

int main(void)
{
    int bill, paid, change;

    printf("Bill (Rp): ");
    if (scanf("%d", &bill) != 1) return 1;
    printf("Paid (Rp): ");
    if (scanf("%d", &paid) != 1) return 1;

    change = paid - bill;
    if (change < 0) {
        printf("Not enough: Rp %d short\n", -change);
        return 0;
    }
    printf("Change: Rp %d\n", change);
    printf("50000 x %d\n", change / 50000);
    change = change % 50000;          /* sisa setelah lembaran 50000 */
    printf("20000 x %d\n", change / 20000);
    change = change % 20000;
    printf("10000 x %d\n", change / 10000);
    change = change % 10000;
    printf(" 5000 x %d\n", change / 5000);
    change = change % 5000;
    printf(" 2000 x %d\n", change / 2000);
    change = change % 2000;
    printf(" 1000 x %d\n", change / 1000);
    change = change % 1000;
    printf("coins: Rp %d\n", change);
    return 0;
}
```

```
Bill (Rp): 34400
Paid (Rp): 100000
Change: Rp 65600
50000 x 1
20000 x 0
10000 x 1
 5000 x 1
 2000 x 0
 1000 x 0
coins: Rp 600
```

Kembaliannya 65.600: satu lembar 50.000 menyisakan 15.600; tidak ada lembar 20.000 yang muat; satu lembar 10.000 menyisakan 5.600; satu lembar 5.000 menyisakan 600; tidak ada lembar 2.000 maupun 1.000 yang muat, dan Rp600 diberikan dalam koin. Satu kali jalan lagi dengan tagihan yang lebih besar daripada pembayaran hanya mencetak kekurangannya:

```
Bill (Rp): 34400
Paid (Rp): 30000
Not enough: Rp 4400 short
```

Daftar Pustaka

[1] D. M. Ritchie, “The development of the C language,” in *Proc. 2nd ACM SIGPLAN Conf. History of Programming Languages (HOPL-II)*, Cambridge, MA, USA, 1993, pp. 201–208.

[2] B. W. Kernighan and D. M. Ritchie, *The C Programming Language*, 2nd ed. Englewood Cliffs, NJ, USA: Prentice Hall, 1988.

[3] *Information Technology — Programming Languages — C*, ISO/IEC 9899:2018, International Organization for Standardization, Geneva, Switzerland, 2018.

[4] TIOBE Software, "TIOBE Index for September 2026." Accessed: Sep. 25, 2026. [Online]. Available: https://www.tiobe.com/tiobe-index/

[5] TechRepublic, "TIOBE Index for September 2026: Top 10 Most Popular Programming Languages," Sep. 2026. Accessed: Sep. 25, 2026. [Online]. Available: https://www.techrepublic.com/article/news-tiobe-index-language-rankings/

[6] Python Software Foundation, "Python 3.14.7," Aug. 5, 2026. Accessed: Sep. 25, 2026. [Online]. Available: https://www.python.org/downloads/latest/python3.14/

[7] Microsoft, "Visual Studio Code updates, version 1.139," Sep. 23, 2026. [Online]. Available: https://code.visualstudio.com/updates

[8] GNU Project, "GCC, the GNU Compiler Collection" (GCC 16.2 released August 7, 2026). Accessed: Sep. 25, 2026. [Online]. Available: https://gcc.gnu.org/

[9] Microsoft, "Using GCC with MinGW," *Visual Studio Code Documentation*. Accessed: Sep. 25, 2026. [Online]. Available: https://code.visualstudio.com/docs/cpp/config-mingw

[10] MSYS2 Project, "MSYS2: Software Distribution and Building Platform for Windows." Accessed: Sep. 25, 2026. [Online]. Available: https://www.msys2.org/

[11] Python Software Foundation, "Using Python on Windows," *Python 3.14 Documentation*. Accessed: Sep. 25, 2026. [Online]. Available: https://docs.python.org/3.14/using/windows.html

[12] Microsoft, "Using Clang in Visual Studio Code," *Visual Studio Code Documentation*. Accessed: Sep. 25, 2026. [Online]. Available: https://code.visualstudio.com/docs/cpp/config-clang-mac

[13] Canonical, "Ubuntu 26.04 LTS summary for LTS users," *Ubuntu Release Notes*. Accessed: Sep. 25, 2026. [Online]. Available: https://documentation.ubuntu.com/release-notes/26.04/summary-for-lts-users/

[14] Microsoft, "Getting Started with Python in VS Code," *Visual Studio Code Documentation*. Accessed: Sep. 25, 2026. [Online]. Available: https://code.visualstudio.com/docs/python/python-tutorial

[15] Python Software Foundation, "Built-in Types: Numeric Types — int, float, complex," *Python 3.14 Documentation*. Accessed: Sep. 25, 2026. [Online]. Available: https://docs.python.org/3.14/library/stdtypes.html

[16] cppreference.com, "Arithmetic operators (C)." Accessed: Sep. 25, 2026. [Online]. Available: https://en.cppreference.com/w/c/language/operator_arithmetic

Bab 2

Pengembangan Program Terstruktur: Algoritma, Pseudokode, dan Kontrol

Program tidak diketik; program dirancang. Sebelum satu baris C atau Python ditulis, seorang pemrogram memutuskan *apa* yang harus dikerjakan dan *dalam urutan apa*, lalu menuliskan keputusan itu dalam bentuk yang mudah diperiksa dan mudah diubah. Bab ini mengajarkan disiplin tersebut. Anda akan mempelajari apa itu algoritma, bagaimana menuliskannya dalam pseudokode, dan bagaimana menghaluskan uraian satu kalimat tentang sebuah tugas, langkah demi langkah, menjadi program. Anda akan berkenalan dengan hasil penting bahwa setiap program dapat dibangun dari tiga jenis struktur saja (runtunan, seleksi, dan perulangan), dan Anda akan mempelajari pernyataan C dan Python untuk masing-masing: `if`, `if`…`else` beserta rantainya, operator kondisional, dan perulangan `while`, yang dikendalikan oleh pencacah atau oleh nilai khusus yang disebut *sentinel*. Di sepanjang jalan hadir pula operator penugasan gabungan dan operator *increment* yang dipakai pemrogram C setiap hari. Bab ini ditutup dengan versi 0.2 kasir Profitina Laundry, yang melayani seluruh antrean pelanggan, menolak layanan yang tidak dikenal, dan mencetak ringkasan di akhir hari.

2.1 Suatu Petang di Meja Kasir

Pukul tujuh malam Profitina Laundry tutup. Pak Hendra, pemiliknya, mampir dan menanyakan tiga hal yang sama kepada Mbak Rina setiap hari: Berapa pelanggan yang datang? Berapa uang yang masuk? Dan berapa rata-rata tagihannya? Jawaban-jawaban itu penting baginya. Jumlah pelanggan menunjukkan apakah spanduk baru di tikungan berhasil; uang yang masuk harus cocok dengan isi laci; dan rata-rata tagihan menunjukkan apakah pelanggan memilih layanan ekspres yang lebih mahal.

Dengan kasir dari Bab 1, Mbak Rina harus menjalankan program sekali untuk setiap pelanggan lalu menjumlahkan empat puluh struk dengan tangan saat tutup, padahal justru pekerjaan itulah yang hendak diambil alih program. Yang ia perlukan adalah program yang terus berjalan sepanjang hari, melayani satu pelanggan demi satu pelanggan, dan menjawab tiga pertanyaan Pak Hendra ketika kasir ditutup. Program itu memerlukan dua kemampuan baru. Ia harus **mengulang** langkah-langkah yang sama untuk setiap pelanggan, dan ia harus **memilih** di antara beberapa harga menurut layanannya. Perulangan dan seleksi adalah pokok bahasan bab ini.

Ada pelajaran kedua yang kurang kasatmata di balik pertanyaan Pak Hendra. “Rata-rata tagihan” terdengar sederhana, tetapi program harus diberi tahu dengan tepat apa artinya: uang masuk dibagi jumlah pelanggan, dengan berapa angka desimal, dan apa yang dicetak pada hari ketika, karena listrik padam, tidak seorang pun datang. Memutuskan rincian semacam itu *sebelum* menulis kode adalah inti pengembangan program terstruktur.

2.2 Algoritma: Aksi dalam Urutan yang Benar

Algoritma adalah prosedur untuk menyelesaikan suatu masalah, yang dinyatakan sebagai daftar **aksi** yang harus dikerjakan beserta **urutan** pengerjaannya. Kedua bagian itu sama pentingnya.

Perhatikan rutinitas Mbak Rina untuk sekantong cucian: timbang kantongnya, tulis struknya, pasang label bernomor pada kantong, masukkan pakaian ke mesin. Ubah urutannya menjadi "masukkan pakaian ke mesin, lalu timbang kantongnya", dan hasilnya adalah kantong basah di atas timbangan dan struk yang salah. Aksinya sama; urutannya salah.

> **DEFINISI.** *Algoritma* adalah daftar berhingga aksi-aksi yang tepat, beserta urutan pelaksanaannya, yang menyelesaikan suatu masalah untuk setiap masukan yang diizinkan. Menetapkan urutan pelaksanaan pernyataan-pernyataan program disebut *kontrol program*.

Tiga sifat menjadikan sederet langkah sebuah algoritma, bukan sekadar harapan. Ia harus **tepat**: "tambahkan sedikit deterjen" cukup bagi manusia tetapi tak berguna bagi mesin, yang memerlukan "tambahkan 30 ml deterjen". Ia harus **berhingga**: ia harus berhenti setelah sejumlah langkah berhingga untuk setiap masukan. Dan ia harus **benar**: untuk setiap masukan yang diizinkan ia harus menghasilkan jawaban yang benar, termasuk untuk masukan yang tidak lazim seperti hari tanpa pelanggan.

Kehidupan sehari-hari penuh algoritma. Resep masakan adalah algoritma; begitu pula petunjuk merakit lemari, langkah menarik uang di ATM, dan aturan yang dipakai wasit untuk memutuskan tendangan penalti. Yang membedakan algoritma di buku ini hanyalah bahwa pelaksananya adalah komputer, yang mengikutinya *persis*, tanpa akal sehat yang memungkinkan manusia membetulkan resep yang samar. Itulah sebabnya kita menuliskannya dengan cermat lebih dahulu.

Mari kita uji sifat-sifat ini pada tugas yang akrab: membuat segelas es teh manis untuk pelanggan yang menunggu di meja. Upaya pertama mungkin berbunyi: *masukkan kantong teh ke gelas, tuang air, tambahkan gula, tambahkan es, aduk, sajikan*. Bacalah seperti mesin membacanya, dan celah-celahnya langsung tampak. Air yang mana, panas atau dingin? Kantong teh di air dingin menghasilkan air berwarna, bukan teh, jadi algoritma itu tidak benar. Berapa banyak gulanya? "Tambahkan gula" tidak tepat. Dan "aduk" tidak menyebut kapan berhenti, jadi algoritma itu bahkan tidak berhingga. Versi yang lebih baik berbunyi: *didihkan 250 ml air; tuangkan ke atas satu kantong teh; tunggu tiga menit; angkat kantongnya; aduk dua sendok teh gula sampai tidak ada butiran yang tampak; tambahkan es sampai gelas penuh; sajikan*. Kini setiap aksi tepat, setiap langkah berakhir, dan urutannya benar: gula larut di dalam teh panas, sesuatu yang tidak akan terjadi di dalam es teh. Pertanyaan-pertanyaan yang sama (apakah setiap langkah tepat, apakah setiap langkah berakhir, apakah urutannya benar, apakah hasilnya benar untuk setiap masukan yang diizinkan?) adalah pertanyaan yang kita ajukan kepada setiap algoritma di buku ini.

2.3 Pseudokode

Pseudokode adalah bahasa informal untuk menuliskan algoritma. Wujudnya campuran bahasa manusia dan bahasa pemrograman, dan ia tidak punya tata bahasa baku: satu-satunya tujuannya adalah membantu Anda memikirkan langkah-langkah dan urutannya sebelum memikirkan titik koma dan kurung kurawal. Pseudokode tidak pernah dikompilasi atau dijalankan; ia dibaca manusia, pada tahap perancangan, ketika mengubah satu baris hanya memakan beberapa detik.

Di buku ini pseudokode mengikuti beberapa kesepakatan sederhana, yang dapat Anda lihat di **Pseudokode 2.1**, algoritma program struk dari Bab 1. Seperti kode programnya, pseudokode di buku ini ditulis dengan kata-kata bahasa Inggris, supaya mudah dicocokkan dengan program C dan Python yang menerjemahkannya:

Pseudokode 2.1 Pseudokode untuk program struk dari Bab 1.

```
RECEIPT
1  price ← 8000
2  print "Weight (kg): "
3  read weight
4  total ← weight × price
5  print the receipt with weight, price and total
```

Nama prosedur ditulis di atas. Setiap baris diberi nomor agar dapat dirujuk. Panah `←` berarti penugasan ("menjadi"), agar tidak tertukar dengan `=` untuk perbandingan. Kata-kata seperti `read` (baca), `print` (cetak), `if` (jika), `else` (selain itu), dan `while` (selama) bermakna seperti biasanya, dan indentasi menunjukkan baris-baris mana yang merupakan satu kesatuan, persis seperti di Python. Pseudokode hanya memuat aksi yang *dapat dijalankan*: deklarasi seperti `int weight;` bukan aksi dan tidak ditulis.

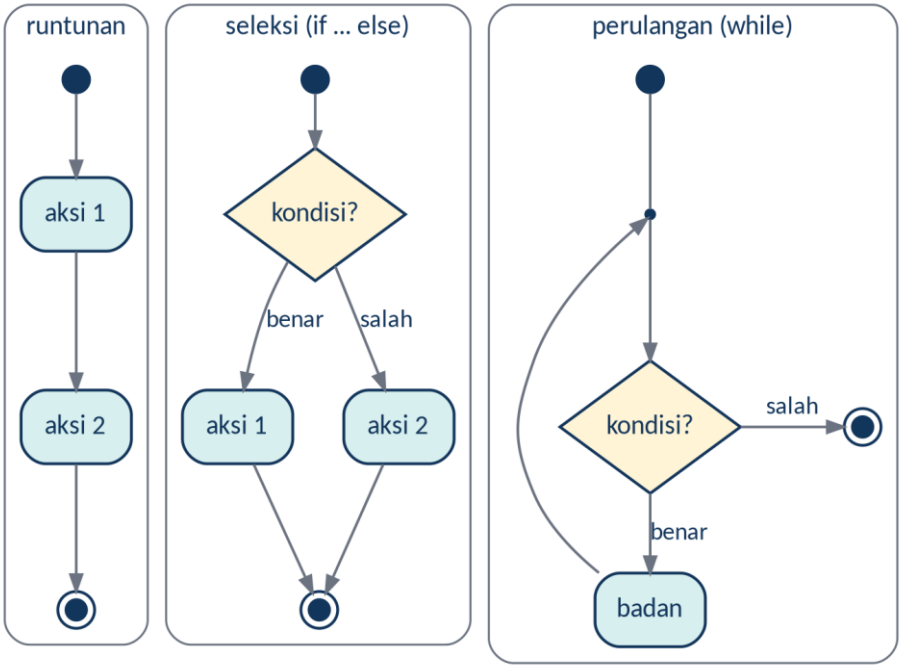
Satu aturan yang baik untuk setiap program di buku ini: habiskan sepuluh menit pertama jauh dari papan ketik. Baca masalahnya, tuliskan apa yang masuk dan apa yang harus keluar, lalu tulis pseudokodenya. Pemrogram yang merancang lebih dahulu hampir selalu selesai lebih cepat daripada yang langsung mengetik, karena kesalahan yang tertangkap di atas kertas tidak perlu dicari lagi di program yang sedang berjalan.

2.4 Tiga Struktur Sudah Cukup

2.4.1 Runtunan, seleksi, dan perulangan

Program-program awal sering berupa gumpalan lompatan. Sebuah pernyataan bernama `goto` memungkinkan program melanjutkan ke baris mana pun, dan program yang memakainya dengan leluasa begitu sulit diikuti sehingga dijuluki "kode spageti". Pada 1966 Corrado Böhm dan Giuseppe Jacopini membuktikan hasil yang luar biasa: *setiap* algoritma dapat ditulis hanya dengan tiga jenis struktur kontrol, yang digabungkan dengan cara ditumpuk dan disarangkan [1]. Dua tahun kemudian surat Edsger Dijkstra, "Go To Statement Considered Harmful", menganjurkan agar pemrogram berhenti memakai `goto` [2], dan gaya yang lahir sesudahnya, **pemrograman terstruktur**, adalah gaya buku ini dan hampir semua program yang ditulis saat ini [3].

Gambar 2.1 menampilkan ketiga struktur itu sebagai **diagram alir** (*flowchart*), yaitu diagram dengan persegi panjang sebagai aksi, belah ketupat sebagai keputusan, dan anak panah sebagai urutan.



Gambar 2.1 Tiga struktur kontrol: runtunan, seleksi, dan perulangan; masing-masing punya tepat satu jalan masuk dan satu jalan keluar.

- Dalam **runtunan** (*sequence*), aksi-aksi dijalankan satu demi satu sesuai urutan tertulisnya. Semua program sejauh ini berupa runtunan.
- Dalam **seleksi** (*selection*), sebuah kondisi menentukan jalur mana dari dua jalur yang dijalankan. C dan Python menyediakan `if` (satu jalur atau tidak sama sekali), `if`…`else` (salah satu dari dua jalur), dan rantai keduanya untuk banyak jalur; C juga punya `switch`, yang dibahas di Bab 3.
- Dalam **perulangan** (*repetition*), sebuah aksi dijalankan berulang-ulang selama suatu kondisi terpenuhi. C menyediakan `while`, `for`, dan `do`…`while`; Python menyediakan `while` dan `for`. Bab ini memakai `while`; Bab 3 menambahkan yang lain.

Sifat terpenting tampak pada gambar: setiap struktur memiliki **tepat satu jalan masuk dan satu jalan keluar**. Itulah yang membuat program terstruktur mudah dibaca. Anda dapat mengganti persegi panjang mana pun di gambar dengan sebuah struktur utuh, yang disebut **penyarangan** (*nesting*), atau meletakkan struktur-struktur berurutan, yang disebut **penumpukan** (*stacking*), dan hasilnya tetap terbaca dari atas ke bawah seperti halaman sebuah buku.

2.4.2 Membaca diagram alir

Diagram alir memakai sekumpulan kecil lambang. Lingkaran penuh menandai tempat struktur dimasuki, dan lingkaran di dalam cincin menandai tempat struktur ditinggalkan; banyak buku menggambarnya sebagai oval berlabel *mulai* dan *selesai*. Persegi panjang bersudut bulat adalah aksi, seperti penugasan atau perhitungan. Belah ketupat adalah keputusan: ia berisi sebuah kondisi, dan tepat dua anak panah keluar darinya, satu berlabel *benar* (atau *ya*) dan satu berlabel *salah* (atau *tidak*). Anak panah menunjukkan urutan kejadian. Sebagian buku menggambar masukan dan keluaran sebagai jajaran genjang; di buku ini keduanya adalah aksi biasa, seperti "cetak prompt, baca weight" pada **Gambar 2.2**.

Untuk membaca diagram alir, letakkan jari Anda di jalan masuk lalu ikuti anak panahnya, dengan memilih cabang di setiap belah ketupat sesuai data yang Anda bayangkan. Diagram alir sangat baik untuk *melihat* bentuk algoritma kecil, terutama di bagian perulangan kembali ke atas, dan untuk menjelaskannya kepada orang lain; untuk algoritma yang lebih besar diagram alir menjadi sesak, dan pseudokode lebih mudah ditulis dan diubah. Buku ini memakai keduanya: diagram alir untuk memperlihatkan sebuah struktur sekali, pseudokode untuk merancang program.

2.4.3 Blok

Sebuah struktur sering perlu mengendalikan lebih dari satu pernyataan. C mengelompokkan beberapa pernyataan menjadi satu **blok** (disebut juga *pernyataan majemuk*) dengan kurung kurawal: di mana pun C mengizinkan satu pernyataan, ia juga mengizinkan `{`, berapa pun pernyataan, dan `}`. Python membentuk blok dengan indentasi: semua baris yang menjorok satu tingkat lebih dalam daripada baris `if` atau `while` adalah miliknya, dan blok berakhir ketika indentasinya kembali. Karena itu indentasi penting dalam kedua bahasa, bagi interpreter di Python dan bagi pembaca di C.

2.5 Seleksi

2.5.1 `if` dan `if`…`else`

Bab 1 memakai `if` seleksi tunggal: sebuah pernyataan dijalankan bila kondisinya benar dan dilewati bila salah. `if`…`else` **seleksi ganda** memilih di antara dua aksi:

```
if (weight < 3.0)
    printf("Minimum charge applied\n");
else
    printf("Charged by weight\n");
```

Bila kondisinya benar, pernyataan pertama dijalankan; bila tidak, pernyataan sesudah `else` yang dijalankan. Tepat satu dari keduanya selalu dijalankan. Di Python pilihan yang sama ditulis dengan titik dua dan indentasi: `if weight < 3.0:` diikuti satu baris yang menjorok, lalu `else:` dan satu baris menjorok lagi.

2.5.2 Rantai keputusan

Sering kali ada lebih dari dua kasus. Profitina Laundry memberi diskon untuk tagihan besar: 10% untuk tagihan Rp100.000 atau lebih, 5% untuk tagihan Rp50.000 atau lebih, dan tidak ada diskon di bawahnya. Pelanggan yang punya kartu anggota mendapat lima poin persentase lagi. Algoritmanya adalah **Pseudokode 2.2**.

Pseudokode 2.2 Pseudokode diskon: rantai tiga kasus, lalu keputusan terpisah untuk anggota.

```
DISCOUNT
1  read bill, member
2  if bill ≥ 100000
3      percent ← 10
4  else if bill ≥ 50000
5      percent ← 5
6  else
7      percent ← 0
8  if member = 1
9      percent ← percent + 5
10 discount ← bill × percent / 100   (whole rupiah)
11 print member or guest, percent, discount, bill − discount
```

Dalam lambang, dengan tagihan b dan persentase p, diskon dalam rupiah bulat adalah

$$d = \left\lfloor \frac{b \cdot p}{100} \right\rfloor \tag{2.1}$$

dengan $\lfloor x \rfloor$, *floor* dari x, adalah bilangan bulat terbesar yang tidak lebih besar daripada x. Untuk bilangan positif, inilah persis yang dihitung pembagian bulat di C, sehingga Persamaan (2.1) menjadi `bill * percent / 100` di C dan `bill * percent // 100` di Python. Mengalikan sebelum membagi itu penting: `bill / 100 * percent` akan membuang dua angka terakhir tagihan sebelum dikalikan.

Urutan pengujian di dalam rantai juga penting. Kondisi-kondisi diperiksa dari atas, dan yang pertama bernilai benar yang menang; sisanya dilewati. Tagihan Rp120.000 memenuhi `bill >= 100000` maupun `bill >= 50000`, dan karena uji 10% ada di depan, ia mendapat 10%. Andai uji 5% ditulis lebih dahulu, setiap tagihan besar hanya akan mendapat 5%. Bila kasus-kasus sebuah rantai saling tumpang tindih, letakkan kondisi yang paling menuntut di depan. **Kode 2.1** adalah programnya.

Kode 2.1 `discount.c`: rantai `if`…`else if`…`else`, sebuah `if` terpisah, dan operator kondisional.

```c
/* discount.c - if...else bersarang dan operator kondisional ?: */
#include <stdio.h>

int main(void)
{
    int bill;       /* tagihan sebelum diskon, rupiah */
    int member;     /* 1 = punya kartu anggota, 0 = tidak punya */
    int percent;

    printf("Bill (Rp): ");
    if (scanf("%d", &bill) != 1) return 1;
    printf("Member card? (1 = yes, 0 = no): ");
    if (scanf("%d", &member) != 1) return 1;

    if (bill >= 100000) {
        percent = 10;
    } else if (bill >= 50000) {
        percent = 5;
    } else {
        percent = 0;
    }
    if (member == 1) {
        percent = percent + 5;      /* anggota mendapat lima poin persen lebih */
    }

    int discount = bill * percent / 100;
    printf("Customer : %s\n", member == 1 ? "member" : "guest");
    printf("Discount : %d%% = Rp %d\n", percent, discount);
    printf("To pay   : Rp %d\n", bill - discount);
    return 0;
}
```

Secara formal C tidak mengenal `else if`: `if` yang kedua hanyalah pernyataan yang mengikuti `else` pertama, dan rantai itu adalah `if`…`else` *bersarang* yang ditulis tanpa indentasi tambahan. Jika ditulis dengan indentasi penuh, ia akan terus bergeser ke kanan, sehingga setiap pemrogram C menuliskannya seperti di atas. Berikut tiga pelanggan:

```
Bill (Rp): 120000
Member card? (1 = yes, 0 = no): 1
Customer : member
Discount : 15% = Rp 18000
To pay   : Rp 102000
Bill (Rp): 60000
Member card? (1 = yes, 0 = no): 0
Customer : guest
Discount : 5% = Rp 3000
To pay   : Rp 57000
Bill (Rp): 30000
Member card? (1 = yes, 0 = no): 1
Customer : member
Discount : 5% = Rp 1500
To pay   : Rp 28500
```

Pelanggan pertama bertagihan besar dan punya kartu anggota: 10% ditambah 5% adalah 15%, dan 15% dari Rp120.000 adalah Rp18.000. Pelanggan kedua masuk kasus tengah, 5% dari Rp60.000. Pelanggan ketiga jatuh sampai ke 0% di dalam rantai tetapi mendapat 5% sebagai anggota, yaitu Rp1.500 dari Rp30.000.

2.5.3 Operator kondisional

`printf` terakhir di **Kode 2.1** memuat satu-satunya operator C yang memiliki tiga operand, **operator kondisional** `?:`. Ekspresi `member == 1 ? "member" : "guest"` dibaca "jika `member` sama dengan 1, maka `"member"`, jika tidak `"guest"`". Ia adalah `if`…`else` yang menghasilkan *nilai* alih-alih menjalankan pernyataan, sehingga dapat diletakkan di dalam `printf`. Pakailah untuk pilihan nilai yang kecil; bila salah satu cabang mengerjakan pekerjaan sungguhan, tulislah `if`…`else` lengkap, yang lebih mudah dibaca.

Kode 2.2 menunjukkan versi Python-nya. Python menulis rantai itu dengan kata kunci `elif`, singkatan dari "else if" [4], dan ekspresi kondisionalnya meletakkan kondisi di tengah: `'member' if member == 1 else 'guest'`.

Kode 2.2 `discount.py`: `if`, `elif`, dan `else`, serta ekspresi kondisional Python.

```python
# discount.py - if/elif/else dan ekspresi kondisional
bill = int(input("Bill (Rp): "))
member = int(input("Member card? (1 = yes, 0 = no): "))

if bill >= 100000:
    percent = 10
elif bill >= 50000:
    percent = 5
else:
    percent = 0
if member == 1:
    percent = percent + 5           # anggota mendapat lima poin persen lebih

discount = bill * percent // 100
print(f"Customer : {'member' if member == 1 else 'guest'}")
print(f"Discount : {percent}% = Rp {discount}")
print(f"To pay   : Rp {bill - discount}")
```

> **PERINGATAN:** Di C, sebuah `else` menjadi milik `if` terdekat di atasnya yang belum punya `else`, apa pun kesan yang ditimbulkan indentasinya. Pada `if (a) if (b) x(); else y();`, `else` itu milik `if (b)`. Masalah ini disebut *dangling else*. Kurung kurawal menghapus semua keraguan, dan itulah salah satu alasan buku ini memakainya setiap kali sebuah `if` berisi `if` lain.

2.6 Perulangan yang Dikendalikan Pencacah

2.6.1 Perulangan `while`

Pernyataan `while` mengulang sebuah pernyataan, atau sebuah blok, selama suatu kondisi bernilai benar:

```
while (condition) {
    statements
}
```

Mula-mula kondisinya diuji. Jika salah, seluruh perulangan dilewati dan program berlanjut sesudah kurung kurawal penutup. Jika benar, badan perulangan dijalankan sekali, lalu kondisinya diuji lagi. Sesuatu di dalam badan harus pada akhirnya membuat kondisi itu salah; jika tidak, perulangan berjalan selamanya, yang disebut **perulangan tak berhingga** (*infinite loop*). Satu kali pelaksanaan badan disebut satu **iterasi**. Aturan lengkap untuk `if`, `while`, dan pernyataan C lainnya terhimpun di rujukan bahasa C [5].

2.6.2 Menghitung sampai lima

Misalkan Mbak Rina ingin mengetahui rata-rata berat lima pesanan pertama pagi ini. Banyaknya pengulangan sudah diketahui sebelumnya, jadi sebuah variabel yang disebut **pencacah** (*counter*) mengendalikan perulangan: ia mulai dari 1, perulangan berjalan selama nilainya paling banyak 5, dan setiap iterasi menambahkan 1 kepadanya. Inilah **perulangan terkendali pencacah** (*counter-controlled repetition*), dan **Pseudokode 2.3** menguraikannya.

Pseudokode 2.3 Perulangan terkendali pencacah: tepat lima berat dibaca.

```
AVERAGE-OF-FIVE
1  total ← 0
2  counter ← 1
3  while counter ≤ 5
4      print "Weight of order counter (kg): "
5      read weight
6      total ← total + weight
7      counter ← counter + 1
8  average ← total / 5   (whole kilograms)
9  print total and average
```

Variabel `total` adalah sebuah **akumulator**: ia mulai dari 0 dan setiap iterasi menambahkan satu berat kepadanya, sehingga setelah perulangan ia menyimpan jumlahnya. Lupa mengisinya dengan 0 lebih dahulu adalah kesalahan klasik; di C, variabel lokal yang belum diinisialisasi berisi apa saja yang kebetulan ada di memori itu, dan jumlahnya menjadi sampah. **Kode 2.3** adalah program C-nya.

Kode 2.3 `average_counter.c`: perulangan `while` yang dikendalikan pencacah.

```c
/* average_counter.c - rata-rata berat tepat lima pesanan */
#include <stdio.h>

int main(void)
{
    int counter = 1;    /* nomor pesanan yang sedang dibaca */
    int weight;         /* berat satu pesanan, kg */
    int total = 0;      /* jumlah berjalan dari berat */

    while (counter <= 5) {                  /* ulangi tepat lima kali */
        printf("Weight of order %d (kg): ", counter);
        if (scanf("%d", &weight) != 1) return 1;
        total = total + weight;
        counter = counter + 1;
    }
    printf("Total weight: %d kg\n", total);
    printf("Average weight: %d kg\n", total / 5);   /* pembagian bilangan bulat */
    return 0;
}
```

```
Weight of order 1 (kg): 4
Weight of order 2 (kg): 7
Weight of order 3 (kg): 3
Weight of order 4 (kg): 6
Weight of order 5 (kg): 6
Total weight: 26 kg
Average weight: 5 kg
```

Badan perulangan dijalankan lima kali, sekali untuk setiap nilai `counter` dari 1 sampai 5; sesudah iterasi kelima `counter` menjadi 6, kondisi `counter <= 5` bernilai salah, dan program berlanjut ke `printf` pertama sesudah perulangan. Rata-rata sebenarnya dari 4, 7, 3, 6, dan 6 adalah 26/5 = 5,2, tetapi program mencetak 5, karena `total / 5` membagi dua bilangan bulat. Subbab berikutnya memperbaikinya.

Kode 2.4 adalah perulangan yang sama dalam Python. Perhatikan bagaimana prompt memakai f-string untuk menyertakan nomor pesanan, di tempat C memakai `%d`.

Kode 2.4 `average_counter.py`: perulangan terkendali pencacah yang sama dalam Python.

```python
# average_counter.py - rata-rata berat tepat lima pesanan
counter = 1     # nomor pesanan yang sedang dibaca
total = 0       # jumlah berjalan dari berat

while counter <= 5:                         # ulangi tepat lima kali
    weight = int(input(f"Weight of order {counter} (kg): "))
    total = total + weight
    counter = counter + 1

print(f"Total weight: {total} kg")
print(f"Average weight: {total // 5} kg")    # pembagian bilangan bulat
```

2.7 Penghalusan Bertahap dari Atas ke Bawah: Perulangan yang Dikendalikan Sentinel

2.7.1 Masalah yang lebih sulit

Pertanyaan Pak Hendra yang sesungguhnya lebih sulit: *berapa rata-rata berat semua pesanan hari ini?* Tidak ada yang tahu di pagi hari berapa pesanan yang akan datang. Kita memerlukan cara untuk memberi tahu program bahwa datanya sudah habis. Jalan keluar yang lazim adalah **nilai sentinel**, disebut juga *nilai bendera* atau *nilai sinyal*: nilai khusus yang tidak mungkin

menjadi data sungguhan dan karena itu berarti "tidak ada data lagi". Tidak ada kantong cucian yang beratnya 0 kg, jadi 0 dapat menjadi sentinel kita. Ketika kasir mengetik 0, hari itu selesai. Inilah **perulangan terkendali sentinel** (*sentinel-controlled repetition*).

Untuk merancang programnya kita memakai teknik yang berlaku untuk masalah sebesar apa pun: **penghalusan bertahap dari atas ke bawah** (*top-down, stepwise refinement*) [6]. Kita mulai dari satu pernyataan tentang seluruh tugas, yaitu **puncaknya**, lalu menghaluskannya berulang kali, setiap kali memecah satu pernyataan menjadi beberapa pernyataan yang lebih kecil, sampai setiap baris cukup sederhana untuk langsung diterjemahkan ke C atau Python.

2.7.2 Puncak dan penghalusan pertama

Puncaknya, **Pseudokode 2.4**, berupa satu kalimat:

Pseudokode 2.4 Puncak: satu kalimat yang menyatakan seluruh tugas.

```
AVERAGE-WEIGHT
1  determine the average weight of the day's orders
```

Kalimat ini lengkap, tetapi terlalu samar untuk diprogram. Penghalusan pertama, **Pseudokode 2.5**, membaginya menjadi tiga tahap yang dimiliki hampir setiap program, yaitu inisialisasi, pemrosesan, dan penyelesaian:

Pseudokode 2.5 Penghalusan pertama: inisialisasi, proses, selesaikan.

```
AVERAGE-WEIGHT
1  initialize the variables
2  read, add and count the weights until the sentinel arrives
3  compute and print the average
```

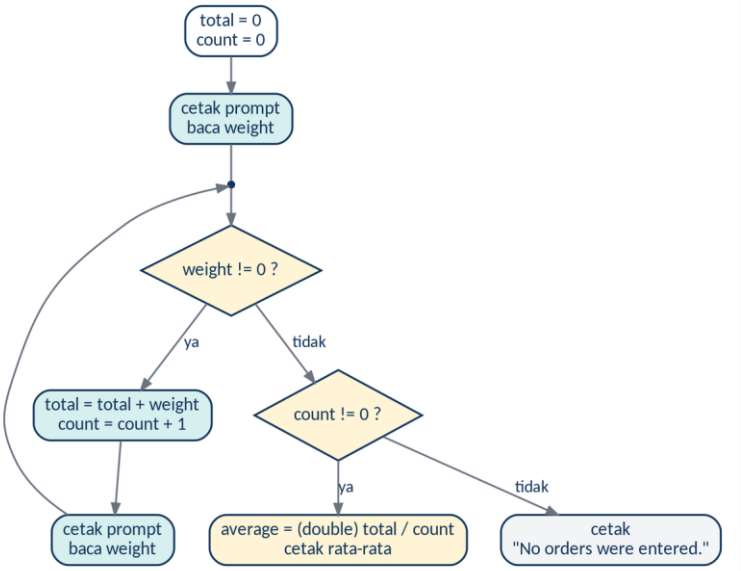
2.7.3 Penghalusan kedua

Sekarang setiap baris dihaluskan bergiliran. *Inisialisasi variabel*: kita memerlukan `total` berjalan dan `count` untuk banyaknya pesanan, keduanya mulai dari 0. *Baca, jumlahkan, dan hitung berat sampai sentinel tiba*: ini sebuah perulangan, dan di sini penghalusan menyingkap satu hal halus. Perulangan harus menguji berat *sebelum* menjumlahkannya, karena sentinel 0 tidak boleh dihitung sebagai pesanan. Jadi kita membaca berat pertama *sebelum* perulangan, mengujinya di puncak perulangan, dan membaca berat berikutnya di *akhir* badan. *Hitung dan cetak rata-rata*: di sini penghalusan menyingkap hal kedua, yang tersembunyi sepenuhnya di balik puncak satu baris tadi. Bila nilai pertama sudah berupa sentinel, `count` bernilai 0, dan membagi dengan 0 akan menjadi bencana. Program harus memeriksa kasus ini. **Pseudokode 2.6** adalah hasilnya.

Pseudokode 2.6 Penghalusan kedua: perulangan terkendali sentinel dengan pengaman terhadap pembagian dengan nol.

```
AVERAGE-WEIGHT
1  total ← 0
2  count ← 0
3  print "Weight in kg (0 to finish): "
4  read weight
5  while weight ≠ 0
6      total ← total + weight
7      count ← count + 1
8      print "Weight in kg (0 to finish): "
9      read weight
10 if count ≠ 0
11     average ← total / count   (with decimals)
12     print count, total and average
13 else
14     print "No orders were entered."
```

Gambar 2.2 menggambarkan algoritma yang sama sebagai diagram alir. Ikutilah anak panahnya: berat pertama dibaca sekali; belah ketupat mengujinya; perulangan menjumlahkan, menghitung, membaca berat berikutnya, lalu kembali ke belah ketupat; hanya ketika sentinel tiba jalurnya meninggalkan perulangan, dan belah ketupat kedua melindungi pembagiannya.



Gambar 2.2 Rata-rata terkendali sentinel: baca sebelum perulangan, uji di puncak, baca lagi di akhir badan, dan bagi hanya bila ada paling sedikit satu pesanan.

Setiap baris **Pseudokode 2.6** kini dapat diterjemahkan langsung, dan penghalusan selesai. Perhatikan bahwa dua keputusan terpenting program ini, membaca lebih dahulu dan melindungi pembagian, ditemukan di atas kertas. Definisi matematis yang kita terapkan adalah rata-rata biasa: untuk n berat $w_1, w_2, \ldots, w_n$,

$$\bar{w} = \frac{w_1 + w_2 + \cdots + w_n}{n} = \frac{1}{n}\sum_{i=1}^{n} w_i \tag{2.2}$$

dan Persamaan (2.2) hanya terdefinisi untuk $n \geq 1$, tepat kasus yang diperiksa baris 10 pseudokode tadi.

2.7.4 Programnya

Kode 2.5 adalah terjemahannya ke C.

Kode 2.5 `average_sentinel.c`: perulangan terkendali sentinel, sebuah *cast*, dan pengaman terhadap pembagian dengan nol.

```c
/* average_sentinel.c - rata-rata berat berapa pun pesanan; 0 mengakhiri */
#include <stdio.h>

int main(void)
{
    int weight;         /* berat satu pesanan, atau sentinel 0 */
    int total = 0;
    int count = 0;

    printf("Weight in kg (0 to finish): ");          /* baca sebelum perulangan */
    if (scanf("%d", &weight) != 1) return 1;
    while (weight != 0) {
        total = total + weight;
        count = count + 1;
        printf("Weight in kg (0 to finish): ");      /* baca lagi di akhir */
        if (scanf("%d", &weight) != 1) return 1;
    }

    if (count != 0) {
        double average = (double) total / count;     /* cast: pertahankan pecahan */
        printf("Orders: %d, total %d kg\n", count, total);
        printf("Average weight: %.2f kg\n", average);
    } else {
        printf("No orders were entered.\n");
    }
    return 0;
}
```

```
Weight in kg (0 to finish): 4
Weight in kg (0 to finish): 7
Weight in kg (0 to finish): 3
Weight in kg (0 to finish): 0
Orders: 3, total 14 kg
Average weight: 4.67 kg
Weight in kg (0 to finish): 0
No orders were entered.
```

Jalan pertama merata-ratakan tiga pesanan: 14 kg dibagi 3 adalah 4,666..., dicetak dengan dua desimal sebagai 4.67. Jalan kedua adalah hari listrik padam: nilai pertama sudah sentinel, badan perulangan tidak pernah dijalankan, dan alih-alih membagi dengan nol program menyatakannya.

2.7.5 Memeriksa rancangan dengan tangan: tabel penelusuran

Sebelum memercayai sebuah perulangan, kita dapat *menjalankannya dengan tangan*. **Tabel penelusuran** (*trace table*) memiliki satu kolom untuk setiap variabel dan satu baris untuk setiap langkah; kita menuliskan nilai setiap variabel sesudah setiap baris yang mengubahnya. **Tabel 2.1** menelusuri **Pseudokode 2.6** untuk masukan 4, 7, 3, 0, masukan yang sama dengan jalan pertama tadi.

Tabel 2.1 Penelusuran **Pseudokode 2.6** untuk masukan 4, 7, 3, 0: sentinel dibaca di akhir iterasi ketiga dan tidak pernah dijumlahkan.

Langkah (baris)	`weight`	`total`	`count`	`weight ≠ 0`?
sebelum perulangan (1–4)	4	0	0	ya
iterasi ke-1 (6–9)	7	4	1	ya
iterasi ke-2 (6–9)	3	11	2	ya
iterasi ke-3 (6–9)	0	14	3	tidak, keluar dari perulangan
sesudah perulangan (10–12)	0	14	3	rata-rata 14 / 3

Tabel ini menjawab pertanyaan-pertanyaan penting sebelum kode apa pun ada. Berapa kali badan dijalankan? Tiga kali, sekali untuk setiap berat sungguhan. Apakah sentinel pernah ditambahkan ke `total`? Tidak: ia dibaca di akhir iterasi ketiga dan langsung menghentikan perulangan. Apakah `count` benar? Ya, 3. Tabel penelusuran hanya memakan dua menit dan menemukan sebagian besar kesalahan perulangan, terutama perulangan yang berjalan sekali terlalu banyak atau sekali terlalu sedikit, yang oleh para pemrogram disebut *off-by-one error*.

2.7.6 *Cast*

Rata-rata pada jalan pertama memiliki pecahan berkat sepotong kecil kode, `(double) total`. `total` dan `count` sama-sama `int`, dan membagi keduanya akan membuang pecahan, seperti pada program pencacah. **Operator cast** `(double)` membuat salinan sementara `total` bertipe `double`; variabelnya sendiri tetap `int`. Kini pembagian itu memiliki `double` di satu ruas dan `int` di ruas lain, dan C menerapkan aturan yang disebut **promosi**: sebelum sebuah operasi, operand bertipe "lebih kecil" diubah ke tipe yang "lebih besar". Jadi `count` juga diubah menjadi `double`, dan pembagiannya mempertahankan pecahan.

Letak *cast* itu penting. `(double) (total / count)` tampak mirip tetapi salah: tanda kurung membuat pembagian bulat dikerjakan *lebih dahulu*, pecahannya sudah hilang, dan *cast* hanya mengubah 4 menjadi 4.0. Lakukan *cast* pada operand, bukan pada hasil yang sudah jadi.

Python tidak memerlukan *cast*: `/`-nya selalu menghasilkan pecahan. **Kode 2.6** adalah program Python-nya.

Kode 2.6 `average_sentinel.py`: algoritma yang sama dalam Python.

```python
# average_sentinel.py - rata-rata berat berapa pun pesanan; 0 mengakhiri
total = 0
count = 0

weight = int(input("Weight in kg (0 to finish): "))      # baca sebelum perulangan
while weight != 0:
    total = total + weight
    count = count + 1
    weight = int(input("Weight in kg (0 to finish): "))  # baca lagi di akhir

if count != 0:
    average = total / count                  # / selalu mempertahankan pecahan
    print(f"Orders: {count}, total {total} kg")
    print(f"Average weight: {average:.2f} kg")
else:
    print("No orders were entered.")
```

Format `{average:.2f}` membulatkan ke dua desimal untuk dicetak, seperti `%.2f` di C. Kedua bahasa membulatkan nilai yang sama dengan cara yang sama, karena `float` di Python dan `double` di C adalah jenis bilangan yang sama.

> **WAWASAN KUNCI:** Dua pola perulangan di bab ini layak dihafal. *Terkendali pencacah:* inisialisasi pencacah, uji di puncak, naikkan di akhir badan. *Terkendali sentinel:* baca nilai pertama sebelum perulangan, uji di puncak, proses, lalu baca nilai berikutnya di akhir badan. Hampir setiap *bug* perulangan pada program pertama berasal dari mencampur keduanya, misalnya hanya membaca di dalam perulangan lalu memproses sentinel seolah-olah data.

2.8 Struktur Kontrol Bersarang

Struktur kontrol dapat diletakkan di dalam struktur lain. Di akhir pekan Profitina Laundry bertanya kepada sepuluh pelanggan apakah mereka puas: 1 berarti puas, 2 berarti tidak puas. Jika lebih dari delapan orang puas, Pak Hendra mentraktir timnya makan siang. Perulangannya berjalan sejumlah kali yang diketahui, yaitu sepuluh, jadi terkendali pencacah; di dalam perulangan setiap jawaban harus digolongkan, jadi sebuah `if`…`else` berada *di dalam* `while`. Sesudah perulangan, satu keputusan lagi memeriksa hasilnya. **Pseudokode 2.7** menunjukkan rancangannya.

Pseudokode 2.7 Pseudokode survei: sebuah seleksi yang bersarang di dalam perulangan terkendali pencacah, diikuti seleksi kedua.

```
SURVEY
1  satisfied ← 0
2  unhappy ← 0
3  customer ← 1
4  while customer ≤ 10
5      read answer
6      if answer = 1
7          satisfied ← satisfied + 1
8      else
9          unhappy ← unhappy + 1
10     customer ← customer + 1
11 print satisfied and unhappy
12 if satisfied > 8
13     print "Excellent week: treat the team to lunch!"
```

Kode 2.7 `survey.c`: `if`…`else` yang bersarang di dalam `while`.

```c
/* survey.c - sepuluh pelanggan menilai layanan: 1 = puas, 2 = tidak */
#include <stdio.h>

int main(void)
{
    int satisfied = 0;
    int unhappy = 0;
    int customer = 1;
    int answer;

    while (customer <= 10) {
        printf("Customer %d (1 = satisfied, 2 = not): ", customer);
        if (scanf("%d", &answer) != 1) return 1;
        if (answer == 1) {                  /* if di dalam while */
            satisfied = satisfied + 1;
        } else {
            unhappy = unhappy + 1;
        }
        customer = customer + 1;
    }
    printf("Satisfied: %d\n", satisfied);
    printf("Not satisfied: %d\n", unhappy);
    if (satisfied > 8) {
        printf("Excellent week: treat the team to lunch!\n");
    }
    return 0;
}
```

```
Customer 1 (1 = satisfied, 2 = not): 1
Customer 2 (1 = satisfied, 2 = not): 1
Customer 3 (1 = satisfied, 2 = not): 2
Customer 4 (1 = satisfied, 2 = not): 1
Customer 5 (1 = satisfied, 2 = not): 1
Customer 6 (1 = satisfied, 2 = not): 1
Customer 7 (1 = satisfied, 2 = not): 1
Customer 8 (1 = satisfied, 2 = not): 1
Customer 9 (1 = satisfied, 2 = not): 1
Customer 10 (1 = satisfied, 2 = not): 1
Satisfied: 9
Not satisfied: 1
Excellent week: treat the team to lunch!
```

Sembilan dari sepuluh orang puas, lebih dari delapan, sehingga pesan penutup muncul. **Kode 2.7** menerjemahkan **Pseudokode 2.7** baris demi baris; tambahannya hanyalah prompt, yang memberi tahu pelanggan apa yang harus diketik, dan pemeriksaan hasil `scanf`. **Kode 2.8** adalah versi Python-nya, dengan struktur `while`, `if`, dan `else` yang persis sama, dan ia mencetak baris-baris yang sama.

Kode 2.8 `survey.py`: struktur bersarang yang sama dalam Python, dengan indentasi saja yang menunjukkan bahwa `if` menjadi milik `while`.

```python
# survey.py - sepuluh pelanggan menilai layanan: 1 = puas, 2 = tidak
satisfied = 0
unhappy = 0
customer = 1

while customer <= 10:
    answer = int(input(f"Customer {customer} (1 = satisfied, 2 = not): "))
    if answer == 1:                          # if di dalam while
        satisfied = satisfied + 1
    else:
        unhappy = unhappy + 1
    customer = customer + 1

print(f"Satisfied: {satisfied}")
print(f"Not satisfied: {unhappy}")
if satisfied > 8:
    print("Excellent week: treat the team to lunch!")
```

Bandingkan kedua kode itu sekali lagi. Di C, kurung kurawal menunjukkan di mana badan `while` berakhir; kurung kurawal penutup sesudah `customer = customer + 1;` adalah akhir perulangan, dan baris-baris `printf` sesudahnya dijalankan sekali. Di Python, informasi yang sama dibawa oleh indentasi semata: `customer = customer + 1` menjorok satu tingkat, jadi ia milik perulangan, sedangkan baris-baris `print` dimulai di tepi kiri, jadi dijalankan sesudahnya. Geser `customer = customer + 1` satu tingkat ke kiri di Python, dan perulangannya tidak pernah berakhir; pindahkan ke luar kurung kurawal penutup di C, dan hal yang sama terjadi. Struktur bersarang hanya mudah dibaca bila indentasinya jujur, dan karena itu pemformatan otomatis VS Code layak dipakai. Program ini menganggap setiap jawaban bernilai 1 atau 2: bilangan lain dihitung sebagai "tidak puas". Soal Latihan 2.4 meminta Anda menolak jawaban semacam itu.

2.9 Operator Penugasan dan Operator Increment

2.9.1 Penugasan gabungan

Pernyataan seperti `total = total + weight;` begitu lazim sehingga C punya bentuk yang lebih pendek: `total += weight;`. Setiap operator aritmetika biner memiliki **operator penugasan gabungan** (*compound assignment operator*) semacam itu, yang didaftar di **Tabel 2.2**. Variabelnya cukup disebut sekali, sehingga menghemat ketikan dan menghapus peluang menulis `total = totl + weight` secara tak sengaja.

Tabel 2.2 Operator penugasan gabungan di C; Python memiliki operator yang sama, dengan `//=` untuk pembagian bulat.

Operator	Contoh	Artinya	Hasil bila `c` bernilai 12
`+=`	`c += 7`	`c = c + 7`	19
`-=`	`c -= 4`	`c = c - 4`	8
`*=`	`c *= 5`	`c = c * 5`	60
`/=`	`c /= 3`	`c = c / 3`	4
`%=`	`c %= 5`	`c = c % 5`	2

2.9.2 Increment dan decrement

Menambah 1 adalah perubahan yang paling sering terjadi, dan C punya dua operator khusus untuk itu: `++` menambah 1 dan `--` mengurangi 1. Masing-masing hadir dalam dua bentuk. Bentuk **prefiks** `++c` menambah 1 lebih dahulu lalu memakai nilai baru di dalam ekspresi di sekitarnya; bentuk **postfiks** `c++` memakai nilai lama lalu menambah 1 sesudahnya. Bila operator itu berdiri sendiri sebagai pernyataan, `++c;` dan `c++;` melakukan hal yang sama; perbedaannya baru tampak ketika nilainya dipakai dalam ekspresi yang sama. **Kode 2.9** memperlihatkan kedua operator bekerja.

Kode 2.9 `increment.c`: penugasan gabungan, serta *increment* prefiks lawan postfiks.

```c
/* increment.c - penugasan gabungan, serta ++c lawan c++ */
#include <stdio.h>

int main(void)
{
    int total = 40000;
    total += 12000;                 /* total = total + 12000 */
    printf("total after += : %d\n", total);
    total -= 2000;
    printf("total after -= : %d\n", total);
    total *= 2;
    printf("total after *= : %d\n", total);
    total /= 4;
    printf("total after /= : %d\n", total);
    total %= 7;
    printf("total after %%= : %d\n", total);

    int c = 5;
    printf("c++ shows %d, ", c++);  /* pakai 5, lalu tambah 1 */
    printf("afterwards c = %d\n", c);
    c = 5;
    printf("++c shows %d, ", ++c);  /* tambah 1, lalu pakai 6 */
    printf("afterwards c = %d\n", c);
    return 0;
}
```

```
total after += : 52000
total after -= : 50000
total after *= : 100000
total after /= : 25000
total after %= : 3
c++ shows 5, afterwards c = 6
++c shows 6, afterwards c = 6
```

Ikutilah paruh pertama baris demi baris: 40.000 ditambah 12.000 adalah 52.000; dikurangi 2.000 menjadi 50.000; dikali 2 menjadi 100.000; dibagi 4 menjadi 25.000; dan 25.000 modulo 7 adalah 3, karena $25000 = 3571 \cdot 7 + 3$. Di paruh kedua, `c++` menampilkan 5 dan meninggalkan 6, sedangkan `++c` langsung menampilkan 6.

Python sengaja tidak memiliki `++` maupun `--`: `c += 1` adalah satu-satunya cara, dan ia adalah pernyataan, bukan ekspresi, sehingga pertanyaan “sebelum atau sesudah” tidak pernah muncul. **Kode 2.10** meniru kedua bentuk C dengan menuliskan dua langkahnya dalam dua urutan.

Kode 2.10 `increment.py`: hasil yang sama di Python, yang memiliki penugasan gabungan tetapi tidak memiliki `++`.

```python
# increment.py - penugasan gabungan; Python tidak punya operator ++
total = 40000
total += 12000                  # total = total + 12000
print(f"total after += : {total}")
total -= 2000
print(f"total after -= : {total}")
total *= 2
print(f"total after *= : {total}")
total //= 4                     # // menjaganya tetap bulat, seperti / di C
print(f"total after /= : {total}")
total %= 7
print(f"total after %= : {total}")

c = 5
print(f"c++ shows {c}, ", end="")    # pakai 5 ...
c += 1                               # ... lalu tambah 1
print(f"afterwards c = {c}")
c = 5
c += 1                               # tambah 1 dulu ...
print(f"++c shows {c}, ", end="")    # ... lalu pakai 6
print(f"afterwards c = {c}")
```

> **PERINGATAN:** Jangan pernah memakai sebuah variabel dua kali dalam ekspresi yang juga menaikkannya, seperti `printf("%d %d", c, c++);` atau `x = x++;`. Standar C tidak menetapkan urutan pengerjaan bagian-bagiannya, dan hasilnya dapat berbeda antarkompilator atau bahkan antartingkat optimasi. GCC memperingatkannya dengan `-Wall` (`operation on 'c' may be undefined`). Lakukan *increment* dalam pernyataan tersendiri.

2.10 Kasir Profitina Laundry v0.2

Semuanya siap untuk kasir yang buka sepanjang hari. Ia harus mengulang untuk setiap pelanggan sampai kasir ditutup (perulangan terkendali sentinel, dengan 0 sebagai sentinel “tutup kasir”), memilih harga menurut layanan (rantai seleksi), menolak layanan yang tidak dikenal, dan mengakumulasi uang masuk serta jumlah pelanggan (penugasan gabungan dan *increment*). Saat tutup, ia menjawab tiga pertanyaan Pak Hendra. **Pseudokode 2.8** adalah rancangan sesudah dua kali penghalusan.

Pseudokode 2.8 Rancangan kasir v0.2: perulangan sentinel yang membungkus sebuah seleksi, dan rata-rata yang dilindungi di akhir.

```
CASHIER-V02
1  revenue ← 0
2  customers ← 0
3  print the menu
4  read service
5  while service ≠ 0
6      if service is 1, 2 or 3
7          price ← the price of that service
8          read weight
9          bill ← weight × price
10         revenue ← revenue + bill
11         customers ← customers + 1
12         print the bill
13     else
14         print "Unknown service, try again."
15     read service
16 print customers and revenue
17 if customers > 0
18     print revenue / customers   (with two decimals)
```

Kode 2.11 adalah program C-nya. Ia mengikuti pseudokode dengan cermat, dengan satu rincian khas C: layanan yang tidak dikenal mencetak pesannya, membaca layanan berikutnya, lalu memakai pernyataan `continue`, yang melewati sisa badan dan langsung melompat kembali ke uji perulangan. Tanpanya, program akan lanjut menanyakan berat untuk layanan yang tidak ada.

Kode 2.11 `cashier_v02.c`: kasir v0.2 melayani antrean pelanggan sampai kasir ditutup.

```c
/* cashier_v02.c - kasir Profitina Laundry v0.2: antrean pelanggan */
#include <stdio.h>

int main(void)
{
    int service;            /* 1..3, atau sentinel 0 = tutup kasir */
    int weightKg;
    int price;              /* harga per kg layanan yang dipilih */
    int bill;               /* tagihan satu pelanggan */
    int revenue = 0;        /* semua tagihan hari ini */
    int customers = 0;

    printf("=== Profitina Laundry - cashier v0.2 ===\n");
    printf("1 = Wash, dry & fold (8000/kg)  2 = Express (12000/kg)\n");
    printf("3 = Ironing only (6000/kg)      0 = close the till\n");
    printf("Service: ");
    if (scanf("%d", &service) != 1) return 1;

    while (service != 0) {
        if (service == 1) {
            price = 8000;
        } else if (service == 2) {
            price = 12000;
        } else if (service == 3) {
            price = 6000;
        } else {
            printf("Unknown service, try again.\n");
            printf("Service: ");
            if (scanf("%d", &service) != 1) return 1;
            continue;       /* kembali ke uji perulangan */
        }
        printf("Weight (kg): ");
        if (scanf("%d", &weightKg) != 1) return 1;
        bill = weightKg * price;
        revenue += bill;
        ++customers;
        printf("Customer #%d pays Rp %d\n", customers, bill);
        printf("Service: ");
        if (scanf("%d", &service) != 1) return 1;
    }
```

Kode 2.11 (Lanjutan) *cashier_v02.c: kasir v0.2 melayani antrean pelanggan sampai kasir ditutup.*

```c
    printf("\n--- END OF DAY ---\n");
    printf("Customers served : %d\n", customers);
    printf("Revenue          : Rp %d\n", revenue);
    if (customers > 0) {
        printf("Average bill     : Rp %.2f\n", (double) revenue / customers);
    }
    return 0;
}
```

Berikut satu hari yang singkat: cuci 5 kg, pesanan ekspres 3 kg, layanan 7 yang salah ketik, setrika 2 kg, lalu tutup.

```
=== Profitina Laundry - cashier v0.2 ===
1 = Wash, dry & fold (8000/kg)  2 = Express (12000/kg)
3 = Ironing only (6000/kg)      0 = close the till
Service: 1
Weight (kg): 5
Customer #1 pays Rp 40000
Service: 2
Weight (kg): 3
Customer #2 pays Rp 36000
Service: 7
Unknown service, try again.
Service: 3
Weight (kg): 2
Customer #3 pays Rp 12000
Service: 0

--- END OF DAY ---
Customers served : 3
Revenue          : Rp 88000
Average bill     : Rp 29333.33
```

Periksalah angkanya dengan tangan, seperti yang akan dilakukan Pak Hendra: 5 × 8.000 = 40.000; 3 × 12.000 = 36.000; 2 × 6.000 = 12.000. Totalnya Rp88.000 dari tiga pelanggan, dan rata-ratanya 88.000 / 3 = 29.333,33. Angka 7 yang salah ketik ditolak dan tidak dihitung. **Kode 2.12** adalah versi Python-nya.

Kode 2.12 `cashier_v02.py`: kasir v0.2 dalam Python.

```python
# cashier_v02.py - kasir Profitina Laundry v0.2: antrean pelanggan
revenue = 0             # semua tagihan hari ini
customers = 0

print("=== Profitina Laundry - cashier v0.2 ===")
print("1 = Wash, dry & fold (8000/kg)  2 = Express (12000/kg)")
print("3 = Ironing only (6000/kg)      0 = close the till")
service = int(input("Service: "))

while service != 0:
    if service == 1:
        price = 8000
    elif service == 2:
        price = 12000
    elif service == 3:
        price = 6000
    else:
        print("Unknown service, try again.")
        service = int(input("Service: "))
        continue        # kembali ke uji perulangan
    weight_kg = int(input("Weight (kg): "))
    bill = weight_kg * price
    revenue += bill
    customers += 1
    print(f"Customer #{customers} pays Rp {bill}")
    service = int(input("Service: "))

print()
print("--- END OF DAY ---")
print(f"Customers served : {customers}")
print(f"Revenue          : Rp {revenue}")
if customers > 0:
    print(f"Average bill     : Rp {revenue / customers:.2f}")
```

`continue` di Python bekerja sama seperti di C. Perhatikan bahwa program Python tidak memerlukan deklarasi variabel di bagian atas: `price`, `weight_kg`, dan `bill` muncul di dalam perulangan, pada saat pertama kali ditugaskan.

2.11 Membuatnya Lebih Baik: Menjaga dari Masukan yang Mustahil

Kasir v0.2 menolak layanan yang tidak dikenal, tetapi dengan senang hati menerima berat 0 atau −3 kg lalu mencetak tagihan Rp0 atau tagihan negatif yang *mengurangi* uang masuk hari itu. Di toko sungguhan hal ini menjadi masalah: satu jari tergelincir, dan total akhir hari tidak lagi cocok dengan isi laci. Perbaikannya adalah pengaman kedua, kali ini pada berat. Dalam pseudokode, baris 8 dan 9 pada **Pseudokode 2.8** menjadi **Pseudokode 2.9**:

Pseudokode 2.9 Masukan berat yang dihaluskan: tanya lagi sampai beratnya masuk akal.

```
8a  read weight
8b  while weight < 1 or weight > 50
8c      print "Please type a weight from 1 to 50."
8d      read weight
9   bill ← weight × price
```

Batas atas 50 kg adalah keputusan usaha, bukan hukum alam: belum pernah seorang pelanggan toko ini membawa lebih dari itu, dan angka 500 yang terketik pasti sebuah kekeliruan. Perulangannya membaca lebih dahulu dan menguji di puncak, pola yang sama dengan perulangan sentinel. Bab 3 memperkenalkan perulangan yang lebih cocok lagi untuk tugas "tanya sampai sah" ini, yaitu `do`…`while`, serta operator logika `||` yang menggabungkan kedua uji; kasir v0.3 di Subbab 3.9 memakai keduanya. Untuk sementara, perhatikan apa yang diberikan penghalusan kepada kita: sebuah kebutuhan baru ("tolak berat yang mustahil") berubah menjadi empat baris pseudokode *sebelum* ada kode apa pun, dan tempatnya tampak jelas dari strukturnya.

2.12 Memakainya Sungguhan

Bentuk kasir v0.2, yaitu perulangan yang membaca permintaan, memeriksanya, menanganinya menurut jenisnya, dan menambahkannya ke total berjalan, adalah bentuk setiap program kasir, dari mesin kasir warung sampai toko daring. Untuk menyesuaikannya dengan usaha Anda, daftarlah layanan dan harganya, beri setiap layanan nomor di menu, dan ganti rantai `if`…`else if` sesuai daftar itu; perulangan, sentinel, dan ringkasannya tetap sama. Tempat fotokopi mungkin menawarkan 1 = hitam putih, 2 = berwarna, 3 = jilid; penjual beras kecil mungkin menawarkan tiga jenis beras per kilogram.

Ambil depot isi ulang air minum sebagai contoh lengkap. Depot itu menjual isi ulang galon 19 liter seharga Rp6.000 untuk air saring biasa dan Rp8.000 untuk air mineral, serta galon baru berikut airnya seharga Rp45.000. Menunya menjadi 1 = isi ulang air saring, 2 = isi ulang air mineral, 3 = galon baru, 0 = tutup. Berat pada kasir v0.2 berubah menjadi *jumlah galon*, dan tagihannya adalah jumlah galon dikali harga. Tidak ada lagi yang berubah: perulangan tetap berjalan sampai 0, nomor yang tidak dikenal tetap ditolak dengan `continue`, dan pemilik tetap mendapat jumlah pelanggan, uang masuk, dan rata-rata tagihan di akhir hari. Bila depot juga mengantar, butir menu keempat dengan ongkos antar tetap, beserta pencacah pengantaran, muat di dalam rantai yang sama. Menulis pseudokode hasil penyesuaian lebih dahulu, seperti **Pseudokode 2.8**, hanya memakan lima menit dan langsung memperlihatkan baris-baris mana yang berubah.

Tiga hal praktis berlaku untuk setiap penyesuaian semacam itu. Simpan uang dalam rupiah bulat, seperti yang kita lakukan; bila rata-rata memerlukan desimal, hitunglah hanya untuk ditampilkan, seperti pada ringkasan. Pilih sentinel yang tidak mungkin menjadi data sungguhan: 0 cocok untuk nomor layanan di sini karena tidak ada layanan bernomor 0, tetapi tidak cocok untuk, misalnya, besarnya diskon, yang memang bisa bernilai 0. Dan ujilah hari yang kosong: tutup kasir sebelum

pelanggan pertama, lalu periksa bahwa program mencetak ringkasan dengan nol pelanggan dan tidak membagi dengan nol.

2.13 Kesalahan yang Sering Terjadi, dan Cara Menghindarinya

- **Lupa menginisialisasi akumulator.** `int total;` tanpa `= 0` bermula dari apa saja yang kebetulan ada di memori. GCC dengan `-Wall` sering memperingatkannya (`'total' may be used uninitialized`). Inisialisasi setiap pencacah dan total di tempat Anda mendeklarasikannya.
- **Memproses sentinel.** Perulangan yang membaca di dalam badan lalu menjumlahkan sebelum menguji akan menghitung 0 sebagai pesanan, atau, dengan sentinel seperti −1, malah menguranginya. Bacalah sebelum perulangan dan sekali lagi di akhir badan.
- **Perulangan tak berhingga.** Lupa `counter = counter + 1` atau lupa `read` yang kedua berarti kondisinya tidak pernah berubah. Bila program berhenti menanggapi, tekan Ctrl+C di terminal lalu carilah pernyataan yang seharusnya mengubah kondisi perulangan.
- **Membagi bilangan bulat padahal ingin pecahan.** `total / count` dengan dua `int` membuang pecahan; `(double) (total / count)` sudah terlambat. Tulislah `(double) total / count`. Di Python pakai `/`, bukan `//`.
- **Membagi dengan nol.** Lindungi setiap pembagian dengan hitungan memakai `if (count != 0)`. Di C, pembagian bulat dengan nol membuat program mogok; di Python ia memunculkan `ZeroDivisionError`.
- **Titik koma yang salah tempat.** `while (weight != 0);` atau `if (bill >= 100000);` mengakhiri pernyataan di titik koma itu: `while` berulang selamanya pada badan kosong, dan `if` tidak mengendalikan apa pun. GCC memperingatkan kasus `if` dengan `-Wextra`.
- **Urutan yang salah dalam rantai `else if`.** Kondisi yang tumpang tindih harus diurutkan dari yang paling menuntut ke yang paling longgar; jika tidak, uji pertama yang lebih lemah menangkap semuanya.
- **Mengandalkan indentasi di C.** Tanpa kurung kurawal, hanya pernyataan *pertama* sesudah `if` atau `while` yang menjadi miliknya, bagaimanapun baris-baris berikutnya diindentasi. Pakailah kurung kurawal bila badan berisi lebih dari satu pernyataan.
- **Sentinel yang bisa menjadi data sungguhan.** Sentinel harus mustahil sebagai data. Berat 0 tidak mungkin terjadi, jadi 0 aman; suhu 0 derajat atau diskon 0 rupiah bisa terjadi, dan program yang berhenti pada nilai semacam itu kehilangan data tanpa pesan galat apa pun. Bila tidak ada nilai yang mustahil, tanyakan lebih dahulu kepada pengguna berapa banyak nilai yang akan menyusul, lalu pakailah pencacah.
- **Membandingkan pecahan dengan `==`.** `if (average == 4.67)` hampir tidak pernah benar, karena 14.0 / 3 bernilai 4,666… di dalam komputer dan hanya *dicetak* sebagai 4.67. Bandingkan bilangan bulat dengan `==`, dan bandingkan pecahan dengan `<` atau `>` atau terhadap toleransi kecil; Bab 3 kembali membahas hal ini.
- **Hanya menguji jalur yang mulus.** Program yang hanya diuji dengan masukan biasa belum benar-benar diuji. Untuk setiap perulangan, cobalah kasus kosong (sentinel sebagai nilai pertama), kasus tunggal (satu nilai), dan kasus yang lazim; untuk setiap rantai, cobalah satu nilai di setiap cabang dan nilai-nilai tepat di batasnya, seperti tagihan tepat Rp50.000 dan Rp100.000. Kebiasaan ini terasa lambat pada awalnya, tetapi ia jauh lebih cepat daripada mencari kesalahan di hadapan pelanggan yang sedang menunggu struknya.

2.14 Ringkasan Bab

- Algoritma adalah daftar berhingga aksi-aksi yang tepat dalam urutan tertentu yang menyelesaikan masalah untuk setiap masukan yang diizinkan. Pseudokode menguraikan algoritma bagi manusia, dengan baris bernomor, `←` untuk penugasan, dan indentasi untuk struktur.
- Setiap algoritma dapat dibangun dari tiga struktur kontrol, runtunan, seleksi, dan perulangan, masing-masing dengan satu jalan masuk dan satu jalan keluar, yang digabungkan dengan penumpukan dan penyarangan (**Gambar 2.1**); program terstruktur tidak memerlukan `goto`.
- Seleksi: `if`, `if`…`else`, dan rantai yang ditulis `else if` di C dan `elif` di Python, dengan kondisi benar yang pertama sebagai pemenang. Operator kondisional `?:` (Python: `x if c else y`) memilih di antara dua nilai.
- Perulangan terkendali pencacah berjalan sejumlah kali yang diketahui (**Pseudokode 2.3**); perulangan terkendali sentinel berjalan sampai nilai khusus tiba, dengan membaca sebelum perulangan dan di akhir badan (**Pseudokode 2.6**, **Gambar 2.2**).
- Penghalusan bertahap dari atas ke bawah mengubah puncak satu kalimat menjadi pseudokode yang siap diterjemahkan, dan memunculkan keputusan rancangan, seperti pengaman terhadap pembagian dengan nol pada Persamaan (2.2), sebelum kode apa pun ditulis.
- *Cast* seperti `(double) total` membuat pembagian mempertahankan pecahannya dengan mempromosikan operand lainnya; melakukan *cast* pada hasil bagi yang sudah jadi sudah terlambat.
- Penugasan gabungan (`+=`, `-=`, `*=`, `/=`, `%=`) menyingkat pembaruan variabel (**Tabel 2.2**); `++c` menaikkan sebelum dipakai, `c++` sesudahnya. Python memiliki penugasan gabungan tetapi tidak memiliki `++`.
- Kasir v0.2 melayani pelanggan sampai sentinel 0, menolak layanan tak dikenal dengan `continue`, dan mencetak jumlah pelanggan, uang masuk, dan rata-rata tagihan.

2.15 Soal Latihan

1. Tulislah pseudokode, dengan dua kali penghalusan, untuk tugas ini: “baca berat pesanan-pesanan hari ini dan laporkan berapa yang beratnya lebih dari 5 kg”. Lalu programkan dalam C dan Python, dengan menolak berat negatif disertai pesan.
2. Tulis ulang rantai diskon pada **Kode 2.1** agar menguji `bill < 50000` lebih dahulu. Kondisi apa yang harus dipakai cabang-cabang lainnya agar hasilnya tetap sama?
3. Apa yang dicetak potongan kode ini, dan mengapa? `int c = 3; c += c++;` Jangan menjalankannya; jelaskan saja mengapa pemrogram yang cermat tidak pernah menulisnya.
4. Ubahlah survei agar jawaban selain 1 atau 2 ditolak dengan pesan `Please type 1 or 2.` dan ditanyakan lagi untuk pelanggan yang sama. Variabel mana yang *tidak* boleh dinaikkan ketika sebuah jawaban ditolak?
5. Telusurilah **Pseudokode 2.6** dengan tangan untuk masukan 5, 0 dan tuliskan nilai `total`, `count`, dan `weight` sesudah setiap baris. Berapa kali badan perulangan dijalankan?
6. Kembangkan kasir v0.2 agar ringkasannya juga mencetak berapa pelanggan yang memilih setiap layanan. Berapa variabel baru yang Anda perlukan, dan di mana variabel-variabel itu dinaikkan?

2.16 Penyelesaian Soal Latihan

1. Puncak: *hitung pesanan berat hari ini*. Penghalusan pertama: inisialisasi; baca dan golongkan berat sampai sentinel; cetak hasilnya. Penghalusan kedua: `orders ← 0`, `heavy ← 0`; baca berat pertama; selama bukan 0: jika negatif cetak pesan, jika tidak naikkan `orders` dan, bila di atas 5, naikkan `heavy`; baca berat berikutnya; akhirnya cetak `heavy` dan `orders`. **Kode 2.13** adalah program C-nya; `heavy_count.py` adalah versi Python-nya.

Kode 2.13 `heavy_count.c`: menghitung pesanan yang lebih berat dari 5 kg, dengan berat negatif ditolak.

```c
/* heavy_count.c - berapa pesanan hari ini yang lebih dari 5 kg? */
#include <stdio.h>

int main(void)
{
    int weight;
    int orders = 0;
    int heavy = 0;

    printf("Weight in kg (0 to finish): ");
    if (scanf("%d", &weight) != 1) return 1;
    while (weight != 0) {
        if (weight < 0) {
            printf("A weight cannot be negative; ignored.\n");
        } else {
            ++orders;
            if (weight > 5) {
                ++heavy;
            }
        }
        printf("Weight in kg (0 to finish): ");
        if (scanf("%d", &weight) != 1) return 1;
    }
    printf("%d of %d orders weigh more than 5 kg\n", heavy, orders);
    return 0;
}
```

```
Weight in kg (0 to finish): 4
Weight in kg (0 to finish): 7
Weight in kg (0 to finish): -2
A weight cannot be negative; ignored.
Weight in kg (0 to finish): 6
Weight in kg (0 to finish): 3
Weight in kg (0 to finish): 0
2 of 4 orders weigh more than 5 kg
```

Berat negatif bukan pesanan dan bukan pesanan berat; dari empat pesanan sungguhan, 7 kg dan 6 kg berada di atas 5 kg.

2. Uji dari bawah ke atas: `if (bill < 50000) percent = 0; else if (bill < 100000) percent = 5; else percent = 10;`. Cabang tengah baru dicapai ketika `bill >= 50000` sudah diketahui, jadi ia hanya memerlukan `bill < 100000`. Kedua versi memberi hasil yang sama untuk setiap tagihan; yang lebih wajar dibaca adalah yang lebih baik.

3. Tidak ada yang bisa memastikan. `c += c++` memakai sekaligus mengubah `c` tanpa urutan yang terdefinisi di antara keduanya, yang oleh standar C disebut *undefined behavior*: kompilator boleh menghasilkan 6, 7, atau apa saja, dan GCC memperingatkannya. Perbaikannya adalah memutuskan apa maksud Anda lalu menuliskannya dalam pernyataan terpisah, misalnya `c += c; c++;`.

4. Letakkan pembacaan di dalam perulangan dalam: baca jawabannya; selama bukan 1 dan bukan 2, cetak pesan lalu baca lagi. Hanya sesudah jawaban sah `satisfied` atau `unhappy` dinaikkan, dan baru sesudah itu `customer` dinaikkan; `customer` harus tetap selama sebuah jawaban diulang, sebab jika tidak, jawaban yang ditolak akan menghabiskan giliran seorang pelanggan.

5. Baris 1–2: `total` = 0, `count` = 0. Baris 3–4: `weight` = 5. Baris 5: 5 ≠ 0, masuk ke badan. Baris 6: `total` = 5. Baris 7: `count` = 1. Baris 8–9: `weight` = 0. Baris 5: 0 ≠ 0 bernilai salah, keluar dari perulangan. Baris 10: `count` = 1 ≠ 0. Baris 11: `average` = 5.0. Badan dijalankan tepat satu kali.

6. Tiga pencacah, `nWash`, `nExpress`, dan `nIroning`, semuanya diinisialisasi 0 sebelum perulangan. Masing-masing dinaikkan di cabangnya sendiri di dalam rantai, di sebelah baris yang menetapkan harga. Ringkasan mencetak ketiga pencacah; jumlahnya harus sama dengan `customers`, dan itu pemeriksaan yang berguna. Kasir v0.3 di Bab 3 melakukan persis hal ini.

Daftar Pustaka

[1] C. Böhm and G. Jacopini, “Flow diagrams, Turing machines and languages with only two formation rules,” *Commun. ACM*, vol. 9, no. 5, pp. 366–371, May 1966.

[2] E. W. Dijkstra, “Go To statement considered harmful,” *Commun. ACM*, vol. 11, no. 3, pp. 147–148, Mar. 1968.

[3] P. J. Deitel and H. M. Deitel, *C How to Program*, 9th ed. Hoboken, NJ, USA: Pearson, 2022.

[4] Python Software Foundation, “More Control Flow Tools,” *The Python Tutorial*, Python 3.14. Accessed: Sep. 25, 2026. [Online]. Available: https://docs.python.org/3.14/tutorial/controlflow.html

[5] cppreference.com, “Statements (C): if, while, continue.” Accessed: Sep. 25, 2026. [Online]. Available: https://en.cppreference.com/w/c/language/statements

[6] N. Wirth, “Program development by stepwise refinement,” *Commun. ACM*, vol. 14, no. 4, pp. 221–227, Apr. 1971.

Bab 3

Kontrol Program: for, do...while, switch, dan Operator Logika

Bab 2 memberi Anda gagasan seleksi dan perulangan beserta satu pernyataan untuk masing-masing jenis. Bab ini melengkapi kotak perkakasnya. Anda akan berkenalan dengan pernyataan `for`, yang menghimpun semua kebutuhan perulangan berhitung ke dalam satu baris; pernyataan `do`...`while`, yang menjalankan badannya paling sedikit sekali dan memang dibuat untuk "tanya terus sampai jawabannya sah"; pernyataan `switch`, yang memilih di antara banyak kasus tetap; pernyataan `break` dan `continue`, yang meninggalkan perulangan lebih awal atau melewati satu putaran; serta operator logika `&&`, `||`, dan `!`, yang memungkinkan satu kondisi menyatakan beberapa hal sekaligus. Python memiliki bentuknya sendiri untuk semua itu, `for` atas sebuah `range`, `while True` dengan `break`, dan pernyataan `match`, dan masing-masing ditampilkan di samping padanannya di C. Bab ini ditutup dengan versi 0.3 kasir Profitina Laundry, yang memeriksa setiap berat yang diterimanya, memilih layanan dengan `switch`, dan mencetak laporan harian berformat dengan kolom-kolom yang rata. Sesudah bab ini Anda mengenal semua struktur kontrol C, dan Bab 4 memakai semuanya untuk lima soal utuh.

3.1 Mesin Cuci Baru dan Laporan yang Lebih Rapi

Profitina Laundry punya tiga mesin cuci, dan yang paling tua mulai bocor. Pak Hendra ingin menggantinya tahun depan dan berencana menyisihkan uang dalam deposito di bank. Sebelum memutuskan berapa jumlahnya, ia ingin melihat tabel sederhana: jika hari ini ia menyetor satu juta rupiah dengan bunga 5% per tahun, berapa isi rekeningnya sesudah satu tahun, dua tahun, dan seterusnya sampai sepuluh tahun? Tidak ada yang mau menghitung sepuluh pangkat dengan tangan. Program dapat mencetak tabel itu dalam sekejap, dengan tahun di satu kolom dan jumlah uang yang rata di kolom lain.

Sementara itu, di meja kasir, kasir v0.2 telah menunjukkan titik-titik lemahnya. Mbak Rina pernah mengetik 60 alih-alih 6 untuk berat sebuah kantong, dan uang masuk hari itu melonjak Rp432.000 yang tidak pernah dibayar siapa pun. Program menerima berat itu tanpa berkata apa-apa. Pak Hendra juga menginginkan lebih dari tiga angka pada laporan harian: berapa pelanggan yang memilih setiap layanan, dan berapa tagihan terbesar hari itu, dicetak sebagai tabel yang dapat ia potret dan kirim ke grup percakapan keluarga.

Kedua keinginan itu memerlukan perkakas bab ini. Tabel tabungan adalah perulangan yang menghitung dari 1 sampai 10, pekerjaan alami `for`. Memeriksa berat berarti bertanya *sampai* jawabannya masuk akal, dan itulah guna `do`...`while`. Memilih di antara layanan adalah keputusan banyak arah, pekerjaan `switch`. Dan "berat harus paling sedikit 1 *dan* paling banyak 50" adalah kondisi dua bagian yang digabungkan dengan operator logika. Suku bunga di bab ini hanyalah contoh untuk aritmetikanya; suku bunga dan ketentuan deposito sungguhan berasal dari bank.

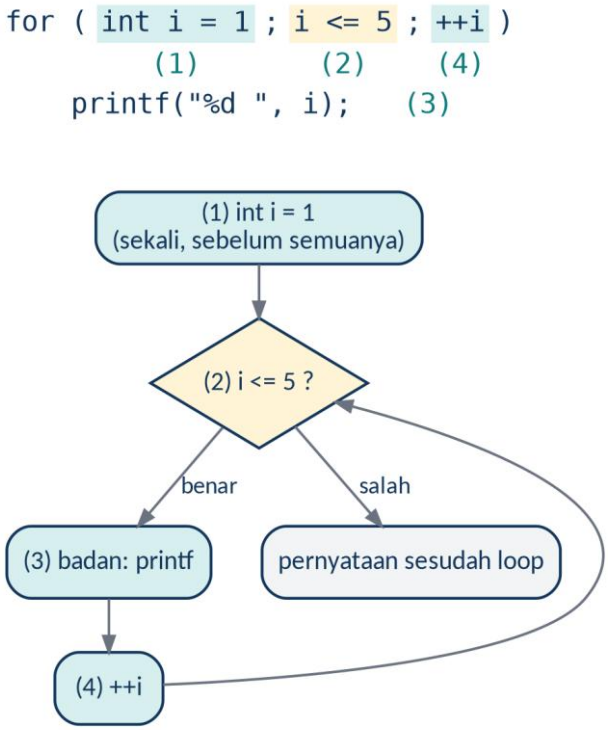
3.2 Pernyataan `for`

3.2.1 Perulangan berhitung dalam satu baris

Perulangan terkendali pencacah di Bab 2 memerlukan tiga bagian terpisah: pencacah diisi sebelum perulangan, diuji di puncak, dan dinaikkan di dasar badan. Bila tersebar di beberapa baris, setiap bagian mudah terlupa, dan melupakan kenaikan menghasilkan perulangan tak berhingga. Pernyataan `for` menghimpun ketiganya di kepalanya:

```
for (initialization; condition; increment)
    statement
```

Gambar 3.1 membongkar sebuah `for` dan menunjukkan urutan pelaksanaan bagian-bagiannya. (1) Inisialisasi dijalankan sekali, sebelum yang lain. (2) Kondisi diuji; jika salah, perulangan langsung berakhir. (3) Jika benar, badan dijalankan. (4) *Increment* dijalankan, lalu kendali kembali ke (2). Jadi urutannya 1, 2, 3, 4, 2, 3, 4, 2, ..., sampai uji pada langkah 2 gagal.



Gambar 3.1 Anatomi pernyataan `for`: inisialisasi (1) dijalankan sekali; lalu kondisi (2), badan (3), dan *increment* (4) berulang sampai kondisinya salah.

Perulangan `for` persis setara dengan perulangan `while` berikut, dan ada gunanya melihat terjemahan itu sekali:

```
initialization;
while (condition) {
    statement
    increment;
}
```

Bentuk `for` lebih disukai setiap kali perulangan dikendalikan pencacah, karena pembaca langsung melihat di mana pencacah mulai, di mana berhenti, dan bagaimana ia bergerak. **Kode 3.1** menunjukkan empat perulangan berhitung yang khas: menghitung naik, menghitung mundur, menghitung dengan langkah 7, dan menjumlahkan bilangan genap dari 2 sampai 100.

Kode 3.1 `for_basics.c`: menghitung naik, menghitung mundur, menghitung berlangkah, dan jumlah berjalan.

```c
/* for_basics.c - pernyataan for: naik, turun, berlangkah, dan penjumlahan */
#include <stdio.h>

int main(void)
{
    for (int counter = 1; counter <= 5; ++counter)   /* awal; uji; langkah */
        printf("%d ", counter);
    printf("\n");

    for (int i = 5; i >= 1; --i)             /* menghitung mundur */
        printf("%d ", i);
    printf("\n");

    for (int i = 7; i <= 77; i += 7)         /* langkah 7 */
        printf("%d ", i);
    printf("\n");

    int sum = 0;
    for (int i = 2; i <= 100; i += 2)        /* bilangan genap 2..100 */
        sum += i;
    printf("Sum of the even numbers 2..100: %d\n", sum);
    return 0;
}
```

```
1 2 3 4 5
5 4 3 2 1
7 14 21 28 35 42 49 56 63 70 77
Sum of the even numbers 2..100: 2550
```

Pencacahnya dideklarasikan di dalam kepala, `for (int counter = 1; …)`. Variabel semacam itu hanya ada di dalam perulangan: sesudah perulangan ia lenyap, dan perulangan lain boleh memakai nama yang sama lagi, seperti ketiga perulangan yang memakai `i`. Hal ini menjaga setiap pencacah tetap dekat dengan tempat pemakaiannya dan mencegah pencacah lama terpakai secara keliru di kemudian hari; standar C mengizinkannya sejak C99, dan `-std=c17` kita mencakupnya [1]. Badan setiap perulangan berupa satu pernyataan, jadi tidak perlu kurung kurawal; aturannya sama dengan `if`, dan bila lebih dari satu pernyataan, kurung kurawal wajib dipakai.

Perulangan terakhir menjumlahkan 2 + 4 + 6 + … + 100. Ada 50 suku, dan memasangkan yang pertama dengan yang terakhir (2 + 100), yang kedua dengan yang kedua terakhir (4 + 98), dan seterusnya menghasilkan 25 pasang bernilai 102, yaitu 2.550, sama dengan yang dicetak program. Mencocokkan program dengan hasil yang dihitung dengan cara lain adalah kebiasaan yang patut dipertahankan.

3.2.2 `for` dan `range` di Python

`for` di Python dibangun secara berbeda. Ia tidak menghitung sendiri; ia menelusuri sederet nilai satu per satu, dan sebuah fungsi bernama `range` menghasilkan deret bilangan yang ditelusuri [2]. **Kode 3.2** mencetak baris-baris yang persis sama.

Kode 3.2 `for_basics.py`: perulangan yang sama dengan `for` dan `range`.

```python
# for_basics.py - for di Python menelusuri sederet bilangan
for counter in range(1, 6):              # 1, 2, 3, 4, 5 (6 tidak termasuk)
    print(counter, end=" ")
print()

for i in range(5, 0, -1):                # menghitung mundur
    print(i, end=" ")
print()

for i in range(7, 78, 7):                # langkah 7
    print(i, end=" ")
print()

total = 0
for i in range(2, 101, 2):               # bilangan genap 2..100
    total += i
print(f"Sum of the even numbers 2..100: {total}")
```

`range(start, stop, step)` menghasilkan bilangan dari `start` sampai, tetapi **tidak termasuk**, `stop`, dengan langkah `step`. Jadi `range(1, 6)` menghasilkan 1, 2, 3, 4, 5, dan `range(5, 0, -1)` menghitung mundur dari 5 sampai 1. Aturan "stop tidak termasuk" membingungkan pemula pada awalnya, tetapi akibatnya menyenangkan: `range(0, n)`, yang dapat disingkat `range(n)`, menghasilkan tepat `n` bilangan. `print(counter, end=" ")` mencetak spasi alih-alih baris baru di akhir, padanan `printf("%d ", counter)`.

> **PERINGATAN:** Jangan mengendalikan perulangan dengan pencacah pecahan seperti `for (double x = 0.1; x != 1.0; x += 0.1)`. Nilai 0.1 tidak dapat disimpan tepat dalam biner, jumlahnya sesudah sembilan langkah adalah 0.9999999999999999 dan bukan 1.0, dan perulangannya tidak pernah berakhir. Hitunglah dengan bilangan bulat dan hitung pecahannya di dalam badan, misalnya `x = i / 10.0` [3].

3.3 Pameran `for`: Bunga Majemuk dan Lebar Kolom

3.3.1 Rumusnya

Deposito Pak Hendra memperoleh **bunga majemuk**: setiap tahun bank menambahkan 5% dari isi rekening, dan bunga tahun berikutnya dihitung dari jumlah yang lebih besar. Jika p adalah setoran awal (pokok), r suku bunga tahunan yang ditulis sebagai pecahan (5% adalah 0.05), dan n banyaknya tahun, maka jumlah a sesudah n tahun adalah

$$a = p\,(1 + r)^n \tag{3.1}$$

Mengapa pangkat? Sesudah satu tahun rekening berisi $p + p \cdot r = p(1 + r)$. Tahun kedua mengalikan jumlah *itu* dengan $(1 + r)$ lagi, sehingga menjadi $p(1 + r)^2$, dan setiap tahun berikutnya menambahkan satu faktor lagi. Untuk $p = 1{,}000{,}000$ dan $r = 0.05$, Persamaan (3.1) memberi 1.050.000 sesudah satu tahun dan 1.102.500 sesudah dua tahun, karena $1.05^2 = 1.1025$.

3.3.2 Programnya

Kode 3.3 mencetak tabel untuk sepuluh tahun. Ia memakai fungsi `pow` dari pustaka matematika: `pow(x, y)` menghitung x^y. *Header* `math.h` mendeklarasikannya, dan di Linux program harus di-*link* dengan pustaka matematika dengan menambahkan `-lm` di akhir perintah kompilasi: `gcc -std=c17 -Wall -Wextra -O2 savings.c -o savings -lm`. Di macOS dan MSYS2 opsi itu tidak berbahaya.

Kode 3.3 `savings.c`: setoran yang tumbuh dengan bunga majemuk, dicetak sebagai tabel yang rata.

```c
/* savings.c - setoran yang tumbuh dengan bunga majemuk, tahun demi tahun */
#include <stdio.h>
#include <math.h>

int main(void)
{
    double principal = 1000000.0;   /* setoran awal, rupiah */
    double rate = 0.05;             /* bunga 5% per tahun */

    printf("%4s%21s\n", "Year", "Amount on deposit");
    for (int year = 1; year <= 10; ++year) {
        double amount = principal * pow(1.0 + rate, year);
        printf("%4d%21.2f\n", year, amount);   /* lebar 4 dan 21 */
    }
    return 0;
}
```

```
Year    Amount on deposit
   1           1050000.00
   2           1102500.00
   3           1157625.00
   4           1215506.25
   5           1276281.56
   6           1340095.64
   7           1407100.42
   8           1477455.44
   9           1551328.22
  10           1628894.63
```

3.3.3 Lebar kolom

Tabel itu rata berkat **lebar kolom** (*field width*) di dalam spesifikasi konversi. `%4d` mencetak bilangan bulat rata kanan dalam ruang 4 karakter, dengan spasi pengisi di kiri; `%21.2f` mencetak `double` dengan 2 desimal, rata kanan dalam ruang 21 karakter. Baris judul memakai lebar yang sama dengan `%4s` dan `%21s` untuk kedua katanya, sehingga judul-judulnya berdiri tepat di atas kolomnya. Tanda minus di depan lebar, seperti `%-18s`, membuatnya rata kiri; kasir v0.3 memakainya untuk label laporannya. **Tabel 3.1** menghimpun bentuk-bentuk yang dipakai di buku ini [4].

Tabel 3.1 Lebar kolom dan presisi pada `printf`; spesifikasi format Python di dalam f-string memakai angka yang sama, dengan `>` untuk rata kanan dan `<` untuk rata kiri.

Spesifikasi	Arti	`printf` untuk 42 atau 3.14159 menghasilkan
`%d`	bilangan bulat, selebar yang diperlukan	`42`
`%5d`	bilangan bulat, rata kanan dalam 5 karakter	`42`
`%-5d`	bilangan bulat, rata kiri dalam 5 karakter	`42`
`%.2f`	double dengan 2 desimal	`3.14`
`%8.2f`	double dengan 2 desimal, rata kanan dalam 8 karakter	`3.14`
`%-10s`	teks, rata kiri dalam 10 karakter	`Express`

Kode 3.4 adalah versi Python-nya. Pangkatnya ditulis `(1.0 + rate) ** year`, dan spesifikasi format di dalam f-string membawa lebar yang sama: `{year:4d}`, `{amount:21.2f}`, dan `{'Year':>4}` untuk teks rata kanan dalam 4 karakter.

Kode 3.4 `savings.py`: tabel yang sama dalam Python.

```
# savings.py - setoran yang tumbuh dengan bunga majemuk, tahun demi tahun
principal = 1000000.0   # setoran awal, rupiah
rate = 0.05             # bunga 5% per tahun

print(f"{'Year':>4}{'Amount on deposit':>21}")
for year in range(1, 11):
    amount = principal * (1.0 + rate) ** year
    print(f"{year:4d}{amount:21.2f}")         # lebar 4 dan 21
```

Lebar kolom dipilih dari data terpanjang yang akan dicetak: jumlah uang terbesar di tabel ini, 1628894.63, memakai sepuluh karakter, jadi lebar 21 menyisakan ruang yang lega untuk judul kolom "Amount on deposit" yang panjangnya tujuh belas karakter. Bila lebarnya terlalu kecil, `printf` tidak memotong angkanya, tetapi kolomnya menjadi tidak rata lagi.

Kedua program sepakat sampai sen terakhir karena keduanya berhitung dengan bilangan titik-mengambang 64-bit yang sama, dan `**` di Python untuk dua float memanggil fungsi pustaka C yang sama dengan `pow`. Perhatikan, meskipun demikian, bahwa uang di sini dihitung dalam `double`, sesuatu yang di Bab 1 tidak dianjurkan. Untuk sebuah *perkiraan* yang dicetak dengan dua desimal hal itu dapat diterima. Untuk rekening sungguhan, ketika setiap rupiah harus tepat sesudah ribuan operasi, bank berhitung dalam satuan bulat dan membulatkan menurut aturan tetap pada saat-saat yang tetap; kita akan berbuat sama setiap kali uang disimpan, seperti yang dilakukan kasir.

3.3.4 Membaca tabel seperti seorang pemilik usaha

Sebuah tabel baru berguna bila seseorang menarik kesimpulan darinya. Misalkan mesin cuci baru berharga Rp1.500.000. Tabel menunjukkan bahwa deposito melewati jumlah itu pada tahun kesembilan: sesudah delapan tahun isinya Rp1.477.455,44, belum cukup, dan sesudah sembilan tahun Rp1.551.328,22. Jadi jika suku bunga bank tetap 5%, setoran satu juta hari ini baru dapat membayar mesin itu sembilan tahun lagi, jauh terlalu lambat untuk mesin yang sudah bocor sekarang. Karena itu Pak Hendra memutuskan menambah uang setiap bulan alih-alih menunggu, sebuah perhitungan lain yang sudah dapat Anda tulis dengan perulangan-perulangan di bab ini.

Pertanyaan "sesudah berapa tahun targetnya tercapai?" sendiri adalah sebuah perulangan kecil, tetapi bukan perulangan berhitung: kita tidak tahu sebelumnya berapa tahun yang diperlukan. Struktur yang wajar adalah perulangan `while` yang mengalikan jumlah uang dengan $1 + r$ dan menghitung tahun *selama* jumlahnya masih di bawah target. Memilih antara `for` dan `while` persis menjawab pertanyaan ini: apakah kita tahu banyaknya pengulangan sebelum perulangan dimulai? Untuk tabel kita tahu (sepuluh tahun), jadi `for` yang tepat; untuk target kita tidak tahu, jadi `while` yang tepat.

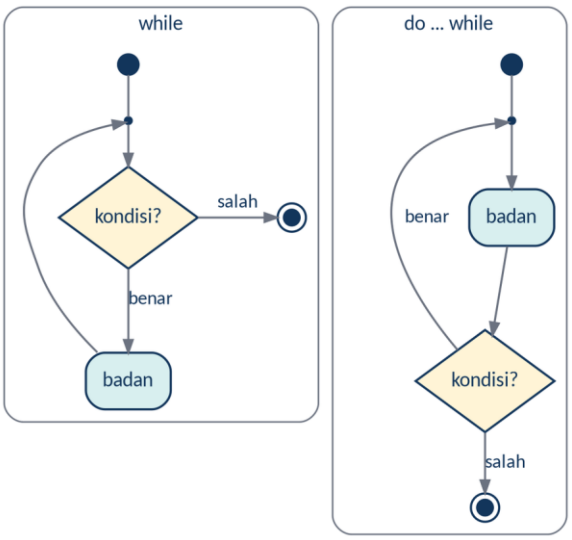
3.4 Pernyataan `do`...`while`

3.4.1 Paling sedikit sekali

Perulangan `while` menguji kondisinya *sebelum* badan, sehingga badannya bisa dijalankan nol kali. Namun beberapa tugas harus mengerjakan sesuatu paling sedikit sekali sebelum ada yang dapat diuji. Menanyakan berat adalah tugas semacam itu: kita tidak dapat memeriksa berat sebelum membacanya. Pernyataan `do`...`while` menguji kondisinya *sesudah* badan:

```
do {
    statements
} while (condition);
```

Badan dijalankan; lalu kondisi diuji; jika benar, badan dijalankan lagi. **Gambar 3.2** membandingkan kedua perulangan. Pada `while` keputusan datang lebih dahulu dan badannya dapat dilewati seluruhnya; pada `do`...`while` badan datang lebih dahulu dan selalu dijalankan paling sedikit sekali. Perhatikan titik koma sesudah kondisi: ia wajib, karena `while (condition);` mengakhiri pernyataan `do`.



badan do ... while selalu dijalankan minimal sekali

Gambar 3.2 `while` menguji sebelum badan, sehingga badan mungkin tak pernah dijalankan; `do`...`while` menguji sesudah badan, sehingga badan selalu dijalankan paling sedikit sekali.

3.4.2 Tanya sampai jawabannya sah

Kode 3.5 menanyakan berat sampai nilainya berada di antara 1 dan 50 kg. Kondisi `weight < 1 || weight > 50` berarti "berat kurang dari 1 **atau** lebih dari 50"; operator `||` dijelaskan di Subbab 3.7.

Kode 3.5 `validate.c`: `do`...`while` terus bertanya sampai beratnya berada dalam rentang.

```c
/* validate.c - do...while: tanya lagi sampai beratnya masuk akal */
#include <stdio.h>

int main(void)
{
    int weight;

    do {
        printf("Weight in kg (1-50): ");
        if (scanf("%d", &weight) != 1) return 1;
        if (weight < 1 || weight > 50) {
            printf("Please type a weight from 1 to 50.\n");
        }
    } while (weight < 1 || weight > 50);     /* perhatikan titik komanya */

    printf("Accepted: %d kg\n", weight);
    return 0;
}
```

```
Weight in kg (1-50): 0
Please type a weight from 1 to 50.
Weight in kg (1-50): 75
Please type a weight from 1 to 50.
Weight in kg (1-50): 12
Accepted: 12 kg
```

Jawaban pertama, 0, terlalu kecil; yang kedua, 75, terlalu besar; setiap kali pesan dicetak dan pertanyaan diulang. Jawaban ketiga, 12, membuat kondisi bernilai salah, perulangan berakhir, dan beratnya diterima. Badan dijalankan tiga kali, dan akan dijalankan sekali, tidak pernah nol kali, sekalipun jawaban pertama sudah benar. Perhatikan pula bahwa pesan kesalahannya sopan dan jelas: ia tidak sekadar berkata "salah", tetapi memberi tahu rentang yang diharapkan. Pengguna yang tahu apa yang diminta akan memperbaiki ketikannya pada percobaan berikutnya, sedangkan pengguna yang hanya mendengar "salah" akan menebak-nebak.

Python tidak memiliki `do...while`. Ungkapan khas Python adalah perulangan yang berjalan "selamanya", `while True:`, dengan `break` yang meninggalkannya begitu jawabannya sah, seperti pada **Kode 3.6**. Ujinya ditulis di tempat yang semestinya, sesudah masukan, dan `break` adalah satu-satunya jalan keluar.

Kode 3.6 `validate.py`: bentuk Python untuk "tanya sampai sah" adalah `while True` dengan `break`.

```python
# validate.py - Python tidak punya do...while: ulang terus, break bila sah
while True:
    weight = int(input("Weight in kg (1-50): "))
    if weight < 1 or weight > 50:
        print("Please type a weight from 1 to 50.")
    else:
        break                           # satu-satunya jalan keluar dari loop

print(f"Accepted: {weight} kg")
```

Python juga dapat menulis perbandingan ganda sebagai `1 <= weight <= 50`, yang terbaca seperti matematika; C tidak dapat, karena di C `1 <= weight <= 50` berarti `(1 <= weight) <= 50`, membandingkan 0 atau 1 dengan 50, dan selalu benar. Program berikutnya menunjukkan bentuk Python itu.

3.5 Pernyataan `switch`

3.5.1 Satu keputusan, banyak kasus

Profitina Laundry menulis kode satu huruf pada label setiap kantong: W untuk cuci, kering, dan lipat, E untuk ekspres, I untuk setrika saja. Di akhir hari Mbak Rina mengetik kode semua label, dan program menghitung setiap jenisnya. Rantai `if...else if` akan berhasil, tetapi ketika satu bilangan bulat atau karakter dibandingkan dengan banyak nilai konstan, C menawarkan pernyataan yang lebih jelas, `switch` [5]:

```
switch (expression) {
case constant1:
    statements
    break;
case constant2:
    statements
    break;
default:
    statements
    break;
}
```

Ekspresinya dievaluasi sekali; kendali melompat ke `case` yang konstantanya sama dengan nilai itu, atau ke `default` bila tidak ada yang sama. Pernyataan-pernyataan sesudah label itu dijalankan sampai bertemu `break`, yang meninggalkan seluruh `switch`. **Kode 3.7** menghitung kode layanan.

Kode 3.7 `service_count.c`: `switch` dengan kasus bersama, `break`, dan `default`.

```c
/* service_count.c - menghitung kode layanan dengan switch; Q mengakhiri */
#include <stdio.h>

int main(void)
{
    char code;
    int wash = 0, express = 0, ironing = 0, unknown = 0;

    printf("Service codes (W, E, I; Q to finish):\n");
    while (scanf(" %c", &code) == 1 && code != 'Q' && code != 'q') {
        switch (code) {
        case 'W':
        case 'w':                   /* kedua case berbagi satu aksi */
            ++wash;
            break;
        case 'E':
        case 'e':
            ++express;
            break;
        case 'I':
        case 'i':
            ++ironing;
            break;
        default:                    /* selain itu */
            ++unknown;
            break;
        }
    }
    printf("Wash, dry & fold: %d\n", wash);
    printf("Express         : %d\n", express);
    printf("Ironing only    : %d\n", ironing);
    printf("Unknown codes   : %d\n", unknown);
    return 0;
}
```

```
Service codes (W, E, I; Q to finish):
W w E
I x W
Q
Wash, dry & fold: 3
Express         : 1
Ironing only    : 1
Unknown codes   : 1
```

Kode boleh diketik beberapa sekaligus dalam satu baris, dengan huruf besar atau kecil. Format `" %c"` dengan spasi sebelum `%c` melompati spasi dan pergantian baris lalu membaca satu karakter, sehingga program melihat W, w, E, I, x, W, dan akhirnya Q, yang mengakhiri perulangan. Kondisi

perulangan menggabungkan tiga uji dengan `&&` ("dan"): `scanf` harus berhasil membaca satu karakter, dan karakter itu bukan `Q` maupun `q`. Dari kode yang terbaca, tiga adalah cuci (W, w, W), satu ekspres, satu setrika, dan `x` tidak dikenal.

3.5.2 Jatuh terus ke bawah

Label `case 'W':` dan `case 'w':` berdiri tepat bertumpuk, tanpa pernyataan di antaranya. Hal ini berhasil karena sifat `switch` yang disebut ***fall-through*** (jatuh terus): begitu kendali melompat ke sebuah label, ia terus saja bergerak ke bawah melewati label-label berikutnya sampai bertemu `break`. Huruf `w` kecil melompat ke labelnya lalu jatuh ke pernyataan-pernyataan milik `W`. Di sini itulah yang kita inginkan; tetapi `break` yang *terlupa* menimbulkan akibat yang sama, dan satu kasus diam-diam menjalankan pernyataan kasus berikutnya. **Gambar 3.1** dan **Gambar 3.2** menampilkan perulangan sebagai gambar; untuk `switch`, ingatlah satu aturan saja: setiap kelompok pernyataan diakhiri `break`, termasuk kelompok terakhir, supaya kasus yang ditambahkan kelak tidak jatuh ke dalamnya. GCC memperingatkan *fall-through* yang tidak disengaja dengan `-Wextra`, dan C23 menambahkan atribut `[[fallthrough]]` untuk menandai yang disengaja.

`switch` hanya dapat menguji nilai bilangan bulat: `int`, `char` (bilangan bulat kecil), dan enumerasi di Bab 10. Ia tidak dapat menguji rentang seperti "dari 1 sampai 50", bilangan pecahan, atau string; untuk itu, pakailah `if`...`else if`.

3.5.3 `switch` atau rantai `else if`?

Penghitungan yang sama dapat ditulis tanpa `switch`:

```
if (code == 'W' || code == 'w')
    ++wash;
else if (code == 'E' || code == 'e')
    ++express;
else if (code == 'I' || code == 'i')
    ++ironing;
else
    ++unknown;
```

Kedua versi benar dan mengerjakan hal yang sama. Untuk keputusan semacam ini `switch` punya dua keunggulan. Ia menyebut variabel yang diuji hanya sekali, di bagian atas, sehingga pembaca langsung tahu bahwa seluruh pernyataan itu berkisar pada `code`; pada rantai, setiap kondisi pada prinsipnya dapat menguji hal yang berbeda, dan pembaca harus memeriksa satu per satu. Selain itu ia mendaftar kasus-kasusnya sebagai satu kolom konstanta, yang terbaca seperti sebuah menu. Rantai punya keunggulannya sendiri: ia dapat menguji rentang dan gabungan, seperti `bill >= 100000`, yang tidak sanggup dilakukan `switch`. Aturan yang baik adalah memakai `switch` bila pertanyaannya "nilai tetap yang mana di antara ini?" dan memakai rantai bila pertanyaannya "kondisi mana di antara ini yang berlaku?".

3.5.4 `match` di Python

Sejak versi 3.10, Python memiliki pernyataan yang dapat memainkan peran `switch`: `match`, dengan klausa-klausa `case` yang dicoba dari atas ke bawah [6]. **Kode 3.8** adalah program Python-nya.

Kode 3.8 `service_count.py`: `match`, dengan `case "W" | "w"` mendaftar dua alternatif dan `case _` menangkap sisanya.

```python
# service_count.py - menghitung kode layanan dengan match; Q mengakhiri
import sys

wash = express = ironing = unknown = 0
finished = False

print("Service codes (W, E, I; Q to finish):")
for line in sys.stdin:                    # satu baris ketikan setiap kali
    for code in line:
        if code.isspace():
            continue                      # lewati spasi, seperti " %c" di C
        if code == "Q" or code == "q":
            finished = True
            break
        match code:
            case "W" | "w":               # kedua case berbagi satu aksi
                wash += 1
            case "E" | "e":
                express += 1
            case "I" | "i":
                ironing += 1
            case _:                       # selain itu
                unknown += 1
    if finished:
        break

print(f"Wash, dry & fold: {wash}")
print(f"Express         : {express}")
print(f"Ironing only    : {ironing}")
print(f"Unknown codes   : {unknown}")
```

Sebuah `case` boleh mendaftar alternatif yang digabung dengan `|`, pengganti label bertumpuk di C; `case _` memainkan peran `default`. Tidak ada *fall-through* dan tidak ada `break`: sesudah satu `case` dijalankan, `match` selesai. (Karena itu `break` di dalam `match`, seperti pada perulangan berikut, meninggalkan *perulangan* yang melingkupinya, bukan `match`.) `match` sanggup berbuat jauh lebih banyak daripada membandingkan konstanta, misalnya membongkar sebuah daftar, tetapi untuk buku ini ia hanyalah `switch`-nya Python. Cara membaca kodenya pun berbeda: program menelusuri masukan baris demi baris dengan `for line in sys.stdin`, lalu setiap baris karakter demi karakter dengan `for code in line`, melewati spasi dengan `isspace` seperti format C `" %c"`. Saat bertemu Q, ia mengisi variabel `finished` lalu meninggalkan kedua perulangan, karena `break` hanya meninggalkan perulangan yang paling dalam.

3.6 `break` dan `continue`

Pernyataan `break`, yang mengakhiri `switch`, juga mengakhiri perulangan: bila dijalankan di dalam `while`, `for`, atau `do`...`while`, perulangan langsung berhenti dan kendali berlanjut sesudahnya. Pernyataan `continue`, yang sudah dipakai kasir v0.2, hanya melewati *sisa iterasi saat ini*: pada `while` atau `do`...`while` kendali melompat ke uji kondisi, dan pada `for` ia melompat ke *increment* lebih dahulu, sehingga pencacahnya tetap maju. **Kode 3.9** menunjukkan perbedaannya pada bilangan 1 sampai 10.

Kode 3.9 `break_continue.c`: `break` mengakhiri perulangan di 5; `continue` hanya melewati 5.

```c
/* break_continue.c - keluar dari loop lebih awal, atau lewati satu putaran */
#include <stdio.h>

int main(void)
{
    for (int x = 1; x <= 10; ++x) {
        if (x == 5)
            break;              /* segera keluar dari loop */
        printf("%d ", x);
    }
    printf("\n");

    for (int x = 1; x <= 10; ++x) {
        if (x == 5)
            continue;           /* lewati sisa putaran ini saja */
        printf("%d ", x);
    }
    printf("\n");
    return 0;
}
```

```
1 2 3 4
1 2 3 4 6 7 8 9 10
```

Perulangan pertama mencetak 1 sampai 4 lalu berhenti ketika `x` mencapai 5. Yang kedua mencetak setiap bilangan kecuali 5. Versi Python-nya, `break_continue.py` di folder kode, memiliki dua perulangan yang sama dengan `range(1, 11)` dan mencetak dua baris yang sama.

Kedua pernyataan itu berguna tetapi harus dipakai dengan hati-hati. Perulangan dengan beberapa `break` memiliki beberapa jalan keluar, dan sifat yang membuat program terstruktur mudah dibaca, satu jalan masuk dan satu jalan keluar, hilang. Pakailah `break` untuk kasus-kasus yang jelas, seperti meninggalkan perulangan `while True` ketika masukan sah sudah tiba atau meninggalkan pencarian begitu barangnya ditemukan, dan utamakan kondisi yang menyatakan kapan perulangan berakhir bila cara itu sama mudahnya.

Sebuah `break` hanya meninggalkan perulangan atau `switch` *terdalam* yang melingkupinya. Bila sebuah pencarian berjalan di dalam dua perulangan bersarang, misalnya atas baris dan kolom daftar harga, dan harus berhenti sepenuhnya begitu menemukan harga di atas Rp80.000, `break` di perulangan dalam hanya mengakhiri baris yang sedang ditelusuri; perulangan luar tetap berlanjut ke baris berikutnya. Jalan keluar yang terstruktur adalah sebuah bendera (*flag*), yaitu variabel seperti `found` yang mulai dari 0, diisi 1 di tempat barangnya ditemukan, dan diuji di dalam kondisi perulangan luar: `for (kg = 1; kg <= 10 && !found; ++kg)`. Program Python `service_count.py` memakai teknik yang persis sama dengan variabelnya, `finished`. Dengan bendera, perulangan tetap memiliki satu jalan keluar yang terbaca di kepalanya, dan pembaca tidak perlu mencari `break` yang tersembunyi di tengah badan.

3.7 Operator Logika

3.7.1 `&&`, `||`, dan `!`

Kondisi sejauh ini membandingkan satu hal. Aturan sungguhan sering terdiri atas beberapa bagian: berat sah bila paling sedikit 1 *dan* paling banyak 50; pelanggan mendapat diskon bila tagihannya besar *atau* ia anggota; perulangan berlanjut bila kodenya *bukan* Q. C membangun kondisi semacam itu dengan tiga **operator logika** [1]:

- `&&` (DAN logika) bernilai benar bila kedua operandnya benar;

- `||` (ATAU logika) bernilai benar bila paling sedikit satu operandnya benar;
- `!` (TIDAK logika) mengubah benar menjadi salah dan salah menjadi benar.

Python mengejanya sebagai kata: `and`, `or`, dan `not`. **Tabel 3.2** adalah **tabel kebenaran** ketiga operator itu, dengan 0 untuk salah dan 1 untuk benar, seperti yang ditulis C.

Tabel 3.2 Tabel kebenaran operator logika AND (`&&`), OR, dan NOT (`!`); di C sebuah kondisi bernilai salah bila 0 dan benar bila selain itu.

a	b	a AND b	a OR b	NOT a
0	0	0	0	1
0	1	0	1	1
1	0	0	1	0
1	1	1	1	0

Operator-operator ini memiliki presedensinya sendiri: `!` mengikat paling kuat, lalu operator pembanding, lalu `&&`, dan `||` paling akhir. Jadi `weight >= 1 && weight <= 50` tidak memerlukan tanda kurung, dan `a || b && c` berarti `a || (b && c)`. Bila `&&` dan `||` bercampur, tanda kurung menjelaskan maksudnya kepada pembaca dan sebaiknya selalu ditulis.

3.7.2 Evaluasi hubung-singkat

C mengevaluasi `&&` dan `||` dari kiri ke kanan dan berhenti begitu jawabannya diketahui. Jika operand kiri `&&` bernilai salah, seluruh ekspresi bernilai salah apa pun operand kanannya, sehingga operand kanan **sama sekali tidak dievaluasi**; demikian pula operand kanan `||` dilewati bila operand kirinya benar. Ini disebut **evaluasi hubung-singkat** (*short-circuit evaluation*), dan ia bukan sekadar trik kecepatan melainkan alat untuk menulis kondisi yang aman. **Kode 3.10** mencetak tabel kebenaran dengan dua perulangan `for` bersarang lalu memakai hubung-singkat untuk melindungi sebuah pembagian.

Kode 3.10 `logic.c`: tabel kebenaran `&&`, `||`, dan `!`, serta pembagian yang dilindungi evaluasi hubung-singkat.

```c
/* logic.c - &&, || dan !, serta evaluasi hubung-singkat */
#include <stdio.h>

int main(void)
{
    printf("a b | a&&b a||b  !a\n");
    for (int a = 0; a <= 1; ++a) {
        for (int b = 0; b <= 1; ++b) {
            printf("%d %d |  %d    %d    %d\n", a, b, a && b, a || b, !a);
        }
    }

    int orders = 0, kilos = 0;
    /* bila orders 0, && berhenti sebelum pembagian dicoba */
    if (orders != 0 && kilos / orders > 5) {
        printf("Big orders on average.\n");
    } else {
        printf("No verdict: no orders yet.\n");
    }
    return 0;
}
```

```
a b | a&&b a||b  !a
0 0 |  0    0    1
0 1 |  0    1    1
1 0 |  0    1    0
1 1 |  1    1    0
No verdict: no orders yet.
```

Pada kondisi `orders != 0 && kilos / orders > 5`, pembagian dengan `orders` hanya dievaluasi bila `orders != 0` bernilai benar. Tanpa pesanan, operand kiri bernilai salah, pembagian tidak pernah terjadi, dan program tidak mungkin mogok. Bila ditulis dengan urutan sebaliknya, `kilos / orders > 5 && orders != 0`, kondisi yang sama membagi dengan nol lebih dahulu. Urutkan bagian-bagian sebuah kondisi sehingga masing-masing melindungi bagian berikutnya.

Kode 3.11 adalah versi Python-nya. `and` dan `or` di Python juga berhubung-singkat dengan cara yang sama. Satu rincian berbeda: keduanya tidak selalu mengembalikan `True` atau `False` tetapi mengembalikan salah satu operandnya (`0 and 1` bernilai 0, `1 or 0` bernilai 1), dan karena itu program membungkusnya dengan `int(...)` agar mencetak 0 atau 1 persis seperti C.

Kode 3.11 `logic.py`: `and`, `or`, dan `not`, dengan perlindungan hubung-singkat yang sama.

```python
# logic.py - and, or dan not, serta evaluasi hubung-singkat
print("a b | a&&b a||b  !a")
for a in range(2):
    for b in range(2):
        print(f"{a} {b} |  {int(a and b)}    {int(a or b)}    {int(not a)}")

orders, kilos = 0, 0
# bila orders 0, "and" berhenti sebelum pembagian dicoba
if orders != 0 and kilos // orders > 5:
    print("Big orders on average.")
else:
    print("No verdict: no orders yet.")
```

3.7.3 Hukum De Morgan

Perulangan validasi di Subbab 3.4 berulang *selama berat tidak sah*: `weight < 1 || weight > 50`. Berat *sah* bila `weight >= 1 && weight <= 50`. Kedua kondisi itu saling meniadakan, dan hubungan di antara keduanya adalah salah satu **hukum De Morgan**:

$$\neg(A \wedge B) \equiv \neg A \vee \neg B \qquad \neg(A \vee B) \equiv \neg A \wedge \neg B \tag{3.2}$$

Dengan kata-kata: “tidak (A dan B)” sama dengan “tidak A, atau tidak B”, dan “tidak (A atau B)” sama dengan “tidak A, dan tidak B”. Dengan A mewakili `weight >= 1` dan B mewakili `weight <= 50`, “tidak sah” adalah `!(weight >= 1 && weight <= 50)`, yang menurut Persamaan (3.2) sama dengan `!(weight >= 1) || !(weight <= 50)`, yaitu `weight < 1 || weight > 50`. Bila Anda harus meniadakan sebuah kondisi, terapkan Persamaan (3.2) di atas kertas alih-alih menebak; mengubah `&&` menjadi `||` adalah sumber sebagian besar kesalahan peniadaan.

3.7.4 Dari aturan usaha menjadi kondisi

Sebagian besar kondisi dalam program sungguhan bermula sebagai kalimat yang diucapkan orang yang menjalankan usahanya. Menerjemahkan kalimat semacam itu dengan cermat adalah keterampilan tersendiri, dan tiga aturan Profitina Laundry memperlihatkan caranya.

“Pesanan ekspres hanya diterima sampai pukul 10 pagi, kecuali untuk anggota.” Pesanan diterima bila layanannya bukan ekspres, atau jamnya sebelum 10, atau pelanggannya anggota: `service != 2 || hour < 10 || member == 1`. Bacalah kembali dengan kata-kata untuk memeriksanya:

pesanan yang bukan ekspres selalu diterima; pesanan ekspres diterima di pagi hari; dan pesanan ekspres seorang anggota diterima kapan saja. Ketiganya sesuai dengan kalimat aslinya.

"Kantong mendapat pewangi gratis bila beratnya paling sedikit 5 kg dan bukan setrika saja." Itu adalah `weight >= 5 && service != 3`. Di sini kedua bagian harus berlaku, jadi operatornya `&&`.

"Tutup kasir ketika kasir mengetik 0, atau ketika sudah lewat pukul 21.00." Perulangan harus *berlanjut* selama keduanya tidak berlaku: `!(service == 0 || hour >= 21)`, yang menurut Persamaan (3.2) sama dengan `service != 0 && hour < 21`. Kondisi perulangan sering lebih mudah ditulis sebagai peniadaan aturan berhentinya, dan hukum De Morgan mengubah peniadaan itu menjadi kondisi yang polos.

Dua kebiasaan membuat terjemahan semacam ini dapat diandalkan. Tulislah kondisinya, lalu bacakan kembali dengan kata-kata sehari-hari dan bandingkan dengan kalimat aslinya. Lalu ujilah pada tabel kecil berisi kasus-kasus, satu baris untuk setiap kombinasi bagian-bagiannya, persis seperti **Tabel 3.2**; dengan dua atau tiga bagian, tabelnya berisi empat atau delapan baris, pekerjaan beberapa menit yang menangkap sebagian besar kesalahan. Kebiasaan ini sama bergunanya bagi pemrogram berpengalaman maupun pemula, karena kesalahan logika jarang ditangkap oleh kompilator mana pun.

3.8 Memilih Struktur yang Tepat

Kini Anda mengenal semua struktur kontrol C. Mana yang dipakai biasanya diputuskan oleh satu pertanyaan untuk masing-masing, seperti yang diringkas **Tabel 3.3**.

Tabel 3.3 Memilih struktur kontrol: pertanyaan yang diajukan dan pernyataan yang menjawabnya.

Situasi	Struktur di C	Di Python
banyak pengulangan diketahui, atau pencacah yang bergerak teratur	`for`	`for … in range(…)`
ulangi selama kondisi berlaku, mungkin nol kali	`while`	`while`
kerjakan sesuatu paling sedikit sekali, lalu putuskan apakah diulang	`do…while`	`while True` … `break`
salah satu dari dua jalur	`if…else`	`if` … `else`
salah satu dari banyak jalur yang ditentukan rentang atau kondisi rumit	rantai `if…else if`	`if` … `elif`
salah satu dari banyak jalur yang ditentukan satu nilai bulat atau karakter	`switch`	`match`
pilihan kecil di antara dua nilai di dalam ekspresi	`?:`	`x if c else y`

Ketiga perulangan dapat mengerjakan setiap tugas: `while` dapat menghitung, dan `for` dengan *increment* kosong dapat menunggu sentinel. Tujuan memilih dengan baik adalah pembacanya. `for` berkata "ini menghitung"; `do…while` berkata "ini terjadi paling sedikit sekali"; `switch` berkata "ini satu keputusan di antara kasus-kasus tetap". Memilih struktur yang menyatakan maksud Anda adalah dokumentasi termurah yang ada.

Empat kasus kecil memperlihatkan bagaimana pilihan itu dibuat dalam praktik. *Cetak nomor struk 1 sampai 40 dalam sehari*: banyaknya pengulangan diketahui, jadi `for`. *Baca kantong dari timbangan sampai kasir mengetik 0*: banyaknya tidak diketahui dan mungkin tidak ada kantong sama sekali, jadi `while` dengan sentinel. *Tanyakan nomor telepon pelanggan sampai panjangnya 10 hingga 13 angka*: paling sedikit satu pertanyaan diperlukan, jadi `do…while`. *Pilih layanan dari nomor menu*: satu bilangan bulat, beberapa nilai tetap, jadi `switch`. Dalam setiap kasus

pertanyaan dari **Tabel 3.3** yang memutuskan, dan pembaca yang mengenal kesepakatan ini memahami tujuan setiap perulangan sebelum membaca badannya. Kebiasaan kecil seperti ini membuat program orang lain, dan program Anda sendiri beberapa bulan kemudian, jauh lebih mudah dipahami.

3.9 Kasir Profitina Laundry v0.3

Kasir v0.3 memakai setiap perkakas bab ini. Perulangan utamanya berupa `do`...`while`, karena kasir ditanya soal layanan paling sedikit sekali. Sebuah `switch` memutuskan apa yang dilakukan terhadap layanan itu: 0 menutup kasir, 1 sampai 3 adalah layanan, dan selain itu tidak dikenal. Di dalam kasus layanan, `do`...`while` kedua menanyakan berat sampai nilainya berada di antara 1 dan 50 kg, yang mencegah kekeliruan 60-alih-alih-6 milik Mbak Rina. Tiga pencacah mencatat berapa pelanggan yang memilih setiap layanan, variabel `largest` mengingat tagihan terbesar, dan laporan dicetak dengan lebar kolom. **Kode 3.12** adalah programnya.

Kode 3.12 `cashier_v03.c`: kasir v0.3 dengan `switch`, berat yang divalidasi, dan laporan harian berformat.

```c
/* cashier_v03.c - kasir Profitina Laundry v0.3: switch, validasi, laporan */
#include <stdio.h>

int main(void)
{
    int service, weightKg, price, bill;
    int nWash = 0, nExpress = 0, nIroning = 0;
    int revenue = 0, largest = 0;

    printf("=== Profitina Laundry - cashier v0.3 ===\n");
    do {
        printf("Service (1 wash, 2 express, 3 ironing, 0 close): ");
        if (scanf("%d", &service) != 1) return 1;

        switch (service) {
        case 0:
            break;                          /* hanya keluar dari switch */
        case 1: case 2: case 3:
            do {                            /* tanya sampai beratnya masuk akal */
                printf("Weight in kg (1-50): ");
                if (scanf("%d", &weightKg) != 1) return 1;
            } while (weightKg < 1 || weightKg > 50);

            if (service == 1) {
                price = 8000;
                ++nWash;
            } else if (service == 2) {
                price = 12000;
                ++nExpress;
            } else {
                price = 6000;
                ++nIroning;
            }
            bill = weightKg * price;
            revenue += bill;
            if (bill > largest)
                largest = bill;
            printf("Bill: Rp %d\n", bill);
            break;
        default:
            printf("Unknown service %d.\n", service);
            break;
        }
    } while (service != 0);
```

Kode 3.12 (Lanjutan) *cashier_v03.c: kasir v0.3 dengan switch, berat yang divalidasi, dan laporan harian berformat.*

```c
    int customers = nWash + nExpress + nIroning;
    printf("\n--- DAILY REPORT ---\n");
    printf("%-18s%6d\n", "Wash, dry & fold", nWash);
    printf("%-18s%6d\n", "Express", nExpress);
    printf("%-18s%6d\n", "Ironing only", nIroning);
    printf("%-18s%6d\n", "Customers", customers);
    printf("%-18s%12d\n", "Revenue (Rp)", revenue);
    printf("%-18s%12d\n", "Largest bill (Rp)", largest);
    if (customers > 0)
        printf("%-18s%12.2f\n", "Average (Rp)", (double) revenue / customers);
    return 0;
}
```

Tiga rincian layak diperhatikan. Pertama, `case 0: break;` tampak aneh: `break` hanya meninggalkan `switch`, bukan `do...while`, dan perulangan lalu berakhir karena kondisinya, `service != 0`, bernilai salah. Kedua, `case 1: case 2: case 3:` memakai *fall-through* untuk mengirim tiga

layanan ke pernyataan-pernyataan yang sama, tempat sebuah rantai `if`...`else if` kemudian memilih harga dan menaikkan pencacah yang tepat. Ketiga, menemukan tagihan terbesar adalah algoritma kecil tersendiri: `largest` mulai dari 0, lebih kecil daripada tagihan sungguhan mana pun, dan setiap tagihan baru yang lebih besar menggantikannya, sehingga sesudah perulangan ia menyimpan nilai maksimum. Berikut satu hari dengan berat yang salah ketik dan layanan yang tidak dikenal:

```
=== Profitina Laundry - cashier v0.3 ===
Service (1 wash, 2 express, 3 ironing, 0 close): 1
Weight in kg (1-50): 5
Bill: Rp 40000
Service (1 wash, 2 express, 3 ironing, 0 close): 2
Weight in kg (1-50): 0
Weight in kg (1-50): 60
Weight in kg (1-50): 3
Bill: Rp 36000
Service (1 wash, 2 express, 3 ironing, 0 close): 4
Unknown service 4.
Service (1 wash, 2 express, 3 ironing, 0 close): 3
Weight in kg (1-50): 2
Bill: Rp 12000
Service (1 wash, 2 express, 3 ironing, 0 close): 1
Weight in kg (1-50): 7
Bill: Rp 56000
Service (1 wash, 2 express, 3 ironing, 0 close): 0

--- DAILY REPORT ---
Wash, dry & fold       2
Express                1
Ironing only           1
Customers              4
Revenue (Rp)          144000
Largest bill (Rp)      56000
Average (Rp)         36000.00
```

Berat pelanggan ekspres terketik 0 lalu 60, dan setiap kali program cukup bertanya lagi sampai 3 diketik; layanan 4 yang tidak dikenal dilaporkan dan tidak dihitung. Periksa laporannya dengan tangan: dua pesanan cuci (5 kg dan 7 kg, Rp40.000 dan Rp56.000), satu pesanan ekspres 3 kg (Rp36.000), dan satu pesanan setrika 2 kg (Rp12.000) menghasilkan Rp144.000 dari empat pelanggan, rata-rata Rp36.000,00, dan tagihan terbesar Rp56.000.

Kode 3.13 adalah versi Python-nya. Perulangan utamanya berupa `while True` dengan `break` di ujungnya bila layanannya 0, `switch` menjadi `match` dengan `case 1 | 2 | 3`, dan perulangan beratnya memakai ungkapan `while True`...`break` dari Subbab 3.4 dengan perbandingan Python `1 <= weight_kg <= 50`.

Kode 3.13 `cashier_v03.py`: kasir v0.3 dalam Python, dengan `match` dan ungkapan `while True`.

```
# cashier_v03.py - kasir Profitina Laundry v0.3: match, validasi, laporan
n_wash = n_express = n_ironing = 0
revenue = largest = 0

print("=== Profitina Laundry - cashier v0.3 ===")
while True:
    service = int(input("Service (1 wash, 2 express, 3 ironing, 0 close): "))
    match service:
        case 0:
            pass
        case 1 | 2 | 3:
            while True:                       # tanya sampai beratnya masuk akal
                weight_kg = int(input("Weight in kg (1-50): "))
                if 1 <= weight_kg <= 50:
                    break
            if service == 1:
                price = 8000
                n_wash += 1
            elif service == 2:
                price = 12000
                n_express += 1
            else:
                price = 6000
                n_ironing += 1
            bill = weight_kg * price
            revenue += bill
            if bill > largest:
                largest = bill
            print(f"Bill: Rp {bill}")
        case _:
            print(f"Unknown service {service}.")
    if service == 0:
        break

customers = n_wash + n_express + n_ironing
print()
print("--- DAILY REPORT ---")
print(f"{'Wash, dry & fold':<18}{n_wash:6d}")
print(f"{'Express':<18}{n_express:6d}")
print(f"{'Ironing only':<18}{n_ironing:6d}")
print(f"{'Customers':<18}{customers:6d}")
print(f"{'Revenue (Rp)':<18}{revenue:12d}")
print(f"{'Largest bill (Rp)':<18}{largest:12d}")
if customers > 0:
    print(f"{'Average (Rp)':<18}{revenue / customers:12.2f}")
```

Spesifikasi format laporannya sepadan dengan milik C: `{'Revenue (Rp)':<18}` adalah `%-18s`, teks rata kiri dalam 18 karakter, dan `{revenue:12d}` adalah `%12d`. Kedua program mencetak laporan yang sama, spasi demi spasi.

3.10 Membuatnya Lebih Baik: Daftar Harga untuk Dinding

Pelanggan terus bertanya kepada Mbak Rina berapa biaya kantong mereka sebelum ditimbang. Daftar harga tercetak di dinding, yang menunjukkan harga 1 sampai 10 kilogram untuk setiap layanan, akan menjawab pertanyaan itu sebelum diajukan. Daftar itu adalah tabel dengan sepuluh baris dan tiga kolom harga, dan sebuah tabel adalah perulangan di dalam perulangan: perulangan luar berjalan atas baris-baris (kilogram), dan untuk setiap baris perulangan dalam berjalan atas kolom-kolom (layanan). **Kode 3.14** mencetaknya.

Kode 3.14 `price_table.c`: perulangan `for` bersarang mencetak daftar harga dengan kolom-kolom yang rata.

```c
/* price_table.c - tabel harga 1..10 kg dengan for bersarang */
#include <stdio.h>

int main(void)
{
    printf("%3s%10s%10s%10s\n", "kg", "Wash", "Express", "Ironing");
    for (int kg = 1; kg <= 10; ++kg) {
        printf("%3d", kg);
        for (int service = 1; service <= 3; ++service) {
            int price = service == 1 ? 8000 : service == 2 ? 12000 : 6000;
            printf("%10d", kg * price);
        }
        printf("\n");
    }
    return 0;
}
```

```
 kg      Wash   Express   Ironing
  1      8000     12000      6000
  2     16000     24000     12000
  3     24000     36000     18000
  4     32000     48000     24000
  5     40000     60000     30000
  6     48000     72000     36000
  7     56000     84000     42000
  8     64000     96000     48000
  9     72000    108000     54000
 10     80000    120000     60000
```

Untuk setiap nilai `kg`, perulangan dalam berjalan tiga kali, sekali per layanan, dan mencetak satu harga dalam ruang 10 karakter; sesudah perulangan dalam, `printf("\n")` mengakhiri barisnya. Seluruhnya badan dalam dijalankan 10 × 3 = 30 kali. Harga layanan dipilih dengan operator kondisional *bersarang*, `service == 1 ? 8000 : service == 2 ? 12000 : 6000`, yang dibaca dari kiri ke kanan seperti rantai `else if`; untuk tiga nilai hal ini masih terbaca, untuk lebih banyak nilai `switch` akan lebih jelas. Versi Python-nya, `price_table.py`, memakai `print(..., end="")` agar tetap di baris yang sama dan ekspresi kondisional `8000 if service == 1 else 12000 if service == 2 else 6000`; ia mencetak tabel yang sama.

Daftar seperti ini juga mudah diperbarui: bila harga berubah, cukup tiga angka di dalam program yang diganti, lalu tabel baru dicetak dan ditempel di dinding. Tabel tercetak juga merupakan uji sederhana bagi kasir. Setiap isiannya harus sama dengan tagihan yang dicetak kasir v0.3 untuk berat dan layanan yang sama; jika tabel menyebut Rp36.000 untuk ekspres 3 kg dan kasir menyebut angka lain, salah satunya keliru. Dua program yang menghitung hal yang sama dengan cara berbeda adalah pemeriksa yang ampuh satu terhadap yang lain.

3.11 Memakainya Sungguhan

Perkakas bab ini muncul di hampir setiap program yang berbicara dengan manusia. `do`...`while` yang bertanya sampai jawabannya sah cocok di mana pun orang mengetik bilangan: umur antara 0 dan 120, jumlah barang antara 1 dan stok di rak, bulan antara 1 dan 12. Tentukan batasnya dari usahanya, bukan dari komputernya: 50 kg tepat untuk *laundry* ini dan keliru untuk pedagang besar beras. `switch` atas nomor menu atau huruf kode cocok untuk setiap program kasir kecil, dan laporan dengan lebar kolom berguna bagi toko mana pun yang ingin melihat harinya dalam satu layar.

Dua keterbatasan kasir v0.3 masih tersisa, dan keduanya patut diketahui sebelum ia dipakai di meja kasir sungguhan. Jika kasir mengetik huruf di tempat yang mengharapkan bilangan, `scanf`

gagal dan program berakhir, seperti yang dikatakan `return 1`; Bab 9 membaca baris teks utuh dan menangani masukan semacam itu dengan anggun. Dan ketika program berakhir, semuanya terlupa: laporannya ada di layar tetapi tidak di tempat lain. Bab 11 menulis pesanan setiap hari ke sebuah berkas, sehingga tidak ada yang hilang ketika listrik padam. Pola yang sama mudah disesuaikan untuk usaha kecil lain. Kantin sekolah dapat meminta kasirnya mengetik satu huruf untuk setiap barang yang terjual, N untuk nasi goreng, M untuk mi, T untuk es teh, lalu menghitungnya dengan `switch` persis seperti `service_count` menghitung kode layanan; di akhir jam istirahat kantin tahu apa yang laku dan apa yang perlu dimasak lebih banyak besok. Seorang penjahit dapat memvalidasi ukuran baju yang diketik dengan `do`...`while`, menolak lingkar dada 5 cm atau 500 cm sebelum sampai ke meja potong. Sebuah klinik kecil dapat mencetak tabel jam janji temu dengan perulangan `for` bersarang, satu baris per dokter dan satu kolom per setengah jam. Dalam setiap kasus strukturnya tetap sama; hanya kata-kata, batas-batas, dan harganya yang berubah.

Sampai saat itu, laporan setidaknya dapat disimpan dengan mengalihkan keluaran program ke sebuah berkas, seperti `./cashier_v03 > report.txt`, sebuah trik yang berlaku untuk setiap program di buku ini.

3.12 Kesalahan yang Sering Terjadi, dan Cara Menghindarinya

- **Titik koma sesudah kepala `for`.** `for (int i = 1; i <= 5; ++i);` memiliki badan kosong; pernyataan di bawahnya dijalankan sekali, sesudah perulangan. GCC memperingatkan indentasi yang menyesatkan itu dengan `-Wall`.
- **Batas yang meleset satu.** `for (i = 1; i < 10; ++i)` berjalan 9 kali, bukan 10. Putuskan apakah nilai terakhir termasuk, tulis `<=` atau `<` sesuai keputusan itu, dan periksa iterasi pertama dan terakhir dengan tangan. Di Python ingatlah bahwa `range(1, 10)` berhenti di 9.
- **`break` yang hilang dalam `switch`.** Tanpanya, pernyataan kasus berikutnya ikut dijalankan. Akhiri setiap kasus dengan `break`, dan kompilasi dengan `-Wextra` untuk menangkap *fall-through* yang tidak disengaja.
- **Lupa titik koma sesudah `do`...`while`.** `} while (weight < 1 || weight > 50)` tanpa `;` adalah galat sintaksis.
- **Menulis `1 <= x <= 50` di C.** Ia terkompilasi, membandingkan 0 atau 1 dengan 50, dan selalu benar. Tulislah `x >= 1 && x <= 50`. (`1 <= x <= 50` di Python benar.)
- **`&` atau `|` alih-alih `&&` atau `||`.** Operator satu karakter bekerja pada bit dan tidak berhubung-singkat; kondisinya mungkin tampak benar pada uji sederhana lalu gagal kemudian.
- **Meniadakan kondisi tanpa De Morgan.** Kebalikan dari `x >= 1 && x <= 50` bukanlah `x < 1 && x > 50`, yang tidak pernah benar, melainkan `x < 1 || x > 50`. Terapkan Persamaan (3.2).
- **Pencacah perulangan berupa pecahan.** Hitunglah dengan bilangan bulat; hitung pecahan dari pencacah di dalam badan. Bilangan pecahan di komputer hampir selalu sedikit meleset dari nilai desimalnya, sehingga uji `!=` pada pecahan hampir tidak pernah berhenti tepat pada waktunya.
- **Mengira `break` meninggalkan `if`.** `break` hanya milik perulangan dan `switch`; di dalam `if` yang berada di dalam perulangan, ia meninggalkan perulangan itu. Untuk melewati sisa sebuah `if`, pakailah `else`.
- **Mengubah pencacah di dalam badan `for`.** Pernyataan seperti `i = i + 2;` di dalam `for (int i = 0; i < 10; ++i)` membuat perulangan melompati nilai dengan cara yang tidak lagi tampak di kepalanya. Bila pencacah harus bergerak secara lain, nyatakan hal itu di kepala perulangan.

- **Menguji pecahan atau string dengan `switch`.** `switch` hanya menerima nilai bulat; `case 2.5:` atau `case "W":` tidak dapat dikompilasi. Pakailah `if`...`else if` untuk itu.
- **Lupa `-lm`.** Di Linux, program yang memanggil `pow` berhasil dikompilasi tetapi gagal di-*link* dengan pesan `undefined reference to 'pow'` kecuali `-lm` ditambahkan di akhir perintah.

3.13 Ringkasan Bab

- `for (initialization; condition; increment)` menghimpun perulangan berhitung dalam satu kepala; inisialisasi dijalankan sekali, lalu kondisi, badan, dan *increment* berulang (**Gambar 3.1**). Pencacah yang dideklarasikan di kepala hanya ada di dalam perulangan. `for` di Python menelusuri `range(start, stop, step)`, yang tidak menyertakan `stop`.
- Bunga majemuk mengikuti Persamaan (3.1); `pow` dari `math.h` menghitung pangkatnya, di-*link* dengan `-lm` di Linux. Lebar kolom seperti `%4d`, `%21.2f`, dan `%-18s` meratakan tabel (**Tabel 3.1**); Python memakai angka yang sama di dalam f-string.
- `do`...`while` menguji sesudah badan, sehingga badan dijalankan paling sedikit sekali (**Gambar 3.2**); ia adalah perulangan yang wajar untuk "tanya sampai sah". Python menuliskannya sebagai `while True` dengan `break`.
- `switch` melompat ke `case` yang sama dengan sebuah nilai bulat atau karakter, berjalan sampai `break`, dan jatuh terus bila tidak ada `break`; `default` menangkap sisanya. Python 3.10 ke atas memiliki `match`, dengan `|` untuk alternatif dan `case _` untuk sisanya, tanpa *fall-through*.
- `break` meninggalkan perulangan atau `switch` terdalam; `continue` melewati sisa iterasi saat ini.
- `&&`, `||`, dan `!` (Python: `and`, `or`, `not`) menggabungkan kondisi (**Tabel 3.2**); ketiganya berhubung-singkat, sehingga bagian kiri kondisi dapat melindungi bagian kanannya. Hukum De Morgan (Persamaan (3.2)) meniadakan kondisi gabungan dengan benar.
- Struktur yang menyatakan maksud Anda adalah struktur yang tepat (**Tabel 3.3**). Kasir v0.3 menggabungkan semuanya: perulangan utama `do`...`while`, `switch` atas layanan, berat yang divalidasi, pencacah per layanan, tagihan terbesar, dan laporan dengan kolom-kolom yang rata.

3.14 Soal Latihan

1. Tulislah program yang mencetak bilangan ganjil dari 1 sampai 19 dalam satu baris, dipisahkan spasi, pertama dengan `for` di C lalu dengan `range` di Python. Kemudian cetak dari 19 turun ke 1.
2. Tulis ulang perulangan pada **Kode 3.5** dengan `while` alih-alih `do`...`while`. Pernyataan mana yang kini harus muncul dua kali, dan mengapa?
3. Profitina Laundry buka pukul 07.00 sampai 21.00 pada hari kerja (1 = Senin sampai 5 = Jumat), pukul 08.00 sampai 18.00 pada hari Sabtu (6), dan tutup pada hari Minggu (7). Tulislah program yang membaca nomor hari lalu mencetak jam bukanya, dengan `switch` yang memakai *fall-through* untuk kelima hari kerja. Tolak bilangan di luar 1 sampai 7.
4. Dengan hukum De Morgan, tulislah peniadaan setiap kondisi berikut tanpa operator `!`: (a) `service >= 1 && service <= 3`; (b) `code == 'Q' || code == 'q'`; (c) `bill >= 100000 || member == 1`.
5. Kembangkan `savings` agar juga mencetak bunga yang diperoleh setiap tahun (jumlah tahun ini dikurangi jumlah tahun lalu) di kolom ketiga selebar 15. Berapa bunga pada tahun ke-10?
6. Apa yang dicetak perulangan ini, dan berapa kali badannya dijalankan? `for (int i = 0; i < 10; i += 3) printf("%d ", i);` Tuliskan perulangan yang sama dalam Python.

3.15 Penyelesaian Soal Latihan

1. C: `for (int i = 1; i <= 19; i += 2) printf("%d ", i);` diikuti `printf("\n");`, dan untuk urutan terbalik `for (int i = 19; i >= 1; i -= 2)`. Python: `for i in range(1, 20, 2): print(i, end=" ")` dan `for i in range(19, 0, -2)`; nilai berhentinya 20 dan 0 karena nilai berhenti tidak disertakan.

2. Dengan `while`, pertanyaannya harus diajukan sekali *sebelum* perulangan dan sekali lagi di dalamnya, karena `while` menguji sebelum badan dan belum ada berat untuk diuji sebelum satu berat dibaca:

```
printf("Weight in kg (1-50): ");
if (scanf("%d", &weight) != 1) return 1;
while (weight < 1 || weight > 50) {
    printf("Please type a weight from 1 to 50.\n");
    printf("Weight in kg (1-50): ");
    if (scanf("%d", &weight) != 1) return 1;
}
```

Inilah pola membaca-lebih-dahulu dari perulangan sentinel di Bab 2. Ia benar, tetapi prompt dan `scanf` muncul dua kali; `do...while` hanya memerlukannya sekali, dan itulah sebabnya ia lebih cocok.

3. Sesudah membaca `day` dan menolak nilai di luar 1 sampai 7 dengan sebuah `if`, `switch` menumpuk label-label hari kerja:

```
switch (day) {
case 1: case 2: case 3: case 4: case 5:
    printf("Open 07:00-21:00\n");
    break;
case 6:
    printf("Open 08:00-18:00\n");
    break;
default:
    printf("Closed\n");
    break;
}
```

Di Python, `match day:` dengan `case 1 | 2 | 3 | 4 | 5:`, `case 6:`, dan `case _:`.

4. (a) `service < 1 || service > 3`; (b) `code != 'Q' && code != 'q'`, yang persis merupakan kondisi perulangan pada **Kode 3.7**; (c) `bill < 100000 && member != 1`. Dalam setiap kasus `&&` dan `||` bertukar tempat dan setiap perbandingan dibalik, seperti kata Persamaan (3.2).

5. Simpan jumlah sebelumnya di variabel `previous`, yang dimulai dari pokoknya. Di dalam perulangan hitung `amount`, cetak `amount - previous` dengan `%15.2f`, lalu isi `previous = amount`. Bunga pada tahun ke-10 adalah 1.628.894,63 − 1.551.328,22 = 77.566,41: 5% dari jumlah di akhir tahun ke-9.

6. Ia mencetak `0 3 6 9` dan badannya dijalankan empat kali: sesudah 9 pencacah menjadi 12, dan `12 < 10` bernilai salah. Di Python: `for i in range(0, 10, 3): print(i, end=" ")`.

Daftar Pustaka

[1] *Information Technology — Programming Languages — C*, ISO/IEC 9899:2018, International Organization for Standardization, Geneva, Switzerland, 2018.

[2] Python Software Foundation, “More Control Flow Tools: for statements and the range() function,” *The Python Tutorial*, Python 3.14. Accessed: Sep. 25, 2026. [Online]. Available: https://docs.python.org/3.14/tutorial/controlflow.html

[3] D. Goldberg, "What every computer scientist should know about floating-point arithmetic," *ACM Comput. Surv.*, vol. 23, no. 1, pp. 5–48, Mar. 1991.

[4] cppreference.com, "printf, fprintf, sprintf, snprintf (C)." Accessed: Sep. 25, 2026. [Online]. Available: https://en.cppreference.com/w/c/io/fprintf

[5] B. W. Kernighan and D. M. Ritchie, *The C Programming Language*, 2nd ed. Englewood Cliffs, NJ, USA: Prentice Hall, 1988.

[6] B. Bucher and G. van Rossum, "PEP 634 – Structural Pattern Matching: Specification," Python Enhancement Proposals, 2020. [Online]. Available: https://peps.python.org/pep-0634/

Bab 4

Evaluasi dan Pembahasan I: Lima Soal Diselesaikan Langkah demi Langkah

Bab ini tidak menambahkan pernyataan baru ke kotak perkakas Anda. Sebaliknya, ia mengambil semua yang telah diberikan Bab 1 sampai 3 (variabel dan aritmetika, masukan dan keluaran, `if` dan `switch`, ketiga perulangan, dan operator logika) lalu memakainya untuk lima soal utuh yang makin lama makin besar. Soal pertama menghitung luas empat bangun datar dan murni berupa runtunan. Soal kedua adalah permainan tebak angka untuk dua pemain, sebuah perulangan sentinel yang menyamar. Soal ketiga mencetak 100 pangkat n untuk n berapa pun, dan sesungguhnya merupakan pelajaran tentang batas bilangan di dalam komputer. Soal keempat adalah batu, kertas, gunting, dengan validasi masukan dan aturan ringkas untuk menentukan pemenang. Soal kelima, pertarungan bergiliran antara seorang anak dan tiga goblin, menggabungkan variabel keadaan, perulangan, urutan yang cermat, dan setiap pengaman yang telah Anda pelajari. Setiap soal diselesaikan dari nol, selalu dengan urutan yang sama: soalnya dinyatakan lalu dirumuskan ulang dengan kata-kata sederhana; contoh dikerjakan dengan tangan; rancangan dibuat di atas kertas; program C yang lengkap beserta keluaran sebenarnya dan penjelasan baris-baris pentingnya; program yang sama dalam Python dengan keluaran yang identik; sebuah *solusi keliru yang menggoda* beserta cara persis ia gagal; dan sebuah varian *periksa diri* beserta jawabannya. Di akhir bab Anda akan memiliki sebuah metode untuk menaklukkan soal pemrograman kecil apa pun, serta lima contoh lengkap penerapannya.

4.1 Posisi Kita Sekarang

Sebelum menyelesaikan apa pun, mari kita gelar kotak perkakasnya. Menyebut perkakas yang diperlukan sebuah soal sudah merupakan separuh pekerjaan, dan setiap soal di bawah ini meraih segenggam perkakas yang berbeda.

Dari **Bab 1** Anda memiliki meja kerja dan kosakata dasar. Anda dapat mengompilasi program C dengan `gcc -std=c17 -Wall -Wextra -O2` dan menjalankan program Python dengan `python3`. Anda tahu bahwa variabel memiliki nama, tipe, nilai, dan alamat; bahwa `scanf` memerlukan alamat dan memeriksa apa yang dibacanya; bahwa `int` berbatas 2.147.483.647 sedangkan `int` di Python tidak berbatas; bahwa pembagian bulat membuang pecahan dan `%` memberikan sisanya; dan bahwa `if` menjalankan sebuah pernyataan hanya bila kondisinya benar.

Dari **Bab 2** Anda memiliki metode dan perulangan-perulangan pertama. Anda merancang dengan pseudokode dan penghalusan bertahap dari atas ke bawah; Anda mengenal runtunan, seleksi, dan perulangan; Anda dapat menulis perulangan `while` terkendali pencacah dan terkendali sentinel, dengan membaca lebih dahulu sebelum perulangan sentinel; Anda dapat melindungi pembagian dengan pemeriksaan; dan Anda dapat memakai `+=`, `++`, dan *cast* `(double)`.

Dari **Bab 3** Anda memiliki struktur kontrol lainnya: `for` untuk menghitung, `do`…`while` untuk “tanya sampai sah”, `switch` (dan `match` di Python) untuk kasus-kasus tetap, `break` dan `continue`, serta operator logika `&&`, `||`, dan `!` beserta evaluasi hubung-singkat dan hukum De Morgan.

Itu cukup untuk kelima soal. Tidak ada satu pun di bawah ini yang memakai fungsi, larik, atau string selain untuk mencetak teks; semuanya hadir di Bab 5, 6, dan 9, dan beberapa program di sini akan menjadi lebih pendek nanti. Itu justru alasan yang baik untuk menyelesaikannya sekarang, dengan cara panjang, dan memahami setiap barisnya.

4.2 Cara Menghadapi Soal Pemrograman

4.2.1 Tujuh langkah

Setiap soal di bab ini diselesaikan dengan tujuh langkah yang sama, versi pemrogram dari nasihat klasik Pólya untuk menyelesaikan soal matematika: pahami soalnya, susun rencana, laksanakan, lalu tinjau kembali [1]. Langkah-langkah ini bukan ritual; masing-masing ada karena melewatkannya adalah penyebab lazim satu jenis kegagalan tertentu.

1. **Rumuskan ulang soalnya dengan kata-kata Anda sendiri.** Tuliskan apa yang masuk, apa yang harus keluar, dan setiap keputusan yang dibiarkan terbuka oleh soal. Soal yang pendek justru paling berbahaya, karena setiap keputusan yang tidak diambilnya harus Anda ambil sendiri, lalu tuliskan.
2. **Kerjakan contoh dengan tangan.** Pilih masukan kecil, hitung keluaran yang benar di atas kertas, dan sertakan paling sedikit satu masukan yang tidak lazim: nol, bilangan negatif, jawaban pertama yang mungkin, jawaban terbesar.
3. **Rancang di atas kertas.** Pilih struktur kontrolnya dan tulis pseudokodenya. Untuk perulangan, putuskan apa yang dibaca sebelumnya, apa yang diuji, dan apa yang berubah di dalam badan.
4. **Telusuri rancangannya** pada salah satu contoh tangan tadi, baris demi baris, dengan tabel penelusuran kecil seperti di Bab 2.
5. **Tulis programnya** dari pseudokode, satu baris demi satu baris, dan kompilasi dengan peringatan menyala. Perbaiki setiap peringatan.
6. **Uji** dengan contoh-contoh tangan dari langkah 2 dan bandingkan keluarannya karakter demi karakter.
7. **Serang program Anda sendiri.** Carilah masukan yang mematahkannya: batas, nilai yang tidak sah, kasus yang membuat perulangan berjalan nol kali atau selamanya. Setiap soal di bawah diakhiri solusi keliru yang menggoda justru untuk melatih langkah ini.

> **WAWASAN KUNCI:** Sebagian besar waktu untuk sebuah soal sebaiknya dipakai untuk langkah 1 sampai 4, sebelum satu baris kode pun ada. Rancangan yang benar di atas kertas menjadi program dalam hitungan menit; program yang diketik tanpa rancangan sering memerlukan berjam-jam tebak-tebakan untuk diperbaiki.

4.2.2 Susunan setiap soal, dan letak kodenya

Setiap subbab soal memiliki bagian-bagian yang sama: **soalnya**, di dalam kotak, diikuti rumusan ulang kita; **memahaminya dengan tangan**, dengan contoh-contoh langkah 2; **merancangnya di atas kertas**, dengan pseudokode; **solusi C**, program lengkap dengan keluaran sebenarnya; **solusi Python**, yang mencetak keluaran yang persis sama; **solusi keliru yang menggoda**, program yang tampak benar, keluaran sebenarnya, dan alasan kegagalannya; serta varian **periksa diri** beserta jawabannya.

Semua program ada di folder `Chapter04` pada kode buku: `c` dan `python` berisi solusinya, `c/tempting` dan `python/tempting` berisi solusi keliru yang menggoda, dan `tests` berisi masukan dan keluaran yang diharapkan yang dipakai di bab ini. Setiap program C terkompilasi dengan `gcc`

`-std=c17 -Wall -Wextra -O2` tanpa satu peringatan pun dan juga dijalankan di bawah AddressSanitizer dan UndefinedBehaviorSanitizer (`-fsanitize=address,undefined`), yang menghentikan program yang membaca memori bukan miliknya atau meluapkan sebuah `int` [2]. Program yang membaca masukan dapat dijalankan dengan mengetik nilainya, atau dengan mengalihkan berkas masukan, misalnya `./guess < ../tests/guess.in` di Linux dan macOS atau `Get-Content ..\tests\guess.in | .\guess.exe` di Windows PowerShell.

4.3 Soal 4.1: Luas Bangun Datar

4.3.1 Soalnya

> **SOAL 4.1.** Write a program that reads the values base, base1, base2, height, side_length and radius (all of them real numbers) and prints the areas of a triangle, a trapezoid, a square and a circle.

Dengan kata lain: tulislah program yang membaca enam bilangan real, yaitu alas, alas1, alas2, tinggi, panjang sisi, dan jari-jari, lalu mencetak luas segitiga, trapesium, persegi, dan lingkaran. Soalnya pendek, jadi kita nyatakan keputusan-keputusannya yang terbuka. *Masukan:* enam bilangan real, ditanyakan satu per satu dengan prompt, sesuai urutan di soal; segitiga memakai `base` dan `height`, trapesium memakai `base1`, `base2`, dan `height` yang sama, persegi memakai `side_length`, dan lingkaran memakai `radius`. *Keluaran:* keempat luas, masing-masing di barisnya sendiri dengan label, dengan dua desimal. *Masukan tidak sah:* panjang tidak mungkin negatif; bila salah satu dari keenam nilai negatif, program menyatakannya dan tidak mencetak luas apa pun. Panjang 0 diizinkan dan menghasilkan luas 0.

4.3.2 Memahaminya dengan tangan

Keempat rumus berasal dari geometri sekolah. Dengan alas b dan tinggi h, luas segitiga adalah $\frac{1}{2}bh$; dengan sisi sejajar b_1 dan b_2 serta tinggi h, luas trapesium adalah $\frac{1}{2}(b_1 + b_2)h$; persegi bersisi s berluas s^2; dan lingkaran berjari-jari r berluas πr^2:

$$A_{\triangle} = \frac{1}{2}\,b\,h, \qquad A_{\text{trap}} = \frac{b_1 + b_2}{2}\,h, \qquad A_{\square} = s^2, \qquad A_{\circ} = \pi r^2 \tag{4.1}$$

Untuk contoh $b = 10$, $b_1 = 8$, $b_2 = 4$, $h = 5$, $s = 3$, dan $r = 2$, Persamaan (4.1) memberi segitiga $\frac{1}{2} \cdot 10 \cdot 5 = 25$, trapesium $\frac{12}{2} \cdot 5 = 30$, persegi 9, dan lingkaran $\pi \cdot 4 \approx 12.566$, yang dicetak 12.57. Contoh kedua memeriksa lingkaran saja: jari-jari 1 memberi $\pi \approx 3.14$. Bila sebuah program mencetak 3.00 untuk lingkaran itu, ia telah membagi atau mengalikan bilangan bulat di suatu tempat.

4.3.3 Merancangnya di atas kertas

Ini runtunan murni dengan satu pengaman: baca, periksa, hitung, cetak. **Pseudokode 4.1** adalah rancangannya.

Pseudokode 4.1 Rancangan Soal 4.1: runtunan pembacaan, satu pengaman, dan empat rumus.

```
GEOMETRY
1  read base, base1, base2, height, side, radius   (one prompt each)
2  if any of the six values is negative
3      print "Lengths cannot be negative." and stop
4  print 0.5 × base × height
5  print (base1 + base2) / 2 × height
6  print side × side
7  print PI × radius × radius
```

C17 standar tidak memiliki konstanta bawaan untuk π, jadi kita mendefinisikannya sendiri sebagai `const double` dengan 15 angka penting, lebih banyak daripada yang mampu dipakai sebuah `double`.

4.3.4 Solusi C

Kode 4.1 mengikuti rancangan baris demi baris.

Kode 4.1 `geometry.c`: luas empat bangun datar, dengan panjang negatif ditolak.

```c
/* geometry.c - Soal 4.1: luas empat bangun datar */
#include <stdio.h>

int main(void)
{
    const double PI = 3.14159265358979;
    double base, base1, base2, height, side, radius;

    printf("Triangle base: ");
    if (scanf("%lf", &base) != 1) return 1;
    printf("Trapezoid base1: ");
    if (scanf("%lf", &base1) != 1) return 1;
    printf("Trapezoid base2: ");
    if (scanf("%lf", &base2) != 1) return 1;
    printf("Height (triangle and trapezoid): ");
    if (scanf("%lf", &height) != 1) return 1;
    printf("Square side length: ");
    if (scanf("%lf", &side) != 1) return 1;
    printf("Circle radius: ");
    if (scanf("%lf", &radius) != 1) return 1;

    if (base < 0 || base1 < 0 || base2 < 0 || height < 0 || side < 0
        || radius < 0) {
        printf("Lengths cannot be negative.\n");
        return 1;
    }

    printf("\nTriangle area  = %10.2f\n", 0.5 * base * height);
    printf("Trapezoid area = %10.2f\n", (base1 + base2) / 2.0 * height);
    printf("Square area    = %10.2f\n", side * side);
    printf("Circle area    = %10.2f\n", PI * radius * radius);
    return 0;
}
```

```
Triangle base: 10
Trapezoid base1: 8
Trapezoid base2: 4
Height (triangle and trapezoid): 5
Square side length: 3
Circle radius: 2

Triangle area  =      25.00
Trapezoid area =      30.00
Square area    =       9.00
Circle area    =      12.57
```

Keluarannya cocok dengan hitungan tangan. Tiga rincian layak diperhatikan. Keenam nilai bertipe `double` dan dibaca dengan `%lf`, sehingga panjang seperti 2.5 diterima. Pengamannya menggabungkan enam perbandingan dengan `||`: seluruh kondisi bernilai benar begitu satu panjang negatif, dan evaluasi hubung-singkat berhenti di yang pertama. Barisnya dipecah dua, yang diizinkan C di mana pun spasi diizinkan, karena kondisi lengkapnya akan lebih panjang daripada satu baris. Akhirnya, rumus-rumusnya ditulis dengan `0.5` dan `2.0`, sehingga setiap operasi melibatkan `double` dan pembagian bulat tidak dapat menyelinap masuk. Satu kali jalan dengan panjang negatif memperlihatkan pengamannya:

```
Triangle base: -1
Trapezoid base1: 8
Trapezoid base2: 4
Height (triangle and trapezoid): 5
Square side length: 3
Circle radius: 2
Lengths cannot be negative.
```

4.3.5 Solusi Python

Kode 4.2 adalah program yang sama dalam Python; `float(input(...))` membaca bilangan real, dan `raise SystemExit(1)` mengakhiri program dengan status 1, padanan `return 1` pada `main` di C.

Kode 4.2 `geometry.py`: luas yang sama dalam Python.

```python
# geometry.py - Soal 4.1: luas empat bangun datar
PI = 3.14159265358979       # konstanta yang sama dengan program C

base = float(input("Triangle base: "))
base1 = float(input("Trapezoid base1: "))
base2 = float(input("Trapezoid base2: "))
height = float(input("Height (triangle and trapezoid): "))
side = float(input("Square side length: "))
radius = float(input("Circle radius: "))

if (base < 0 or base1 < 0 or base2 < 0 or height < 0 or side < 0
        or radius < 0):
    print("Lengths cannot be negative.")
    raise SystemExit(1)

print()
print(f"Triangle area  = {0.5 * base * height:10.2f}")
print(f"Trapezoid area = {(base1 + base2) / 2.0 * height:10.2f}")
print(f"Square area    = {side * side:10.2f}")
print(f"Circle area    = {PI * radius * radius:10.2f}")
```

Kondisi panjang di Python harus tetap berada di dalam tanda kurung agar dapat berlanjut ke baris berikutnya, dan itulah sebabnya pengamannya dibungkus `( … )`. Pustaka standar Python memang menyediakan π sebagai `math.pi`, tetapi memakai konstanta yang sama dengan program C menjamin keluaran yang identik.

4.3.6 Solusi keliru yang menggoda

Jalan pintas yang paling menggoda adalah menyalin rumus sekolah secara harfiah: “setengah kali alas kali tinggi”, `1 / 2 * base * height`. **Kode 4.3** melakukan persis itu, untuk segitiga saja.

Kode 4.3 `geometry_wrong.c`: `1 / 2` dievaluasi lebih dahulu, sebagai pembagian bulat.

```c
/* geometry_wrong.c - segitiga keliru yang menggoda: 1 / 2 bernilai 0 */
#include <stdio.h>

int main(void)
{
    double base, height;

    printf("Triangle base: ");
    if (scanf("%lf", &base) != 1) return 1;
    printf("Height: ");
    if (scanf("%lf", &height) != 1) return 1;

    printf("Triangle area = %.2f\n", 1 / 2 * base * height);
    return 0;
}
```

```
Triangle base: 10
Height: 5
Triangle area = 0.00
```

Program ini terkompilasi tanpa peringatan dan mencetak 0.00 untuk setiap segitiga. Ekspresinya dievaluasi dari kiri ke kanan: `1 / 2` membagi dua konstanta `int`, menghasilkan 0, dan apa pun yang dikalikan 0 adalah 0. Bahwa `base` dan `height` bertipe `double` tidak menolong, karena kerusakannya terjadi sebelum keduanya terlibat. Versi Python di `python/tempting` membuat kesalahan yang sama dengan `1 // 2` dan mencetak baris yang sama; dengan `/` biasa di Python kekeliruan ini tidak mungkin terjadi, karena `1 / 2` di sana bernilai 0.5. Obatnya di C adalah menulis `0.5`, atau `1.0 / 2`, atau mendahulukan bilangan `double`-nya: `base * height / 2`.

> **PERIKSA DIRI.** Tambahkan persegi panjang dengan `length` dan `width`. Baris mana di **Kode 4.1** yang berubah, dan apa yang dicetak baris barunya untuk panjang 6 dan lebar 4?
>
> *Jawaban.* Tiga tempat berubah: sebuah deklarasi baru dan dua prompt baru dengan panggilan `scanf`; dua perbandingan tambahan di pengaman panjang negatif; dan satu baris baru `printf("Rectangle area = %10.2f\n", length * width);`. Untuk 6 dan 4 ia mencetak `Rectangle area =      24.00`. Tidak ada lagi yang berubah, dan itulah keuntungan runtunan murni: bangun baru cukup ditambahkan di akhir.

4.4 Soal 4.2: Permainan Tebak Angka

4.4.1 Soalnya

> **SOAL 4.2.** Write a guessing game. The program reads the answer, then keeps reading guesses until a guess equals the answer. After each wrong guess it prints “Your guess is too low!” or “Your guess is too high!”. When the guess is right, it congratulates the player.

Dengan kata lain: dua orang bermain. Pemain 1 mengetik sebuah bilangan bulat rahasia; pemain 2 lalu menebak sampai tebakannya benar, dan sesudah setiap tebakan yang salah program mengatakan ke arah mana bilangan rahasia itu berada. Kita mengambil tiga keputusan. Pesan-pesannya dicetak persis seperti tertulis, termasuk tanda seru, karena program yang diuji oleh program atau orang lain dinilai dari keluarannya yang persis. Ucapan selamatnya juga melaporkan bilangan rahasia dan banyaknya tebakan, dengan “try” untuk satu dan “tries” untuk lebih. Dan bilangan rahasia itu terlihat di layar saat pemain 1 mengetiknya; permainan sungguhan akan membiarkan komputer memilih bilangan acak, yang memerlukan fungsi-fungsi dari Bab 5, jadi untuk sementara pemain 1 sebaiknya menutupi layarnya atau menggulirnya.

4.4.2 Memahaminya dengan tangan

Ambil bilangan rahasia 37. Pemain 2 menebak 50: terlalu tinggi. Lalu 25: terlalu rendah. Lalu 37: benar, sesudah 3 tebakan. Pemain yang cerdik membelah dua rentang yang tersisa setiap kali, sehingga 1 sampai 100 memerlukan paling banyak 7 tebakan, tetapi program tidak peduli bagaimana pemain menebak. Kasus yang tidak lazim adalah tebakan *pertama* yang langsung benar: program hanya boleh mencetak ucapan selamat, dengan "1 try", dan tidak pernah "too low" atau "too high".

4.4.3 Merancangnya di atas kertas

Strukturnya adalah perulangan sentinel dari Bab 2 yang menyamar. "Sentinel"-nya bukan nilai tetap seperti 0, melainkan jawaban itu sendiri: baca satu tebakan sebelum perulangan, ulangi *selama* tebakan berbeda dari jawaban, beri umpan balik dan baca lagi di dalamnya. Di dalam perulangan kita sudah tahu bahwa tebakannya salah, jadi `if`…`else` sederhana cukup untuk memilih antara rendah dan tinggi. **Pseudokode 4.2** memperlihatkannya.

Pseudokode 4.2 Rancangan Soal 4.2: perulangan baca-lebih-dahulu yang sentinelnya adalah jawaban itu sendiri.

```
GUESS
1  read answer
2  tries ← 1
3  read guess
4  while guess ≠ answer
5      if guess < answer
6          print "Your guess is too low!"
7      else
8          print "Your guess is too high!"
9      tries ← tries + 1
10     read guess
11 print congratulations with answer and tries
```

4.4.4 Solusi C

Kode 4.4 `guess.c`: permainan tebak angka sebagai perulangan `while` baca-lebih-dahulu.

```c
/* guess.c - Soal 4.2: permainan tebak angka untuk dua pemain */
#include <stdio.h>

int main(void)
{
    int answer, guess;
    int tries = 1;

    printf("Player 1, type the secret number: ");
    if (scanf("%d", &answer) != 1) return 1;

    printf("Player 2, your guess: ");
    if (scanf("%d", &guess) != 1) return 1;
    while (guess != answer) {
        if (guess < answer) {
            printf("Your guess is too low!\n");
        } else {
            printf("Your guess is too high!\n");
        }
        ++tries;
        printf("Player 2, your guess: ");
        if (scanf("%d", &guess) != 1) return 1;
    }
    printf("Congratulations! You found %d in %d %s.\n", answer, tries,
           tries == 1 ? "try" : "tries");
    return 0;
}
```

```
Player 1, type the secret number: 37
Player 2, your guess: 50
Your guess is too high!
Player 2, your guess: 25
Your guess is too low!
Player 2, your guess: 37
Congratulations! You found 37 in 3 tries.

Player 1, type the secret number: 7
Player 2, your guess: 7
Congratulations! You found 7 in 1 try.
```

Jalan pertama adalah contoh tangan tadi: dua tebakan salah dengan umpan balik yang tepat, lalu ucapan selamat sesudah 3 tebakan. Pada jalan kedua tebakan pertama langsung benar: kondisi perulangan langsung salah, badannya tidak pernah dijalankan, dan ucapan selamatnya berbunyi "1 try". Kata itu dipilih oleh operator kondisional `tries == 1 ? "try" : "tries"`, dicetak dengan `%s`, konversi untuk teks. Pencacah `tries` dimulai dari 1, bukan 0, karena tebakan yang dibaca sebelum perulangan sudah merupakan tebakan pertama.

4.4.5 Solusi Python

Kode 4.5 `guess.py`: perulangan baca-lebih-dahulu yang sama dalam Python.

```python
# guess.py - Soal 4.2: permainan tebak angka untuk dua pemain
tries = 1

answer = int(input("Player 1, type the secret number: "))

guess = int(input("Player 2, your guess: "))
while guess != answer:
    if guess < answer:
        print("Your guess is too low!")
    else:
        print("Your guess is too high!")
    tries += 1
    guess = int(input("Player 2, your guess: "))

word = "try" if tries == 1 else "tries"
print(f"Congratulations! You found {answer} in {tries} {word}.")
```

Kode 4.5 mencerminkan program C pernyataan demi pernyataan. Ia mencetak dua kali jalan yang sama; pilihan "try" atau "tries" dibuat dengan ekspresi kondisional Python sebelum `print` terakhir.

4.4.6 Solusi keliru yang menggoda

`do`…`while` tampak diciptakan untuk soal ini: "baca tebakan, nilai, ulangi sampai benar". **Kode 4.6** menuliskan persis itu.

Kode 4.6 `guess_wrong.c`: badannya menilai setiap tebakan, termasuk tebakan yang menang.

```c
/* guess_wrong.c - loop keliru yang menggoda: tebakan benar ikut dinilai */
#include <stdio.h>

int main(void)
{
    int answer, guess;

    printf("Player 1, type the secret number: ");
    if (scanf("%d", &answer) != 1) return 1;

    do {
        printf("Player 2, your guess: ");
        if (scanf("%d", &guess) != 1) return 1;
        if (guess < answer)
            printf("Your guess is too low!\n");
        else
            printf("Your guess is too high!\n");
    } while (guess != answer);
    printf("Congratulations!\n");
    return 0;
}
```

```
Player 1, type the secret number: 37
Player 2, your guess: 50
Your guess is too high!
Player 2, your guess: 25
Your guess is too low!
Player 2, your guess: 37
Your guess is too high!
Congratulations!
```

Tebakan pemenang 37 diikuti "Your guess is too high!" dan baru kemudian ucapan selamat. Badan `do`…`while` dijalankan sepenuhnya sebelum kondisinya diuji, sehingga tebakan yang benar ikut dinilai, dan cabang `else`, yang menganggap "tidak rendah berarti tinggi", menyala untuk "sama". Program ini bekerja pada setiap tebakan yang salah dan gagal pada tebakan yang paling

penting. Ada dua perbaikan: pertahankan `while` baca-lebih-dahulu pada **Kode 4.4**, yang badannya hanya pernah melihat tebakan salah, atau tambahkan kasus ketiga `else if (guess > answer)` sehingga kesamaan tidak mencetak apa pun. Yang pertama lebih bersih, karena meletakkan uji di tempat maknanya: nilai hanya selama tebakannya salah.

> **PERIKSA DIRI.** Batasi permainan menjadi 5 tebakan: sesudah tebakan salah yang kelima, program mencetak "Out of tries! The number was 37." alih-alih ucapan selamat. Menjadi apa kondisi perulangannya, dan bagaimana program tahu sesudah perulangan pesan mana yang harus dicetak?
>
> *Jawaban.* Perulangan berjalan selama `guess != answer && tries < 5`, dengan menaikkan `tries` dan membaca lagi di dalamnya. Sesudah perulangan, salah satu dari kedua kondisi telah gagal; ujilah `if (guess == answer)` untuk mencetak ucapan selamat, dan cetak pesan "Out of tries" bila tidak. Menguji `tries == 5` sebagai gantinya adalah keliru, karena pemain yang tepat menebak pada tebakan kelima juga memiliki `tries == 5`.

4.5 Soal 4.3: 100 Pangkat n

4.5.1 Soalnya

> **SOAL 4.3.** Write a program that reads an integer n and prints the value of 100^n. Hint: an `int` has 32 bits, and its range is from −2,147,483,648 to 2,147,483,647.

Dengan kata lain: program membaca sebuah bilangan bulat lalu mencetak 100 dipangkatkan bilangan itu, dengan petunjuk bahwa `int` hanya 32 bit. Petunjuk itulah seluruh soalnya. $100^5 = 10{,}000{,}000{,}000$ sudah lebih dari empat kali `int` terbesar, sehingga program yang paling jelas, yang mengalikan 100 dengan dirinya sendiri n kali di dalam `int`, keliru untuk setiap $n \geq 5$. Soal ini bertanya apakah Anda menyadarinya, dan apa yang Anda lakukan terhadapnya. Kita memutuskan: program harus mencetak 100^n *dengan tepat*, digit demi digit, untuk setiap n yang masuk akal diketik orang; kita menerima n dari −100.000 sampai 100.000; dan untuk n negatif kita mencetak pecahan desimal yang tepat, karena $100^{-2} = 0.0001$ adalah bilangan yang sah. Kita juga melaporkan apakah nilainya akan muat di dalam `int` atau `long long`.

4.5.2 Memahaminya dengan tangan

Pengamatan kuncinya adalah bahwa kita tidak perlu *menghitung* bilangannya untuk *mencetaknya*. Karena $100 = 10^2$,

$$100^n = (10^2)^n = 10^{2n} \tag{4.2}$$

dan 10^{2n}, bila ditulis dalam desimal, adalah angka 1 diikuti $2n$ nol. Jadi $100^0 = 1$, $100^1 = 100$, $100^3 = 1\,000\,000$ (angka 1 dan enam nol), dan 100^{12} adalah angka 1 dan 24 nol. Untuk pangkat negatif $n = -m$, Persamaan (4.2) memberi 10^{-2m}, yang ditulis "0." diikuti $2m - 1$ nol lalu angka 1: $100^{-1} = 0.01$ dan $100^{-2} = 0.0001$.

Sejauh mana tipe-tipe bawaan menjangkau? **Tabel 4.1** membandingkan pangkat-pangkat itu dengan batasnya: `int` memuat $100^4 = 10^8$ tetapi tidak 10^{10}; `long long`, yang nilai terbesarnya sekitar 9.22×10^{18}, memuat $100^9 = 10^{18}$ tetapi tidak 10^{20}.

Tabel 4.1 Pangkat-pangkat 100 terhadap batas tipe bilangan bulat C: `int` berhenti sesudah $n = 4$, `long long` sesudah $n = 9$.

n	100^n	Muat di `int` (maks. sekitar 2.1×10^9)?	Muat di `long long` (maks. sekitar 9.2×10^{18})?
4	10^8	ya	ya
5	10^{10}	tidak	ya
9	10^{18}	tidak	ya
10	10^{20}	tidak	tidak

4.5.3 Merancangnya di atas kertas

Rancangannya mencetak digit alih-alih menghitung bilangan, seperti yang ditunjukkan **Pseudokode 4.3**. Sebuah perulangan `for` mencetak nol-nolnya, karena banyaknya, $2n$ atau $2m - 1$, sudah diketahui sebelum perulangan dimulai.

Pseudokode 4.3 Rancangan Soal 4.3: cetak digit-digit 10^{2n} alih-alih menghitung bilangannya.

```
POWER-OF-100
1  read n
2  if n < -100000 or n > 100000
3      print a message and stop
4  print "100^n = "
5  if n ≥ 0
6      print "1"
7      repeat 2n times: print "0"
8  else
9      print "0."
10     repeat -2n - 1 times: print "0"
11     print "1"
12 print whether n fits int (0 ≤ n ≤ 4) and long long (0 ≤ n ≤ 9)
```

4.5.4 Solusi C

Kode 4.7 adalah **Pseudokode 4.3** yang ditulis dalam C; kedua perulangan `for` mencetak nol-nolnya, dan operator kondisional memilih “yes” atau “no” untuk kedua tipe.

Kode 4.7 `power100.c`: 100 pangkat n, dicetak digit demi digit, tanpa pernah meluap.

```c
/* power100.c - Soal 4.3: cetak 100 pangkat n dengan tepat, untuk n berapa pun */
#include <stdio.h>

int main(void)
{
    int n;

    printf("n: ");
    if (scanf("%d", &n) != 1) return 1;
    if (n < -100000 || n > 100000) {
        printf("Please keep n between -100000 and 100000.\n");
        return 1;
    }

    printf("100^%d = ", n);
    if (n >= 0) {
        printf("1");                          /* 100^n = 1 lalu 2n nol */
        for (int i = 1; i <= 2 * n; ++i)
            printf("0");
    } else {
        printf("0.");                         /* 100^-m = 0. (2m-1 nol) 1 */
        for (int i = 1; i <= -2 * n - 1; ++i)
            printf("0");
        printf("1");
    }
    printf("\n");
    printf("fits in int: %s, fits in long long: %s\n",
           n >= 0 && n <= 4 ? "yes" : "no", n >= 0 && n <= 9 ? "yes" : "no");
    return 0;
}
```

```
n: 0
100^0 = 1
fits in int: yes, fits in long long: yes
n: 3
100^3 = 1000000
fits in int: yes, fits in long long: yes
n: 12
100^12 = 1000000000000000000000000
fits in int: no, fits in long long: no
n: -2
100^-2 = 0.0001
fits in int: no, fits in long long: no
```

Hitunglah nol pada jalan ketiga: 24, sehingga bilangannya terdiri atas 25 digit, jauh melampaui tipe bilangan bulat bawaan mana pun, namun tak satu pun aritmetika mendekati batas. Bilangan terbesar yang dihitung program adalah batas perulangan `2 * n`, paling besar 200.000. Karena itulah pemeriksaan rentang n ada: tanpanya, pengguna yang mengetik 2.000.000.000 akan membuat `2 * n` meluapkan `int`, yang oleh standar C disebut *undefined behavior* [3] dan dilaporkan oleh *sanitizer*. Untuk $n = 0$ perulangannya berjalan nol kali dan keluarannya cukup 1, dan untuk $n = -2$ perulangannya mencetak $2 \cdot 2 - 1 = 3$ nol di antara "0." dan angka 1 terakhir.

4.5.5 Solusi Python

Kode 4.8 `power100.py`: perulangan pencetak digit yang sama dalam Python.

```
# power100.py - Soal 4.3: cetak 100 pangkat n dengan tepat, untuk n berapa pun
n = int(input("n: "))
if n < -100000 or n > 100000:
    print("Please keep n between -100000 and 100000.")
    raise SystemExit(1)

print(f"100^{n} = ", end="")
if n >= 0:
    print("1", end="")                      # 100^n = 1 lalu 2n nol
    for i in range(2 * n):
        print("0", end="")
else:
    print("0.", end="")                     # 100^-m = 0. (2m-1 nol) 1
    for i in range(-2 * n - 1):
        print("0", end="")
    print("1", end="")
print()
fits_int = "yes" if 0 <= n <= 4 else "no"
fits_ll = "yes" if 0 <= n <= 9 else "no"
print(f"fits in int: {fits_int}, fits in long long: {fits_ll}")
```

Kode 4.8 mencetak keempat jalan yang sama. Ia layak mendapat catatan, karena Python sebenarnya dapat mengambil jalan pintas: bilangan bulatnya tidak berbatas, sehingga `print(100 ** n)` mencetak 100^{12} dengan tepat [4], dan bahkan `"1" + "0" * (2 * n)` menyusun digitnya dalam satu ekspresi. Kita mempertahankan perulangannya agar kedua program memiliki rancangan yang sama, tetapi patut diketahui bahwa seluruh kesulitan soal ini adalah sifat bilangan bulat berukuran tetap di C, bukan sifat matematikanya.

4.5.6 Solusi keliru yang menggoda

C memang memiliki fungsi pangkat, `pow` dari Bab 3. **Kode 4.9** menyerahkan pekerjaannya kepada fungsi itu dan mencetak hasilnya tanpa desimal.

Kode 4.9 `power100_wrong.c`: `pow` mengembalikan `double`, yang hanya memiliki sekitar 16 angka penting.

```
/* power100_wrong.c - jawaban keliru yang menggoda: serahkan pada pow() */
#include <stdio.h>
#include <math.h>

int main(void)
{
    int n;

    printf("n: ");
    if (scanf("%d", &n) != 1) return 1;
    printf("100^%d = %.0f\n", n, pow(100.0, n));
    return 0;
}
```

```
n: 3
100^3 = 1000000
n: 12
100^12 = 999999999999999983222784
```

Untuk $n = 3$ jawabannya benar, dan pemrogram yang hanya menguji nilai kecil akan percaya bahwa programnya bekerja. Untuk $n = 12$ ia mencetak 999999999999999983222784 alih-alih 1 diikuti 24 nol. Penyebabnya, `pow` mengembalikan `double`, yang menyimpan sekitar 15 sampai 17 angka desimal penting [5]. Bilangan 10^{24} tidak memiliki wujud `double` yang tepat; `double` terdekatnya adalah 999.999.999.999.999.983.222.784, dan `%.0f` dengan setia mencetak bilangan *itu*. Versi Python-nya, `100.0 ** n` dengan format `:.0f`, mencetak digit keliru yang sama, karena ia

menghitung dengan `double` yang sama. Pelajarannya berlaku umum: `double` tepat hanya untuk bilangan bulat sampai 2^{53}, sekitar 9×10^{15}, dan apa pun yang lebih besar dibulatkan.

> **PERIKSA DIRI.** Ubahlah program agar mencetak 1000^n dengan tepat untuk $n \geq 0$. Berapa nol yang dicetak perulangannya, dan sampai n berapa 1000^n masih muat di `long long`?
>
> *Jawaban.* $1000^n = 10^{3n}$, jadi perulangannya mencetak $3n$ nol sesudah angka 1. `long long` memuat 10^{18}, yaitu 1000^6, tetapi tidak $10^{21} = 1000^7$, jadi jawabannya $n \leq 6$.

4.6 Soal 4.4: Batu, Kertas, Gunting

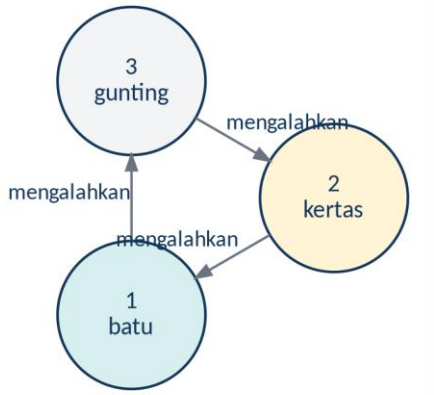
4.6.1 Soalnya

> **SOAL 4.4.** Write a rock, paper, scissors game for two players. Each player enters 1 (rock), 2 (paper) or 3 (scissors); an invalid number must be asked again. The program announces the result: rock beats scissors, scissors beats paper, and paper beats rock. As long as the process is the same, the design is free.

Dengan kata lain: buatlah permainan batu, kertas, gunting untuk dua pemain; masing-masing mengetik 1 (batu), 2 (kertas), atau 3 (gunting), angka yang tidak sah ditanyakan ulang, dan program mengumumkan hasilnya; selama prosesnya sama, rancangannya bebas. Kita memutuskan rinciannya. Setiap pemain ditanya dengan prompt yang mendaftar ketiga angka; angka lain mencetak "Please type 1, 2 or 3." dan menanyai pemain yang sama lagi. Hasilnya dicetak dengan kata-kata, pertama apa yang ditunjukkan setiap pemain lalu putusannya: "It is a draw.", "Player 1 wins!", atau "Player 2 wins!". Satu permainan untuk setiap kali jalan; memainkan beberapa ronde menjadi varian periksa diri.

4.6.2 Memahaminya dengan tangan

Ada $3 \times 3 = 9$ kombinasi dan tiga putusan. **Gambar 4.1** menggambar aturannya sebagai sebuah siklus: setiap benda mengalahkan benda yang ditunjuk anak panahnya. Batu (1) mengalahkan gunting (3), gunting (3) mengalahkan kertas (2), dan kertas (2) mengalahkan batu (1).



Gambar 4.1 Aturan batu, kertas, gunting sebagai siklus: setiap benda mengalahkan benda berikutnya menurut arah anak panah.

Sembilan pernyataan `if`, satu untuk setiap kombinasi, akan berhasil dan mudah diperiksa. Tetapi siklus itu mengisyaratkan aturan yang lebih pendek. Menelusuri siklus sekali dengan arah $1 \to 3 \to 2 \to 1$, nomor pemenang selalu satu *lebih besar* daripada nomor yang kalah, dihitung modulo 3: kertas (2) mengalahkan batu (1), gunting (3) mengalahkan kertas (2), dan batu (1) mengalahkan gunting (3), karena $1 \equiv 3 + 1 \pmod 3$. Jadi pemain 1 menang tepat bila $p_1 - p_2$ bersisa 1 ketika

dibagi 3. Karena $p_1 - p_2$ dapat bernilai negatif dan `%` di C mempertahankan tanda bilangan yang dibagi (Bab 1), kita menambahkan 3 lebih dahulu, yang tidak mengubah sisanya tetapi membuat bilangannya positif; jadi pemain 1 menang tepat bila

$$(p_1 - p_2 + 3) \bmod 3 = 1 \tag{4.3}$$

Rumus seperti ini harus diperiksa, bukan dipercaya begitu saja. **Tabel 4.2** menghitung Persamaan (4.3) untuk kesembilan kasus.

Tabel 4.2 Kesembilan kombinasi diperiksa terhadap Persamaan (4.3): sisa 0 berarti seri, 1 berarti pemain 1 menang, 2 berarti pemain 2 menang.

p_1	p_2	p_1-p_2+3	Mod 3	Putusan menurut rumus	Benar?
1 batu	1 batu	3	0	seri ($p_1 = p_2$)	ya
1 batu	2 kertas	2	2	pemain 2	ya
1 batu	3 gunting	1	1	pemain 1	ya
2 kertas	1 batu	4	1	pemain 1	ya
2 kertas	2 kertas	3	0	seri	ya
2 kertas	3 gunting	2	2	pemain 2	ya
3 gunting	1 batu	5	2	pemain 2	ya
3 gunting	2 kertas	4	1	pemain 1	ya
3 gunting	3 gunting	3	0	seri	ya

4.6.3 Merancangnya di atas kertas

Dua perulangan "tanya sampai sah", satu untuk setiap pemain, adalah `do`…`while` dari Bab 3. Mencetak nama adalah pilihan banyak arah atas satu bilangan bulat, sebuah `switch`. Putusannya berupa `if`…`else if`…`else` dengan Persamaan (4.3) di tengahnya. **Pseudokode 4.4** merangkainya.

Pseudokode 4.4 Rancangan Soal 4.4: dua perulangan validasi, sebuah `switch` untuk nama-nama, dan aturan Persamaan (4.3).

```
ROCK-PAPER-SCISSORS
1  repeat: read p1, and if it is not 1, 2 or 3 print a message
2  until p1 is 1, 2 or 3
3  repeat the same for p2
4  print the names of p1 and p2
5  if p1 = p2
6      print "It is a draw."
7  else if (p1 – p2 + 3) mod 3 = 1
8      print "Player 1 wins!"
9  else
10     print "Player 2 wins!"
```

4.6.4 Solusi C

Kode 4.10 menerjemahkan **Pseudokode 4.4**: dua perulangan `do`…`while`, dua pernyataan `switch`, dan satu rantai.

Kode 4.10 `rps.c`: masukan tervalidasi, nama dicetak dengan `switch`, dan pemenang ditemukan dengan satu sisa pembagian.

```c
/* rps.c - Soal 4.4: batu, kertas, gunting untuk dua pemain */
#include <stdio.h>

int main(void)
{
    int p1, p2;

    do {
        printf("Player 1 (1 = rock, 2 = paper, 3 = scissors): ");
        if (scanf("%d", &p1) != 1) return 1;
        if (p1 < 1 || p1 > 3)
            printf("Please type 1, 2 or 3.\n");
    } while (p1 < 1 || p1 > 3);
    do {
        printf("Player 2 (1 = rock, 2 = paper, 3 = scissors): ");
        if (scanf("%d", &p2) != 1) return 1;
        if (p2 < 1 || p2 > 3)
            printf("Please type 1, 2 or 3.\n");
    } while (p2 < 1 || p2 > 3);

    printf("Player 1 shows ");
    switch (p1) {
    case 1: printf("rock"); break;
    case 2: printf("paper"); break;
    default: printf("scissors"); break;
    }
    printf(", player 2 shows ");
    switch (p2) {
    case 1: printf("rock"); break;
    case 2: printf("paper"); break;
    default: printf("scissors"); break;
    }
    printf(".\n");

    if (p1 == p2)
        printf("It is a draw.\n");
    else if ((p1 - p2 + 3) % 3 == 1)        /* batu>gunting>kertas>batu */
        printf("Player 1 wins!\n");
    else
        printf("Player 2 wins!\n");
    return 0;
}
```

```
Player 1 (1 = rock, 2 = paper, 3 = scissors): 0
Please type 1, 2 or 3.
Player 1 (1 = rock, 2 = paper, 3 = scissors): 4
Please type 1, 2 or 3.
Player 1 (1 = rock, 2 = paper, 3 = scissors): 2
Player 2 (1 = rock, 2 = paper, 3 = scissors): 1
Player 1 shows paper, player 2 shows rock.
Player 1 wins!
Player 1 (1 = rock, 2 = paper, 3 = scissors): 1
Player 2 (1 = rock, 2 = paper, 3 = scissors): 3
Player 1 shows rock, player 2 shows scissors.
Player 1 wins!
Player 1 (1 = rock, 2 = paper, 3 = scissors): 3
Player 2 (1 = rock, 2 = paper, 3 = scissors): 3
Player 1 shows scissors, player 2 shows scissors.
It is a draw.
```

Jalan pertama menguji validasinya: pemain 1 mengetik 0 dan 4, masing-masing ditolak, lalu 2 (kertas); pemain 2 mengetik 1 (batu); kertas mengalahkan batu. Jalan kedua adalah kasus ketika "+ 3" berperan: batu melawan gunting menghasilkan $1 - 3 + 3 = 1$, bersisa 1, pemain 1 menang. Yang ketiga seri. Setiap `switch` mencetak satu kata dan mengakhiri setiap kasusnya dengan

`break`; kasus terakhir memakai `default` untuk 3, yang aman karena perulangan-perulangan di atasnya menjamin hanya 1, 2, atau 3 yang dapat tiba.

4.6.5 Solusi Python

Kode 4.11 `rps.py`: permainan yang sama dengan validasi `while True` dan `match`.

```
# rps.py - Soal 4.4: batu, kertas, gunting untuk dua pemain
while True:
    p1 = int(input("Player 1 (1 = rock, 2 = paper, 3 = scissors): "))
    if 1 <= p1 <= 3:
        break
    print("Please type 1, 2 or 3.")
while True:
    p2 = int(input("Player 2 (1 = rock, 2 = paper, 3 = scissors): "))
    if 1 <= p2 <= 3:
        break
    print("Please type 1, 2 or 3.")

print("Player 1 shows ", end="")
match p1:
    case 1: print("rock", end="")
    case 2: print("paper", end="")
    case _: print("scissors", end="")
print(", player 2 shows ", end="")
match p2:
    case 1: print("rock", end="")
    case 2: print("paper", end="")
    case _: print("scissors", end="")
print(".")

if p1 == p2:
    print("It is a draw.")
elif (p1 - p2 + 3) % 3 == 1:                # batu>gunting>kertas>batu
    print("Player 1 wins!")
else:
    print("Player 2 wins!")
```

Pada **Kode 4.11** perulangan validasinya memakai ungkapan `while True`…`break`, dan setiap `match` menaruh `case` beserta pernyataannya dalam satu baris, yang diizinkan Python bila pernyataannya pendek. Rumusnya ditulis persis seperti di C. Di Python `+ 3` sebenarnya bahkan tidak diperlukan, karena `%` di Python selalu mengembalikan sisa bertanda sama dengan pembaginya; kita mempertahankannya agar rumusnya sama di kedua bahasa.

4.6.6 Solusi keliru yang menggoda

Setelah menemukan siklusnya, sangat menggoda untuk menulis rumus tanpa `+ 3`: “pemain 1 menang bila $(p_1 - p_2) \bmod 3 = 1$”. Di atas kertas, dalam matematika, itu benar. **Kode 4.12** mencobanya di C dan sekaligus mencetak sisanya.

Kode 4.12 `rps_wrong.c`: tanpa `+ 3`, `%` di C memberi −2 untuk batu melawan gunting.

```c
/* rps_wrong.c - aturan keliru yang menggoda: (p1 - p2) % 3 tanpa + 3 */
#include <stdio.h>

int main(void)
{
    int p1, p2;

    printf("Player 1 (1-3): ");
    if (scanf("%d", &p1) != 1) return 1;
    printf("Player 2 (1-3): ");
    if (scanf("%d", &p2) != 1) return 1;

    printf("(p1 - p2) %% 3 = %d\n", (p1 - p2) % 3);
    if (p1 == p2)
        printf("It is a draw.\n");
    else if ((p1 - p2) % 3 == 1)
        printf("Player 1 wins!\n");
    else
        printf("Player 2 wins!\n");
    return 0;
}
```

```
Player 1 (1-3): 1
Player 2 (1-3): 3
(p1 - p2) % 3 = -2
Player 2 wins!
```

Batu melawan gunting: $1 - 3 = -2$, dan di C `-2 % 3` bernilai −2, bukan 1, karena sisa pembagian di C bertanda sama dengan bilangan yang dibagi. Uji `== 1` gagal, dan program menyatakan bahwa gunting mengalahkan batu. Gunting melawan kertas tidak terkena, karena $3 - 2 = 1$ positif. Kesalahannya mengenai tepat satu kombinasi ketika pemenang memiliki nomor yang *lebih kecil*, batu atas gunting, dan itulah sebabnya uji cepat dengan kertas melawan batu lolos dan *bug*-nya bertahan. Di Python ekspresi polos `(p1 - p2) % 3` akan memberi 1 dan benar; versi Python di `python/tempting` meniru C dengan `math.fmod` untuk memperlihatkan kegagalan yang sama. Rumus Persamaan (4.3), dengan `+ 3`-nya, benar di kedua bahasa.

> **PERIKSA DIRI.** Ubahlah program menjadi “dua dari tiga”: ronde dimainkan sampai salah satu pemain memenangi dua ronde; hasil seri tidak dihitung. Perulangan apa yang Anda perlukan di sekeliling permainan, variabel apa, dan apa kondisinya?
>
> *Jawaban.* Dua pencacah `wins1` dan `wins2`, keduanya 0, dan perulangan `while (wins1 < 2 && wins2 < 2)` yang membungkus pembacaan dan penilaian; kemenangan menaikkan pencacah pemenangnya dan seri tidak menaikkan apa pun. Sesudah perulangan, `wins1 == 2` memberi tahu siapa yang memenangi pertandingan. Perulangannya berupa `while`, bukan `for`, karena banyaknya ronde tidak diketahui sebelumnya: paling sedikit dua dan, bila ada seri, tidak terbatas.

4.7 Soal 4.5: Pertarungan Goblin

4.7.1 Soalnya

> **SOAL 4.5.** A boy fights three goblins, turn by turn. The boy has 200 hit points (HP) and an attack of 10; every third turn he uses a skill that deals 20 instead. Each goblin has 165 HP; their attacks are 3, 4 and 1. In every turn the boy attacks first, then the goblins attack him. Print the battle turn by turn until one side is wiped out, then print the number of turns and the winner’s remaining HP.

Dengan kata lain: seorang anak dengan 200 poin nyawa (HP) dan serangan 10, yang setiap giliran ketiga memakai jurus berkekuatan 20, melawan tiga goblin dengan 165 HP masing-masing dan serangan 3, 4, dan 1; setiap giliran si anak menyerang lebih dahulu, lalu para goblin membalas; cetak pertarungannya giliran demi giliran sampai salah satu pihak habis, lalu cetak banyaknya giliran dan sisa HP pemenangnya. Soal ini membiarkan beberapa keputusan terbuka, dan masing-masing mengubah hasilnya, jadi kita nyatakan semuanya. *Siapa yang diserang si anak?* Goblin pertama yang masih hidup: goblin 1 sampai ia tumbang, lalu goblin 2, lalu goblin 3. *Siapa yang membalas?* Hanya goblin yang masih hidup *sesudah* serangan si anak; goblin yang tumbang pada giliran ini tidak memukul. *Bolehkah HP di bawah nol?* Tidak: pukulan yang melebihi sisa HP menyisakan 0, baik untuk goblin maupun untuk si anak. *Apa yang dicetak?* Satu baris per giliran, sebagai tabel: giliran, jenis serangan, sasaran, HP ketiga goblin, kerusakan yang dibalas para goblin, dan HP si anak. *Apa "sisa HP pemenang"?* Untuk si anak, HP-nya; untuk para goblin, jumlah sisa HP mereka.

4.7.2 Memahaminya dengan tangan

Simulasikan giliran-giliran pertama di atas kertas. Giliran 1: si anak memberi 10 kepada goblin 1 (165 → 155); ketiga goblin hidup dan membalas 3 + 4 + 1 = 8; si anak tinggal 192. Giliran 2: sama, goblin 1 tinggal 145 dan si anak 184. Giliran 3 habis dibagi 3, jadi jurusnya memberi 20: goblin 1 tinggal 125, si anak 176. Polanya jelas: setiap tiga giliran goblin 1 kehilangan 10 + 10 + 20 = 40 HP sementara si anak kehilangan 24.

Kita dapat meramalkan hasilnya sebelum menulis kode apa pun. Goblin 1 memerlukan kerusakan 165 HP: 12 giliran memberi 4 × 40 = 160, dan giliran 13 menghabisinya, jadi goblin 1 tumbang pada giliran 13 dan tidak membalas pada giliran itu. Sejak saat itu hanya 4 + 1 = 5 kerusakan yang kembali per giliran. Goblin 2 tumbang pada giliran 26 dan goblin 3 pada giliran 39. Si anak menerima 8 × 12 = 96 pada giliran 1 sampai 12, 5 × 13 = 65 pada giliran 13 sampai 25, dan 1 × 13 = 13 pada giliran 26 sampai 38, seluruhnya 174, dan tidak menerima apa pun pada giliran 39. Ia menang sesudah 39 giliran dengan sisa 200 − 174 = 26 HP. Ramalan seperti ini adalah uji terbaik yang mungkin: jika program tidak sepakat, salah satunya keliru, dan mencari tahu yang mana selalu mengajarkan sesuatu.

4.7.3 Merancangnya di atas kertas

Keadaan (*state*) pertarungan adalah lima bilangan: HP si anak, HP ketiga goblin, dan nomor giliran. Satu iterasi perulangan adalah satu giliran utuh, dan **Gambar 4.2** memperlihatkan langkah-langkahnya. Perulangan berlanjut selama si anak hidup *dan* paling sedikit satu goblin hidup, `boy > 0 && (g1 > 0 || g2 > 0 || g3 > 0)`, sebuah kondisi yang memakai kedua operator logika dan memerlukan tanda kurungnya.



Gambar 4.2 Satu giliran pertarungan: pilih kerusakan, pukul goblin hidup yang pertama, biarkan goblin yang hidup membalas, cetak satu baris, dan putuskan apakah pertarungan berlanjut.

Dalam pseudokode, **Pseudokode 4.5**:

Pseudokode 4.5 Rancangan Soal 4.5: lima variabel keadaan dan satu perulangan, satu iterasi per giliran.

```
BATTLE
1  boy ← 200; g1, g2, g3 ← 165; turn ← 0
2  print the table header
3  while boy > 0 and (g1 > 0 or g2 > 0 or g3 > 0)
4      turn ← turn + 1
5      damage ← 20 if turn mod 3 = 0, else 10
6      target ← the first goblin with HP > 0
7      target's HP ← max(0, target's HP − damage)
8      back ← 3·[g1 > 0] + 4·[g2 > 0] + 1·[g3 > 0]
9      boy ← max(0, boy − back)
10     print turn, attack, target, g1, g2, g3, back, boy
11 if boy > 0
12     print "The boy wins" with turn and boy
13 else
14     print "The goblins win" with turn and g1 + g2 + g3
```

Pada baris 8, $[g_1 > 0]$ berarti 1 bila goblin 1 hidup dan 0 bila tidak, sehingga hanya goblin hidup yang menyumbang. Urutan baris 7 dan 8 menyandikan keputusan bahwa goblin yang tumbang pada giliran ini tidak membalas.

4.7.4 Solusi C

Kode 4.13 `battle.c`: pertarungan sebagai satu perulangan `while` atas lima variabel keadaan, dicetak sebagai tabel.

```c
/* battle.c - Soal 4.5: seorang anak melawan tiga goblin, giliran demi giliran */
#include <stdio.h>

int main(void)
{
    int boy = 200;                          /* poin nyawa (HP) */
    int g1 = 165, g2 = 165, g3 = 165;
    int turn = 0;

    printf("Turn  Attack  Target  G1   G2   G3   Hit back  Boy\n");
    while (boy > 0 && (g1 > 0 || g2 > 0 || g3 > 0)) {
        ++turn;
        int damage = turn % 3 == 0 ? 20 : 10;   /* jurus setiap giliran ketiga */
        int target;

        if (g1 > 0) {                       /* goblin pertama yang masih hidup */
            target = 1;
            g1 = g1 > damage ? g1 - damage : 0;
        } else if (g2 > 0) {
            target = 2;
            g2 = g2 > damage ? g2 - damage : 0;
        } else {
            target = 3;
            g3 = g3 > damage ? g3 - damage : 0;
        }

        int back = 0;                       /* hanya goblin hidup yang memukul */
        if (g1 > 0) back += 3;
        if (g2 > 0) back += 4;
        if (g3 > 0) back += 1;
        boy = boy > back ? boy - back : 0;

        printf("%4d  %-6s     G%d  %3d  %3d  %3d  %8d  %3d\n", turn,
               damage == 20 ? "skill" : "normal", target, g1, g2, g3, back, boy);
    }

    if (boy > 0)
        printf("The boy wins after %d turns with %d HP left.\n", turn, boy);
    else
        printf("The goblins win after %d turns with %d HP left in total.\n",
               turn, g1 + g2 + g3);
    return 0;
}
```

```
Turn  Attack  Target  G1   G2   G3   Hit back  Boy
   1  normal     G1  155  165  165          8  192
   2  normal     G1  145  165  165          8  184
   3  skill      G1  125  165  165          8  176
   4  normal     G1  115  165  165          8  168
   5  normal     G1  105  165  165          8  160
   6  skill      G1   85  165  165          8  152
   7  normal     G1   75  165  165          8  144
   8  normal     G1   65  165  165          8  136
   9  skill      G1   45  165  165          8  128
  10  normal     G1   35  165  165          8  120
  11  normal     G1   25  165  165          8  112
  12  skill      G1    5  165  165          8  104
  13  normal     G1    0  165  165          5   99
  14  normal     G2    0  155  165          5   94
  15  skill      G2    0  135  165          5   89
  16  normal     G2    0  125  165          5   84
  17  normal     G2    0  115  165          5   79
  18  skill      G2    0   95  165          5   74
  19  normal     G2    0   85  165          5   69
  20  normal     G2    0   75  165          5   64
  21  skill      G2    0   55  165          5   59
  22  normal     G2    0   45  165          5   54
  23  normal     G2    0   35  165          5   49
  24  skill      G2    0   15  165          5   44
  25  normal     G2    0    5  165          5   39
  26  normal     G2    0    0  165          1   38
  27  skill      G3    0    0  145          1   37
  28  normal     G3    0    0  135          1   36
  29  normal     G3    0    0  125          1   35
  30  skill      G3    0    0  105          1   34
  31  normal     G3    0    0   95          1   33
  32  normal     G3    0    0   85          1   32
  33  skill      G3    0    0   65          1   31
  34  normal     G3    0    0   55          1   30
  35  normal     G3    0    0   45          1   29
  36  skill      G3    0    0   25          1   28
  37  normal     G3    0    0   15          1   27
  38  normal     G3    0    0    5          1   26
  39  skill      G3    0    0    0          0   26
The boy wins after 39 turns with 26 HP left.
```

Program sepakat dengan ramalan: goblin 1 tumbang pada giliran 13 (dan kerusakan balasan turun dari 8 menjadi 5 pada giliran itu juga), goblin 2 pada giliran 26, goblin 3 pada giliran 39, dan si anak menang dengan 26 HP. HP tidak pernah turun di bawah 0 karena setiap pengurangan dijaga oleh operator kondisional: `g1 = g1 > damage ? g1 - damage : 0;` mengurangi bila HP yang tersisa cukup dan mengisi 0 bila tidak. Kerusakan balasan disusun dengan `back += …` hanya untuk goblin yang HP-nya di atas 0. Tabelnya memakai lebar kolom sehingga kolom-kolomnya rata sepanjang pertarungan: `%4d` untuk giliran, `%-6s` untuk jenis serangan, `%3d` untuk setiap HP. Akhirnya, `damage == 20 ? "skill" : "normal"` menamai serangan dari kerusakannya, sehingga aturan “setiap giliran ketiga” hanya tinggal di satu tempat, yaitu baris yang memilih kerusakan.

4.7.5 Solusi Python

Kode 4.14 `battle.py`: pertarungan yang sama dalam Python.

```
# battle.py - Soal 4.5: seorang anak melawan tiga goblin, giliran demi giliran
boy = 200                                   # poin nyawa (HP)
g1 = g2 = g3 = 165
turn = 0

print("Turn  Attack  Target  G1   G2   G3   Hit back  Boy")
while boy > 0 and (g1 > 0 or g2 > 0 or g3 > 0):
    turn += 1
    damage = 20 if turn % 3 == 0 else 10    # jurus setiap giliran ketiga

    if g1 > 0:                              # goblin pertama yang masih hidup
        target = 1
        g1 = g1 - damage if g1 > damage else 0
    elif g2 > 0:
        target = 2
        g2 = g2 - damage if g2 > damage else 0
    else:
        target = 3
        g3 = g3 - damage if g3 > damage else 0

    back = 0                                # hanya goblin hidup yang memukul
    if g1 > 0:
        back += 3
    if g2 > 0:
        back += 4
    if g3 > 0:
        back += 1
    boy = boy - back if boy > back else 0

    kind = "skill" if damage == 20 else "normal"
    print(f"{turn:4d}  {kind:<6s}     G{target}  {g1:3d}  {g2:3d}  {g3:3d}"
          f"  {back:8d}  {boy:3d}")

if boy > 0:
    print(f"The boy wins after {turn} turns with {boy} HP left.")
else:
    print(f"The goblins win after {turn} turns with {g1 + g2 + g3} HP left "
          "in total.")
```

Kode 4.14 mencetak 39 baris yang sama dan putusan yang sama. Pernyataan `g1 = g2 = g3 = 165` menugaskan satu nilai kepada tiga variabel sekaligus. `print` yang panjang dipecah menjadi dua f-string yang ditulis berdampingan di dalam tanda kurung; Python menggabungkan literal string yang bersebelahan menjadi satu, cara yang praktis untuk menjaga baris tetap pendek.

4.7.6 Solusi keliru yang menggoda

Versi yang menggoda melupakan pengaman-pengamannya. Goblin selalu membalas, hidup atau mati, dan HP cukup dikurangi, tanpa lantai di angka nol; agar keluarannya pendek, ia hanya mencetak setiap giliran kelima dan giliran terakhir.

Kode 4.15 `battle_wrong.c`: tanpa pemeriksaan “masih hidup?”, goblin yang mati terus bertarung dan HP menjadi negatif.

```c
/* battle_wrong.c - pertarungan keliru: tak ada yang cek siapa hidup */
#include <stdio.h>

int main(void)
{
    int boy = 200;
    int g1 = 165, g2 = 165, g3 = 165;
    int turn = 0;

    while (boy > 0 && g1 + g2 + g3 > 0) {
        ++turn;
        int damage = turn % 3 == 0 ? 20 : 10;
        if (g1 > 0)
            g1 -= damage;               /* bisa turun di bawah nol */
        else if (g2 > 0)
            g2 -= damage;
        else
            g3 -= damage;
        boy -= 3 + 4 + 1;               /* setiap goblin selalu membalas */
        if (turn % 5 == 0 || boy <= 0)
            printf("turn %2d: G1 %4d  G2 %4d  G3 %4d  boy %4d\n",
                   turn, g1, g2, g3, boy);
    }
    printf("%s after %d turns.\n", boy > 0 ? "The boy wins" : "The goblins win",
           turn);
    return 0;
}
```

```
turn  5: G1  105  G2  165  G3  165  boy  160
turn 10: G1   35  G2  165  G3  165  boy  120
turn 15: G1   -5  G2  135  G3  165  boy   80
turn 20: G1   -5  G2   75  G3  165  boy   40
turn 25: G1   -5  G2    5  G3  165  boy    0
The goblins win after 25 turns.
```

Hasilnya bukan ketidaktepatan kecil melainkan putusan yang terbalik. Sejak giliran 13, goblin 1 ber-HP −5, tercetak sebagai bilangan negatif, dan tetap memberi 3 kerusakan setiap giliran; si anak terus kehilangan 8 HP per giliran alih-alih 5 lalu 1, dan tumbang pada giliran 25. Dua pengaman yang hilang mengubah siapa yang menang. Perhatikan pula bahwa kondisi perulangan `g1 + g2 + g3 > 0` keliru dengan cara yang lebih halus: bila HP negatif diizinkan, jumlah di atas 0 tidak berarti ada goblin yang hidup. Kondisi yang benar menguji setiap goblin secara terpisah, seperti pada **Kode 4.13**.

> **PERIKSA DIRI.** (a) Apa yang terjadi bila serangan goblin 2 bernilai 6 alih-alih 4? (b) Dengan serangan aslinya, berapa HP awal terkecil yang masih membuat si anak menang?
>
> *Jawaban.* (a) Para goblin menang: dengan 3 + 6 + 1 = 10 kerusakan per giliran selama ketiganya hidup, si anak tumbang pada giliran 24, ketika goblin 2 masih ber-HP 15 dan goblin 3 belum tersentuh; program mencetak “The goblins win after 24 turns with 180 HP left in total.” (b) Si anak kehilangan tepat 174 HP sepanjang pertarungan, seperti dihitung di Subbab 4.7.2, dan jalannya pertarungan tidak bergantung pada HP-nya selama ia bertahan. Jadi ia memerlukan 175 HP: ia lalu menang sesudah 39 giliran dengan sisa 1 HP. Dengan 174 HP ia tumbang pada giliran 38, satu giliran sebelum goblin terakhir.

4.8 Soal-soal Berdampingan

Tabel 4.3 mendaftar kelima soal beserta struktur yang diperlukannya dan jebakan yang menjerat setiap solusi yang menggoda. Bacalah kolom terakhir sekali lagi: tidak satu pun dari kelima jebakan itu aneh. Pembagian bulat, menilai sesudah kejadian, batas sebuah tipe, tanda sisa

pembagian, dan pengaman yang hilang adalah kesalahan yang muncul di program sungguhan setiap hari.

Tabel 4.3 Kelima soal bab ini, strukturnya, dan jebakan setiap solusi keliru yang menggoda.

Soal	Struktur utama	Jebakan solusi yang menggoda
4.1 Bangun datar	runtunan, satu pengaman yang menggabungkan enam uji dengan OR	`1 / 2` adalah pembagian bulat, jadi setiap segitiga berluas 0
4.2 Tebak angka	`while` baca-lebih-dahulu, `if`…`else`	`do`…`while` menilai tebakan pemenang sebagai "too high"
4.3 100^n	`if`…`else`, `for` atas digit	`pow` mengembalikan `double` dengan sekitar 16 digit
4.4 Batu, kertas, gunting	validasi `do`…`while`, `switch`, satu sisa pembagian	tanpa `+ 3`, `%` di C negatif untuk batu melawan gunting
4.5 Pertarungan goblin	variabel keadaan, `while` dengan AND dan OR, pembaruan terjaga	tanpa pemeriksaan "hidup", goblin mati bertarung, HP negatif, putusan terbalik

4.9 Memakainya Sungguhan

Kelima soal tampak seperti latihan, tetapi bentuk-bentuknya ada di mana-mana dalam perangkat lunak sungguhan. Soal 4.1 adalah setiap halaman kalkulator: baca ukuran, validasi, terapkan rumus, cetak hasil; toko keramik yang menghitung berapa dus yang diperlukan sebuah lantai adalah program yang sama dengan rumus yang berbeda. Soal 4.2 adalah setiap perulangan coba-lagi: papan PIN yang berkata "PIN salah" sampai PIN yang benar tiba, dengan batas percobaan dari varian periksa diri yang dimiliki setiap kartu bank. Soal 4.3 adalah pengingat bahwa bilangan di dalam program punya batas; uang yang dihitung dalam rupiah bulat muat dengan lega di `long long` (sampai sekitar sembilan juta triliun rupiah), tetapi harga yang dikalikan jumlah besar di dalam `int` dapat meluap jauh sebelum itu. Soal 4.4 adalah setiap menu yang divalidasi. Dan Soal 4.5 adalah setiap simulasi: antrean di meja *laundry* yang disimulasikan menit demi menit, stok gudang hari demi hari, atau tabungan bulan demi bulan, semuanya menyimpan beberapa variabel keadaan dan memperbaruinya dalam satu perulangan, dengan pengaman yang menjaga keadaannya tetap masuk akal.

Bagi Profitina Laundry, pola pertarungan itu menjawab pertanyaan sungguhan. Jika tiga mesin cuci masing-masing memerlukan 60 menit per muatan dan pelanggan membawa satu kantong setiap 15 menit pada pagi yang ramai, seberapa panjang antrean tumbuh sampai tengah hari? Keadaannya adalah banyaknya kantong yang menunggu dan menit yang tersisa di setiap mesin; satu iterasi adalah satu menit; dan pengaman-pengamannya menjaga hitungan agar tidak negatif. Dengan perkakas Bab 1 sampai 3 Anda sudah dapat menulis simulasi itu.

4.10 Kesalahan yang Sering Terjadi, dan Cara Menghindarinya

- **Memutuskan rincian yang tak tertulis secara diam-diam.** Setiap soal di sini membiarkan keputusan terbuka: goblin mana yang diserang, bolehkah HP negatif, apa arti "tidak sah". Tuliskan keputusan Anda di bagian atas program sebagai komentar; pembaca, atau penguji, lalu tahu apa yang dijanjikan program.
- **Hanya menguji kasus yang lazim.** Setiap solusi yang menggoda lolos uji biasa dan gagal pada uji yang tidak lazim: tebakan yang langsung benar, pangkat yang besar, batu melawan gunting. Ujilah batas dan kasus khusus dengan sengaja.

- **Teks keluaran yang tidak cocok.** Bila soal memberikan pesan yang persis, seperti "Your guess is too low!", cetaklah karakter demi karakter. Uji otomatis membandingkan teks dengan persis, dan satu tanda seru yang hilang membuatnya gagal.
- **Memercayai rumus tanpa memeriksa setiap kasus.** Persamaan (4.3) benar, tetapi hanya tabel kesembilan kasus yang membuktikannya. Bila banyaknya kasus kecil, periksalah semuanya.
- **Luapan yang tidak terlihat kompilator.** Aritmetika `int` yang melampaui 2.147.483.647 adalah *undefined behavior* di C, dan program boleh mencetak apa saja. Perkirakan besarnya bilangan sebelum memilih tipe; jalankan dengan `-fsanitize=undefined` selama pengujian.
- **Mencetak di cabang yang salah.** Dalam pertarungan, mencetak sebelum para goblin membalas, atau sesudah perulangan alih-alih di dalamnya, menghasilkan tabel yang tidak sesuai aturan. Telusuri satu giliran di atas kertas dan bandingkan dengan baris pertama keluaran.
- **Validasi setengah jalan.** Menolak panjang negatif tetapi tidak menolak huruf, atau memeriksa pemain 1 tetapi tidak pemain 2, memberi rasa aman yang palsu. Validasilah setiap masukan dengan cara yang sama.

4.11 Ringkasan Bab

- Soal kecil diselesaikan dalam tujuh langkah: rumuskan ulang, kerjakan contoh dengan tangan, rancang di atas kertas, telusuri rancangannya, tulis programnya, uji terhadap contoh tangan, dan serang dengan masukan yang tidak lazim.
- Soal 4.1 adalah runtunan murni dengan satu pengaman; rumus-rumusnya (Persamaan (4.1)) harus ditulis dengan konstanta `double` seperti `0.5`, karena `1 / 2` bernilai 0 dalam aritmetika bilangan bulat.
- Soal 4.2 adalah perulangan `while` baca-lebih-dahulu yang sentinelnya adalah jawaban; `do`…`while` yang menilai sebelum menguji salah memberi label pada tebakan pemenang.
- Soal 4.3 memperlihatkan bahwa `int` berhenti sesudah 100^4 dan `long long` sesudah 100^9 (**Tabel 4.1**); mencetak digit 10^{2n} (Persamaan (4.2)) tepat untuk n berapa pun, sedangkan `pow` mengembalikan `double` yang dibulatkan.
- Soal 4.4 memvalidasi kedua pemain dengan `do`…`while`, mencetak nama dengan `switch`, dan menentukan pemenang dengan $(p_1 - p_2 + 3) \bmod 3$ (Persamaan (4.3)), yang diperiksa pada kesembilan kasus (**Tabel 4.2**); tanpa `+ 3`, sisa negatif di C mematahkan satu kasus.
- Soal 4.5 menyimpan lima variabel keadaan dan menjalankan satu iterasi per giliran (**Gambar 4.2**); menjaga setiap serangan dengan "masih hidup?" dan tidak pernah membiarkan HP di bawah 0 menentukan pemenangnya, dan ramalan di atas kertas (sisa 26 HP sesudah 39 giliran) mengukuhkan programnya.
- Solusi-solusi keliru yang menggoda (**Tabel 4.3**) gagal pada jebakan sehari-hari, dan masing-masing lolos uji biasa; uji yang tidak lazimlah yang menemukannya.

4.12 Soal Latihan

1. Kembangkan Soal 4.1 dengan luas jajaran genjang (alas dan tinggi) dan keliling setiap bangun. Nilai baru apa yang diperlukan untuk keliling trapesium, dan mengapa program tidak dapat menghitungnya dari nilai-nilai yang diberikan?
2. Dalam permainan tebak angka, cetaklah sesudah setiap tebakan salah rentang tempat jawaban pasti berada, misalnya "The number is between 26 and 49." Dua variabel apa yang Anda perlukan, dan bagaimana keduanya diperbarui?

3. Tulislah program yang membaca n dan mencetak 2^n dengan tepat untuk $0 \le n \le 62$ dengan sebuah `long long` dan perulangan `for`. Mengapa 62 batasnya, dan apa yang terjadi pada 63 di bawah `-fsanitize=undefined`?
4. Batu, kertas, gunting memiliki versi yang diperluas dengan kadal dan Spock, lima benda yang masing-masing mengalahkan dua benda lain. Beri nomor 1 sampai 5 dengan urutan batu, Spock, kertas, kadal, gunting. Tunjukkan bahwa pemain 1 menang tepat bila $(p_1 - p_2 + 5) \bmod 5$ bernilai 1 atau 2 dengan memeriksa kasus batu melawan gunting.
5. Dalam pertarungan, biarkan jurus si anak juga menyembuhkannya sebanyak 5 HP (paling tinggi sampai 200 awalnya) setiap kali jurus itu dipakai. Di mana pernyataan barunya diletakkan, dan apa hasilnya?
6. Jelaskan dalam dua kalimat mengapa kondisi perulangan `g1 + g2 + g3 > 0` benar pada **Kode 4.13** bila HP tidak pernah di bawah 0, tetapi keliru pada **Kode 4.15**.

4.13 Penyelesaian Soal Latihan

1. Luas jajaran genjang adalah alas × tinggi, satu baris lagi dari jenis yang sama. Keliling segitiga dan trapesium memerlukan panjang sisi-sisi miringnya, yang tidak ada di dalam masukan: tinggi dan alas tidak menentukan dua sisi lain sebuah segitiga. Program harus menanyakan sisi yang hilang itu atau dibatasi pada bangun khusus, seperti segitiga siku-siku, yang sisi ketiganya mengikuti $\sqrt{b^2 + h^2}$. Keliling persegi adalah $4s$ dan keliling lingkaran $2\pi r$.

2. Simpan `low` dan `high`, yang dimulai dari bilangan rahasia terkecil dan terbesar yang diizinkan, misalnya 1 dan 100. Sesudah tebakan yang terlalu rendah, isi `low = guess + 1`; sesudah tebakan yang terlalu tinggi, isi `high = guess - 1`; lalu cetak “The number is between %d and %d.”. Untuk bilangan rahasia 37 dengan tebakan 50 dan 25, program berkata 1 sampai 49 lalu 26 sampai 49.

3. Mulailah dengan `long long p = 1;` lalu kalikan dengan 2 di dalam perulangan `for (int i = 1; i <= n; ++i) p *= 2;`, dan cetak dengan `%lld`. `long long` memuat sampai $2^{63} - 1$, jadi 2^{62} adalah pangkat 2 terbesar yang muat. Pada $n = 63$ perkalian terakhir meluap; *sanitizer* menghentikan program dengan pesan “signed integer overflow”, dan tanpanya program mungkin mencetak bilangan negatif. Di Python, `2 ** 63` hanyalah 9223372036854775808.

4. Batu bernomor 1 dan gunting 5: $(1 - 5 + 5) \bmod 5 = 1$, jadi pemain 1 menang, dan itu benar, karena batu meremukkan gunting. Polanya berhasil karena dalam urutan batu, Spock, kertas, kadal, gunting setiap benda mengalahkan benda tepat sebelumnya dan benda dua tempat sebelumnya, dihitung mengelilingi lingkaran lima benda.

5. Sesudah serangan si anak dan sebelum para goblin membalas, tambahkan: bila kerusakannya 20, isi `boy = boy + 5 > 200 ? 200 : boy + 5;`. Jalannya pertarungan sendiri tidak berubah, karena HP para goblin tidak bergantung pada HP si anak; ia sembuh 5 HP pada setiap dari 13 giliran jurus dan tidak pernah mencapai batas 200, sehingga ia menang sesudah 39 giliran dengan 26 + 13 × 5 = 91 HP. Urutan penyembuhan dan serangan goblin harus dinyatakan, karena urutan itu akan mengubah hasilnya bila HP si anak sedang rendah.

6. Bila HP tidak pernah di bawah 0, jumlahnya lebih besar dari 0 tepat bila paling sedikit satu goblin ber-HP di atas 0, sehingga ujinya benar, meskipun kurang jelas dibanding menguji setiap goblin. Bila HP boleh negatif, goblin mati ber-HP −5 dan goblin hidup ber-HP 3 berjumlah −2, sehingga pertarungan berakhir padahal masih ada goblin yang hidup; jumlah itu tidak lagi berarti “ada yang hidup”.

Daftar Pustaka

[1] G. Pólya, *How to Solve It: A New Aspect of Mathematical Method*, 2nd ed. Princeton, NJ, USA: Princeton Univ. Press, 1957.

[2] GCC Project, "Program Instrumentation Options: -fsanitize=address, -fsanitize=undefined," *Using the GNU Compiler Collection (GCC)*. Accessed: Sep. 25, 2026. [Online]. Available: https://gcc.gnu.org/onlinedocs/gcc/Instrumentation-Options.html

[3] *Information Technology — Programming Languages — C*, ISO/IEC 9899:2018, International Organization for Standardization, Geneva, Switzerland, 2018.

[4] Python Software Foundation, "Built-in Types: Numeric Types — int, float, complex," *Python 3.14 Documentation*. Accessed: Sep. 25, 2026. [Online]. Available: https://docs.python.org/3.14/library/stdtypes.html

[5] D. Goldberg, "What every computer scientist should know about floating-point arithmetic," *ACM Comput. Surv.*, vol. 23, no. 1, pp. 5–48, Mar. 1991.

Bab 5

Fungsi: Bagi, Beri Nama, dan Taklukkan

Semua program di Bab 1 sampai 3 tinggal di dalam satu fungsi, `main`, dan bertambah panjang setiap kali kita mengajarinya hal baru. Cara itu masih jalan untuk tiga puluh baris, tetapi gagal untuk tiga ratus: tidak ada orang yang sanggup memegang tiga ratus baris di kepalanya, dan kalau sebuah aturan seperti harga muncul di tiga tempat, satu dari ketiganya pasti terlupa ketika harga berubah. Bab ini mengajarkan obat yang dipakai setiap pemrogram: potong program menjadi bagian-bagian kecil, beri setiap bagian sebuah nama, lalu biarkan `main` bercerita dengan memanggil nama-nama itu. Di akhir bab ini Anda akan mampu mendefinisikan dan memanggil fungsi buatan sendiri dalam C dan Python; menjelaskan apa yang dijanjikan sebuah prototipe dan mengapa kompilator memerlukannya; menyebutkan dengan tepat apa yang terjadi pada argumen ketika diteruskan ke parameter; meminjam fungsi siap pakai dari pustaka standar, termasuk pustaka matematika yang memerlukan `-lm`; menentukan di mana sebuah variabel terlihat dan berapa lama ia hidup (lokal, global, blok, dan `static`); membangun pembangkit bilangan acak kecil yang hasilnya sama di semua komputer; menulis fungsi rekursif, menelusurinya di atas kertas, dan meramalkan biayanya; serta memecah kasir Profitina Laundry menjadi berkas header, berkas aturan bisnis, `main` yang pendek, dan program uji yang memeriksa setiap aturan terhadap jawaban hitungan tangan.

5.1 Kasir Mbak Rina Sudah Terlalu Panjang

Profitina Laundry adalah usaha cuci pakaian kecil di sebuah gang ramai di Surabaya. Pemiliknya, Pak Hendra, bersama Mbak Rina yang menjaga meja kasir, mencuci, mengeringkan, melipat, dan menyetrika pakaian mahasiswa dari kampus terdekat, keluarga yang mesin cucinya rusak, dan tiga hotel kecil yang mengirim seprai setiap pagi. Keponakan Mbak Rina, Dimas, mahasiswa jurusan informatika, menulis program kasir yang dipakai Mbak Rina di meja depan: ia mengetik jenis layanan dan berat cucian, lalu program mencetak jumlah yang harus dibayar. Program itu lahir di Bab 1, belajar mengulang untuk antrean pelanggan di Bab 2, dan belajar `switch` serta total harian di Bab 3.

Bulan lalu Pak Hendra menyusun ulang daftar harganya. *Cuci lipat* kini Rp7.000 per kilogram, layanan baru *cuci setrika* Rp10.000, *setrika saja* tetap Rp6.000, layanan ekspres menjadi tambahan separuh harga untuk layanan apa pun, dan tagihan minimum turun dari tiga kilogram menjadi dua. Dimas membuka program dan mencari `8000`, harga lama cuci, kering, dan lipat. Ia menemukannya di perhitungan tagihan dan mengubahnya. Yang tidak ia sadari, angka yang sama muncul untuk kedua kalinya di baris yang mencetak daftar harga untuk pelanggan, dan untuk ketiga kalinya di perhitungan total harian. Selama empat hari nota berkata satu hal dan laporan harian berkata lain, dan hitungan uang laci yang teliti setiap malam oleh Mbak Rina-lah yang menemukan selisihnya.

Tidak ada yang aneh dari cerita itu. Siapa pun yang pernah menyimpan buku resep mengenal masalah ini. Resep *nasi goreng* yang baik tidak menjelaskan lagi cara membuat sambal: resep itu cukup menulis “dua sendok sambal (lihat halaman 12)”. Resep sambal ditulis sekali, di halaman

12, dan setiap masakan yang memerlukannya menunjuk ke sana. Ketika juru masak menemukan sambal yang lebih enak, hanya halaman 12 yang berubah, dan semua masakan ikut membaik sekaligus. Program memerlukan kebiasaan yang sama. Aturan "cuci lipat Rp7.000 per kilogram" seharusnya ditulis tepat di satu tempat, dengan sebuah nama, dan setiap bagian program yang memerlukan harga itu cukup bertanya kepada nama tersebut.

Bagian program yang bernama dan dapat dipakai ulang itu disebut **fungsi**. Anda sudah memakai fungsi sejak program pertama: `printf` dan `scanf` adalah fungsi yang ditulis orang lain, dan `main` adalah fungsi yang Anda tulis sendiri. Di bab ini Anda mulai menulis fungsi-fungsi lainnya. Manfaatnya bukan hanya untuk Mbak Rina. Penjahit yang menarik biaya per potong, kios fotokopi yang menarik biaya per lembar, bengkel motor yang menarik biaya per pekerjaan: masing-masing punya segelintir aturan, dan setiap aturan layak mendapat nama dan rumahnya sendiri.

5.2 Gagasannya dengan Kata-Kata Sederhana

5.2.1 Fungsi adalah sepotong pekerjaan yang diberi nama

> **DEFINISI.** *Fungsi* adalah blok kode bernama yang mengerjakan satu tugas. Fungsi dapat menerima nilai masukan yang disebut *parameter*, dan dapat memberikan satu hasil yang disebut *nilai kembalian* (*return value*). Menulis fungsi (kepala dan badannya) disebut *mendefinisikan*; memakainya dengan menulis namanya diikuti tanda kurung disebut *memanggil*. Nilai-nilai yang ditulis di antara tanda kurung pada pemanggilan disebut *argumen*.

Bayangkan fungsi sebagai mesin kecil yang diberi label. Label itu menyebut nama mesin dan apa yang dimasukkan: `bill_for(price_per_kg, kg)`. Anda memasukkan dua bilangan, mesin bekerja, dan keluarlah satu bilangan, yaitu tagihannya. Pemakai mesin tidak perlu melihat isi perutnya; cukup labelnya. Persis begitulah Anda memakai `printf` selama empat bab tanpa pernah membaca kodenya.

Definisi fungsi C terdiri atas empat bagian. Bacalah contoh ini dari kiri ke kanan:

```
long   bill_for   (int price_per_kg, int kg)   { ... }
^^^^   ^^^^^^^^    ^^^^^^^^^^^^^^^^^^^^^^^^   ^^^^^^^
type   name         parameters                 body
```

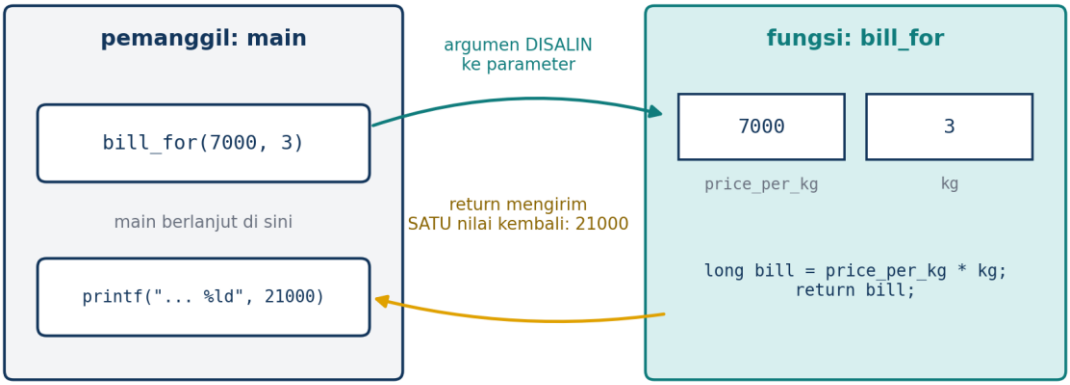
Tipe kembalian (`long`) menyatakan jenis nilai yang keluar. **Nama** mengikuti aturan yang sama dengan nama variabel. **Daftar parameter** mendeklarasikan masukan yang diharapkan, seperti deklarasi variabel biasa yang dipisahkan koma. **Badan** di antara kurung kurawal berisi kode C biasa, dan di suatu tempat di dalamnya pernyataan `return` menentukan nilai mana yang keluar. Fungsi yang tidak mengembalikan apa-apa bertipe kembalian `void`, dan fungsi yang tidak menerima apa-apa berdaftar parameter `(void)`, seperti `main` sejak Bab 1.

5.2.2 Apa yang terjadi selama pemanggilan

Ketika komputer bertemu pemanggilan seperti `bill_for(7000, 3)`, empat hal terjadi berurutan, dan **Gambar 5.1** memperlihatkannya.

1. Argumen dievaluasi: `7000` dan `3` sudah berupa nilai, tetapi argumen boleh berupa ekspresi apa saja, misalnya `price * 2`.
2. Variabel baru dibuat untuk parameter, dan setiap argumen **disalin** ke parameternya: `price_per_kg` menerima 7000 dan `kg` menerima 3.
3. Badan fungsi berjalan. Di sini badan menghitung `7000 * 3` dan tiba di `return bill;`.

4. Nilai kembalian, 21000, berjalan kembali ke tempat pemanggilan, parameter dan variabel lokal fungsi lenyap, dan pemanggil melanjutkan tepat dari tempat ia berhenti, kini dengan 21000 sebagai nilai pemanggilan itu.



Gambar 5.1 Pemanggilan `bill_for(7000, 3)`: argumen disalin ke parameter, badan fungsi berjalan, dan `return` mengirim satu nilai kembali ke tempat pemanggilan, tempat `main` melanjutkan kerjanya.

Langkah 2 layak dibaca dua kali, karena menjelaskan kejutan yang dialami setiap pemula. Parameter menerima *salinan*. Jika fungsi mengubah parameternya, yang berubah adalah salinannya sendiri, dan variabel milik pemanggil tidak tersentuh. Aturan ini disebut **pass-by-value** (pengiriman nilai), dan C memakainya untuk setiap argumen. Bab 7 memperlihatkan bagaimana sebuah fungsi tetap dapat mengubah variabel milik pemanggilnya, yaitu dengan menerima alamat variabel alih-alih nilainya; untuk sekarang, fungsi berbicara kembali kepada pemanggilnya hanya lewat nilai kembalian.

5.2.3 Bagi dan taklukkan: mengapa potongan kecil menang

Mengapa repot-repot? Para pemrogram punya empat jawaban, dan masing-masing layak diingat karena masing-masing adalah pertanyaan yang dapat Anda ajukan kepada kode Anda sendiri.

- **Satu aturan, satu tempat.** Harga sebuah layanan tinggal di satu fungsi. Ketika harga berubah, satu baris yang berubah, dan empat hari laporan keliru seperti yang dialami Mbak Rina tidak mungkin terjadi.
- **Nama menjelaskan.** `charged_kg(weight)` mengatakan apa yang dikerjakannya. Tiga baris di dalamnya tidak mengatakannya sampai Anda membacanya dengan teliti. `main` yang tersusun dari pemanggilan bernama baik terbaca seperti cerita program itu.
- **Potongan dapat diuji sendirian.** Anda dapat memanggil `bill_for` dengan bilangan yang jawabannya sudah dihitung dengan tangan, lalu membandingkan, tanpa mengetik pesanan sehari penuh. Subbab 5.10 melakukan persis hal itu.
- **Potongan dapat dipakai ulang.** Fungsi yang mencetak jumlah uang sebagai `Rp 1.234.567` akan mencetak nota, laporan harian, dan daftar harga, dan Bab 6 akan memakainya lagi tanpa perubahan.

Strategi memotong masalah besar menjadi masalah-masalah kecil, menyelesaikan masing-masing, lalu menggabungkan jawabannya disebut **divide and conquer** (bagi dan taklukkan). Dalam perancangan perangkat lunak, gagasan ini berakar pada *stepwise refinement* (penghalusan bertahap) yang diuraikan Niklaus Wirth [1]: mulailah dari deskripsi satu baris tentang seluruh program, lalu haluskan setiap baris menjadi beberapa langkah lebih kecil sampai setiap langkah mudah ditulis. Kernighan dan Ritchie merumuskan aturan praktisnya dalam satu kalimat di buku

klasik mereka: fungsi memecah tugas komputasi besar menjadi tugas-tugas kecil, dan memungkinkan orang membangun di atas karya orang lain alih-alih memulai dari nol [2].

5.3 Fungsi Pertama Anda

5.3.1 Satu aturan, ditulis sekali

Kode 5.1 adalah program terkecil yang berguna dengan fungsi buatan sendiri. Aturan "tagihan adalah harga per kilogram dikali jumlah kilogram" tinggal di `bill_for`, dan `main` memanggilnya empat kali untuk tiga pelanggan.

Kode 5.1 `first_function.c`: satu aturan tagihan, didefinisikan sekali di atas `main` dan dipanggil empat kali.

```c
/* first_function.c - satu aturan tagihan, ditulis sekali, dipakai tiga kali */
#include <stdio.h>

/* definisi: kepala menamai masukan, badan menghitung jawaban */
long bill_for(int price_per_kg, int kg)
{
    long bill = (long) price_per_kg * kg;
    return bill;
}

int main(void)
{
    printf("Bu Ani     : Rp %ld\n", bill_for(7000, 3));
    printf("Pak Budi   : Rp %ld\n", bill_for(10000, 5));

    long both = bill_for(7000, 4) + bill_for(6000, 2);   /* dua pemanggilan */
    printf("Mbak Citra : Rp %ld\n", both);
    return 0;
}
```

Perhatikan cast `(long) price_per_kg * kg`. Parameternya bertipe `int`, dan di sebagian komputer sebuah `int` paling banyak memuat 32767 (standar C tidak menjamin lebih dari itu), sedangkan di semua komputer desktop masa kini `int` memuat sedikit di atas dua miliar. Hasil kali dua nilai `int` dihitung sebagai `int`, sehingga tanpa cast pesanan besar bisa meluap (*overflow*) sebelum hasilnya sampai ke variabel `long`. Mengubah satu faktor menjadi `long` terlebih dahulu membuat seluruh perkalian berlangsung dalam `long`. Ini pelajaran yang sama dengan cast sebelum pembagian di Bab 2: tipe sebuah operasi ditentukan oleh operannya, bukan oleh variabel yang akan menerima hasilnya.

Kode 5.2 adalah program yang sama dalam Python. Kata kunci `def` memulai definisi, parameter tidak diberi tipe, dan badan fungsi adalah blok yang menjorok di bawah kepalanya.

Kode 5.2 `first_function.py`: aturan yang sama sebagai fungsi Python.

```python
# first_function.py - satu aturan tagihan, ditulis sekali, dipakai tiga kali


# definisi: kepala menamai masukan, badan menghitung jawaban
def bill_for(price_per_kg, kg):
    bill = price_per_kg * kg
    return bill


def main():
    print(f"Bu Ani     : Rp {bill_for(7000, 3)}")
    print(f"Pak Budi   : Rp {bill_for(10000, 5)}")

    both = bill_for(7000, 4) + bill_for(6000, 2)   # dua pemanggilan
    print(f"Mbak Citra : Rp {both}")


if __name__ == "__main__":
    main()
```

Kedua program mencetak tiga baris yang sama:

```
Bu Ani     : Rp 21000
Pak Budi   : Rp 50000
Mbak Citra : Rp 40000
```

Program Python diakhiri dua baris yang akan Anda jumpai di bagian bawah setiap program Python dalam buku ini: `if __name__ == "__main__": main()`. Python menjalankan berkas dari atas ke bawah. Pernyataan `def` hanya *membuat* fungsi; tidak ada yang dipanggil sampai baris terakhir memanggil `main`. Pengujian terhadap `__name__` berarti "hanya jika berkas ini dijalankan sebagai program, bukan diimpor sebagai modul oleh berkas lain". Subbab 5.8 memakai persis hal ini: `cashier.py` mengimpor fungsi-fungsi `laundry.py` tanpa menjalankan apa pun di dalamnya.

5.3.2 Prototipe, fungsi `void`, dan pass-by-value

Di **Kode 5.1** fungsi didefinisikan di atas `main`, sehingga kompilator sudah membaca kepalanya sebelum bertemu pemanggilan pertama. Program yang lebih besar lebih menyukai urutan sebaliknya, `main` lebih dulu, karena `main` adalah daftar isi program. Akibatnya kompilator bertemu pemanggilan fungsi yang belum pernah dilihatnya. C mengatasinya dengan **prototipe**: kepala fungsi diikuti titik koma, diletakkan sebelum pemanggilan pertama. Prototipe adalah janji: "fungsi dengan nama ini, tipe parameter ini, dan tipe kembalian ini akan didefinisikan di suatu tempat". Dengan janji itu kompilator dapat memeriksa setiap pemanggilan, mengubah argumen ke tipe parameter, dan protes ketika jumlah argumen sebuah pemanggilan salah.

Kode 5.3 meletakkan empat prototipe di atas, `main` sesudahnya, dan definisi-definisi di akhir. Program ini juga memperlihatkan dua jenis fungsi baru. `print_rule` tidak mengembalikan apa-apa (`void`) dan ada hanya untuk apa yang dikerjakannya, yaitu mencetak sebaris simbol. `max3` dibangun dari `max2`: fungsi boleh memanggil fungsi lain, dan fungsi kecil yang dipakai ulang di dalam fungsi yang lebih besar adalah wujud sehari-hari dari bagi dan taklukkan.

Kode 5.3 `prototypes.c`: prototipe di atas, `main` lebih dulu, fungsi `void`, fungsi yang dibangun dari fungsi lain, dan peragaan pass-by-value.

```c
/* prototypes.c - prototipe, fungsi void, dan pass-by-value */
#include <stdio.h>

void print_rule(char symbol, int width);    /* prototipe: janji-janjinya */
int max2(int a, int b);
int max3(int a, int b, int c);
long add_express_fee(long bill);

int main(void)
{
    print_rule('=', 32);
    printf("Heaviest bag today: %d kg\n", max3(4, 9, 6));
    print_rule('-', 32);

    long bill = 30000;
    long with_fee = add_express_fee(bill);
    printf("bill in main     : %ld\n", bill);      /* tetap 30000 */
    printf("with express fee : %ld\n", with_fee);
    print_rule('=', 32);
    return 0;
}

void print_rule(char symbol, int width)
{
    for (int i = 0; i < width; i++)
        putchar(symbol);
    putchar('\n');
}

int max2(int a, int b)
{
    return a > b ? a : b;
}

int max3(int a, int b, int c)
{
    return max2(max2(a, b), c);                 /* fungsi yang memakai fungsi */
}

long add_express_fee(long bill)
{
    bill = bill + bill / 2;                     /* hanya mengubah SALINAN */
    printf("bill in function : %ld\n", bill);
    return bill;
}
```

Fungsi terakhir adalah percobaan yang membuktikan pass-by-value. `add_express_fee` menerima tagihan, menambahkan separuhnya, dan mencetak nilai baru parameternya: 45000. Kembali di `main`, variabel `bill` masih berisi 30000, karena fungsi itu hanya pernah memegang salinan. Pemanggil mendapat jumlah baru hanya lewat nilai kembalian, yang disimpan `main` di `with_fee`.

```
================================
Heaviest bag today: 9 kg
--------------------------------
bill in function : 45000
bill in main     : 30000
with express fee : 45000
================================
```

> **PERINGATAN:** Pada C lama, memanggil fungsi tanpa prototipe diizinkan, dan kompilator menebak bahwa fungsi itu mengembalikan `int`. Tebakan itu sering keliru dan kesalahannya aneh-aneh. Sejak C99 bahasa ini tidak lagi mengizinkannya [3], dan `gcc` melaporkannya. Selalu berikan prototipe, atau definisikan fungsi sebelum pemakaian pertamanya.

5.3.3 Cara pandang Python: nama harus ada saat dipanggil

Python tidak mengenal prototipe. **Kode 5.4** mendefinisikan `main` lebih dulu dan fungsi-fungsi pembantu sesudahnya, persis seperti program C, dan program itu jalan, karena Python tidak mencari nama `max3` ketika *membaca* `main`; Python mencarinya ketika `main` *dijalankan*, dan pada saat itu baris terakhir berkas sudah menjalankan setiap `def`. Jika pemanggilan `main()` dipindahkan ke atas berkas, Python akan berhenti dengan `NameError: name 'print_rule' is not defined`.

Kode 5.4 `prototypes.py`: program yang sama dalam Python, tempat fungsi harus ada ketika dipanggil, bukan ketika dibaca.

```
# prototypes.py - Python tak perlu prototipe: nama harus ada saat DIPANGGIL


def main():
    print_rule("=", 32)
    print(f"Heaviest bag today: {max3(4, 9, 6)} kg")
    print_rule("-", 32)

    bill = 30000
    with_fee = add_express_fee(bill)
    print(f"bill in main     : {bill}")        # tetap 30000
    print(f"with express fee : {with_fee}")
    print_rule("=", 32)


def print_rule(symbol, width):
    print(symbol * width)


def max2(a, b):
    return a if a > b else b


def max3(a, b, c):
    return max2(max2(a, b), c)              # fungsi yang memakai fungsi


def add_express_fee(bill):
    bill = bill + bill // 2                 # hanya mengikat ulang nama LOKAL
    print(f"bill in function : {bill}")
    return bill


if __name__ == "__main__":
    main()                                  # semua def di atas sudah dijalankan
```

Keluaran Python sama persis, termasuk bukti pass-by-value. Aturan Python dirumuskan secara berbeda (parameter adalah *nama* baru bagi objek argumen, dan mengisi parameter hanya mengikat ulang nama itu [4]), tetapi untuk bilangan akibatnya sama dengan C: variabel milik pemanggil tidak berubah. Bab 7 kembali ke perbedaan ini ketika hal itu penting, yaitu untuk list.

5.4 Meminjam Fungsi: Pustaka Standar

Tidak ada yang menulis akar kuadrat dari nol. C datang bersama **pustaka standar** (*standard library*), kumpulan fungsi yang diuraikan oleh standar C sendiri [3] dan dikelompokkan menurut pokok bahasan. Untuk memakai sebuah kelompok, Anda menulis `#include` untuk **header**-nya, yaitu berkas berisi prototipe. **Tabel 5.1** memuat header yang paling sering dipakai buku ini.

Tabel 5.1 Header standar yang paling sering dipakai buku ini; header memuat prototipe, pustaka memuat fungsi yang sudah dikompilasi.

Header	Isinya	Contoh
`<stdio.h>`	masukan dan keluaran	`printf`, `scanf`, `putchar`
`<math.h>`	matematika pada `double`	`sqrt`, `pow`, `ceil`, `floor`, `round`, `fabs`
`<stdlib.h>`	utilitas umum	`abs`, `rand`, `srand`, `qsort`, `malloc`
`<string.h>`	string karakter	`strlen`, `strcpy`, `strcmp`
`<ctype.h>`	karakter tunggal	`isdigit`, `toupper`
`<time.h>`	jam dan kalender	`time`, `clock`
`<limits.h>`	batas tipe bilangan bulat	`INT_MAX`, `INT_MIN`
`<stdint.h>`	bilangan bulat berlebar pasti	`uint32_t`, `int64_t`

Profitina Laundry langsung memerlukan satu fungsi matematika. Mbak Rina menghitung **setiap kilogram yang sudah dimulai**: kantong seberat 3,4 kg dihitung 4 kg. Ia juga menarik minimal 2 kg, karena mencuci satu kemeja saja menghabiskan air dan listrik hampir sebanyak mencuci satu kantong kecil. Dalam notasi matematika, dengan w berat dalam kilogram dan $\lceil w \rceil$ *ceiling* (pembulatan ke atas) dari w, yaitu bilangan bulat terkecil yang tidak kurang dari w:

$$k = \max(2,\ \lceil w \rceil) \tag{5.1}$$

Pustaka C menyediakan ceiling sebagai `ceil` di `<math.h>`. **Kode 5.5** mencetak tabel kecil yang membandingkan `floor` (bulatkan ke bawah), `round` (bulatkan ke bilangan bulat terdekat), dan `ceil` (bulatkan ke atas), lalu menerapkan Persamaan (5.1) dalam fungsi `charged_kg`.

Kode 5.5 `rounding.c`: `floor`, `round`, dan `ceil` dari `<math.h>`, serta aturan penagihan laundry sebagai fungsi.

```c
/* rounding.c - meminjam ceil, floor, dan round dari pustaka matematika */
#include <stdio.h>
#include <math.h>

int charged_kg(double weight)
{
    int kg = (int) ceil(weight);          /* tiap kilogram yang dimulai dihitung */
    return kg < 2 ? 2 : kg;               /* tetapi tidak pernah kurang dari 2 kg */
}

int main(void)
{
    double weights[] = {0.8, 2.0, 2.1, 3.5, 4.0, 7.25};
    printf("weight   floor   round    ceil  charged\n");
    for (int i = 0; i < 6; i++) {
        double w = weights[i];
        printf("%6.2f  %6.1f  %6.1f  %6.1f  %5d kg\n",
               w, floor(w), round(w), ceil(w), charged_kg(w));
    }
    printf("sqrt(2) = %.6f, pow(1.1, 12) = %.4f\n", sqrt(2.0), pow(1.1, 12));
    return 0;
}
```

```
weight   floor   round    ceil  charged
  0.80     0.0     1.0     1.0      2 kg
  2.00     2.0     2.0     2.0      2 kg
  2.10     2.0     2.0     3.0      3 kg
  3.50     3.0     4.0     4.0      4 kg
  4.00     4.0     4.0     4.0      4 kg
  7.25     7.0     7.0     8.0      8 kg
sqrt(2) = 1.414214, pow(1.1, 12) = 3.1384
```

Bacalah baris untuk 2.10: `floor` dan `round` sama-sama memberi 2, tetapi pelanggan sudah memulai kilogram ketiga, sehingga `ceil` memberi 3 dan tagihannya 3 kg. Baris untuk 0.80 memperlihatkan batas minimum bekerja: `ceil` memberi 1, dan `charged_kg` menaikkannya menjadi 2.

Di Linux, program yang memakai `<math.h>` harus ditautkan dengan pustaka matematika dengan menambahkan `-lm` di *akhir* baris perintah, karena *linker* (penaut) membaca pustaka secara berurutan dan hanya mengambil dari sebuah pustaka fungsi-fungsi yang diminta berkas sebelumnya [5]:

```
gcc -std=c17 -Wall -Wextra -o rounding rounding.c -lm
```

Di macOS dan di Windows dengan MSYS2, fungsi matematika sudah menjadi bagian pustaka C utama, sehingga `-lm` diterima tetapi tidak diperlukan. Setiap perintah build di buku ini menyertakannya, agar satu perintah berlaku di mana saja.

Kode 5.6 adalah versi Python. Modul `math` milik Python memiliki `floor`, `ceil`, `sqrt`, dan `pow` [6], tetapi fungsi bawaan `round` berbeda dari milik C: bilangan yang tepat berada di tengah dua bilangan bulat dibulatkan ke bilangan *genap*, sehingga `round(2.5)` bernilai 2 dan `round(3.5)` bernilai 4. Aturan ini, yang disebut *banker's rounding* (pembulatan bankir), menghindari kecenderungan kecil ke atas ketika banyak bilangan setengah dibulatkan [7]. `round` milik C selalu membulatkan setengah menjauhi nol. Agar tabelnya tercetak sama persis, program Python mendefinisikan `c_round`, yang bekerja seperti C.

Kode 5.6 `rounding.py`: tabel yang sama dalam Python, dengan fungsi kecil yang membulatkan setengah seperti C.

```python
# rounding.py - meminjam ceil, floor, dan round dari modul math
import math


def charged_kg(weight):
    kg = math.ceil(weight)                  # tiap kilogram yang dimulai dihitung
    return 2 if kg < 2 else kg              # tetapi tidak pernah kurang dari 2 kg


def c_round(x):
    """Bulatkan setengah menjauhi nol, seperti round() C; round() Python tidak."""
    return math.floor(x + 0.5) if x >= 0 else math.ceil(x - 0.5)


def main():
    weights = [0.8, 2.0, 2.1, 3.5, 4.0, 7.25]
    print("weight   floor   round    ceil  charged")
    for w in weights:
        print(f"{w:6.2f}  {math.floor(w):6.1f}  {c_round(w):6.1f}  "
              f"{math.ceil(w):6.1f}  {charged_kg(w):5d} kg")
    print(f"sqrt(2) = {math.sqrt(2.0):.6f}, pow(1.1, 12) = {math.pow(1.1, 12):.4f}")


if __name__ == "__main__":
    main()
```

5.5 Cakupan dan Masa Hidup: Tempat Nama Tinggal

Setiap variabel memiliki dua sifat yang jarang dipikirkan pemula sampai keduanya menimbulkan kesalahan. **Cakupan** (*scope*) adalah bagian teks program tempat namanya dapat dipakai. **Masa hidup** (*lifetime*), yang oleh standar C disebut *storage duration*, adalah rentang waktu berjalan ketika variabel itu ada dan menyimpan nilainya [3]. **Tabel 5.2** merangkum empat jenis yang Anda perlukan.

Tabel 5.2 Cakupan dan masa hidup empat jenis variabel dalam C.

Jenis	Dideklarasikan	Terlihat (cakupan)	Ada (masa hidup)
lokal	di dalam fungsi	dari deklarasi sampai akhir bloknya	sejak blok dimasuki sampai ditinggalkan
parameter	di kepala fungsi	seluruh badan fungsi	selama satu pemanggilan
global	di luar semua fungsi	dari deklarasi sampai akhir berkas	sepanjang program berjalan
lokal `static`	di dalam fungsi, dengan `static`	seperti variabel lokal	sepanjang program berjalan

Kode 5.7 memperlihatkan keempatnya di meja kasir Profitina Laundry. Variabel global `orders_today` terlihat oleh semua fungsi, yang sekaligus nyaman dan berbahaya: fungsi mana pun boleh mengubahnya, sehingga ketika nilainya salah, semua fungsi menjadi tersangka. Variabel lokal `fee` di dalam `take_order` dibuat pada setiap pemanggilan dan dimusnahkan di akhirnya. Blok di dalam `main` mendeklarasikan `kg` kedua yang *menyembunyikan* `kg` di luar sampai kurung kurawal penutup. Dan `next_receipt_number` menyimpan penghitungnya dalam variabel lokal `static`: hanya fungsi ini yang dapat melihatnya, tetapi ia bertahan dari satu pemanggilan ke pemanggilan berikutnya, sehingga nomor nota berjalan 1, 2, 3.

Kode 5.7 `scope.c`: penghitung global, variabel lokal, blok yang menyembunyikan nama, dan penghitung nota `static`.

```c
/* scope.c - di mana sebuah nama terlihat, dan berapa lama variabel hidup */
#include <stdio.h>

int orders_today = 0;                   /* global: terlihat oleh setiap fungsi */

int next_receipt_number(void)
{
    static int last = 0;                /* static: nilainya tetap antarpanggilan */
    last++;
    return last;
}

void take_order(int kg)
{
    int fee = kg * 7000;                /* lokal: ada selama panggilan saja */
    orders_today++;
    printf("receipt #%d: %d kg, Rp %d\n", next_receipt_number(), kg, fee);
}

int main(void)
{
    int kg = 3;
    take_order(kg);
    take_order(5);
    {
        int kg = 10;                    /* kg baru yang menyembunyikan kg di luar */
        take_order(kg);
    }
    printf("kg in main is still %d\n", kg);
    printf("orders today: %d\n", orders_today);
    return 0;
}
```

```
receipt #1: 3 kg, Rp 21000
receipt #2: 5 kg, Rp 35000
receipt #3: 10 kg, Rp 70000
kg in main is still 3
orders today: 3
```

Penghitung nota `static` menggabungkan kebaikan dua dunia dan layak ditiru: nomornya harus bertahan antarpemanggilan, seperti variabel global, tetapi tidak ada fungsi lain yang berhak mengubahnya, sehingga ia disembunyikan di dalam satu-satunya fungsi pemiliknya. Sebaliknya, variabel global sebaiknya disimpan untuk nilai yang benar-benar milik seluruh program, dan itu pun dipakai secukupnya.

Python menarik garis secara berbeda, seperti diperlihatkan **Kode 5.8**. Variabel yang diisi di dalam fungsi bersifat lokal bagi fungsi itu, kecuali fungsi tersebut menyatakannya `global`. Variabel di tingkat teratas berkas bersifat global bagi modul. Tidak ada cakupan blok: variabel yang dibuat di dalam `if` atau perulangan masih terlihat sesudahnya, itulah sebabnya program Python memakai nama lain, `inner_kg`, untuk meniru blok C. Tidak ada pula variabel lokal `static`; penggantinya yang lazim adalah variabel modul berawalan garis bawah, seperti di sini, atau sebuah objek (Bab 14 memperkenalkan kelas dalam C++, dan Python juga memilikinya).

Kode 5.8 `scope.py`: program yang sama dalam Python, dengan deklarasi `global` dan variabel modul yang berperan sebagai `static` milik C.

```python
# scope.py - di mana sebuah nama terlihat, dan berapa lama variabel hidup

orders_today = 0                        # global: terlihat oleh setiap fungsi
_last_receipt = 0                       # berperan sebagai variabel static di C


def next_receipt_number():
    global _last_receipt
    _last_receipt += 1
    return _last_receipt


def take_order(kg):
    global orders_today                 # kita MENGISINYA, jadi nyatakan
    fee = kg * 7000                     # lokal: ada selama panggilan saja
    orders_today += 1
    print(f"receipt #{next_receipt_number()}: {kg} kg, Rp {fee}")


def main():
    kg = 3
    take_order(kg)
    take_order(5)
    inner_kg = 10                       # Python tidak punya cakupan blok
    take_order(inner_kg)
    print(f"kg in main is still {kg}")
    print(f"orders today: {orders_today}")


if __name__ == "__main__":
    main()
```

5.6 Keacakan yang Dapat Diulang

Mbak Rina ingin promosi kecil untuk hari Sabtu: setiap pelanggan dengan tagihan minimal Rp50.000 melempar dua dadu, dan angka kembar memenangkan voucher setrika gratis. Dimas akan membuat dadunya di dalam program, supaya tidak ada perdebatan soal dadu yang jatuh dari meja.

Pustaka C menyediakan `rand()` di `<stdlib.h>`, yang mengembalikan bilangan bulat *pseudo-acak* (*pseudo-random*) antara 0 dan `RAND_MAX`, serta `srand(seed)`, yang memilih titik awal barisan. Kata *pseudo* itu penting: bilangan-bilangannya berasal dari sebuah rumus, sehingga benih (*seed*) yang sama selalu menghasilkan barisan yang sama. Ini justru keunggulan. Program yang bilangan acaknya dapat diulang dapat diuji, dan hasil yang mengejutkan dapat diselidiki.

Ada satu masalah bagi buku yang mencetak keluaran programnya: rumus di balik `rand` tidak ditetapkan oleh standar, sehingga bilangannya berbeda antara Windows, macOS, dan Linux, dan modul `random` milik Python memakai pembangkit lain lagi, yaitu Mersenne Twister [8]. Supaya bilangannya sama di mana pun, kita menulis pembangkit sendiri, dan pembangkit ini sekaligus mengajarkan dua gagasan bab ini: variabel `static` yang menyimpan nilainya antarpemanggilan, dan fungsi dengan *keadaan tersembunyi*. Rumusnya adalah *linear congruential generator* (pembangkit kongruensial linear) yang ditunjukkan oleh standar C sendiri sebagai contoh implementasi `rand` [3], dan dianalisis panjang lebar oleh Knuth [9]:

$$x_{k+1} = (1103515245\, x_k + 12345) \bmod 2^{32} \tag{5.2}$$

Setiap keadaan baru x_{k+1} dihitung dari keadaan sebelumnya x_k; `my_rand` mengembalikan bit ke-16 sampai ke-30 dari keadaan itu, yaitu bilangan dari 0 sampai 32767. Memakai tipe `uint32_t`

dari `<stdint.h>`, bilangan bulat tak bertanda selebar tepat 32 bit, membuat "mod 2^{32}" terjadi dengan sendirinya, karena aritmetika tak bertanda di C berputar kembali pada lebar tipenya. **Kode 5.9** adalah seluruh promosi itu.

Kode 5.9 `lucky_draw.c`: pembangkit bilangan acak yang dapat diulang dengan keadaan `static` tersembunyi, dadu yang dibuat darinya, dan uji keadilan.

```c
/* lucky_draw.c - promosi hari Sabtu dengan bilangan acak buatan sendiri:
   pelanggan dengan tagihan minimal Rp 50.000 melempar dua dadu,
   dan angka kembar memenangkan voucher setrika gratis. */
#include <stdio.h>
#include <stdint.h>

static uint32_t state = 1;              /* keadaan tersembunyi pembangkit */

void my_srand(uint32_t seed)
{
    state = seed;
}

int my_rand(void)                       /* 0 .. 32767, contoh dari ISO C */
{
    state = state * 1103515245u + 12345u;
    return (int) ((state >> 16) & 0x7FFF);
}

int roll_die(void)
{
    return 1 + my_rand() % 6;           /* perkecil 0..32767 menjadi 1..6 */
}

int main(void)
{
    int counts[7] = {0};
    my_srand(2026);
    int winners = 0;
    for (int customer = 1; customer <= 12; customer++) {
        int a = roll_die(), b = roll_die();
        counts[a]++;
        counts[b]++;
        printf("customer %2d rolls %d and %d%s\n", customer, a, b,
               a == b ? "  -> free ironing!" : "");
        if (a == b)
            winners++;
    }
    printf("winners: %d of 12\n", winners);

    for (int i = 0; i < 6000; i++)      /* uji keadilan: 6000 lemparan lagi */
        counts[roll_die()]++;
    for (int face = 1; face <= 6; face++)
        printf("face %d: %d\n", face, counts[face]);
    return 0;
}
```

`roll_die` mengubah bilangan 0 sampai 32767 menjadi mata dadu 1 sampai 6 dengan `1 + my_rand() % 6`: sisa pembagian oleh 6 adalah 0 sampai 5, dan menambah 1 menggesernya menjadi 1 sampai 6. Karena 32768 bukan kelipatan 6, mata 1 dan 2 sedikit sekali lebih mungkin muncul (masing-masing 5462 peluang dari 32768, dibanding 5461 untuk mata lainnya), selisih yang terlalu kecil untuk urusan voucher tetapi penting untuk undian berhadiah besar atau kriptografi, yang memerlukan pembangkit jauh lebih baik.

Versi Python di **Kode 5.10** harus meniru perputaran 32 bit, karena bilangan bulat Python tidak pernah meluap; `& 0xFFFFFFFF` mempertahankan 32 bit terendah, yang tepat sama dengan "mod 2^{32}".

Kode 5.10 `lucky_draw.py`: pembangkit yang sama dalam Python, dengan perputaran 32 bit yang dinyatakan secara eksplisit.

```
# lucky_draw.py - promosi hari Sabtu dengan bilangan acak buatan sendiri:
# pelanggan dengan tagihan minimal Rp 50.000 melempar dua dadu,
# dan angka kembar memenangkan voucher setrika gratis.

_state = 1                              # keadaan tersembunyi pembangkit


def my_srand(seed):
    global _state
    _state = seed & 0xFFFFFFFF


def my_rand():                          # 0 .. 32767, contoh dari ISO C
    global _state
    _state = (_state * 1103515245 + 12345) & 0xFFFFFFFF   # berputar pada 32 bit
    return (_state >> 16) & 0x7FFF


def roll_die():
    return 1 + my_rand() % 6            # perkecil 0..32767 menjadi 1..6


def main():
    counts = [0] * 7
    my_srand(2026)
    winners = 0
    for customer in range(1, 13):
        a = roll_die()
        b = roll_die()
        counts[a] += 1
        counts[b] += 1
        prize = "  -> free ironing!" if a == b else ""
        print(f"customer {customer:2d} rolls {a} and {b}{prize}")
        if a == b:
            winners += 1
    print(f"winners: {winners} of 12")

    for _ in range(6000):               # uji keadilan: 6000 lemparan lagi
        counts[roll_die()] += 1
    for face in range(1, 7):
        print(f"face {face}: {counts[face]}")


if __name__ == "__main__":
    main()
```

Kedua program mencetak, di komputer mana pun:

```
customer  1 rolls 5 and 1
customer  2 rolls 2 and 3
customer  3 rolls 3 and 6
customer  4 rolls 3 and 2
customer  5 rolls 4 and 1
customer  6 rolls 1 and 3
customer  7 rolls 1 and 5
customer  8 rolls 4 and 3
customer  9 rolls 3 and 1
customer 10 rolls 3 and 3  -> free ironing!
customer 11 rolls 4 and 1
customer 12 rolls 4 and 4  -> free ironing!
winners: 2 of 12
face 1: 1014
face 2: 1020
face 3: 969
face 4: 988
face 5: 1020
face 6: 1013
```

Dua dari dua belas pelanggan menang, dekat dengan satu banding enam yang dijanjikan sepasang dadu yang adil, dan seluruh 6024 lemparan jatuh antara 969 dan 1020 kali pada setiap mata, dekat dengan harapan 1004. Untuk hari Sabtu sungguhan, `main` akan memanggil `my_srand((uint32_t) time(NULL))` sebagai ganti `my_srand(2026)`, supaya setiap hari dimulai dari titik yang berbeda; `time` dari `<time.h>` mengembalikan banyaknya detik sejak tahun 1970, benih yang berbeda setiap detik.

5.7 Rekursi: Fungsi yang Memanggil Dirinya Sendiri

5.7.1 Kasus dasar dan masalah yang lebih kecil

Sebuah fungsi boleh memanggil fungsi apa pun, termasuk dirinya sendiri. Fungsi yang berbuat demikian disebut **rekursif**. Awalnya ini terdengar seperti muslihat yang tidak akan pernah berhenti, dan fungsi rekursif yang ceroboh memang tidak pernah berhenti. Fungsi rekursif yang cermat memiliki dua bagian.

> **DEFINISI.** Fungsi *rekursif* menyelesaikan masalah dengan memanggil dirinya sendiri pada contoh yang lebih kecil dari masalah yang sama. Fungsi itu memiliki *kasus dasar* (*base case*), contoh yang cukup kecil untuk langsung dijawab, dan *langkah rekursif*, yang mengubah setiap contoh lain menjadi contoh yang lebih kecil lalu menggabungkan jawabannya.

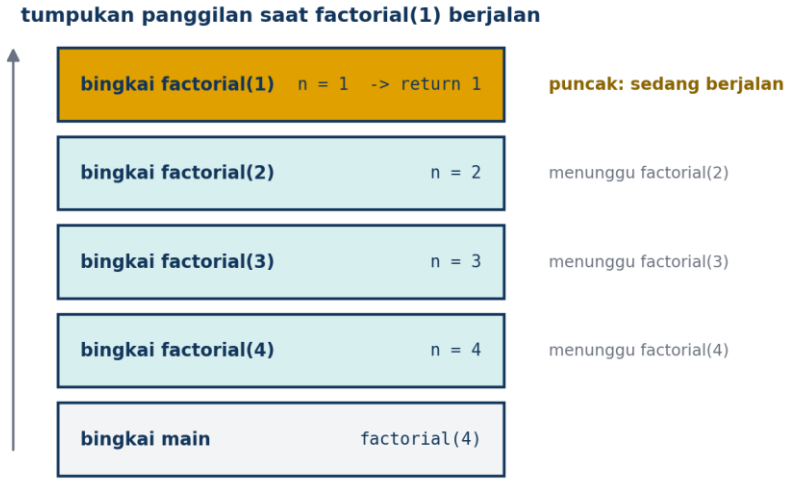
Faktorial adalah contoh pertama yang klasik. Faktorial n, ditulis $n!$, adalah hasil kali $1 \times 2 \times \cdots \times n$, dan $0! = 1$ menurut kesepakatan. Hasil kali n bilangan pertama sama dengan hasil kali $n-1$ bilangan pertama, dikali n:

$$n! = \begin{cases} 1 & \text{jika } n \le 1, \\ n \cdot (n-1)! & \text{jika } n > 1. \end{cases} \tag{5.3}$$

Persamaan (5.3) diterjemahkan ke C baris demi baris, seperti diperlihatkan fungsi pertama **Kode 5.11**. Telusurilah `factorial(4)` dengan tangan. Ia bukan kasus dasar, jadi ia harus menghitung `4 * factorial(3)`, dan tidak dapat menyelesaikan perkaliannya sebelum `factorial(3)` menjawab. `factorial(3)` pada gilirannya menunggu `factorial(2)`, yang menunggu `factorial(1)`. Yang terakhir ini kasus dasar dan langsung mengembalikan 1. Sekarang pemanggilan yang menunggu selesai dalam urutan terbalik: `factorial(2)` mengembalikan

$2 \times 1 = 2$, `factorial(3)` mengembalikan $3 \times 2 = 6$, dan `factorial(4)` mengembalikan $4 \times 6 = 24$.

Di mana pemanggilan-pemanggilan itu menunggu? Setiap pemanggilan mendapat **bingkai** (*frame*, disebut juga *activation record*) sendiri: area memori kecil yang menyimpan parameter, variabel lokal, dan tempat untuk kembali. Bingkai-bingkai itu bertumpuk di **tumpukan panggilan** (*call stack*); pemanggilan menaruh bingkai baru di atas, dan `return` mengangkat bingkai teratas. **Gambar 5.2** memperlihatkan tumpukan pada titik terdalam `factorial(4)`. Empat bingkai dari fungsi yang sama ada pada saat bersamaan, masing-masing dengan `n` miliknya sendiri, dan inilah sebabnya rekursi dapat bekerja: parameter `n` sebuah pemanggilan adalah variabel yang berbeda dari `n` pemanggilan berikutnya.



Gambar 5.2 Tumpukan panggilan pada titik terdalam `factorial(4)`: setiap pemanggilan memiliki bingkai sendiri dengan `n` sendiri, dan setiap bingkai di bawah puncak menunggu jawaban bingkai di atasnya.

Tumpukan itu besar tetapi tidak tak terbatas: beberapa megabita di komputer desktop biasa. Fungsi rekursif tanpa kasus dasar yang benar terus memanggil dirinya sampai tumpukan penuh, dan program berhenti mendadak karena *stack overflow* (di Linux pesannya biasanya `Segmentation fault`). Python melindungi dirinya dengan batas kedalaman rekursi, 1000 secara bawaan, dan berhenti dengan `RecursionError` [10].

5.7.2 Rekursi yang berguna: rupiah bertitik

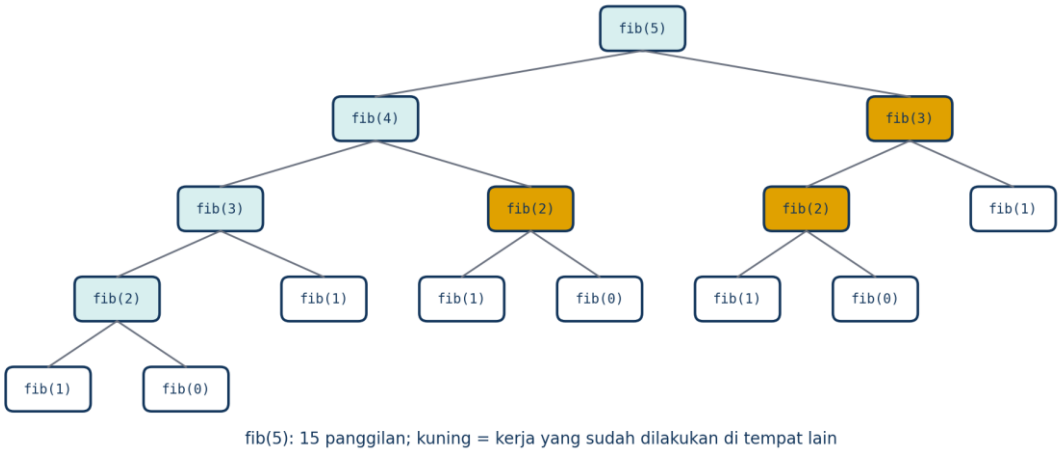
Kita menulis satu juta dua ratus tiga puluh empat ribu lima ratus enam puluh tujuh rupiah sebagai Rp1.234.567, dengan titik di antara setiap kelompok tiga digit. `printf` tidak dapat melakukannya sendiri, tetapi rekursi melakukannya dalam enam baris. Gagasannya: untuk mencetak 1234567, cetak dulu semuanya kecuali tiga digit terakhir (yaitu 1234, masalah sejenis yang lebih kecil), lalu cetak titik dan tiga digit terakhir, 567, selalu tiga digit, sehingga 5007 menjadi 5.007, bukan 5.7. Kasus dasarnya adalah bilangan di bawah 1000, yang dicetak apa adanya. Inilah `print_thousands` di **Kode 5.11**. Perhatikan letak pemanggilan rekursifnya: *sebelum* `printf`. Pemanggilan terdalam mencetak lebih dulu, tepat sesuai urutan munculnya kelompok-kelompok digit.

5.7.3 Harga rekursi: Fibonacci

Bilangan Fibonacci dimulai dengan 0 dan 1, dan setiap bilangan berikutnya adalah jumlah dua bilangan sebelumnya: 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, … Definisinya sendiri rekursif,

$$F(0) = 0, \quad F(1) = 1, \quad F(n) = F(n-1) + F(n-2) \text{ untuk } n \geq 2, \tag{5.4}$$

dan Persamaan (5.4) diterjemahkan menjadi fungsi tiga baris `fib` di **Kode 5.11**. Fungsi itu benar, dan sekaligus pelajaran tentang biaya. **Gambar 5.3** menggambar setiap pemanggilan yang dilakukan `fib(5)`. Akar memanggil `fib(4)` dan `fib(3)`; `fib(4)` memanggil `fib(3)` lagi, dan seluruh subpohon di bawahnya dihitung dua kali. Kotak kuning adalah pemanggilan yang jawabannya sudah pernah dihitung di tempat lain dalam pohon.



Gambar 5.3 Setiap pemanggilan oleh `fib(5)`: lima belas pemanggilan untuk menghitung satu bilangan, dan pemanggilan berwarna kuning mengulang pekerjaan yang sudah dilakukan di tempat lain dalam pohon.

Misalkan $C(n)$ adalah banyaknya pemanggilan yang dilakukan `fib(n)`, termasuk dirinya sendiri. Untuk $n < 2$ jumlahnya satu pemanggilan. Selain itu, jumlahnya satu pemanggilan ditambah semua pemanggilan dari dua pemanggilan rekursifnya:

$$C(n) = C(n-1) + C(n-2) + 1, \qquad C(0) = C(1) = 1. \tag{5.5}$$

Dari Persamaan (5.5): $C(2) = 3$, $C(3) = 5$, $C(4) = 9$, $C(5) = 15$, cocok dengan lima belas kotak di **Gambar 5.3**. Menambah 1 pada kedua ruas memperlihatkan bahwa $C(n) + 1$ sendiri mematuhi aturan Fibonacci, dan menghitung beberapa nilai pertamanya memberikan $C(n) = 2F(n+1) - 1$. Jadi banyaknya pemanggilan tumbuh secepat bilangan Fibonacci, yang tumbuh dengan faktor sekitar 1,618 pada setiap langkah [11]. Setiap kenaikan n sebesar 5 mengalikan pekerjaan dengan sekitar $1.618^5 \approx 11$. Program menghitung pemanggilan dalam variabel global untuk memperlihatkannya.

Kode 5.11 `recursion.c`: faktorial, rupiah bertitik, dan Fibonacci dengan penghitung pemanggilan.

```c
/* recursion.c - fungsi yang memanggil dirinya sendiri */
#include <stdio.h>

long factorial(int n)
{
    if (n <= 1)                         /* kasus dasar: berhenti di sini */
        return 1;
    return n * factorial(n - 1);        /* langkah rekursif: masalah lebih kecil */
}

void print_thousands(long amount)       /* 1234567 -> 1.234.567 */
{
    if (amount < 1000) {
        printf("%ld", amount);
        return;
    }
    print_thousands(amount / 1000);     /* dulu semua kecuali 3 digit akhir */
    printf(".%03ld", amount % 1000);    /* lalu tiga digit terakhir */
}

long calls = 0;                         /* berapa kali fib dimasuki */

long fib(int n)
{
    calls++;
    if (n < 2)
        return n;
    return fib(n - 1) + fib(n - 2);
}

int main(void)
{
    for (int n = 0; n <= 5; n++)
        printf("%d! = %ld\n", n, factorial(n));

    long amounts[] = {7, 28000, 187500, 1234567};
    for (int i = 0; i < 4; i++) {
        printf("Rp ");
        print_thousands(amounts[i]);
        printf("\n");
    }
```

Kode 5.11 (Lanjutan) *recursion.c: faktorial, rupiah bertitik, dan Fibonacci dengan penghitung pemanggilan.*

```c
    int ns[] = {5, 10, 20, 25, 30};
    for (int i = 0; i < 5; i++) {
        calls = 0;
        long f = fib(ns[i]);
        printf("fib(%2d) = %6ld after %8ld calls\n", ns[i], f, calls);
    }
    return 0;
}
```

Kode 5.12 adalah versi Python. `thousands` mengembalikan string alih-alih mencetak, karena menyambung string itu mudah di Python; rekursinya sama.

Kode 5.12 `recursion.py`: tiga rekursi yang sama dalam Python.

```python
# recursion.py - fungsi yang memanggil dirinya sendiri


def factorial(n):
    if n <= 1:                          # kasus dasar: berhenti di sini
        return 1
    return n * factorial(n - 1)         # langkah rekursif: masalah lebih kecil


def thousands(amount):                  # 1234567 -> "1.234.567"
    if amount < 1000:
        return str(amount)
    return thousands(amount // 1000) + f".{amount % 1000:03d}"


calls = 0                               # berapa kali fib dimasuki


def fib(n):
    global calls
    calls += 1
    if n < 2:
        return n
    return fib(n - 1) + fib(n - 2)


def main():
    global calls
    for n in range(6):
        print(f"{n}! = {factorial(n)}")

    for amount in [7, 28000, 187500, 1234567]:
        print("Rp " + thousands(amount))

    for n in [5, 10, 20, 25, 30]:
        calls = 0
        f = fib(n)
        print(f"fib({n:2d}) = {f:6d} after {calls:8d} calls")


if __name__ == "__main__":
    main()
```

```
0! = 1
1! = 1
2! = 2
3! = 6
4! = 24
5! = 120
Rp 7
Rp 28.000
Rp 187.500
Rp 1.234.567
fib( 5) =      5 after       15 calls
fib(10) =     55 after      177 calls
fib(20) =   6765 after    21891 calls
fib(25) =  75025 after   242785 calls
fib(30) = 832040 after  2692537 calls
```

Lima baris terakhir adalah pelajarannya. `fib(30)` melakukan 2.692.537 pemanggilan untuk menghitung 832040, dan setiap baris kira-kira sebelas kali baris sebelumnya, seperti diramalkan Persamaan (5.5). Perulangan sederhana yang menyimpan dua bilangan terakhir menghitung `fib(30)` dalam 30 langkah. Bab 8 mengadu kedua cara itu dengan stopwatch.

5.7.4 Rekursi atau perulangan?

Setiap fungsi rekursif dapat ditulis ulang dengan perulangan, dan setiap perulangan sebagai rekursi. Pilihlah berdasarkan kejelasan dan biaya. Faktorial enak dibaca dengan kedua cara, dan perulangan menghemat tumpukan. Pencetak rupiah lebih jelas secara rekursif, karena masalahnya sendiri berbunyi "sisanya, lalu kelompok terakhir". Fibonacci yang ditulis sebagai salinan langsung definisinya adalah jebakan: jelas, tetapi lambat secara eksponensial. Masalah yang datanya sendiri bersarang, seperti folder di dalam disk atau Menara Hanoi di Soal Latihan 5.6, adalah tempat rekursi paling unggul, karena versi perulangannya harus mengelola tumpukannya sendiri secara manual. Abelson dan Sussman menyebut pola Fibonacci *tree recursion* (rekursi pohon) dan memakainya untuk peringatan yang sama: proses yang bercabang dua pada setiap langkah mengulang sebagian besar pekerjaannya [12].

5.8 Membangun Kasir Versi 4

5.8.1 Satu berkas aturan, satu berkas cerita

Sekarang kita membangun ulang kasir Mbak Rina dengan semua bahan di atas. Aturan bisnis pindah ke berkas sumbernya sendiri, `laundry.c`, dan prototipenya ke **berkas header**, `laundry.h`. Setiap program yang memerlukan aturan itu cukup menyertakan header-nya: kasir, laporan harian Bab 6, ringkasan tutup toko Bab 7, dan program uji bab ini. **Kode 5.13** adalah header-nya. Dua baris pertama dan baris terakhirnya adalah *include guard* (penjaga penyertaan): jika sebuah berkas tak sengaja disertakan dua kali, salinan kedua dilewati, karena `LAUNDRY_H` sudah terdefinisi.

Kode 5.13 `laundry.h`: janji-janji modul aturan bisnis, beserta konstanta dan include guard.

```
/* laundry.h - aturan bisnis Profitina Laundry: janji-janjinya (Bab 5) */
#ifndef LAUNDRY_H
#define LAUNDRY_H

#define MIN_KG 2
#define MAX_KG 50
#define EXPRESS_PERCENT 50

int price_per_kg(int service);          /* rupiah per kg; 0 = layanan tak dikenal */
void print_service(int service);        /* nama, dipadatkan menjadi 12 karakter */
int charged_kg(double weight);          /* kilogram yang dimulai, minimal MIN_KG */
long bill_for(int service, double weight, int express);
void print_rupiah(long amount);         /* 1234567 -> Rp 1.234.567 */
int next_receipt_number(void);          /* 1, 2, 3, ... */

#endif
```

Konstanta ditulis dengan `#define`. **Praprosesor**, yang berjalan sebelum kompilator, mengganti setiap `MIN_KG` dalam program dengan `2`. Tarif layanan ekspres, batas minimum, dan kantong terbesar yang masuk akal masing-masing punya satu rumah dan satu nama.

Kode 5.14 menepati janji-janji itu. Setiap fungsi mengerjakan satu hal. `price_per_kg` mengetahui daftar harga, dan nilai 0 berarti "layanan tak dikenal". `print_service` mengetahui nama-namanya. `charged_kg` menerapkan Persamaan (5.1). `bill_for` menggabungkan harga dan kilogram yang ditagih lalu menambahkan biaya ekspres. `print_rupiah` mencetak jumlah uang bertitik dengan rekursi Subbab 5.7.2, yang fungsi pembantunya ditandai `static` di tingkat berkas: di depan sebuah fungsi, `static` berarti *hanya milik berkas ini*, sehingga tidak ada berkas lain yang

dapat memanggil `print_thousands` atau bertabrakan dengan namanya. `next_receipt_number` adalah penghitung `static` dari Subbab 5.5.

Kode 5.14 `laundry.c`: aturan bisnis Profitina Laundry, satu fungsi untuk setiap aturan.

```c
/* laundry.c - aturan bisnis Profitina Laundry: satu fungsi per aturan */
#include <stdio.h>
#include <math.h>
#include "laundry.h"

int price_per_kg(int service)
{
    switch (service) {
    case 1: return 7000;                /* cuci lipat */
    case 2: return 10000;               /* cuci setrika */
    case 3: return 6000;                /* setrika saja */
    default: return 0;
    }
}

void print_service(int service)
{
    switch (service) {
    case 1: printf("%-12s", "Wash & fold"); break;
    case 2: printf("%-12s", "Wash & iron"); break;
    case 3: printf("%-12s", "Iron only"); break;
    default: printf("%-12s", "Unknown"); break;
    }
}

int charged_kg(double weight)
{
    int kg = (int) ceil(weight);
    return kg < MIN_KG ? MIN_KG : kg;
}

long bill_for(int service, double weight, int express)
{
    long bill = (long) price_per_kg(service) * charged_kg(weight);
    if (express)
        bill += bill * EXPRESS_PERCENT / 100;
    return bill;
}
```

Kode 5.14 (Lanjutan) *laundry.c: aturan bisnis Profitina Laundry, satu fungsi untuk setiap aturan.*

```c
static void print_thousands(long amount)   /* static: hanya milik berkas ini */
{
    if (amount < 1000) {
        printf("%ld", amount);
        return;
    }
    print_thousands(amount / 1000);
    printf(".%03ld", amount % 1000);
}

void print_rupiah(long amount)
{
    printf("Rp ");
    print_thousands(amount);
}

int next_receipt_number(void)
{
    static int last = 0;
    return ++last;
}
```

Dengan aturan sudah di tempatnya, kasir di **Kode 5.15** cukup pendek untuk dibaca sebagai cerita: selama ada pesanan, baca, periksa, hitung tagihan, cetak nota, dan tambahkan ke total hari itu. Dua fungsi pembantu tinggal di berkas ini karena keduanya milik meja kasir, bukan milik bisnis: `order_is_valid` menjelaskan mengapa sebuah pesanan ditolak, dan `print_receipt` menata satu baris nota.

Kode 5.15 `cashier.c`: kasir versi 4, yang `main`-nya bercerita dan menyerahkan pekerjaan kepada fungsi-fungsi bernama.

```c
/* cashier.c - kasir Profitina Laundry versi 4: main bercerita,
   fungsi-fungsi di laundry.c yang bekerja.
   Masukan: satu pesanan per baris, "service weight express", diakhiri 0. */
#include <stdio.h>
#include "laundry.h"

int order_is_valid(int service, double weight)
{
    if (price_per_kg(service) == 0) {
        printf("skipped: unknown service %d\n", service);
        return 0;
    }
    if (weight <= 0 || weight > MAX_KG) {
        printf("skipped: %.1f kg is not a laundry bag\n", weight);
        return 0;
    }
    return 1;
}

void print_receipt(int service, double weight, int express, long bill)
{
    printf("#%03d  ", next_receipt_number());
    print_service(service);
    printf(" %5.1f kg -> %2d kg", weight, charged_kg(weight));
    printf(express ? "  express  " : "           ");
    print_rupiah(bill);
    printf("\n");
}

int main(void)
{
    int service, express, orders = 0;
    double weight;
    long revenue = 0;

    printf("PROFITINA LAUNDRY - cashier v4\n");
    while (scanf("%d", &service) == 1 && service != 0) {
        if (scanf("%lf %d", &weight, &express) != 2)
            break;
        if (!order_is_valid(service, weight))
            continue;
        long bill = bill_for(service, weight, express);
        print_receipt(service, weight, express, bill);
        orders++;
        revenue += bill;
    }
```

Kode 5.15 (Lanjutan) *cashier.c: kasir versi 4, yang main-nya bercerita dan menyerahkan pekerjaan kepada fungsi-fungsi bernama.*

```c
    printf("orders: %d, revenue: ", orders);
    print_rupiah(revenue);
    printf("\n");
    return 0;
}
```

Syarat perulangan `scanf("%d", &service) == 1 && service != 0` berhenti di akhir masukan dan pada kode layanan 0, yang dipakai kasir untuk menutup hari. `continue` di dalam perulangan melewati sisa badan perulangan untuk pesanan yang tidak sah, lalu pesanan berikutnya dibaca.

Modul Python **Kode 5.16** memuat aturan yang sama. Daftar harga dan nama-nama layanan menjadi *dictionary* (kamus), tabel yang memetakan kunci ke nilai, dan `.get(service, 0)` menjawab 0 untuk kunci yang tidak dikenal, kesepakatan yang sama dengan C. Di tempat fungsi C

mencetak, fungsi Python mengembalikan string, dan pemanggilnya yang mencetak; kedua rancangan sama-sama jalan, dan Bab 9 memperlihatkan bagaimana C juga membangun string.

Kode 5.16 `laundry.py`: aturan bisnis sebagai modul Python.

```
# laundry.py - aturan bisnis Profitina Laundry: satu fungsi per aturan
import math

MIN_KG = 2
MAX_KG = 50
EXPRESS_PERCENT = 50

_PRICES = {1: 7000, 2: 10000, 3: 6000}   # cuci lipat, cuci setrika, setrika saja
_NAMES = {1: "Wash & fold", 2: "Wash & iron", 3: "Iron only"}
_last_receipt = 0


def price_per_kg(service):
    return _PRICES.get(service, 0)       # 0 = layanan tak dikenal


def service_name(service):
    return f"{_NAMES.get(service, 'Unknown'):<12}"


def charged_kg(weight):
    kg = math.ceil(weight)
    return MIN_KG if kg < MIN_KG else kg


def bill_for(service, weight, express):
    bill = price_per_kg(service) * charged_kg(weight)
    if express:
        bill += bill * EXPRESS_PERCENT // 100
    return bill


def _thousands(amount):                  # awalan _: hanya milik modul ini
    if amount < 1000:
        return str(amount)
    return _thousands(amount // 1000) + f".{amount % 1000:03d}"


def rupiah(amount):
    return "Rp " + _thousands(amount)


def next_receipt_number():
    global _last_receipt
    _last_receipt += 1
    return _last_receipt
```

Kode 5.17 adalah kasir Python. Program ini membaca seluruh masukan sekaligus dengan `sys.stdin.read().split()`, yang menghasilkan list kata-kata, lalu menelusuri list itu tiga kata sekali jalan.

Kode 5.17 `cashier.py`: kasir versi 4 dalam Python, yang mengimpor aturan dari `laundry.py`.

```python
# cashier.py - kasir Profitina Laundry versi 4: main bercerita,
# fungsi-fungsi di laundry.py yang bekerja.
# Masukan: satu pesanan per baris, "service weight express", diakhiri 0.
import sys
from laundry import (MAX_KG, price_per_kg, service_name, charged_kg,
                     bill_for, rupiah, next_receipt_number)


def order_is_valid(service, weight):
    if price_per_kg(service) == 0:
        print(f"skipped: unknown service {service}")
        return False
    if weight <= 0 or weight > MAX_KG:
        print(f"skipped: {weight:.1f} kg is not a laundry bag")
        return False
    return True


def print_receipt(service, weight, express, bill):
    tag = "  express  " if express else "           "
    print(f"#{next_receipt_number():03d}  {service_name(service)}"
          f" {weight:5.1f} kg -> {charged_kg(weight):2d} kg{tag}{rupiah(bill)}")


def main():
    words = sys.stdin.read().split()
    pos, orders, revenue = 0, 0, 0
    print("PROFITINA LAUNDRY - cashier v4")
    while pos < len(words) and int(words[pos]) != 0:
        if pos + 2 >= len(words):
            break
        service = int(words[pos])
        weight = float(words[pos + 1])
        express = int(words[pos + 2])
        pos += 3
        if not order_is_valid(service, weight):
            continue
        bill = bill_for(service, weight, express)
        print_receipt(service, weight, express, bill)
        orders += 1
        revenue += bill
    print(f"orders: {orders}, revenue: {rupiah(revenue)}")


if __name__ == "__main__":
    main()
```

5.8.2 Mengompilasi beberapa berkas

Program yang terdiri atas beberapa berkas dikompilasi dengan menyebut semua berkas `.c`-nya; header tidak disebut, karena baris `#include` sudah membawanya masuk:

```
gcc -std=c17 -Wall -Wextra -o cashier cashier.c laundry.c -lm
```

Di balik satu perintah ini, `gcc` mengompilasi setiap berkas `.c` secara terpisah menjadi *berkas objek*, lalu **linker** menggabungkan berkas-berkas objek dan pustaka menjadi satu program, mencocokkan setiap pemanggilan di `cashier.c` dengan definisinya di `laundry.c`. Jika sebuah fungsi dijanjikan di header tetapi tidak pernah didefinisikan, kompilator puas dan linker protes dengan `undefined reference to ...`; pesan itu selalu berarti "ada janji yang tidak ditepati". Di VS Code, tugas build dari Bab 1 hanya mengompilasi berkas yang sedang terbuka, sehingga program yang terdiri atas beberapa berkas dibangun dari terminal terpadu (**Terminal > New Terminal**) dengan perintah di atas.

Untuk Python tidak ada yang berubah: `python3 cashier.py` menemukan `laundry.py` di folder yang sama dan mengimpornya.

5.9 Menjalankannya

Berikut satu pagi sungguhan di meja kasir, seperti yang diketik Mbak Rina: satu pesanan per baris berisi layanan, berat, dan 1 untuk ekspres atau 0 untuk biasa, lalu 0 untuk menutup.

```
1 3.4 0
2 1.2 1
3 5 0
7 2 0
2 6.5 0
1 0 0
1 12.75 1
0
```

```
PROFITINA LAUNDRY - cashier v4
#001  Wash & fold    3.4 kg ->  4 kg           Rp 28.000
#002  Wash & iron    1.2 kg ->  2 kg  express  Rp 30.000
#003  Iron only      5.0 kg ->  5 kg           Rp 30.000
skipped: unknown service 7
#004  Wash & iron    6.5 kg ->  7 kg           Rp 70.000
skipped: 0.0 kg is not a laundry bag
#005  Wash & fold   12.8 kg -> 13 kg  express  Rp 136.500
orders: 5, revenue: Rp 294.500
```

Periksalah baris-barisnya dengan tangan, karena kebiasaan itulah yang akan menangkap empat hari laporan keliru tadi. Kantong pertama beratnya 3,4 kg, jadi empat kilogram ditagih dengan harga Rp7.000: Rp28.000. Kantong kedua 1,2 kg cuci setrika ekspres: minimum 2 kg dengan harga Rp10.000 memberi Rp20.000, ditambah separuhnya untuk ekspres menjadi Rp30.000. Layanan 7 tidak ada dan ditolak beserta alasannya, begitu pula kantong 0 kg, barangkali salah ketik. Pesanan terakhir, 12,75 kg cuci lipat ekspres, tercetak sebagai 12.8 kg karena `%5.1f` menampilkan satu desimal; tagihannya 13 kg, Rp91.000, ditambah separuhnya, Rp136.500. Lima pesanan sah menghasilkan Rp294.500.

5.10 Mengujinya

Program uji memanggil setiap fungsi dengan masukan yang jawabannya sudah dihitung dengan tangan, lalu membandingkan. **Kode 5.18** memeriksa dua belas fakta seperti itu. Fungsi pembantu `check` menghitung banyaknya pemeriksaan dan hanya melaporkan kegagalan, lengkap dengan nomor baris pemeriksaan yang gagal, yang disediakan makro bawaan `__LINE__`.

Kode 5.18 `test_laundry.c`: dua belas pemeriksaan aturan bisnis terhadap jawaban hitungan tangan.

```c
/* test_laundry.c - menguji setiap aturan bisnis terhadap jawaban hitungan tangan */
#include <stdio.h>
#include "laundry.h"

static int checks = 0, failures = 0;

static void check(long got, long expected, int line)
{
    checks++;
    if (got != expected) {
        failures++;
        printf("FAILED on line %d: got %ld, expected %ld\n", line, got, expected);
    }
}

int main(void)
{
    check(price_per_kg(1), 7000, __LINE__);
    check(price_per_kg(2), 10000, __LINE__);
    check(price_per_kg(3), 6000, __LINE__);
    check(price_per_kg(9), 0, __LINE__);
    check(charged_kg(0.8), 2, __LINE__);      /* batas minimum */
    check(charged_kg(2.0), 2, __LINE__);      /* tepat 2 tetap 2 */
    check(charged_kg(2.1), 3, __LINE__);      /* kilogram yang sudah dimulai */
    check(bill_for(1, 3.4, 0), 28000, __LINE__);
    check(bill_for(2, 1.2, 1), 30000, __LINE__);
    check(bill_for(3, 5.0, 1), 45000, __LINE__);
    check(next_receipt_number(), 1, __LINE__);
    check(next_receipt_number(), 2, __LINE__);
    printf("%d checks, %d failures\n", checks, failures);
    return failures != 0;
}
```

Kasus tepi (*edge case*) adalah pemeriksaan yang paling berharga: 0,8 kg harus menjadi minimum 2 kg, tepat 2,0 kg harus tetap 2 (aturan ceroboh seperti “kilogram bulat ditambah satu” akan menagih 3), dan 2,1 kg harus menjadi 3. Penghitung nota harus mulai dari 1 dan naik. Program mengembalikan 1 kepada sistem operasi ketika ada pemeriksaan yang gagal, sehingga sebuah skrip dapat mendeteksi kegagalan secara otomatis; `run_tests.py` di folder kode bab ini mengompilasi dan menjalankan setiap program bab ini dengan cara tersebut.

Kode 5.19 adalah uji yang sama dalam Python, ditambah satu pemeriksaan yang sulit dilakukan versi C: bahwa `rupiah(1234567)` adalah teks `Rp 1.234.567`.

Kode 5.19 `test_laundry.py`: pemeriksaan yang sama dalam Python, ditambah satu untuk teks rupiah.

```python
# test_laundry.py - menguji setiap aturan bisnis terhadap jawaban hitungan tangan
import inspect
import sys
from laundry import (price_per_kg, charged_kg, bill_for, rupiah,
                     next_receipt_number)

checks = 0
failures = 0


def check(got, expected):
    global checks, failures
    checks += 1
    if got != expected:
        failures += 1
        line = inspect.stack()[1].lineno
        print(f"FAILED on line {line}: got {got}, expected {expected}")


def main():
    check(price_per_kg(1), 7000)
    check(price_per_kg(2), 10000)
    check(price_per_kg(3), 6000)
    check(price_per_kg(9), 0)
    check(charged_kg(0.8), 2)            # batas minimum
    check(charged_kg(2.0), 2)            # tepat 2 tetap 2
    check(charged_kg(2.1), 3)            # kilogram yang sudah dimulai
    check(bill_for(1, 3.4, 0), 28000)
    check(bill_for(2, 1.2, 1), 30000)
    check(bill_for(3, 5.0, 1), 45000)
    check(next_receipt_number(), 1)
    check(next_receipt_number(), 2)
    if rupiah(1234567) != "Rp 1.234.567":  # Python bisa menguji teks juga
        print("FAILED: rupiah(1234567)")
    print(f"{checks} checks, {failures} failures")
    return 1 if failures else 0


if __name__ == "__main__":
    sys.exit(main())
```

```
12 checks, 0 failures
```

Dua belas pemeriksaan tanpa kegagalan. Tulislah uji sebelum Anda mengubah sebuah aturan, jalankan sesudahnya, dan dalam sedetik Anda akan tahu apakah perubahan itu merusak sesuatu. Python memiliki kerangka pengujian lengkap, `unittest`, di pustaka standarnya [13]; kita memakai fungsi `check` yang kecil agar uji C dan Python tampak serupa dan mencetak baris yang sama.

5.11 Membuatnya Lebih Baik

Nilai pemecahan ini terlihat saat pertama kali Mbak Rina berubah pikiran. Misalkan ia menambah layanan keempat, *bed cover*, dengan harga Rp15.000 per kilogram. Di versi 3, daftar harga, nama layanan, dan pemeriksaan keabsahan tersebar di `main`. Sekarang tepat dua fungsi yang berubah, `price_per_kg` dan `print_service`, masing-masing dengan satu baris `case`, dan kasir, pemeriksaan `order_is_valid`, serta setiap laporan di masa depan langsung mengenali layanan baru tanpa satu suntingan pun, karena semuanya bertanya kepada `price_per_kg`. Tambahkan satu baris ke program uji, `check(price_per_kg(4), 15000, __LINE__);`, lalu jalankan.

Misalkan sebaliknya tarif ekspres turun dari 50% menjadi 40%. `EXPRESS_PERCENT` berubah di header, `bill_for` memakainya, dan uji `check(bill_for(2, 1.2, 1), 30000, ...)` kini gagal dengan `got 28000, expected 30000`. Kegagalan itu kabar baik: uji telah memperhatikan perubahan, dan Anda memperbarui jawaban yang diharapkan dengan sadar, alih-alih mengetahui harga baru dari pelanggan yang marah.

Dua kebiasaan kecil membuat fungsi seperti ini lebih baik lagi. Pertama, jauhkan keluaran dari aturan: `bill_for` menghitung dan mengembalikan, tidak pernah mencetak. Fungsi yang menghitung sekaligus mencetak tidak dapat dipakai ulang di tempat yang tidak menginginkan cetakan, seperti laporan harian Bab 6. Kedua, berikan setiap fungsi komentar yang menyatakan janjinya, seperti di `laundry.h`, termasuk apa yang dikembalikannya untuk masukan yang buruk (`0 = unknown service`). Pembaca header tidak boleh sampai perlu membuka berkas `.c` untuk memakai sebuah fungsi dengan benar.

5.12 Memakainya Sungguhan

Kasir ini milik Mbak Rina, tetapi bentuknya cocok untuk hampir setiap usaha kecil yang menagih berdasarkan aturan. Untuk menyesuaikannya, ubahlah aturannya, bukan ceritanya.

- **Penjahit** menagih per potong dan per jenis pekerjaan: memotong kelim celana Rp15.000, mengecilkan kemeja Rp25.000, kebaya baru sesuai penawaran. `price_per_kg` menjadi `price_of(job)`, dan `charged_kg` hilang atau berubah menjadi aturan untuk pekerjaan kilat.
- **Kios fotokopi** dekat kampus menagih per lembar, lebih murah untuk hitam-putih daripada berwarna, dengan potongan di atas 100 lembar. Potongan itu satu fungsi lagi, `discount_for(pages)`, yang dipakai `bill_for`; pencetak nota tidak berubah.
- **Bengkel motor** menagih ongkos servis ditambah suku cadang. Baris pesanan mendapat bilangan keempat, harga suku cadang, dan `bill_for` menambahkannya; selebihnya tetap.
- **Dapur katering** menjual kotak nasi per lusin: tagihan minimum dan pembulatan ke atas `charged_kg` menjadi "minimal 10 kotak, dengan kelipatan 5", bentuk yang sama dengan Persamaan (5.1) dengan langkah berbeda.

Dalam setiap kasus, tulislah aturan lebih dulu sebagai header berisi prototipe dengan janji satu baris untuk masing-masing, tulislah program uji dengan jawaban yang dihitung pemilik usaha di atas kertas, dan baru sesudah itu tulislah `main`. Pemilik usaha jauh lebih mudah membaca dan memeriksa uji daripada kode.

5.13 Kesalahan yang Sering Terjadi, dan Cara Menghindarinya

- **Mengira fungsi mengubah argumennya.** `add_express_fee(bill);` saja tidak berbuat apa-apa terhadap `bill`; hasilnya dikembalikan lalu dibuang. Tulislah `bill = add_express_fee(bill);`, atau tunggu pointer di Bab 7.
- **Lupa `return`.** Fungsi non-`void` yang sampai di kurung kurawal penutup tanpa `return` mengembalikan nilai yang tidak terdefinisi. `gcc -Wall` memperingatkan: *control reaches end of non-void function*. Jangan pernah mengabaikan peringatan itu.
- **Memanggil tanpa prototipe.** Galat *implicit declaration of function* berarti ada pemanggilan sebelum definisi dan tidak ada prototipe. Tambahkan prototipe atau sertakan header yang tepat (misalnya `<math.h>` untuk `sqrt`).
- **Lupa `-lm`.** *undefined reference to `ceil`* di Linux berarti pustaka matematika belum ditautkan. Tambahkan `-lm` di akhir perintah.

- **Luapan bilangan bulat di dalam perkalian.** `price * kg` dengan dua nilai `int` dihitung sebagai `int`, apa pun tipe variabel yang menerimanya. Cast salah satu faktornya, seperti `(long) price * kg`.
- **Rekursi tanpa kasus dasar yang dapat dicapai.** `factorial(-1)` dengan pengujian `n == 1` alih-alih `n <= 1` tidak pernah mencapai kasus dasar dan berakhir dengan stack overflow. Pastikan kasus dasar menangkap setiap masukan yang kecil atau janggal.
- **Rekursi pohon untuk masukan besar.** Rekursi yang memanggil dirinya dua kali pada masukan yang hanya sedikit lebih kecil, seperti `fib`, mengulang pekerjaan dalam jumlah eksponensial. Hitunglah pemanggilannya, seperti di **Kode 5.11**, sebelum mempercayakannya pada $n = 50$.
- **`round` Python bukan `round` C.** `round(2.5)` bernilai 2 di Python dan `round(2.5)` bernilai 3.0 di C. Ketika uang dibulatkan, tuliskan aturan mana yang Anda maksud dan pakailah dengan sengaja.
- **Variabel lokal yang menyembunyikan variabel global.** Mendeklarasikan `int orders_today = 0;` di dalam fungsi membuat variabel lokal baru yang menyembunyikan variabel global; variabel global itu tidak pernah berubah. `gcc -Wshadow` memperingatkan tentang nama yang menyembunyikan nama lain.

5.14 Ringkasan Bab

- Fungsi adalah sepotong pekerjaan bernama dengan parameter dan nilai kembalian. Definisikan sekali, panggil di mana pun diperlukan; pemanggil hanya memerlukan kepalanya.
- Argumen disalin ke parameter (pass-by-value): fungsi tidak dapat mengubah variabel pemanggil lewat parameternya, dan memberikan jawabannya lewat `return`.
- Prototipe adalah janji tentang nama, tipe parameter, dan tipe kembalian sebuah fungsi, diletakkan sebelum pemanggilan pertama; header adalah berkas berisi prototipe, dan `#include` membawanya masuk. Python tidak memerlukan prototipe, hanya mensyaratkan nama sudah ada ketika dipanggil.
- Pustaka standar menyediakan fungsi yang sudah teruji menurut pokok bahasan: `<math.h>` (dengan `-lm` di Linux) untuk `ceil`, `floor`, `round`, `sqrt`, `pow`; `<stdlib.h>`, `<string.h>`, `<time.h>`, dan lainnya. Laundry menagih max(2, $\lceil w \rceil$) kilogram, Persamaan (5.1).
- Cakupan adalah tempat sebuah nama terlihat, masa hidup adalah berapa lama variabel ada: variabel lokal dan parameter hidup selama satu pemanggilan, variabel global dan lokal `static` selama program berjalan. Utamakan variabel lokal, pakai `static` untuk ingatan pribadi, dan jarangkan variabel global.
- Pembangkit pseudo-acak adalah fungsi dengan keadaan tersembunyi; dengan rumus sendiri, Persamaan (5.2), bilangannya berulang sama di setiap komputer dan di kedua bahasa.
- Fungsi rekursif memiliki kasus dasar dan langkah rekursif; setiap pemanggilan yang aktif memiliki bingkai sendiri di tumpukan panggilan. Rekursi wajar untuk masalah bersarang, dan berbahaya bila mengulang pekerjaan, seperti `fib` dengan $2F(n+1) - 1$ pemanggilan.
- Program yang terdiri atas beberapa berkas menyimpan aturan di sebuah modul (`laundry.h`, `laundry.c`), cerita di `main`, dan program uji yang memeriksa setiap aturan terhadap jawaban hitungan tangan.

5.15 Soal Latihan

1. **Menelusuri.** Tanpa menjalankannya, tuliskan apa yang dicetak program ini, lalu periksa dengan menjalankannya. Jelaskan setiap baris dengan kata *salinan* dan *return*. `int twice(int`

`x) { x = x * 2; return x; }` dan di `main`: `int a = 5; int b = twice(a); printf("%d %d\n", a, b); twice(b); printf("%d\n", b);`

2. **Fungsi buatan sendiri.** Tulislah `long round_to_500(long amount)` yang membulatkan jumlah uang ke kelipatan Rp500 terdekat, setengah dibulatkan ke atas (Rp28.250 menjadi Rp28.500, Rp28.249 menjadi Rp28.000), hanya dengan aritmetika bilangan bulat. Berikan lima nilai uji beserta hasil yang diharapkan.
3. **Detektif cakupan.** Sebuah program memiliki variabel global `int total = 0;` dan fungsi `void add(int x) { int total = 0; total += x; }`. Sesudah `add(5); add(7);`, berapakah `total` global, dan mengapa? Bagaimana Anda memperbaiki `add`, dan opsi peringatan `gcc` mana yang akan menunjuk masalah itu?
4. **Euclid, secara rekursif.** Faktor persekutuan terbesar (FPB) dua bilangan memenuhi $\gcd(a, 0) = a$ dan $\gcd(a, b) = \gcd(b, a \bmod b)$. Tulislah sebagai fungsi rekursif yang mencetak setiap pemanggilan, lalu pakailah untuk menentukan seberapa sering dua rute penjemputan yang datang setiap 4 dan setiap 6 hari jatuh pada hari yang sama (kelipatan persekutuan terkecilnya, $a \cdot b / \gcd(a, b)$).
5. **Jumlah digit.** Tulislah jumlah digit desimal sebuah bilangan dua kali, secara rekursif dan dengan perulangan, serta *digit pemeriksa* yang mengulang penjumlahan digit sampai tersisa satu digit (2026 → 10 → 1). Cetak kedua jumlah dan digit pemeriksa untuk 0, 7, 2026, dan 987654321.
6. **Menara Hanoi.** Tiga keranjang cucian berbeda ukuran bertumpuk di rak A, yang terbesar di bawah. Pindahkan semuanya ke rak C, satu keranjang setiap kali, tanpa pernah menaruh keranjang yang lebih besar di atas yang lebih kecil, dengan rak B sebagai bantuan. Tulislah fungsi rekursif yang mencetak langkah-langkah untuk n keranjang dan menghitungnya. Berapa langkah yang diperlukan untuk n keranjang?
7. **Menghitung pemanggilan.** Dengan Persamaan (5.5), hitunglah $C(10)$ dengan tangan dan cocokkan dengan keluaran **Kode 5.11**. Kemudian perkirakan berapa pemanggilan yang dilakukan `fib(40)`, dan jelaskan mengapa perulangan hanya memerlukan 40 langkah.

5.16 Penyelesaian Soal Latihan

1. Program mencetak `5  10` lalu `10`. Pada `twice(a)` parameter `x` menerima *salinan* 5, menggandakan salinannya, dan mengembalikan 10, yang disimpan `main` di `b`; `a` tidak tersentuh. Pemanggilan `twice(b)` menggandakan salinan 10 dan *mengembalikan* 20, tetapi tidak ada yang menyimpan nilai kembalian itu, sehingga hilang dan `b` tetap 10.

2. Tambahkan 250 lalu potong ke kelipatan 500 dengan pembagian bilangan bulat: `return (amount + 250) / 500 * 500;`. Nilai uji: 28250 → 28500, 28249 → 28000, 28000 → 28000, 0 → 0, 136499 → 136500. Dua yang pertama adalah tepi aturan dan paling berharga; setengah dibulatkan ke atas karena 28250 + 250 = 28500 tepat kelipatan 500.

3. `total` global tetap 0. Di dalam `add`, deklarasi `int total = 0;` membuat variabel lokal baru yang menyembunyikan variabel global di seluruh fungsi; penjumlahan mengubah variabel lokal, yang lenyap di akhir setiap pemanggilan. Perbaikannya adalah menghapus deklarasi lokal itu, sehingga `total += x;` merujuk ke variabel global. `gcc -Wshadow` melaporkan bahwa deklarasi lokal itu membayangi (*shadows*) deklarasi global.

4. **Kode 5.20** dan **Kode 5.21** mengikuti kedua persamaan secara langsung; setiap pemanggilan pasti memperkecil argumen kedua, sehingga kasus dasar $b = 0$ selalu tercapai.

Kode 5.20 `ex5_gcd.c`: algoritma Euclid secara rekursif, dan kelipatan persekutuan terkecil yang dibangun darinya.

```c
/* ex5_gcd.c - Latihan 5.4: FPB Euclid, secara rekursif */
#include <stdio.h>

int gcd(int a, int b)
{
    printf("  gcd(%d, %d)\n", a, b);
    if (b == 0)
        return a;
    return gcd(b, a % b);
}

int main(void)
{
    int g = gcd(84, 36);
    printf("gcd = %d\n", g);
    int a = 4, b = 6;                   /* rute jemput setiap 4 dan 6 hari */
    int lcm = a / gcd(a, b) * b;
    printf("both rounds meet every %d days\n", lcm);
    return 0;
}
```

Kode 5.21 `ex5_gcd.py`: hal yang sama dalam Python.

```python
# ex5_gcd.py - Latihan 5.4: FPB Euclid, secara rekursif


def gcd(a, b):
    print(f"  gcd({a}, {b})")
    if b == 0:
        return a
    return gcd(b, a % b)


def main():
    g = gcd(84, 36)
    print(f"gcd = {g}")
    a, b = 4, 6                         # rute jemput setiap 4 dan 6 hari
    lcm = a // gcd(a, b) * b
    print(f"both rounds meet every {lcm} days")


if __name__ == "__main__":
    main()
```

```
  gcd(84, 36)
  gcd(36, 12)
  gcd(12, 0)
gcd = 12
  gcd(4, 6)
  gcd(6, 4)
  gcd(4, 2)
  gcd(2, 0)
both rounds meet every 12 days
```

Kelipatan persekutuan terkecil dihitung sebagai `a / gcd(a, b) * b`, bukan `a * b / gcd(a, b)`: membagi lebih dulu menjaga bilangan antara tetap kecil, sehingga menghindari luapan untuk masukan besar. Kedua rute bertemu setiap 12 hari.

5. Kode 5.22 dan **Kode 5.23**. Rekursinya memisahkan digit terakhir dengan `% 10` dan berekursi pada sisanya dengan `/ 10`; kasus dasarnya bilangan satu digit. Digit pemeriksa pun rekursif: ia menerapkan penjumlahan digit sampai hasilnya tinggal satu digit.

Kode 5.22 `ex5_digits.c`: jumlah digit secara rekursif dan dengan perulangan, serta digit pemeriksa.

```c
/* ex5_digits.c - Latihan 5.5: jumlah digit, rekursif dan dengan perulangan */
#include <stdio.h>

int digit_sum_rec(long n)
{
    if (n < 10)
        return (int) n;
    return (int) (n % 10) + digit_sum_rec(n / 10);
}

int digit_sum_loop(long n)
{
    int sum = 0;
    while (n > 0) {
        sum += (int) (n % 10);
        n /= 10;
    }
    return sum;
}

int check_digit(long receipt)            /* ulangi sampai tersisa satu digit */
{
    int s = digit_sum_rec(receipt);
    return s < 10 ? s : check_digit(s);
}

int main(void)
{
    long numbers[] = {0, 7, 2026, 987654321};
    for (int i = 0; i < 4; i++)
        printf("%9ld: %2d %2d  check digit %d\n", numbers[i],
               digit_sum_rec(numbers[i]), digit_sum_loop(numbers[i]),
               check_digit(numbers[i]));
    return 0;
}
```

Kode 5.23 `ex5_digits.py`: hal yang sama dalam Python.

```python
# ex5_digits.py - Latihan 5.5: jumlah digit, rekursif dan dengan perulangan


def digit_sum_rec(n):
    if n < 10:
        return n
    return n % 10 + digit_sum_rec(n // 10)


def digit_sum_loop(n):
    total = 0
    while n > 0:
        total += n % 10
        n //= 10
    return total


def check_digit(receipt):                # ulangi sampai tersisa satu digit
    s = digit_sum_rec(receipt)
    return s if s < 10 else check_digit(s)


def main():
    for n in [0, 7, 2026, 987654321]:
        print(f"{n:9d}: {digit_sum_rec(n):2d} {digit_sum_loop(n):2d}"
              f"  check digit {check_digit(n)}")


if __name__ == "__main__":
    main()
```

```
        0:  0  0  check digit 0
        7:  7  7  check digit 7
     2026: 10 10  check digit 1
987654321: 45 45  check digit 9
```

6. Untuk memindahkan n keranjang dari A ke C, pindahkan $n-1$ keranjang teratas dari A ke B (dengan C sebagai bantuan), pindahkan yang terbesar dari A ke C, lalu pindahkan $n-1$ keranjang dari B ke C (dengan A sebagai bantuan). Memindahkan nol keranjang adalah kasus dasarnya. **Kode 5.24** dan **Kode 5.25** mencetak langkah-langkah untuk tiga keranjang.

Kode 5.24 `ex5_hanoi.c`: Menara Hanoi, masalah klasik yang mudah secara rekursif dan sulit dengan perulangan.

```c
/* ex5_hanoi.c - Latihan 5.6: Menara Hanoi dengan keranjang cucian */
#include <stdio.h>

int moves = 0;

void hanoi(int n, char from, char to, char spare)
{
    if (n == 0)
        return;
    hanoi(n - 1, from, spare, to);       /* kosongkan jalan */
    moves++;
    printf("move basket %d from %c to %c\n", n, from, to);
    hanoi(n - 1, spare, to, from);       /* taruh sisanya kembali di atas */
}

int main(void)
{
    hanoi(3, 'A', 'C', 'B');
    printf("%d moves\n", moves);
    return 0;
}
```

Kode 5.25 `ex5_hanoi.py`: hal yang sama dalam Python.

```python
# ex5_hanoi.py - Latihan 5.6: Menara Hanoi dengan keranjang cucian

moves = 0


def hanoi(n, source, target, spare):
    global moves
    if n == 0:
        return
    hanoi(n - 1, source, spare, target)  # kosongkan jalan
    moves += 1
    print(f"move basket {n} from {source} to {target}")
    hanoi(n - 1, spare, target, source)  # taruh sisanya kembali di atas


def main():
    hanoi(3, "A", "C", "B")
    print(f"{moves} moves")


if __name__ == "__main__":
    main()
```

```
move basket 1 from A to C
move basket 2 from A to B
move basket 1 from C to B
move basket 3 from A to C
move basket 1 from B to A
move basket 2 from B to C
move basket 1 from A to C
7 moves
```

Jika $M(n)$ adalah banyaknya langkah, $M(0) = 0$ dan $M(n) = 2M(n-1) + 1$, yang memberikan $M(n) = 2^n - 1$: 7 langkah untuk 3 keranjang, 1023 untuk 10.

7. Dari Persamaan (5.5): $C(2) = 3$, $C(3) = 5$, $C(4) = 9$, $C(5) = 15$, $C(6) = 25$, $C(7) = 41$, $C(8) = 67$, $C(9) = 109$, $C(10) = 177$, yaitu bilangan yang dicetak program untuk `fib(10)`. Dengan $C(n) = 2F(n+1) - 1$ dan $F(41) = 165580141$, `fib(40)` melakukan 331,160,281 pemanggilan. Perulangan hanya menyimpan dua bilangan terakhir dan menjumlahkannya sekali per langkah,

sehingga memerlukan 40 penjumlahan: setiap bilangan Fibonacci dihitung sekali saja, bukan dihitung ulang di setiap cabang pohon.

Daftar Pustaka

[1] N. Wirth, "Program development by stepwise refinement," *Communications of the ACM*, vol. 14, no. 4, pp. 221–227, Apr. 1971.

[2] B. W. Kernighan and D. M. Ritchie, *The C Programming Language*, 2nd ed. Englewood Cliffs, NJ, USA: Prentice Hall, 1988.

[3] ISO/IEC 9899:2018, *Information technology — Programming languages — C*. Geneva, Switzerland: International Organization for Standardization, 2018.

[4] Python Software Foundation, "Defining functions," in *The Python Tutorial*, Python 3.14 documentation. [Online]. Available: https://docs.python.org/3/tutorial/controlflow.html#defining-functions

[5] Free Software Foundation, "Options for linking," in *Using the GNU Compiler Collection (GCC)*. [Online]. Available: https://gcc.gnu.org/onlinedocs/gcc/Link-Options.html

[6] Python Software Foundation, "math — Mathematical functions," Python 3.14 documentation. [Online]. Available: https://docs.python.org/3/library/math.html

[7] Python Software Foundation, "Built-in functions: round," Python 3.14 documentation. [Online]. Available: https://docs.python.org/3/library/functions.html#round

[8] Python Software Foundation, "random — Generate pseudo-random numbers," Python 3.14 documentation. [Online]. Available: https://docs.python.org/3/library/random.html

[9] D. E. Knuth, *The Art of Computer Programming, Vol. 2: Seminumerical Algorithms*, 3rd ed. Reading, MA, USA: Addison-Wesley, 1997.

[10] Python Software Foundation, "sys.getrecursionlimit," Python 3.14 documentation. [Online]. Available: https://docs.python.org/3/library/sys.html#sys.getrecursionlimit

[11] R. L. Graham, D. E. Knuth, and O. Patashnik, *Concrete Mathematics*, 2nd ed. Reading, MA, USA: Addison-Wesley, 1994.

[12] H. Abelson and G. J. Sussman, *Structure and Interpretation of Computer Programs*, 2nd ed. Cambridge, MA, USA: MIT Press, 1996.

[13] Python Software Foundation, "unittest — Unit testing framework," Python 3.14 documentation. [Online]. Available: https://docs.python.org/3/library/unittest.html

Bab 6

Array: Banyak Nilai, Satu Nama

Kasir Bab 5 mencetak nota yang benar lalu melupakannya begitu pelanggan berikutnya maju. Itu cukup untuk nota, tetapi tidak berguna untuk semua hal yang ditanyakan pemilik saat tutup toko: pesanan mana yang terbesar, berapa tagihan yang lazim, berapa pelanggan yang meminta setrika, hari apa yang paling ramai dalam seminggu. Untuk menjawabnya, program harus mengingat banyak nilai sejenis, dan memberi setiap nilai nama variabelnya sendiri tidak akan sanggup melampaui segelintir nilai. Bab ini memperkenalkan **array** (larik), satu nama untuk sederet nilai, dan tiga hal yang dilakukan setiap pemrogram terhadap deretan: menghitung, mengurutkan, dan mencari. Di akhir bab ini Anda akan mampu mendeklarasikan, menginisialisasi, dan menelusuri array dalam C serta list dalam Python; menjelaskan mengapa indeks dimulai dari 0 dan apa yang terjadi ketika program melangkah keluar dari array; memakai array sebagai kumpulan penghitung; meneruskan array ke fungsi dengan aman beserta ukuran dan `const`; mengurutkan dengan bubble sort dan insertion sort serta menghitung kerjanya; mencari dengan pencarian linear dan biner serta menentukan kapan memakai yang mana; menyimpan tabel dalam array dua dimensi dan menghitung alamat sel mana pun; serta membangun kasir Profitina Laundry versi 5, yang mengingat seluruh hari dan mencetak laporan berisi total per layanan, rata-rata, median, dan tiga tagihan terbesar.

6.1 Catatan di Penghujung Hari

Ketika Profitina Laundry tutup pukul sembilan malam, Mbak Rina duduk dengan secangkir teh dan setumpuk tembusan karbon nota hari itu. Ia ingin tahu empat hal. Berapa uang yang masuk, dan berapa tagihan rata-ratanya? Pesanan mana yang terbesar, supaya ia bisa berterima kasih kepada pelanggan itu lain kali? Berapa pesanan cuci lipat, cuci setrika, dan setrika saja, supaya ia tahu apakah harus membeli lebih banyak deterjen atau lebih banyak pelicin setrika? Dan, dengan menengok ke seminggu terakhir, hari apa yang paling ramai, supaya pembantu paruh waktunya datang pada hari yang tepat?

Dengan kertas, ia menjawab pertanyaan pertama dengan menjumlahkan nota dua kali, dan sisanya dengan memilah nota menjadi beberapa tumpukan di meja. Dimas ingin kasirnya yang mengerjakan. Kesulitannya, program Bab 5 hanya menyimpan total berjalan. Total cukup untuk menghitung jumlah dan rata-rata, tetapi tidak untuk median, tidak untuk tiga tagihan terbesar, dan tidak untuk daftar yang terurut, karena semua itu memerlukan setiap tagihan di akhir hari, bukan hanya jumlahnya.

Mendeklarasikan `long bill1, bill2, bill3, ... bill100;` bukan penyelesaian. Perulangan tidak dapat berkata “variabel berikutnya”, karena bagi kompilator `bill1` dan `bill2` adalah nama yang tidak berhubungan, dan diperlukan seratus pernyataan `if` yang hampir sama untuk menyimpan pesanan kelima puluh tujuh di `bill57`. Yang dibutuhkan Dimas adalah satu nama untuk seluruh tumpukan nota, dan sebuah bilangan yang menyatakan nota mana yang dimaksud. Persis itulah tumpukan tembusan di atas meja: satu tumpukan, dan nota-nota di dalamnya bernomor dari atas.

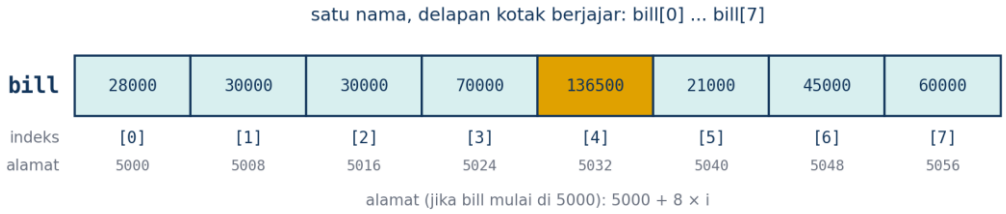
Kebutuhan yang sama muncul di mana pun ada daftar benda sejenis: penjualan harian sebuah warung selama sebulan, curah hujan setiap hari di kebun buah dan durian, nilai rapor satu kelas, berat beras setiap karung dalam satu kiriman. Dalam setiap kasus pemiliknya bertanya hal yang sama: berapa totalnya, rata-ratanya, yang terbesar, berapa banyak tiap jenis, tolong urutkan, dan di mana yang satu ini.

6.2 Gagasannya dengan Kata-Kata Sederhana

6.2.1 Array adalah deretan kotak bernomor

> **DEFINISI.** *Array* adalah sejumlah tetap variabel bertipe sama yang disimpan berdampingan di memori dengan satu nama. Setiap variabel adalah *elemen* array, dan dipilih dengan *indeks*, bilangan bulat yang ditulis di dalam kurung siku: `bill[0]`, `bill[1]`, dan seterusnya. Dalam C indeks pertama adalah 0, dan array berisi n elemen memiliki indeks 0 sampai $n - 1$.

Deklarasi `long bill[8];` meminta delapan variabel `long` yang berderet. **Gambar 6.1** menggambarkannya. Nama `bill` mewakili seluruh deretan; `bill[4]` adalah kotak kelima, yang di sini berisi 136500. Apa pun yang dapat Anda lakukan terhadap variabel `long` biasa dapat Anda lakukan terhadap sebuah elemen: membacanya, mengisinya, meneruskannya ke `printf`, memakainya dalam syarat.



Gambar 6.1 Array `bill` dengan delapan elemen: satu nama, indeks 0 sampai 7, dan elemen-elemen berdampingan di memori, berjarak delapan bita di komputer yang menyimpan `long` dalam delapan bita.

Kekuatan array terletak pada indeks yang boleh berupa variabel. `bill[i]` berarti "kotak yang nomornya sedang tersimpan di `i`", sehingga satu perulangan dengan `i` berjalan dari 0 sampai 7 mengunjungi setiap kotak. Itulah seluruh perbedaan antara delapan variabel dan satu array: delapan variabel punya delapan nama, array punya satu nama dan sebuah nomor yang dapat dihitung oleh perulangan.

6.2.2 Mengapa indeks dimulai dari 0

Kotak-kotak array duduk bersebelahan. Jika array dimulai di alamat memori a, dan setiap elemen memakan s bita, maka elemen bernomor i dimulai di

$$\text{alamat}(\text{bill}[i]) = a + i \cdot s \tag{6.1}$$

Inilah sebabnya C menghitung dari 0: indeks adalah *jarak* dari awal array, diukur dalam elemen. Elemen pertama berjarak 0 dari awal, dan Persamaan (6.1) tidak memerlukan koreksi. Dengan $a = 5000$ dan $s = 8$, seperti di **Gambar 6.1**, `bill[4]` berada di $5000 + 4 \cdot 8 = 5032$. Komputer menemukan elemen mana pun dengan satu perkalian dan satu penjumlahan, sepanjang apa pun arraynya; itulah sebabnya membaca `bill[4]` sama cepatnya dengan membaca `bill[0]`, dan mengapa array adalah struktur tercepat untuk permintaan "berikan elemen nomor i". (Alamat

sungguhan adalah bilangan yang jauh lebih besar, dan di Windows 64-bit sebuah `long` memakan empat bita, bukan delapan; rumusnya tetap sama.)

Rumus yang sama menjelaskan mengapa ukuran array tetap. Elemen-elemennya harus berdampingan, sehingga ruang untuk semuanya dipesan sekaligus ketika array dibuat. Tidak ada tempat untuk menambah kotak kesembilan sesudah `bill[7]`, karena bita-bita sesudahnya mungkin sudah menjadi milik variabel lain. Program yang perlu bertambah besar harus memesan ruang lebih banyak daripada yang diperkirakan akan dipakai, seperti dilakukan kasir di Subbab 6.9, atau meminta blok baru yang lebih besar lalu menyalin elemen-elemennya ke sana, yang diam-diam dilakukan list Python di balik `append`.

6.2.3 Tidak ada yang menjaga pagar

C memercayai pemrogramnya. Ketika program menulis `bill[8]` pada array berisi delapan elemen, kompilator tidak menghentikannya dan program yang berjalan tidak memeriksanya: Persamaan (6.1) begitu saja menghitung alamat tepat sesudah ujung array, dan program membaca atau menulis apa pun yang kebetulan tinggal di sana, barangkali variabel lain, barangkali sesuatu yang tidak boleh disentuhnya. Standar C menyebutnya *undefined behavior* (perilaku tak terdefinisi) [1]: apa saja boleh terjadi, termasuk program tampak bekerja sampai suatu hari tidak lagi. SEI CERT C Coding Standard memasukkannya sebagai salah satu aturan terpentingnya, ARR30-C, karena operasi tulis di luar batas telah menyebabkan lubang keamanan yang tak terhitung jumlahnya [2].

Pertahanannya adalah kebiasaan, bukan alat: setiap perulangan atas array berjalan `for (i = 0; i < n; i++)` dengan `<` dan tidak pernah `<=`, setiap indeks yang dihitung dari masukan diperiksa sebelum dipakai, dan ukuran array adalah konstanta bernama. Opsi build `-fsanitize=address,undefined`, yang dilalui setiap program buku ini, membuat `gcc` menyisipkan pemeriksaan yang menghentikan program dengan laporan yang jelas pada akses di luar batas yang pertama; pakailah selama Anda mengembangkan program [3].

6.2.4 List milik Python

Python tidak memiliki array berukuran tetap di inti bahasanya; perannya dimainkan oleh **list**. List ditulis dalam kurung siku, `bill = [28000, 30000, 30000]`, dan diindeks persis seperti array C, mulai dari 0. Tiga perbedaan penting. List tahu panjangnya sendiri, `len(bill)`, sehingga tidak perlu penghitung terpisah. List dapat memanjang dan memendek, dengan `bill.append(45000)` dan `bill.pop()`. Dan Python *memang* menjaga pagar: `bill[8]` pada list berisi delapan elemen menghentikan program dengan `IndexError: list index out of range` [4]. List bahkan boleh memuat nilai berbeda tipe, walaupun list di buku ini tidak pernah demikian.

Ada pula perbedaan dalam isi kotak-kotaknya. Array `long` di C memuat bilangan-bilangan itu sendiri, masing-masing delapan bita, tersusun rapat. List Python memuat *referensi*: setiap elemen menunjuk ke objek bilangan terpisah di tempat lain di memori. Inilah yang memungkinkan list memuat nilai bertipe apa saja dan memanjang dengan leluasa, dan harganya adalah memori dan kecepatan: list berisi sejuta bilangan memakan memori beberapa kali lipat array C berisi sejuta nilai `long`, dan menelusurinya lebih lambat. Untuk beberapa ratus pesanan sebuah laundry, perbedaan itu sama sekali tidak penting. Untuk jutaan hasil pengukuran, pemrogram Python memakai modul standar `array` atau pustaka NumPy, yang menyimpan bilangan secara rapat seperti C.

6.3 Mendeklarasikan, Mengisi, dan Menelusuri Array

Kode 6.1 mendeklarasikan delapan tagihan hari itu dengan sebuah **inisialisator**, daftar nilai di dalam kurung kurawal, lalu menelusuri array empat kali: untuk mencetaknya, menjumlahkannya, mencari tagihan terbesar, dan menghitung tagihan yang bernilai Rp50.000 atau lebih.

Kode 6.1 `day_bills.c`: delapan tagihan dalam satu array, dicetak, dijumlahkan, dicari yang terbesar, dan dihitung.

```c
/* day_bills.c - delapan tagihan di bawah satu nama: array */
#include <stdio.h>

#define N 8

int main(void)
{
    long bill[N] = {28000, 30000, 30000, 70000, 136500, 21000, 45000, 60000};

    printf("index  bill\n");
    for (int i = 0; i < N; i++)          /* i berjalan 0, 1, ..., N-1 */
        printf("%5d  %6ld\n", i, bill[i]);

    long total = 0;
    for (int i = 0; i < N; i++)
        total += bill[i];

    int best = 0;                        /* indeks tagihan terbesar sejauh ini */
    for (int i = 1; i < N; i++)
        if (bill[i] > bill[best])
            best = i;

    int big = 0;
    for (int i = 0; i < N; i++)
        if (bill[i] >= 50000)
            big++;

    printf("total   : %ld\n", total);
    printf("average : %.2f\n", (double) total / N);
    printf("biggest : %ld at index %d\n", bill[best], best);
    printf("bills of Rp 50000 or more: %d\n", big);
    printf("elements: %zu\n", sizeof bill / sizeof bill[0]);  /* bita / bita */
    return 0;
}
```

Beberapa aturan inisialisator layak diketahui. Jika kurung kurawal memuat lebih sedikit nilai daripada banyaknya elemen, sisanya diisi nol, sehingga `int count[4] = {0};` menghasilkan empat nol; itulah cara baku untuk memulai sekumpulan penghitung. Jika ukurannya tidak ditulis, seperti `long bill[] = {28000, 30000};`, kompilator menghitung nilainya dan membuat array tepat sepanjang itu. Array yang dideklarasikan di dalam fungsi *tanpa* inisialisator tidak diisi nol; elemennya berisi apa saja yang sebelumnya ada di memori, itulah sebabnya setiap array penghitung di buku ini dimulai dengan `= {0}`.

Ada gunanya menelusuri salah satu perulangan ini dengan tangan sekali saja, seperti cara komputer menjalankannya. Sebelum perulangan penjumlahan, `total` bernilai 0. Dengan `i` = 0 badan perulangan menambahkan `bill[0]`, 28000, dan `total` menjadi 28000; dengan `i` = 1 ditambahkan 30000 dan `total` menjadi 58000; lalu 88000, 158000, 294500, 315500, 360500, dan akhirnya 420500 ketika `i` = 7. Sekarang `i++` membuat `i` bernilai 8, pengujian `i < N` bernilai salah, dan perulangan berakhir tanpa pernah menyentuh `bill[8]`, yang memang tidak ada. Empat bagian yang sama (awal, uji, badan, langkah) berjalan di setiap perulangan bab ini; hanya badannya yang berganti.

Pencarian tagihan terbesar menyimpan *indeks* tagihan terbaik sejauh ini, `best`, bukan nilainya. Mengetahui letak tagihan terbesar berarti mengetahui nilainya juga (`bill[best]`), dan di kasir Subbab 6.9 letak itu juga memberikan nomor pesanan. Perulangannya dimulai dari 1 karena `best = 0` sudah melihat elemen 0.

Baris terakhir mencetak banyaknya elemen dengan `sizeof bill / sizeof bill[0]`: ukuran seluruh array dalam bita dibagi ukuran satu elemen. Trik ini hanya berlaku di tempat kompilator dapat melihat deklarasi array, seperti di `main` ini; Subbab 6.5 memperlihatkan di mana trik itu gagal.

Rata-rata adalah total dibagi banyaknya nilai, yaitu **rerata** (*mean*) aritmetika:

$$\bar{x} = \frac{1}{n} \sum_{i=0}^{n-1} x_i \tag{6.2}$$

Pada Persamaan (6.2), x_i adalah elemen i dan lambang $\sum$ berarti "jumlahkan untuk i dari 0 sampai $n - 1$", persis perulangan kedua. Cast `(double) total / N` membuat pembagiannya tepat, bukan pembagian bilangan bulat.

Versi Python, **Kode 6.2**, menelusuri list dengan `range(N)`, yang menghasilkan indeks 0 sampai $N - 1$, agar kedua program tampak serupa. Python yang idiomatis sering menulis `for b in bill:` dan memakai `sum(bill)` serta `max(bill)` dari pustaka standar; kita menulis perulangannya sendiri karena bab ini membahas apa yang dikerjakan fungsi-fungsi pustaka itu di dalamnya.

Kode 6.2 `day_bills.py`: penelusuran yang sama atas list Python.

```python
# day_bills.py - delapan tagihan di bawah satu nama: list

N = 8


def main():
    bill = [28000, 30000, 30000, 70000, 136500, 21000, 45000, 60000]

    print("index  bill")
    for i in range(N):                  # i berjalan 0, 1, ..., N-1
        print(f"{i:5d}  {bill[i]:6d}")

    total = 0
    for i in range(N):
        total += bill[i]

    best = 0                            # indeks tagihan terbesar sejauh ini
    for i in range(1, N):
        if bill[i] > bill[best]:
            best = i

    big = 0
    for i in range(N):
        if bill[i] >= 50000:
            big += 1

    print(f"total   : {total}")
    print(f"average : {total / N:.2f}")
    print(f"biggest : {bill[best]} at index {best}")
    print(f"bills of Rp 50000 or more: {big}")
    print(f"elements: {len(bill)}")      # list tahu panjangnya sendiri


if __name__ == "__main__":
    main()
```

```
index  bill
    0   28000
    1   30000
    2   30000
    3   70000
    4  136500
    5   21000
    6   45000
    7   60000
total   : 420500
average : 52562.50
biggest : 136500 at index 4
bills of Rp 50000 or more: 3
elements: 8
```

Periksalah angkanya sekali dengan tangan: delapan tagihan berjumlah 420500, dan 420500 / 8 = 52562,5. Tiga tagihan (70000, 136500, dan 60000) bernilai paling sedikit 50000. Rata-ratanya lebih tinggi daripada lima dari delapan tagihan, karena satu pesanan besar senilai Rp136.500 menariknya ke atas; Subbab 6.12 kembali ke hal ini.

6.4 Array sebagai Penghitung

Pertanyaan kedua adalah “berapa banyak tiap layanan?”. Di sini array memainkan peran baru: *elemen*-nya adalah penghitung, dan *indeks*-nya menyatakan apa yang dihitung. **Kode 6.3** memiliki kode layanan dua puluh pesanan hari itu dan array `count` berisi empat penghitung. Untuk setiap pesanan berkode `s`, `count[s]++` menambah satu pada penghitung nomor `s`. Nilai data menjadi

indeks. Tidak ada yang perlu dibandingkan atau dicari: kode layanan langsung menuju penghitungnya.

Kode 6.3 `service_count.c`: array penghitung yang diindeks dengan kode layanan, dan sebatang bintang untuk masing-masing.

```c
/* service_count.c - array penghitung: berapa kali tiap layanan dipesan? */
#include <stdio.h>

int main(void)
{
    int service[20] = {1, 2, 1, 3, 1, 1, 2, 3, 1, 2,
                       1, 1, 3, 2, 1, 9, 1, 2, 1, 1};
    int count[4] = {0};                 /* count[1..3]; count[0] tidak dipakai */
    int unknown = 0;

    for (int i = 0; i < 20; i++) {
        int s = service[i];
        if (s >= 1 && s <= 3)           /* penjaga: jangan keluar dari 0..3 */
            count[s]++;                 /* nilainya sendiri adalah indeks */
        else
            unknown++;
    }

    for (int s = 1; s <= 3; s++) {
        printf("service %d: %2d  ", s, count[s]);
        for (int k = 0; k < count[s]; k++)
            putchar('*');
        putchar('\n');
    }
    printf("unknown codes: %d\n", unknown);
    return 0;
}
```

Satu pesanan berkode mustahil 9, salah ketik. Tanpa penjaga `if (s >= 1 && s <= 3)`, pernyataan `count[9]++` akan menulis melewati ujung array empat elemen: tepat perilaku tak terdefinisi Subbab 6.2.3, dan kesalahan yang mungkin tak ketahuan berminggu-minggu sementara diam-diam mengubah variabel lain. Dengan penjaga, salah ketik itu dihitung terpisah dan dilaporkan. Penghitung 0 tidak pernah dipakai; mengorbankan satu elemen supaya kode layanan dapat langsung menjadi indeks adalah pertukaran yang baik.

Versi Python, **Kode 6.4**, membuat penghitung dengan `[0] * 4`, list berisi empat nol, dan menggambar setiap batang dengan `"*" * count[s]`, string yang diulang `count[s]` kali.

Kode 6.4 `service_count.py`: penghitung yang sama dalam Python.

```python
# service_count.py - list penghitung: berapa kali tiap layanan dipesan?


def main():
    service = [1, 2, 1, 3, 1, 1, 2, 3, 1, 2,
               1, 1, 3, 2, 1, 9, 1, 2, 1, 1]
    count = [0] * 4                      # count[1..3]; count[0] tidak dipakai
    unknown = 0

    for s in service:
        if 1 <= s <= 3:                  # penjaga: Python akan melempar IndexError
            count[s] += 1                # nilainya sendiri adalah indeks
        else:
            unknown += 1

    for s in range(1, 4):
        print(f"service {s}: {count[s]:2d}  " + "*" * count[s])
    print(f"unknown codes: {unknown}")


if __name__ == "__main__":
    main()
```

```
service 1: 11  ***********
service 2:  5  *****
service 3:  3  ***
unknown codes: 1
```

Pola yang sama bekerja ketika yang dihitung bukan kode kecil melainkan rentang nilai. Untuk melihat sebaran tagihan, hitunglah dalam pita Rp25.000: tagihan sebesar `b` rupiah jatuh di pita `b / 25000`, sehingga Rp28.000 berada di pita 1 (25.000 sampai 49.999) dan Rp136.500 di pita 5. Dengan `int band[6] = {0};` dan penjaga yang memasukkan setiap tagihan Rp125.000 atau lebih ke pita terakhir, `band[b / 25000]++` membangun *histogram*, gambaran banyaknya tagihan di setiap rentang. Pembagian bilangan bulat mengerjakan penggolongannya, dan array mengerjakan penghitungannya.

Sebelas dari dua puluh pesanan, lebih dari separuh, adalah cuci lipat. Mbak Rina kini tahu deterjen mana yang perlu dibeli dalam jumlah besar. *Frequency counting* (penghitungan frekuensi) ini salah satu pola paling berguna dalam pemrograman: menghitung suara per calon, penjualan per produk, pengunjung per jam, atau jenis huruf dalam sebuah teks (Bab 9) semuanya mengikuti pola ini.

6.5 Meneruskan Array ke Fungsi

Setiap pertanyaan malam itu memerlukan pekerjaan kecil yang sama: menjumlahkan, mencari yang terbesar, menghitung rata-rata. Pekerjaan itu layak menjadi fungsi, seperti diajarkan Bab 5, dan fungsinya memerlukan array sebagai parameter. **Kode 6.5** memperlihatkan caranya. Parameternya ditulis `const long a[]`, dengan kurung siku kosong, dan banyaknya elemen datang sebagai parameter kedua `n`.

Kode 6.5 `array_functions.c`: fungsi yang menerima array beserta ukurannya; `const` berjanji tidak mengubahnya, dan `add_to_all` mengubah array milik pemanggil.

```c
/* array_functions.c - menyerahkan array ke fungsi */
#include <stdio.h>

long total(const long a[], int n)        /* const: kita berjanji tidak mengubah a */
{
    long sum = 0;
    for (int i = 0; i < n; i++)
        sum += a[i];
    return sum;
}

double average(const long a[], int n)
{
    return n > 0 ? (double) total(a, n) / n : 0.0;
}

int index_of_max(const long a[], int n)
{
    int best = 0;
    for (int i = 1; i < n; i++)
        if (a[i] > a[best])
            best = i;
    return best;
}

void add_to_all(long a[], int n, long amount)   /* mengubah array milik PEMANGGIL */
{
    for (int i = 0; i < n; i++)
        a[i] += amount;
}

void print_all(const long a[], int n)
{
    for (int i = 0; i < n; i++)
        printf("%ld%s", a[i], i < n - 1 ? " " : "\n");
}

int main(void)
{
    long morning[5] = {28000, 30000, 30000, 70000, 21000};
    long evening[3] = {136500, 45000, 60000};
```

Kode 6.5 (Lanjutan) *array_functions.c: fungsi yang menerima array beserta ukurannya; const berjanji tidak mengubahnya, dan add_to_all mengubah array milik pemanggil.*

```c
    printf("morning: ");
    print_all(morning, 5);
    printf("total %ld, average %.2f, biggest at %d\n", total(morning, 5),
           average(morning, 5), index_of_max(morning, 5));
    printf("evening: total %ld, average %.2f, biggest at %d\n",
           total(evening, 3), average(evening, 3), index_of_max(evening, 3));

    add_to_all(evening, 3, 5000);        /* ongkos antar tiap kantong sore */
    printf("evening after delivery fee: ");
    print_all(evening, 3);
    return 0;
}
```

Dua fakta tentang parameter array mengejutkan setiap orang sekali.

Array tidak disalin. Bab 5 menyatakan bahwa C menyalin setiap argumen ke parameternya. Untuk array, C hanya meneruskan alamat elemen pertamanya, seolah-olah pemanggilannya `total(&morning[0], 5)`. Fungsi bekerja pada kotak-kotak milik pemanggil sendiri. Cara ini cepat (menyalin seribu tagihan pada setiap pemanggilan akan lambat), dan berarti fungsi *dapat*

mengubah array pemanggil, seperti yang dilakukan `add_to_all` ketika menambahkan ongkos antar Rp5.000 ke setiap tagihan sore. Bab 7 menjelaskan mekanismenya, yaitu pointer.

Fungsi tidak tahu panjangnya. Karena yang tiba hanyalah alamat, `sizeof a` di dalam fungsi memberikan ukuran sebuah alamat (delapan bita di kebanyakan komputer), bukan ukuran array, dan `gcc -Wall` memperingatkan jika Anda mencobanya. Itulah sebabnya setiap fungsi di sini menerima `n` sebagai parameter kedua, dan mengapa fungsi yang sama dapat melayani lima elemen pagi dan tiga elemen sore.

Kata kunci `const` pada `const long a[]` adalah janji bahwa fungsi hanya membaca array. Jika badan fungsi mencoba `a[0] = 0;`, kompilator akan menolak. Pasanglah `const` pada setiap parameter array yang hanya dibaca: ia mendokumentasikan maksud, memungkinkan pemanggil meneruskan array yang tidak boleh berubah, dan mengubah penugasan yang tak disengaja menjadi galat kompilasi alih-alih kesalahan tersembunyi.

Cobalah sekali untuk melihat janji itu ditegakkan: tambahkan baris `a[0] = 0;` ke dalam `total` lalu kompilasi. `gcc` berhenti dengan pesan *error: assignment of read-only location* sambil menunjuk baris itu. Kesalahan yang sebenarnya akan mengubah data Mbak Rina secara diam-diam sudah tertangkap sebelum programnya sempat ada.

Dalam Python, **Kode 6.6**, list diteruskan dengan cara yang sama: fungsi menerima list milik pemanggil itu sendiri, bukan salinan, sehingga `add_to_all` mengubahnya. Python tidak memiliki `const`; fungsi yang tidak boleh mengubah list cukup ditulis agar tidak mengubahnya, atau membuat salinannya sendiri dengan `list(a)`.

Kode 6.6 `array_functions.py`: fungsi-fungsi yang sama atas list Python.

```
# array_functions.py - menyerahkan list ke fungsi


def total(a):
    s = 0
    for x in a:
        s += x
    return s


def average(a):
    return total(a) / len(a) if a else 0.0


def index_of_max(a):
    best = 0
    for i in range(1, len(a)):
        if a[i] > a[best]:
            best = i
    return best


def add_to_all(a, amount):              # mengubah list milik PEMANGGIL
    for i in range(len(a)):
        a[i] += amount


def print_all(a):
    print(" ".join(str(x) for x in a))


def main():
    morning = [28000, 30000, 30000, 70000, 21000]
    evening = [136500, 45000, 60000]

    print("morning: ", end="")
    print_all(morning)
    print(f"total {total(morning)}, average {average(morning):.2f}, "
          f"biggest at {index_of_max(morning)}")
    print(f"evening: total {total(evening)}, average {average(evening):.2f}, "
          f"biggest at {index_of_max(evening)}")

    add_to_all(evening, 5000)           # ongkos antar tiap kantong sore
    print("evening after delivery fee: ", end="")
    print_all(evening)
```

Kode 6.6 (Lanjutan) *array_functions.py: fungsi-fungsi yang sama atas list Python.*

```
if __name__ == "__main__":
    main()
```

```
morning: 28000 30000 30000 70000 21000
total 179000, average 35800.00, biggest at 3
evening: total 241500, average 80500.00, biggest at 0
evening after delivery fee: 141500 50000 65000
```

6.6 Pengurutan: Bubble Sort

Pengurutan (*sorting*) adalah menyusun benda menurut urutan: tagihan dari kecil ke besar, nama dari A sampai Z. Ketika Knuth menulis jilid bukunya tentang pengurutan, para pembuat komputer memperkirakan bahwa seperempat seluruh waktu jalan komputer habis untuk mengurutkan [5],

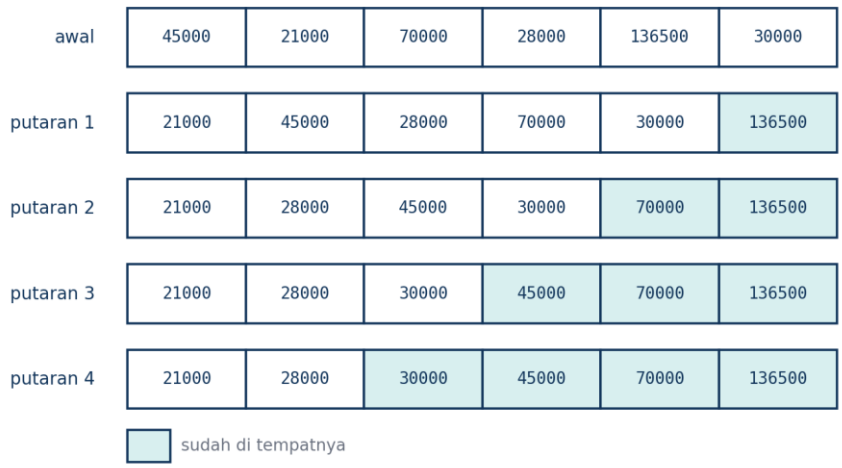
dan walaupun porsinya sudah berubah, data yang terurut tetap membuat hampir segala hal lain lebih mudah: median adalah bagian tengah daftar terurut, tiga teratas adalah ujungnya, dan daftar terurut dapat dicari dalam beberapa langkah saja, bukan banyak langkah.

Bubble sort adalah cara yang paling mudah dipahami. Berjalanlah dari kiri ke kanan dan bandingkan setiap pasangan bertetangga; ketika sebuah pasangan salah urutan, tukarlah. Setelah satu *putaran* (*pass*) seperti itu, nilai terbesar telah terbawa, tukar demi tukar, sampai ke ujung kanan, seperti gelembung yang naik ke permukaan, dan sudah berada di tempat akhirnya. Ulangi putaran pada bagian yang belum di tempatnya. Setelah $n - 1$ putaran semua nilai terurut; dan jika satu putaran penuh tidak melakukan satu tukaran pun, array sudah terurut dan kita boleh berhenti lebih awal. **Pseudokode 6.1** menuliskan caranya dalam pseudokode, bahasa langkah demi langkah dari Bab 2.

Pseudokode 6.1 Bubble sort dengan keluar lebih awal: setiap putaran membawa nilai terbesar yang tersisa ke ujung bagian yang belum terurut.

```
BUBBLE-SORT(A, n)
1  for pass = 1 to n - 1
2      swapped = FALSE
3      for i = 0 to n - pass - 1
4          if A[i] > A[i + 1]
5              exchange A[i] with A[i + 1]
6              swapped = TRUE
7      if not swapped
8          return            // tak ada tukaran satu putaran: terurut
```

Gambar 6.2 memperlihatkan putaran-putaran itu pada enam tagihan dari meja kasir. Setelah putaran 1, tagihan terbesar, 136500, berada di ujung kanan. Setelah putaran 2, 70000 berada di sebelahnya. Putaran 4 tidak menukar apa pun, dan cara ini berhenti tanpa putaran kelima.



Gambar 6.2 Bubble sort pada enam tagihan: setelah setiap putaran satu nilai lagi (berarsir) berada di tempat akhirnya di kanan; putaran 4 tidak menukar apa pun, sehingga pengurutan berhenti lebih awal.

Ikutilah putaran 1 perbandingan demi perbandingan, karena setiap cara pengurutan dibangun dari langkah-langkah kecil seperti ini. Array dimulai sebagai 45000, 21000, 70000, 28000, 136500, 30000. Bandingkan posisi 0 dan 1: 45000 lebih besar daripada 21000, jadi tukar, menghasilkan 21000, 45000, 70000, 28000, 136500, 30000. Bandingkan posisi 1 dan 2: 45000 dan 70000 sudah berurutan, tidak ditukar. Bandingkan 2 dan 3: 70000 lebih besar daripada 28000, tukar, menghasilkan 21000, 45000, 28000, 70000, 136500, 30000. Bandingkan 3 dan 4: 70000 dan 136500 sudah berurutan. Bandingkan 4 dan 5: 136500 lebih besar daripada 30000,

tukar, menghasilkan 21000, 45000, 28000, 70000, 30000, 136500. Lima perbandingan dan tiga tukaran, dan tagihan terbesar telah berpindah dari posisi 4 ke ujung. Perhatikan bahwa 136500 tidak bergerak sampai perbandingan terakhir, lalu hanya bergeser satu tempat: nilai besar terbawa ke kanan dalam satu putaran, tetapi nilai kecil di dekat ujung, seperti 30000, hanya bergeser ke kiri satu tempat per putaran. Ketidakseimbangan itulah yang membuat bubble sort memerlukan banyak putaran ketika nilai-nilai kecil berada di kanan.

> **WAWASAN KUNCI:** Setelah putaran p, p posisi terakhir memuat p nilai terbesar, dalam urutan akhirnya. Fakta ini, yang benar setelah setiap putaran, adalah *invarian perulangan* (*loop invariant*) bubble sort, dan menjelaskan dua hal sekaligus: mengapa perulangan dalam boleh berhenti di `n - pass` (ekornya sudah benar), dan mengapa $n - 1$ putaran selalu cukup (sesudahnya $n - 1$ posisi terakhir sudah benar, sehingga posisi pertama pun pasti benar).

Seberapa besar kerjanya? Putaran 1 membandingkan $n - 1$ pasangan, putaran 2 membandingkan $n - 2$, dan seterusnya sampai 1. Pada kasus terburuk, daftar dengan urutan terbalik, setiap putaran diperlukan dan banyaknya perbandingan adalah

$$(n-1) + (n-2) + \cdots + 1 = \frac{n(n-1)}{2} \tag{6.3}$$

Untuk 6 tagihan di **Gambar 6.2** itu paling banyak 15 perbandingan, tidak berbahaya. Untuk pesanan setahun, katakanlah 20.000, Persamaan (6.3) memberikan hampir 200 juta perbandingan, yang masih dikerjakan komputer modern dalam waktu jauh di bawah satu detik, tetapi bertambah empat kali lipat setiap kali daftarnya menjadi dua kali lebih panjang. Bubble sort adalah cara yang tepat untuk dipelajari pertama kali dan hampir tidak pernah tepat untuk data besar; sejarah algoritma ini yang ditulis Astrachan adalah uraian yang hidup tentang mengapa ia tetap diajarkan [6]. Cara yang lebih baik, seperti merge sort dan quicksort, memerlukan sekitar $n \log_2 n$ perbandingan; fungsi pustaka `qsort` di Bab 7 dan `sorted` milik Python memakai cara semacam itu [7].

Satu lagi sifat bubble sort penting dalam praktik. Metode ini menukar dua tetangga hanya ketika yang kiri *lebih besar*, tidak ketika sama besar, sehingga dua nilai yang sama tidak pernah bertukar tempat: dua tagihan Rp30.000 dari pelanggan yang berbeda tetap dalam urutan nota yang dibuat. Pengurutan yang memiliki sifat ini disebut **stabil** (*stable*). Kestabilan menjadi penting begitu data membawa lebih dari sekadar kunci yang diurutkan: urutkan pesanan menurut tagihan, dan pengurutan yang stabil mempertahankan pesanan bertagihan sama dalam urutan nota, persis yang diharapkan pembaca laporan. Insertion sort juga stabil; `sorted` milik Python dijamin stabil [7]; `qsort` milik C tidak.

6.7 Pencarian: Linear dan Biner

Untuk menemukan sebuah nilai dalam array yang *tidak* terurut, hanya ada satu cara: lihat elemennya satu per satu dari depan sampai ketemu atau habis. Itulah **pencarian linear**, fungsi `linear_search` di **Kode 6.7**. Fungsi ini mengembalikan indeks tempat nilai ditemukan, atau -1, yang tidak mungkin menjadi indeks, untuk berkata "tidak ada". Pada kasus terburuk ia melihat seluruh n elemen.

Dalam array yang *terurut* ada cara yang jauh lebih baik, cara yang Anda pakai saat membuka kamus. Bukalah di tengah. Jika kata yang Anda cari datang sebelum kata di tengah, kata itu ada di separuh kiri, dan Anda tidak perlu melihat separuh kanan lagi; jika tidak, kata itu ada di separuh kanan. Setiap tengokan membuang separuh dari yang tersisa. Itulah **pencarian biner** (*binary*

search). Variabel `low` dan `high` menandai bagian yang masih dipertanyakan, `mid` adalah tengahnya, dan perulangan berakhir ketika nilai ditemukan atau bagian itu kosong (`low > high`).

Kode 6.7 `sort_search.c`: bubble sort dengan jejak setiap putaran yang dicetak, pencarian linear, dan pencarian biner dengan jejak setiap tengokan yang dicetak.

```c
/* sort_search.c - bubble sort dengan jejak, lalu pencarian linear dan biner */
#include <stdio.h>

void print_all(const long a[], int n)
{
    for (int i = 0; i < n; i++)
        printf("%7ld", a[i]);
    printf("\n");
}

void bubble_sort(long a[], int n)
{
    for (int pass = 1; pass < n; pass++) {
        int swapped = 0;
        for (int i = 0; i < n - pass; i++) {    /* ekornya sudah terurut */
            if (a[i] > a[i + 1]) {
                long t = a[i];
                a[i] = a[i + 1];
                a[i + 1] = t;
                swapped = 1;
            }
        }
        printf("pass %d:", pass);
        print_all(a, n);
        if (!swapped)                           /* tak ada yang pindah: selesai */
            break;
    }
}

int linear_search(const long a[], int n, long key)
{
    for (int i = 0; i < n; i++)
        if (a[i] == key)
            return i;
    return -1;                                  /* -1 berarti "tidak ada" */
}
```

Kode 6.7 (Lanjutan) *sort_search.c: bubble sort dengan jejak setiap putaran yang dicetak, pencarian linear, dan pencarian biner dengan jejak setiap tengokan yang dicetak.*

```c
int binary_search(const long a[], int n, long key)   /* a harus terurut */
{
    int low = 0, high = n - 1;
    while (low <= high) {
        int mid = low + (high - low) / 2;
        printf("  low=%d mid=%d high=%d looks at %ld\n", low, mid, high, a[mid]);
        if (a[mid] == key)
            return mid;
        if (a[mid] < key)
            low = mid + 1;                      /* kunci ada di separuh kanan */
        else
            high = mid - 1;                     /* kunci ada di separuh kiri */
    }
    return -1;
}

int main(void)
{
    long bill[6] = {45000, 21000, 70000, 28000, 136500, 30000};
    printf("start: ");
    print_all(bill, 6);
    printf("linear search for 28000: index %d\n", linear_search(bill, 6, 28000));

    bubble_sort(bill, 6);
    printf("binary search for 70000:\n");
    printf("found at index %d\n", binary_search(bill, 6, 70000));
    printf("binary search for 50000:\n");
    printf("found at index %d\n", binary_search(bill, 6, 50000));
    return 0;
}
```

Titik tengah dihitung sebagai `low + (high - low) / 2`, bukan `(low + high) / 2`. Keduanya sama dalam matematika, tetapi untuk array yang sangat besar `low + high` dapat melampaui `int` terbesar, dan bentuk kedua lalu menghasilkan indeks negatif. Kesalahan ini bersembunyi di pencarian biner pustaka standar Java selama sembilan tahun sebelum Joshua Bloch melaporkannya pada 2006 [8]; bentuk pertama tidak pernah meluap.

Kode 6.8 adalah versi Python. Menukar dua elemen cukup satu baris di Python, `a[i], a[i + 1] = a[i + 1], a[i]`, sedangkan C memerlukan variabel sementara `t`.

Kode 6.8 `sort_search.py`: pengurutan dan pencarian yang sama dalam Python.

```python
# sort_search.py - bubble sort dengan jejak, lalu pencarian linear dan biner


def print_all(a):
    print("".join(f"{x:7d}" for x in a))


def bubble_sort(a):
    n = len(a)
    for pass_no in range(1, n):
        swapped = False
        for i in range(n - pass_no):         # ekornya sudah terurut
            if a[i] > a[i + 1]:
                a[i], a[i + 1] = a[i + 1], a[i]
                swapped = True
        print(f"pass {pass_no}:", end="")
        print_all(a)
        if not swapped:                      # tak ada yang pindah: selesai
            break


def linear_search(a, key):
    for i in range(len(a)):
        if a[i] == key:
            return i
    return -1                                # -1 berarti "tidak ada"


def binary_search(a, key):                   # a harus terurut
    low, high = 0, len(a) - 1
    while low <= high:
        mid = low + (high - low) // 2
        print(f"  low={low} mid={mid} high={high} looks at {a[mid]}")
        if a[mid] == key:
            return mid
        if a[mid] < key:
            low = mid + 1                    # kunci ada di separuh kanan
        else:
            high = mid - 1                   # kunci ada di separuh kiri
    return -1
```

Kode 6.8 (Lanjutan) *sort_search.py: pengurutan dan pencarian yang sama dalam Python.*

```python
def main():
    bill = [45000, 21000, 70000, 28000, 136500, 30000]
    print("start: ", end="")
    print_all(bill)
    print(f"linear search for 28000: index {linear_search(bill, 28000)}")

    bubble_sort(bill)
    print("binary search for 70000:")
    print(f"found at index {binary_search(bill, 70000)}")
    print("binary search for 50000:")
    print(f"found at index {binary_search(bill, 50000)}")


if __name__ == "__main__":
    main()
```

```
start:   45000  21000  70000  28000 136500  30000
linear search for 28000: index 3
pass 1:  21000  45000  28000  70000  30000 136500
pass 2:  21000  28000  45000  30000  70000 136500
pass 3:  21000  28000  30000  45000  70000 136500
pass 4:  21000  28000  30000  45000  70000 136500
binary search for 70000:
  low=0 mid=2 high=5 looks at 30000
  low=3 mid=4 high=5 looks at 70000
found at index 4
binary search for 50000:
  low=0 mid=2 high=5 looks at 30000
  low=3 mid=4 high=5 looks at 70000
  low=3 mid=3 high=3 looks at 45000
found at index -1
```

Keluarannya memperlihatkan putaran-putaran **Gambar 6.2** lalu dua pencarian biner dalam array terurut. Untuk 70000, tengokan pertama, di indeks 2, melihat 30000, yang lebih kecil, sehingga pencarian berlanjut di separuh kanan, indeks 3 sampai 5; tengokan kedua menemukan 70000 di indeks 4. Untuk 50000, yang tidak ada, tengokan ketiga mempersempit bagian itu menjadi indeks tunggal 3, melihat 45000, dan langkah berikutnya akan membuat `low` lebih besar daripada `high`: tidak ditemukan.

Berapa tengokan yang paling banyak diperlukan pencarian biner? Setiap tengokan paling sedikit membagi dua bagian yang tersisa. Mulai dari n elemen, setelah k tengokan paling banyak tersisa $n/2^k$, dan pencarian harus berhenti ketika yang tersisa kurang dari satu. Maka banyaknya tengokan terbesar adalah

$$k_{\max} = \lfloor \log_2 n \rfloor + 1 \tag{6.4}$$

dengan $\log_2 n$ adalah pangkat yang harus dikenakan pada 2 untuk menghasilkan n, dan $\lfloor \cdot \rfloor$ membulatkan ke bawah. Untuk $n = 6$, $\log_2 6 \approx 2.58$, sehingga Persamaan (6.4) memberikan 3 tengokan, persis yang diperlukan pencarian 50000. Untuk sejuta nota, hasilnya 20 tengokan, dibanding sampai sejuta untuk pencarian linear. Itulah upah dari mengurutkan lebih dulu, dan itulah sebabnya buku telepon, kamus, dan indeks buku ini terurut [9].

Pencarian mana yang sebaiknya dipakai program? Jawabannya bergantung pada seberapa sering data berubah dan seberapa sering data dicari. Mengurutkan jauh lebih mahal daripada satu kali pencarian linear, sehingga untuk daftar yang hanya dicari sekali, pencarian linear yang menang. Untuk daftar yang diurutkan sekali lalu dicari ribuan kali, seperti daftar harga supermarket atau nomor nota sebulan, mengurutkan sekali lalu mencari dengan pencarian biner menang telak. Kasir Mbak Rina mencari sebuah nota barangkali lima kali sehari di antara lima puluh nota: pencarian linear adalah pilihan yang jujur, dan kode yang lebih sederhana lebih berharga daripada beberapa mikrodetik yang dihemat. Pada hari laundry itu menyimpan nota setahun dan seorang pelanggan bertanya “di mana pesanan saya nomor 18342?”, pencarian biner atas nomor nota yang terurut menjadi alat yang tepat, dan Bab 8 membangun persis itu.

6.8 Array Dua Dimensi

Pertanyaan terakhir Mbak Rina mencakup seluruh minggu: kilogram yang dicuci per hari dan per layanan. Datanya membentuk *tabel*, tujuh baris untuk hari dan tiga kolom untuk layanan. C menyimpan tabel dalam **array dua dimensi**, yang dideklarasikan dengan dua ukuran: `int kg[7][3];`. Elemen di baris `d` dan kolom `s` adalah `kg[d][s]`.

Kode 6.9 mengisi tabel dengan inisialisator (satu pasang kurung kurawal di dalam untuk setiap baris), mencetaknya dengan total setiap baris, menambahkan baris berisi total setiap kolom, dan mencari hari tersibuk. Nama-nama hari adalah array karakter dua dimensi, `char day[7][4]`: tujuh baris berisi empat karakter, cukup untuk tiga huruf dan `'\0'` yang mengakhiri setiap string di C (Bab 9 menjelaskan string selengkapnya).

Kode 6.9 `week_table.c`: kilogram per hari dan layanan dalam array dua dimensi, dengan total baris, total kolom, dan hari tersibuk.

```c
/* week_table.c - array dua dimensi: kilogram per hari dan layanan */
#include <stdio.h>

#define DAYS 7
#define SERVICES 3

int main(void)
{
    char day[DAYS][4] = {"Mon", "Tue", "Wed", "Thu", "Fri", "Sat", "Sun"};
    int kg[DAYS][SERVICES] = {          /* baris: hari; kolom: layanan 1..3 */
        {42, 18, 10}, {38, 20, 12}, {40, 15, 9}, {45, 22, 11},
        {51, 30, 14}, {66, 41, 20}, {58, 35, 16}
    };
    int col_total[SERVICES] = {0};
    int grand = 0, busiest = 0, busiest_kg = -1;

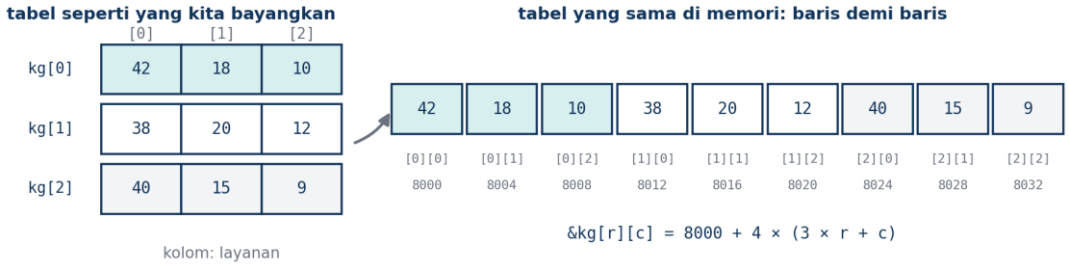
    printf("day   fold  iron  only  total\n");
    for (int d = 0; d < DAYS; d++) {
        int row_total = 0;
        printf("%s ", day[d]);
        for (int s = 0; s < SERVICES; s++) {
            printf("%6d", kg[d][s]);
            row_total += kg[d][s];
            col_total[s] += kg[d][s];
        }
        printf("%7d\n", row_total);
        grand += row_total;
        if (row_total > busiest_kg) {
            busiest_kg = row_total;
            busiest = d;
        }
    }
    printf("all ");
    for (int s = 0; s < SERVICES; s++)
        printf("%6d", col_total[s]);
    printf("%7d\n", grand);
    printf("busiest day: %s with %d kg\n", day[busiest], busiest_kg);
    return 0;
}
```

Pola dua perulangan bersarang, yang luar atas baris dan yang dalam atas kolom, adalah cara baku mengunjungi setiap sel tabel. Total baris diatur ulang untuk setiap baris di dalam perulangan luar; total kolom adalah array tiga penghitung yang hidup melintasi semua baris.

Bagaimana tabel disimpan di memori, yang hanya berupa satu deretan panjang bita? Baris demi baris: mula-mula tiga elemen baris 0, lalu tiga elemen baris 1, dan seterusnya. Urutan ini disebut *row-major* (utama-baris), dan **Gambar 6.3** menggambarkannya untuk tiga hari pertama. Alamat `kg[r][c]` dalam array dengan C kolom dan elemen berukuran s bita adalah

$$\text{alamat}(\text{kg}[r][c]) = a + (r \cdot C + c) \cdot s \tag{6.5}$$

karena r baris utuh berisi C elemen datang sebelum baris r, dan c elemen datang sebelum kolom c di dalamnya. Itulah sebabnya C harus mengetahui banyaknya kolom parameter array dua dimensi (`int kg[][3]`): tanpa C, Persamaan (6.5) tidak dapat dihitung.



Gambar 6.3 Tiga baris pertama tabel mingguan, seperti yang kita bayangkan dan seperti letaknya di memori: baris demi baris, sehingga `kg[r][c]` terletak $3r + c$ elemen sesudah awal (empat bita per `int`).

Fungsi yang menerima seluruh tabel harus diberi tahu banyaknya kolom pada parameternya, dengan alasan yang baru saja dijelaskan, sedangkan banyaknya baris boleh diteruskan terpisah seperti panjang array satu dimensi: `int row_total(const int kg[][3], int d)` menjumlahkan baris `d`, dan `int column_total(const int kg[][3], int days, int s)` menjumlahkan kolom `s` sepanjang `days` baris. Di dalamnya, `kg[d][s]` bekerja persis seperti di `main`. C99 juga membolehkan banyaknya kolom itu sendiri menjadi parameter, `int total(int rows, int cols, const int t[rows][cols])`, yaitu parameter *variable-length array* (array berpanjang variabel); fitur ini praktis tetapi bersifat opsional sejak C11, sehingga buku ini tetap memakai banyak kolom yang tetap.

Python tidak memiliki array dua dimensi; penggantinya yang lazim adalah **list berisi list**, satu list di dalam untuk setiap baris, dan `kg[d][s]` bekerja dengan cara yang sama, seperti diperlihatkan **Kode 6.10**.

Kode 6.10 `week_table.py`: tabel yang sama sebagai list berisi list di Python.

```
# week_table.py - list berisi list: kilogram per hari dan layanan


def main():
    day = ["Mon", "Tue", "Wed", "Thu", "Fri", "Sat", "Sun"]
    kg = [                                  # baris: hari; kolom: layanan 1..3
        [42, 18, 10], [38, 20, 12], [40, 15, 9], [45, 22, 11],
        [51, 30, 14], [66, 41, 20], [58, 35, 16],
    ]
    col_total = [0] * 3
    grand, busiest, busiest_kg = 0, 0, -1

    print("day   fold  iron  only  total")
    for d in range(7):
        row_total = 0
        line = f"{day[d]} "
        for s in range(3):
            line += f"{kg[d][s]:6d}"
            row_total += kg[d][s]
            col_total[s] += kg[d][s]
        print(f"{line}{row_total:7d}")
        grand += row_total
        if row_total > busiest_kg:
            busiest_kg, busiest = row_total, d
    print("all " + "".join(f"{c:6d}" for c in col_total) + f"{grand:7d}")
    print(f"busiest day: {day[busiest]} with {busiest_kg} kg")


if __name__ == "__main__":
    main()
```

```
day   fold  iron  only  total
Mon     42    18    10     70
Tue     38    20    12     70
Wed     40    15     9     64
Thu     45    22    11     78
Fri     51    30    14     95
Sat     66    41    20    127
Sun     58    35    16    109
all    340   181    92    613
busiest day: Sat with 127 kg
```

Sabtu adalah hari tersibuk dengan 127 kg, disusul Minggu, sehingga pembantu paruh waktu diperlukan di akhir pekan. Sepanjang minggu, cuci lipat menyumbang 340 dari 613 kg, lebih dari separuh, cocok dengan hitungan Subbab 6.4.

> **PERINGATAN:** Di Python, jangan pernah membuat tabel dengan `[[0] * 3] * 7`. Ekspresi itu membuat *satu* list di dalam dan tujuh referensi kepadanya, sehingga mengubah `kg[0][0]` mengubah sel pertama setiap baris. Tulislah `[[0] * 3 for _ in range(7)]`, yang membuat tujuh baris terpisah [10].

6.9 Membangun Kasir Versi 5

Sekarang kasir belajar mengingat. Versi 5 membaca pesanan hari itu ke dalam array lalu menjawab pertanyaan-pertanyaan malam. Versi ini memakai ulang `laundry.h` dan `laundry.c` dari Bab 5 tanpa perubahan (harga, aturan penagihan, dan pencetak rupiah) dan menambahkan modul kecil kedua untuk array tagihan. **Kode 6.11** adalah header-nya, dan **Kode 6.12** kodenya.

Kode 6.11 `bills.h`: janji-janji modul untuk array tagihan.

```c
/* bills.h - alat kecil untuk array tagihan (Bab 6) */
#ifndef BILLS_H
#define BILLS_H

long total(const long a[], int n);
int index_of_max(const long a[], int n);
void copy_bills(const long from[], long to[], int n);
void sort_descending(long a[], int n);
long median(const long a[], int n);      /* 0 untuk array kosong */

#endif
```

Kode 6.12 `bills.c`: total, indeks maksimum, penyalinan, pengurutan dari terbesar ke terkecil, dan median.

```c
/* bills.c - alat kecil untuk array tagihan */
#include "bills.h"

#define MAX_COPY 1000

long total(const long a[], int n)
{
    long sum = 0;
    for (int i = 0; i < n; i++)
        sum += a[i];
    return sum;
}

int index_of_max(const long a[], int n)
{
    int best = 0;
    for (int i = 1; i < n; i++)
        if (a[i] > a[best])
            best = i;
    return best;
}

void copy_bills(const long from[], long to[], int n)
{
    for (int i = 0; i < n; i++)
        to[i] = from[i];
}

void sort_descending(long a[], int n)    /* bubble sort, terbesar dulu */
{
    for (int pass = 1; pass < n; pass++) {
        int swapped = 0;
        for (int i = 0; i < n - pass; i++) {
            if (a[i] < a[i + 1]) {
                long t = a[i];
                a[i] = a[i + 1];
                a[i + 1] = t;
                swapped = 1;
            }
        }
        if (!swapped)
            break;
    }
}
```

Kode 6.12 (Lanjutan) *bills.c: total, indeks maksimum, penyalinan, pengurutan dari terbesar ke terkecil, dan median.*

```
long median(const long a[], int n)
{
    long sorted[MAX_COPY];
    if (n <= 0 || n > MAX_COPY)
        return 0;
    copy_bills(a, sorted, n);            /* jangan urutkan data pemanggil */
    sort_descending(sorted, n);
    if (n % 2 == 1)
        return sorted[n / 2];
    return (sorted[n / 2 - 1] + sorted[n / 2]) / 2;
}
```

Median adalah nilai tengah data yang terurut: separuh tagihan paling sedikit sebesar itu, separuh lagi paling banyak sebesar itu. Untuk banyak nilai ganjil, median adalah nilai di tengah; untuk banyak nilai genap, median adalah rerata dua nilai tengah, di sini dalam rupiah bulat. Fungsi `median` mengurutkan *salinan*, karena fungsi yang tugasnya menjawab pertanyaan tidak boleh mengubah urutan data pemanggil sebagai efek samping: urutan tagihan adalah urutan nota, dan laporan nanti mencetak pesanan terbesar menurut nomor notanya.

Laporannya sendiri, **Kode 6.13**, menyimpan pesanan dalam tiga **array paralel**: `service[i]`, `weight[i]`, dan `bill[i]` bersama-sama menggambarkan pesanan nomor *i*. Perulangan membaca paling banyak `MAX_ORDERS` pesanan, yang melindungi array dari hari yang ramai di luar dugaan. Total per layanan adalah array yang diindeks dengan kode layanan, pola penghitungan Subbab 6.4 yang diterapkan sekaligus pada banyak pesanan, kilogram, dan uang.

Kode 6.13 `day_report.c`: kasir versi 5, yang menyimpan hari itu dalam array paralel dan mencetak laporan tutup toko.

```c
/* day_report.c - kasir Profitina Laundry versi 5: ia mengingat hari itu.
   Masukan: satu pesanan per baris, "service weight express", diakhiri 0. */
#include <stdio.h>
#include "laundry.h"
#include "bills.h"

#define MAX_ORDERS 100

int main(void)
{
    int service[MAX_ORDERS];            /* array paralel: pesanan i adalah */
    double weight[MAX_ORDERS];          /* service[i], weight[i], bill[i] */
    long bill[MAX_ORDERS];
    int n = 0, s, express;
    double w;

    while (n < MAX_ORDERS && scanf("%d", &s) == 1 && s != 0) {
        if (scanf("%lf %d", &w, &express) != 2)
            break;
        if (price_per_kg(s) == 0 || w <= 0 || w > MAX_KG)
            continue;                   /* kasir v4 sudah menyebut alasannya */
        service[n] = s;
        weight[n] = w;
        bill[n] = bill_for(s, w, express);
        n++;
    }
    if (n == 0) {
        printf("no orders today\n");
        return 0;
    }

    int count[4] = {0};
    double kg[4] = {0};
    long revenue[4] = {0};
    for (int i = 0; i < n; i++) {
        count[service[i]]++;
        kg[service[i]] += weight[i];
        revenue[service[i]] += bill[i];
    }
```

Kode 6.13 (Lanjutan) *day_report.c: kasir versi 5, yang menyimpan hari itu dalam array paralel dan mencetak laporan tutup toko.*

```
    printf("PROFITINA LAUNDRY - day report (v5)\n");
    printf("service       orders      kg  revenue\n");
    for (int k = 1; k <= 3; k++) {
        print_service(k);
        printf("  %5d  %6.1f  ", count[k], kg[k]);
        print_rupiah(revenue[k]);
        printf("\n");
    }
    long sum = total(bill, n);
    int best = index_of_max(bill, n);
    printf("orders %d, revenue ", n);
    print_rupiah(sum);
    printf(", average ");
    print_rupiah(sum / n);
    printf("\nbiggest: order #%03d, ", best + 1);
    print_rupiah(bill[best]);
    printf("\nmedian bill: ");
    print_rupiah(median(bill, n));

    long top[MAX_ORDERS];
    copy_bills(bill, top, n);
    sort_descending(top, n);
    printf("\ntop three:");
    for (int i = 0; i < 3 && i < n; i++) {
        printf("  ");
        print_rupiah(top[i]);
    }
    printf("\n");
    return 0;
}
```

Bacalah perulangan masukan `day_report.c` pelan-pelan, karena syaratnya mengerjakan tiga tugas dalam urutan yang tetap. `n < MAX_ORDERS` diuji lebih dulu: ketika array sudah penuh, perulangan berhenti *sebelum* `scanf` membaca pesanan berikutnya, sehingga tidak ada pesanan yang terbaca lalu hilang. Berikutnya, `scanf("%d", &s) == 1` gagal di akhir masukan, dan `s != 0` berhenti pada kode penutup. Karena `&&` dievaluasi dari kiri ke kanan dan berhenti pada bagian pertama yang salah (Bab 3), urutan ketiga pengujian itu penting. Di dalam perulangan, pesanan yang tidak sah dilewati dengan `continue` *sebelum* apa pun disimpan, sehingga ketiga array selalu memuat tepat `n` pesanan sah pada posisi yang sama; *invarian* ini, fakta yang tetap benar pada setiap putaran perulangan, itulah yang membuat perulangan-perulangan sesudahnya atas `i < n` aman.

Array paralel bekerja, tetapi rapuh: mengurutkan `bill` saja akan memisahkan setiap tagihan dari layanan dan beratnya. Itulah sebabnya laporan mengurutkan salinan, `top`. Bab 10 mengganti ketiga array itu dengan satu array *struktur*, yang setiap elemennya memuat semua fakta satu pesanan bersama-sama.

Versi Python mengikuti rancangan yang sama. **Kode 6.14** adalah modulnya; `median` menyalin list dengan `list(a)` sebelum mengurutkan.

Kode 6.14 `bills.py`: alat-alat yang sama untuk list Python.

```
# bills.py - alat kecil untuk list tagihan


def total(a):
    s = 0
    for x in a:
        s += x
    return s


def index_of_max(a):
    best = 0
    for i in range(1, len(a)):
        if a[i] > a[best]:
            best = i
    return best


def sort_descending(a):                  # bubble sort, terbesar dulu
    n = len(a)
    for pass_no in range(1, n):
        swapped = False
        for i in range(n - pass_no):
            if a[i] < a[i + 1]:
                a[i], a[i + 1] = a[i + 1], a[i]
                swapped = True
        if not swapped:
            break


def median(a):
    if not a:
        return 0
    s = list(a)                          # jangan urutkan data pemanggil
    sort_descending(s)
    n = len(s)
    if n % 2 == 1:
        return s[n // 2]
    return (s[n // 2 - 1] + s[n // 2]) // 2
```

Kode 6.15 adalah laporannya. Di sini list dimulai kosong dan memanjang dengan `append`, sehingga `n` tidak diperlukan: `len(bill)` selalu sama dengan banyaknya pesanan yang tersimpan.

Kode 6.15 `day_report.py`: kasir versi 5 dalam Python.

```
# day_report.py - kasir Profitina Laundry versi 5: ia mengingat hari itu.
# Masukan: satu pesanan per baris, "service weight express", diakhiri 0.
import sys
from laundry import (MAX_KG, price_per_kg, service_name, bill_for, rupiah)
from bills import total, index_of_max, sort_descending, median

MAX_ORDERS = 100


def main():
    words = sys.stdin.read().split()
    service, weight, bill = [], [], []   # list paralel: pesanan i adalah
    pos = 0                              # service[i], weight[i], bill[i]
    while (len(bill) < MAX_ORDERS and pos < len(words)
           and int(words[pos]) != 0 and pos + 2 < len(words)):
        s, w = int(words[pos]), float(words[pos + 1])
        express = int(words[pos + 2])
        pos += 3
        if price_per_kg(s) == 0 or w <= 0 or w > MAX_KG:
            continue                     # kasir v4 sudah menyebut alasannya
        service.append(s)
        weight.append(w)
        bill.append(bill_for(s, w, express))
    n = len(bill)
    if n == 0:
        print("no orders today")
        return

    count, kg, revenue = [0] * 4, [0.0] * 4, [0] * 4
    for i in range(n):
        count[service[i]] += 1
        kg[service[i]] += weight[i]
        revenue[service[i]] += bill[i]

    print("PROFITINA LAUNDRY - day report (v5)")
    print("service       orders      kg  revenue")
    for k in range(1, 4):
        print(f"{service_name(k)}  {count[k]:5d}  {kg[k]:6.1f}  "
              f"{rupiah(revenue[k])}")
    s = total(bill)
    best = index_of_max(bill)
    print(f"orders {n}, revenue {rupiah(s)}, average {rupiah(s // n)}")
    print(f"biggest: order #{best + 1:03d}, {rupiah(bill[best])}")
    print(f"median bill: {rupiah(median(bill))}")
```

Kode 6.15 (Lanjutan) *day_report.py: kasir versi 5 dalam Python.*

```
    top = list(bill)
    sort_descending(top)
    print("top three:" + "".join("  " + rupiah(x) for x in top[:3]))


if __name__ == "__main__":
    main()
```

6.10 Menjalankannya

Satu hari Sabtu di meja kasir, dengan dua belas pesanan, satu di antaranya berlayanan tak dikenal 7:

```
1 3.4 0
2 1.2 1
3 5 0
2 6.5 0
1 12.75 1
1 4.2 0
3 2.5 0
2 3 1
1 7.9 0
7 3 0
3 1 0
1 5.5 1
0
```

```
PROFITINA LAUNDRY - day report (v5)
service       orders      kg  revenue
Wash & fold        5    33.8  Rp 318.500
Wash & iron        3    10.7  Rp 145.000
Iron only          3     8.5  Rp 60.000
orders 11, revenue Rp 523.500, average Rp 47.590
biggest: order #005, Rp 136.500
median bill: Rp 35.000
top three:  Rp 136.500  Rp 70.000  Rp 63.000
```

Periksalah laporan itu dengan tangan. Lima pesanan cuci lipat beratnya 3,4 + 12,75 + 4,2 + 7,9 + 5,5 = 33,75 kg, tercetak dengan satu desimal sebagai 33.8. Tagihannya Rp28.000, 136.500, 35.000 (4,2 kg ditagih 5 kg), 56.000 (8 kg), dan 63.000 (6 kg dengan harga Rp7.000 ditambah separuhnya untuk ekspres), yang berjumlah Rp318.500. Sebelas pesanan sah menghasilkan Rp523.500; dibagi 11 hasilnya 47590,9, dan laporan mencetak Rp47.590 karena `sum / n` adalah pembagian bilangan bulat. Diurutkan dari yang terbesar, tagihannya 136500, 70000, 63000, 56000, 45000, 35000, 30000, 30000, 28000, 18000, 12000; yang keenam dari sebelas adalah median, Rp35.000.

Dua layanan lainnya diperiksa dengan cara yang sama. Cuci setrika memiliki tiga pesanan: 1,2 kg ekspres (minimum 2 kg dengan harga Rp10.000 adalah Rp20.000, ditambah separuhnya, Rp30.000), 6,5 kg (7 kg, Rp70.000), dan 3 kg ekspres (Rp30.000 ditambah separuhnya, Rp45.000), seluruhnya Rp145.000 untuk 10,7 kg. Setrika saja memiliki 5 kg (Rp30.000), 2,5 kg (3 kg, Rp18.000), dan 1 kg (minimum, Rp12.000), seluruhnya Rp60.000 untuk 8,5 kg. Ketiga pendapatan itu berjumlah 318500 + 145000 + 60000 = 523500, total hari itu, dan hitungannya 5 + 3 + 3 sama dengan sebelas pesanan: bagian-bagiannya cocok dengan keseluruhan. Pesanan berlayanan 7 tidak muncul di mana pun, sebagaimana mestinya. Laporan yang bagian-bagiannya tidak berjumlah sama dengan keseluruhannya pasti mengandung kesalahan, dan pemeriksaan silang ini adalah cara tercepat untuk menemukannya.

Rata-rata Rp47.590 lebih tinggi daripada tujuh dari sebelas tagihan. Median Rp35.000 lebih tepat menggambarkan berapa yang dibayar pelanggan biasa, karena satu pesanan Rp136.500 menarik rata-rata ke atas tetapi hanya menggeser median satu tempat. Ketika Mbak Rina ingin tahu "berapa belanja pelanggan yang biasa?", median adalah jawaban yang jujur; ketika ia ingin tahu "berapa uang masuk per pesanan?", rata-ratalah jawabannya.

6.11 Mengujinya

Kode 6.16 menguji modul pada array kecil yang jawabannya jelas: lima nilai yang totalnya 150 dan nilai tengahnya 30, array satu elemen, dan array berpanjang genap yang mediannya rerata 2 dan 3 dalam bilangan bulat, yaitu 2. Dua pemeriksaan menyangkut perilaku, bukan hasil: setelah `median(a, 5)` array `a` harus tidak berubah, dan hari tanpa pesanan (`n` = 0) harus memberi total 0, bukan berhenti mendadak.

Kode 6.16 `test_bills.c`: sembilan pemeriksaan alat array, termasuk array kosong dan janji bahwa `median` tidak mengubah masukannya.

```c
/* test_bills.c - menguji alat array pada array kecil yang dihitung tangan */
#include <stdio.h>
#include "bills.h"

static int checks = 0, failures = 0;

static void check(long got, long expected, int line)
{
    checks++;
    if (got != expected) {
        failures++;
        printf("FAILED on line %d: got %ld, expected %ld\n", line, got, expected);
    }
}

int main(void)
{
    long a[5] = {30, 10, 50, 20, 40};
    long one[1] = {7};
    long even[4] = {4, 1, 3, 2};

    check(total(a, 5), 150, __LINE__);
    check(total(a, 0), 0, __LINE__);           /* hari tanpa pesanan */
    check(index_of_max(a, 5), 2, __LINE__);
    check(index_of_max(one, 1), 0, __LINE__);
    check(median(a, 5), 30, __LINE__);
    check(median(even, 4), 2, __LINE__);       /* (3 + 2) / 2 dalam rupiah bulat */
    check(a[0], 30, __LINE__);                 /* median tidak menyentuh a */
    sort_descending(a, 5);
    check(a[0], 50, __LINE__);
    check(a[4], 10, __LINE__);
    printf("%d checks, %d failures\n", checks, failures);
    return failures != 0;
}
```

Kode 6.17 adalah uji yang sama dalam Python. Hari tanpa pesanan ditulis `total([])`, list kosong, dan janji tentang masukan diperiksa dengan cara yang sama, yaitu dengan melihat `a[0]` setelah pemanggilan.

Kode 6.17 `test_bills.py`: pemeriksaan yang sama dalam Python.

```
# test_bills.py - menguji alat list pada list kecil yang dihitung tangan
import inspect
import sys
from bills import total, index_of_max, sort_descending, median

checks = 0
failures = 0


def check(got, expected):
    global checks, failures
    checks += 1
    if got != expected:
        failures += 1
        line = inspect.stack()[1].lineno
        print(f"FAILED on line {line}: got {got}, expected {expected}")


def main():
    a = [30, 10, 50, 20, 40]
    check(total(a), 150)
    check(total([]), 0)                  # hari tanpa pesanan
    check(index_of_max(a), 2)
    check(index_of_max([7]), 0)
    check(median(a), 30)
    check(median([4, 1, 3, 2]), 2)       # (3 + 2) // 2 dalam rupiah bulat
    check(a[0], 30)                      # median tidak menyentuh a
    sort_descending(a)
    check(a[0], 50)
    check(a[4], 10)
    print(f"{checks} checks, {failures} failures")
    return 1 if failures else 0


if __name__ == "__main__":
    sys.exit(main())
```

```
9 checks, 0 failures
```

Perhatikan pula apa yang *tidak* diuji di sini: pencetakan laporan. Keluaran laporan diperiksa dengan cara lain, yaitu dengan membandingkan seluruh keluarannya terhadap berkas keluaran yang diharapkan, persis yang dilakukan `run_tests.py` untuk setiap program bab ini. Kedua jenis uji saling melengkapi: uji fungsi menunjuk dengan tepat fungsi mana yang salah, sedangkan uji keluaran memastikan bahwa semua bagian tersambung dengan benar.

Mengapa menguji array sekecil itu, padahal hari sungguhan berisi puluhan pesanan? Karena array kecil dapat diperiksa di kepala, dan karena kesalahan yang penting tinggal di tepian: elemen pertama, elemen terakhir, satu elemen, tanpa elemen, banyak elemen genap atau ganjil. Fungsi yang benar pada lima kasus itu hampir selalu benar pada kasus besar, dan ketika salah, kasus kecil yang gagal memperlihatkan *mengapa* dengan cara yang tidak pernah bisa dilakukan laporan sebelas pesanan. Ketika kelak sebuah kesalahan ditemukan pada data sungguhan, langkah pertama adalah memperkecil data menjadi array terkecil yang masih memperlihatkan kesalahan itu, lalu menambahkan array tersebut ke program uji sebelum memperbaiki apa pun, supaya kesalahan itu tidak pernah kembali tanpa ketahuan.

6.12 Membuatnya Lebih Baik

Jaga kapasitas. Array C berukuran tetap. Syarat perulangan `n < MAX_ORDERS` menghentikan pembacaan pada 100 pesanan alih-alih menulis melewati ujung; versi yang lebih kuat akan

mencetak peringatan bahwa ada pesanan yang diabaikan, supaya hari yang memecahkan rekor tidak terpotong diam-diam. Memilih kapasitas adalah keputusan sungguhan: 100 pesanan adalah tiga kali hari tersibuk Mbak Rina, dan 100 nilai `long` memakan 800 bita. Bab 10 memperlihatkan cara meminta memori selagi program berjalan, ketika ukurannya tidak dapat diketahui sebelumnya.

Pertahankan urutan data. Setiap fungsi yang harus mengurutkan untuk menjawab mengurutkan salinan. Alternatifnya, yang berguna bila datanya besar, adalah mengurutkan array *indeks*: array `order` berisi 0 sampai $n - 1$, diurutkan menurut `bill[order[i]]`, sehingga array-array paralel itu sendiri tidak pernah berpindah. Soal Latihan 6.7 memintanya.

Laporkan angka yang tahan banting. Seperti diperlihatkan Subbab 6.10, satu pesanan yang tidak biasa sangat menggeser rata-rata dan hanya sedikit menggeser median. Laporan yang dimaksudkan untuk mengambil keputusan sebaiknya menampilkan keduanya, dan mungkin juga tagihan terkecil dan terbesar, supaya pembaca melihat seberapa tersebar hari itu. Bab 8 menambahkan *simpangan baku*, satu bilangan untuk sebaran itu.

Bulatkan rata-rata dengan sadar. `sum / n` membuang pecahan; `(sum + n / 2) / n` membulatkan ke rupiah terdekat. Selisihnya satu rupiah di sini, tetapi memilih aturannya dengan sengaja, dan menyatakannya, adalah kebiasaan yang penting ketika menyangkut uang.

Simpan data setelah tutup. Ketika program berakhir, array-array itu lenyap, dan laporan besok tidak dapat membandingkan dengan hari ini. Bab 11 menulis setiap pesanan ke sebuah berkas begitu diterima, supaya hari itu selamat dari listrik padam dan berkas sebulan dapat dibaca kembali ke dalam array untuk laporan bulanan. Fungsi-fungsi `bills.c` sama sekali tidak berubah: fungsi-fungsi itu tidak peduli apakah array diisi dari papan ketik atau dari berkas, dan itulah upah dari memisahkan masukan, perhitungan, dan keluaran ke dalam fungsi-fungsi yang berbeda.

6.13 Memakainya Sungguhan

Laporan harian ini cocok untuk setiap usaha yang mencatat banyak barang sejenis.

- **Warung** mencatat jumlah setiap penjualan dan kode produknya; pola penghitungan memberikan penjualan per produk, median memberikan belanja yang lazim, dan tabel mingguan menjadi penjualan per hari dan per produk.
- **Kebun buah dan durian** mencatat kilogram yang dipetik per pohon per minggu; array dua dimensi `kg[week][tree]` dengan total baris dan kolom memperlihatkan minggu terbaik dan pohon terbaik, dan mengurutkan total kolom memeringkat pohon-pohonnya.
- **Meja pendaftaran klinik kecil** mencatat lama tunggu setiap pasien; rata-rata, median, dan waktu tunggu terlama menunjukkan apakah diperlukan satu petugas pendaftaran lagi di pagi hari. (Laporan seperti itu hanya tentang antrean; keputusan tentang pasien tetap milik dokter atau bidan.)
- **Kantin sekolah** menghitung minuman yang terjual per jam dengan array `count[24]` yang diindeks dengan jam, dan jam tersibuk menunjukkan kapan diperlukan dua penjual.
- **Juru parkir motor** mencatat jam setiap motor masuk dan keluar; array berisi 24 penghitung memberikan banyaknya motor yang ada pada setiap jam, dan nilai terbesarnya menunjukkan apakah lahan parkir cukup luas pada hari pasaran.

Dalam setiap kasus, tentukan dulu nilai apa yang disimpan untuk setiap barang (array paralel, kelak struktur), pertanyaan apa yang dijawab di akhir (total, hitungan, urutan, pencarian), dan jumlah barang terbesar yang mungkin terjadi, yang menjadi kapasitasnya.

6.14 Kesalahan yang Sering Terjadi, dan Cara Menghindarinya

- **Kelebihan satu di ujung.** `for (i = 0; i <= n; i++)` mengunjungi $n + 1$ elemen, satu melewati ujung. Tulislah `i < n`, selalu.
- **Mulai dari 1.** Buku matematika sering menomori dari 1; array C dimulai dari 0. `bill[1]` adalah tagihan *kedua*.
- **Penghitung yang tidak diinisialisasi.** `int count[4];` di dalam fungsi berisi sampah. Tulislah `int count[4] = {0};`.
- **Memakai nilai sebagai indeks tanpa memeriksanya.** `count[s]++` dengan `s` dari masukan menulis ke sembarang tempat ketika `s` di luar jangkauan. Periksa jangkauannya lebih dulu.
- **`sizeof` pada parameter array.** Di dalam fungsi, `sizeof a / sizeof a[0]` membagi ukuran alamat dengan ukuran elemen; hasilnya tidak ada hubungannya dengan array. Teruskan panjangnya sebagai parameter.
- **Tak sengaja mengurutkan data pemanggil.** Fungsi yang mengurutkan parameter arraynya mengubah array pemanggil. Urutkan salinan bila urutan itu penting bagi pemanggil.
- **Pencarian biner pada data yang tidak terurut.** Hasilnya salah tanpa peringatan apa pun. Urutkan dulu, atau pakai pencarian linear.
- **`(low + high) / 2`.** Meluap untuk array yang sangat besar. Tulislah `low + (high - low) / 2`.
- **Baris bersama di Python.** `[[0] * 3] * 7` membuat tujuh referensi ke satu baris. Pakailah list comprehension.
- **Menugaskan array.** `a = b;` untuk dua array C tidak dapat dikompilasi; salin elemen demi elemen (seperti `copy_bills`). Di Python, `a = b` dapat dijalankan, tetapi menjadikan `a` nama kedua untuk list yang sama; pakailah `a = list(b)` untuk salinan.

6.15 Ringkasan Bab

- Array adalah sejumlah tetap elemen bertipe sama di bawah satu nama, disimpan berdampingan; elemen i berjarak i dari awal, Persamaan (6.1), itulah sebabnya indeks berjalan dari 0 sampai $n - 1$ dan setiap elemen dicapai dalam satu langkah.
- C tidak memeriksa indeks: melangkah keluar dari array adalah perilaku tak terdefinisi. Berulanglah dengan `i < n`, periksa indeks yang berasal dari masukan, dan kembangkan program dengan `-fsanitize=address,undefined`. List Python memeriksa dan melempar `IndexError`.
- Inisialisator mengisi elemen di dalam kurung kurawal, elemen yang tidak disebut menjadi nol; `= {0}` memulai sekumpulan penghitung. Memakai nilai data sebagai indeks mengubah array menjadi penghitung untuk setiap jenis.
- Parameter array menerima alamat array pemanggil: fungsi dapat mengubahnya, dan tidak tahu panjangnya, sehingga panjangnya diteruskan sebagai parameter dan `const` menandai array yang hanya dibaca.
- Bubble sort membawa nilai terbesar yang tersisa ke ujung pada setiap putaran dan memerlukan sampai $n(n-1)/2$ perbandingan, Persamaan (6.3); insertion sort adalah kerabat dekatnya. Pengurutan pustaka jauh lebih cepat untuk data besar.

- Pencarian linear bekerja pada array apa pun dan mungkin melihat seluruh n elemen; pencarian biner memerlukan data terurut dan paling banyak $\lfloor \log_2 n \rfloor + 1$ tengokan, Persamaan (6.4).
- Array dua dimensi menyimpan tabel baris demi baris; alamat `kg[r][c]` mengikuti Persamaan (6.5). Python memakai list berisi list.
- Kasir versi 5 menyimpan hari itu dalam array paralel dan melaporkan total per layanan, rata-rata, median, dan tiga teratas; median lebih tepat menggambarkan pelanggan biasa ketika ada satu pesanan yang tidak biasa.

6.16 Soal Latihan

1. **Menelusuri indeks.** Untuk `int a[5] = {4, 8, 15, 16, 23};`, berapakah `a[0]`, `a[4]`, `a[a[0] - 3]`, dan nilai `s` setelah `int s = 0; for (int i = 1; i < 5; i += 2) s += a[i];`? Ekspresi mana yang merupakan perilaku tak terdefinisi: `a[5]`, `a[-1]`, atau keduanya?
2. **Alamat.** Array `double price[10]` dimulai di alamat 2000, dan sebuah `double` memakan 8 bita. Berapakah alamat `price[7]`? Tabel `int t[4][6]` dimulai di 9000 dengan nilai `int` 4 bita; berapakah alamat `t[2][5]`?
3. **Membalik di tempat.** Tulislah `void reverse(long a[], int n)` yang membalik array tanpa array kedua, dengan menukar elemen pertama dan terakhir, lalu kedua dan kedua dari akhir, dan seterusnya. Ujilah pada array berpanjang ganjil dan genap.
4. **Yang terkecil dan letaknya.** Tulislah `int index_of_min(const long a[], int n)`. Apa yang seharusnya dikembalikan untuk `n = 0`, dan bagaimana pemanggil menanganinya?
5. **Insertion sort.** Tulislah insertion sort, cara pemain kartu mengurutkan kartu di tangan: ambil kartu satu per satu dari kiri dan sisipkan masing-masing ke tempatnya di antara kartu yang sudah terurut, sambil menggeser kartu yang lebih besar satu tempat ke kanan. Cetaklah array setelah setiap penyisipan untuk enam tagihan di **Gambar 6.2**. Berapa perbandingan yang dilakukannya pada array yang sudah terurut?
6. **Menghitung perbandingan.** Dengan Persamaan (6.3) dan Persamaan (6.4), isilah tabel untuk n = 10, 1000, dan 1.000.000: perbandingan kasus terburuk bubble sort, serta tengokan kasus terburuk pencarian linear dan biner.
7. **Mengurutkan indeks.** Tulislah fungsi yang mengisi `int order[]` dengan 0 sampai $n - 1$ dan mengurutkannya sehingga `bill[order[0]]` adalah tagihan terbesar, *tanpa memindahkan elemen* `bill`. Mengapa ini berguna bersama array paralel?

6.17 Penyelesaian Soal Latihan

1. `a[0]` bernilai 4, `a[4]` bernilai 23, dan `a[a[0] - 3]` adalah `a[1]`, yaitu 8. Perulangan mengunjungi `i` = 1 dan 3, sehingga `s` = 8 + 16 = 24. `a[5]` (satu melewati ujung) dan `a[-1]` (satu sebelum awal) keduanya perilaku tak terdefinisi.

2. Menurut Persamaan (6.1), `price[7]` berada di $2000 + 7 \cdot 8 = 2056$. Menurut Persamaan (6.5) dengan $C = 6$ dan $s = 4$, `t[2][5]` berada di $9000 + (2 \cdot 6 + 5) \cdot 4 = 9000 + 68 = 9068$.

3. Dua indeks mulai dari kedua ujung dan bergerak saling mendekat sambil menukar, sampai bertemu atau bersilang (**Kode 6.18**, **Kode 6.19**). Pada array berpanjang ganjil elemen tengahnya tetap di tempat.

Kode 6.18 `ex6_reverse.c`: membalik array di tempat dengan dua indeks.

```c
/* ex6_reverse.c - Latihan 6.3: balik array di tempat */
#include <stdio.h>

void reverse(long a[], int n)
{
    for (int i = 0, j = n - 1; i < j; i++, j--) {   /* dua indeks bertemu */
        long t = a[i];
        a[i] = a[j];
        a[j] = t;
    }
}

int main(void)
{
    long odd[5] = {1, 2, 3, 4, 5};
    long even[4] = {10, 20, 30, 40};
    reverse(odd, 5);
    reverse(even, 4);
    for (int i = 0; i < 5; i++)
        printf("%ld ", odd[i]);
    printf("\n");
    for (int i = 0; i < 4; i++)
        printf("%ld ", even[i]);
    printf("\n");
    return 0;
}
```

Kode 6.19 `ex6_reverse.py`: hal yang sama dalam Python (Python yang idiomatis cukup menulis `a.reverse()`).

```python
# ex6_reverse.py - Latihan 6.3: balik list di tempat


def reverse(a):
    i, j = 0, len(a) - 1
    while i < j:                            # dua indeks bertemu
        a[i], a[j] = a[j], a[i]
        i += 1
        j -= 1


def main():
    odd = [1, 2, 3, 4, 5]
    even = [10, 20, 30, 40]
    reverse(odd)
    reverse(even)
    print("".join(f"{x} " for x in odd))
    print("".join(f"{x} " for x in even))


if __name__ == "__main__":
    main()
```

```
5 4 3 2 1
40 30 20 10
```

4. Fungsinya mencerminkan `index_of_max` dengan `<` sebagai ganti `>`. Untuk `n = 0` tidak ada elemen terkecil, sehingga tidak ada indeks yang benar; kembalikan -1, kesepakatan yang sama dengan fungsi pencarian, dan biarkan pemanggil memeriksanya sebelum memakai hasilnya sebagai indeks: `int k = index_of_min(a, n); if (k >= 0) printf("%ld\n", a[k]);`.

5. Kode 6.20 dan **Kode 6.21**. Perulangan `while` menggeser setiap elemen yang lebih besar satu tempat ke kanan dan berhenti pada elemen pertama yang tidak lebih besar, tempat nilai yang diambil dituliskan.

Kode 6.20 `ex6_insertion.c`: insertion sort dengan jejak setelah setiap penyisipan.

```c
/* ex6_insertion.c - Latihan 6.5: insertion sort, cara pemain kartu */
#include <stdio.h>

void insertion_sort(long a[], int n)
{
    for (int i = 1; i < n; i++) {
        long key = a[i];                /* kartu yang kita ambil */
        int j = i - 1;
        while (j >= 0 && a[j] > key) {  /* geser kartu yang lebih besar ke kanan */
            a[j + 1] = a[j];
            j--;
        }
        a[j + 1] = key;
        printf("after i=%d:", i);
        for (int k = 0; k < n; k++)
            printf(" %ld", a[k]);
        printf("\n");
    }
}

int main(void)
{
    long bill[6] = {45000, 21000, 70000, 28000, 136500, 30000};
    insertion_sort(bill, 6);
    return 0;
}
```

Kode 6.21 `ex6_insertion.py`: hal yang sama dalam Python.

```python
# ex6_insertion.py - Latihan 6.5: insertion sort, cara pemain kartu


def insertion_sort(a):
    for i in range(1, len(a)):
        key = a[i]                      # kartu yang kita ambil
        j = i - 1
        while j >= 0 and a[j] > key:    # geser kartu yang lebih besar ke kanan
            a[j + 1] = a[j]
            j -= 1
        a[j + 1] = key
        print(f"after i={i}:" + "".join(f" {x}" for x in a))


def main():
    insertion_sort([45000, 21000, 70000, 28000, 136500, 30000])


if __name__ == "__main__":
    main()
```

```
after i=1: 21000 45000 70000 28000 136500 30000
after i=2: 21000 45000 70000 28000 136500 30000
after i=3: 21000 28000 45000 70000 136500 30000
after i=4: 21000 28000 45000 70000 136500 30000
after i=5: 21000 28000 30000 45000 70000 136500
```

Pada array yang sudah terurut, syarat `while` langsung gagal untuk setiap `i`, sehingga insertion sort hanya melakukan $n - 1$ perbandingan. Inilah keunggulan besarnya dibanding bubble sort tanpa keluar lebih awal, dan alasan pengurutan pustaka memakai insertion sort untuk potongan yang pendek atau hampir terurut.

6. Bubble sort: $n(n-1)/2 = 45.499.500$, dan $499.999.500.000$. Pencarian linear: 10, 1000, dan 1.000.000 tengokan. Pencarian biner, $\lfloor \log_2 n \rfloor + 1$: 4, 10, dan 20 tengokan.

7. Isi `order[i] = i`, lalu jalankan bubble sort pada `order`, dengan membandingkan `bill[order[i]] < bill[order[i + 1]]` dan menukar kedua *indeks* ketika urutannya salah. Sesudahnya `order[0]` adalah letak tagihan terbesar, dan `service[order[0]]` serta `weight[order[0]]` masih menggambarkan pesanan yang sama, karena tidak ada array paralel yang berpindah. Dengan cara inilah laporan dapat mendaftar pesanan menurut besarnya sambil tetap mencetak nomor notanya, yaitu `order[i] + 1`.

Daftar Pustaka

[1] ISO/IEC 9899:2018, *Information technology — Programming languages — C*. Geneva, Switzerland: International Organization for Standardization, 2018.

[2] Carnegie Mellon University Software Engineering Institute, "ARR30-C. Do not form or use out-of-bounds pointers or array subscripts," in *SEI CERT C Coding Standard*. [Online]. Available: https://wiki.sei.cmu.edu/confluence/display/c/ARR30-C.+Do+not+form+or+use+out-of-bounds+pointers+or+array+subscripts

[3] Free Software Foundation, "Program instrumentation options," in *Using the GNU Compiler Collection (GCC)*. [Online]. Available: https://gcc.gnu.org/onlinedocs/gcc/Instrumentation-Options.html

[4] Python Software Foundation, "Data structures: more on lists," in *The Python Tutorial*, Python 3.14 documentation. [Online]. Available: https://docs.python.org/3/tutorial/datastructures.html

[5] D. E. Knuth, *The Art of Computer Programming, Vol. 3: Sorting and Searching*, 2nd ed. Reading, MA, USA: Addison-Wesley, 1998.

[6] O. Astrachan, "Bubble sort: an archaeological algorithmic analysis," in *Proc. 34th SIGCSE Technical Symposium on Computer Science Education*, Reno, NV, USA, 2003, pp. 1–5.

[7] Python Software Foundation, "Sorting techniques," Python 3.14 documentation. [Online]. Available: https://docs.python.org/3/howto/sorting.html

[8] J. Bloch, "Extra, extra - read all about it: nearly all binary searches and mergesorts are broken," Google Research Blog, Jun. 2, 2006. [Online]. Available: https://research.google/blog/extra-extra-read-all-about-it-nearly-all-binary-searches-and-mergesorts-are-broken/

[9] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*, 4th ed. Cambridge, MA, USA: MIT Press, 2022.

[10] Python Software Foundation, "How do I create a multidimensional list?," in *Programming FAQ*, Python 3.14 documentation. [Online]. Available: https://docs.python.org/3/faq/programming.html#how-do-i-create-a-multidimensional-list

Bab 7

Pointer: Alamat sebagai Nilai

Bab 5 meninggalkan dua janji yang belum ditepati. Sebuah fungsi hanya dapat mengembalikan satu nilai, padahal pemilik toko meminta empat nilai saat tutup. Dan sebuah fungsi menerima salinan argumennya, padahal `scanf` entah bagaimana mengisi variabel pemanggilnya, dan fungsi-fungsi array di Bab 6 entah bagaimana mengubah array milik pemanggil. Kedua teka-teki itu punya jawaban yang sama: *alamat*, bilangan yang menyatakan di mana sebuah variabel tinggal di memori, adalah nilai juga, yang dapat disimpan dalam variabel, diteruskan ke fungsi, dan diikuti kembali ke variabel yang ditunjuknya. Variabel seperti itu disebut **pointer** (penunjuk), dan gagasan inilah yang memberi C kekuatan sekaligus reputasinya. Bab ini membahas pointer pelan-pelan, dengan gambar. Di akhir bab ini Anda akan mampu menjelaskan apa yang dikerjakan `&` dan `*` serta menggambar memori sebuah program kecil lengkap dengan panahnya; menulis fungsi yang mengubah variabel pemanggilnya, seperti `swap`; mengembalikan beberapa jawaban dari satu pemanggilan lewat parameter keluaran, beserta status yang menyatakan apakah jawaban itu sah; menelusuri array dengan pointer, berhitung dengan aritmetika pointer, dan menjelaskan mengapa `a[i]` dan `*(a + i)` adalah hal yang sama; membaca dan menulis `const` dalam deklarasi pointer; meneruskan fungsi ke fungsi lain dan memakai `qsort` dari pustaka dengan fungsi pembanding; mengenali kesalahan pointer yang klasik (pointer null, pointer yang tidak diinisialisasi, pointer ke variabel yang sudah tidak ada) dan menangkapnya dengan sanitizer; menyebutkan apa yang dilakukan Python sebagai gantinya; serta membangun kasir Profitina Laundry versi 6, yang ringkasan tutup tokonya mendapatkan total, rata-rata, tagihan terkecil, dan tagihan terbesar dari satu pemanggilan.

7.1 Satu Panggilan, Tiga Jawaban

Saat tutup toko, Mbak Rina meminta ringkasan dari kasir: berapa pesanan, berapa kilogram yang dicuci, berapa pendapatan, berapa tagihan rata-rata, mana yang terkecil dan mana yang terbesar. Dimas ingin satu fungsi, `day_stats`, menelusuri tagihan hari itu sekali saja dan menghasilkan semuanya. Fungsi C mengembalikan satu nilai. Ia bisa menulis empat fungsi, `total`, `average`, `smallest`, dan `biggest`, yang masing-masing menelusuri array lagi, dan untuk lima puluh pesanan tidak akan ada yang menyadari pemborosannya. Tetapi masalah ini muncul kembali dalam bentuk yang tidak bisa diselesaikan dengan empat fungsi: memecah berat seperti 3,45 kg menjadi 3 kg dan 450 g menghasilkan dua bilangan yang saling terkait, dan fungsi yang membaca pesanan dari papan ketik harus mengembalikan layanan, berat, dan tanda ekspres, sekaligus menyatakan apakah pembacaannya berhasil.

Masalah kedua ada di rak. Dua kantong menunggu diambil, dan program Dimas menyimpan beratnya di dua variabel, `left` dan `right`. Supaya kantong yang lebih ringan selalu di kiri, ia ingin fungsi `swap(left, right)`. Bab 5 sudah membuktikan bahwa ini tidak mungkin berhasil dengan nilai: fungsi menerima salinan, menukar salinannya, dan rak tidak berubah.

Pikirkan bagaimana laundry itu sendiri memecahkan masalah yang mirip. Ketika pelanggan menitipkan kantong, Mbak Rina tidak menyerahkan isi kantong kepada pembantu yang akan mencucinya. Ia menyerahkan *label* bertuliskan nomor rak: “rak 12”. Pembantu berjalan ke rak 12 dan mengerjakan kantong yang ada di sana, dan ketika Mbak Rina menengok rak 12 kemudian, ia

melihat kantong yang sudah dicuci. Label itu kecil, mudah disalin, dan mudah diteruskan, dan memungkinkan siapa pun yang memegangnya menjangkau satu-satunya kantong yang asli. Pointer persis label seperti itu: nilai kecil yang menyatakan *di mana* sesuatu berada, sehingga fungsi yang memegangnya dapat menjangkau variabel asli milik pemanggil, bukan salinannya.

Profitina Laundry sudah bertahun-tahun memakai label seperti itu tanpa menyebutnya pointer. Nota yang dibawa pulang pelanggan memuat sebuah nomor, dan nomor itu mengarah ke tepat satu kantong di tepat satu rak. Ketika pelanggan kembali, tidak ada yang menggeledah seluruh toko: nomornya menunjukkan tempat mencarinya. Jika nota hilang, kantongnya tetap ada, tetapi tidak ada yang bisa menemukannya dengan cepat; jika dua nota tak sengaja bernomor sama, dua pelanggan akan mengaku memiliki satu kantong. Setiap keadaan sehari-hari ini punya kembarannya dalam program yang memakai pointer, dan bab ini menjumpai semuanya.

7.2 Gagasannya dengan Kata-Kata Sederhana

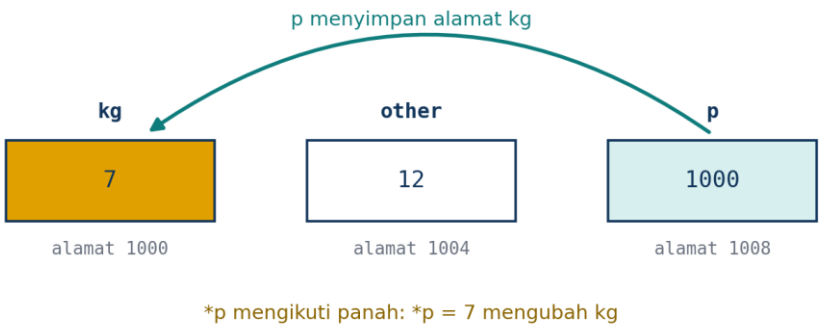
7.2.1 Setiap variabel punya alamat

Memori komputer adalah deretan panjang bita bernomor, seperti deretan loker bernomor yang sangat panjang di sebuah pasar. Setiap variabel menempati beberapa loker yang berurutan: `int` biasanya empat, `double` delapan. Nomor loker pertamanya adalah **alamat** variabel itu. Operator `&` (dibaca “alamat dari”) memberikannya: jika `kg` adalah variabel `int`, `&kg` adalah nomor loker tempat `kg` dimulai. Anda sudah menulis `&` sejak `scanf("%d", &kg)` yang pertama; sekarang Anda tahu apa yang Anda serahkan kepada `scanf`: bukan nilai `kg`, yang tidak diperlukan `scanf`, melainkan tempat tinggal `kg`, yang justru diperlukan `scanf` untuk menaruh sebuah bilangan di sana.

> **DEFINISI.** *Pointer* adalah variabel yang nilainya adalah alamat variabel lain. Jika `p` menyimpan alamat `kg`, kita katakan bahwa `p` *menunjuk* `kg`. Deklarasi `int *p;` menjadikan `p` pointer ke `int`. Operator `&` memberikan alamat sebuah variabel; operator `*`, bila dikenakan pada pointer, memberikan variabel yang ditunjuknya, langkah yang disebut *dereferensi* atau *mengikuti* pointer.

7.2.2 Mengikuti panah

Gambar 7.1 menggambar memori tiga variabel. `kg` tinggal di alamat 1000 dan berisi 7; `other` tinggal di 1004 dan berisi 12; `p` tinggal di 1008 dan berisi bilangan 1000, alamat `kg`. Kita menggambar bilangan itu sebagai panah dari `p` ke `kg`, karena memang itulah artinya. Alamat-alamat pada gambar adalah bilangan kecil rekaan; alamat sungguhan jauh lebih besar dan berubah dari satu kali jalan ke kali berikutnya, tetapi gambarnya tetap sama.



Gambar 7.1 Tiga variabel di memori: `p` menyimpan alamat `kg`, digambar sebagai panah; `*p` mengikuti panah itu, sehingga `*p = 7` mengubah `kg`.

Dengan `p` menunjuk `kg`, ekspresi `*p` *adalah* `kg`: membaca `*p` berarti membaca `kg`, dan penugasan `*p = 7` menulis 7 ke dalam `kg`. Sebagai aturan, untuk setiap variabel `x`,

$$*(\&x) \equiv x \tag{7.1}$$

Persamaan (7.1) menyatakan bahwa mengikuti alamat sebuah variabel membawa kita kembali ke variabel itu sendiri: `&` dan `*` saling meniadakan. Lambang ≡ berarti "adalah hal yang sama dengan", bukan sekadar "bernilai sama": `*(&x)` boleh diisi, persis seperti `x`.

Tanda bintang muncul di dua tempat dengan dua tugas berbeda, dan ini membingungkan semua orang pada awalnya. Dalam **deklarasi**, `int *p;`, bintang berarti "p adalah pointer ke int". Dalam **ekspresi**, `*p = 7;`, bintang berarti "ikuti p". Cara membaca deklarasi yang membuat keduanya selaras: `int *p` menyatakan bahwa `*p` adalah sebuah `int`.

Latihlah cara membaca itu pada beberapa deklarasi. `double *price;` menyatakan bahwa `*price` adalah `double`, jadi `price` adalah pointer ke `double`. `long *a, b;` adalah jebakan: bintangnya hanya milik `a`, sehingga `a` pointer ke `long` dan `b` hanya `long` biasa; tulislah satu deklarasi per baris untuk menghindarinya. `int **pp;` menyatakan bahwa `**pp` adalah `int`, jadi `*pp` pointer ke `int` dan `pp` adalah pointer ke pointer, yang diperlukan Bab 10 untuk linked list. Dan pada `int *p = &kg;` bintangnya milik deklarasi: nilai `&kg` masuk ke `p`, bukan ke `*p`.

7.2.3 Tipe sebuah pointer, dan pointer yang tidak menunjuk apa-apa

Pointer mengetahui tipe dari yang ditunjuknya: `int *` menunjuk `int`, `long *` menunjuk `long`, `double *` menunjuk `double`. Alamat saja hanya menyatakan di mana bita pertama berada; tipenya menyatakan berapa bita yang menjadi milik variabel itu dan bagaimana membacanya. Kompilator menolak menaruh alamat sebuah `double` ke dalam `int *` tanpa cast, karena mengikuti pointer itu berarti membaca delapan bita milik `double` seolah-olah sebuah `int`, dan hasilnya tidak bermakna.

Pointer juga boleh tidak menunjuk apa-apa. Konstanta `NULL` (dari `<stddef.h>`, juga tersedia lewat `<stdio.h>` dan `<stdlib.h>`) adalah nilai pointer yang dijamin bukan alamat variabel mana pun. Nilai ini dipakai untuk berkata "tidak ada variabel di sini": pencarian yang tidak menemukan apa-apa boleh mengembalikan `NULL`, dan fungsi boleh menolak bekerja ketika menerima `NULL`. Mengikuti pointer null adalah perilaku tak terdefinisi, dan di sistem desktop langsung membuat program berhenti mendadak [1], [2]. Penemu referensi null, Tony Hoare, belakangan menyebutnya "kesalahan miliaran dolar", karena tak terhitung banyaknya program yang berhenti mendadak akibat mengikuti pointer null tanpa memeriksanya lebih dulu [3]. Aturannya sederhana: sebelum mengikuti pointer yang mungkin `NULL`, ujilah.

7.2.4 Pointer yang sudah Anda pakai

Anda sudah bertemu pointer sejak Bab 1 tanpa mengenal namanya. `&` pada `scanf("%d", &kg)` menghasilkan pointer ke `kg`, dan `scanf` menulis bilangan yang dibacanya melalui pointer itu. Teks pada `printf("Hello\n")` adalah array karakter, dan yang diterima `printf` adalah pointer ke karakter pertamanya. Parameter array `const long a[]` di Bab 6 adalah pointer, itulah sebabnya fungsi-fungsi di bab itu dapat mengubah array pemanggil dan tidak dapat mengetahui panjangnya. Bab ini tidak menambahkan fitur baru yang aneh ke dalam C; bab ini menjelaskan fitur yang selama ini sudah bekerja di balik layar, dan memberi Anda kata-kata serta gambar untuk memakainya dengan sengaja.

7.3 Pointer Pertama Anda

Kode 7.1 melakukan semua yang ada di **Gambar 7.1**. Program ini menyimpan alamat `kg` di `p`, menulis melalui `p`, mengarahkan `p` ke variabel lain, menyalin pointer ke `q`, dan akhirnya mendeklarasikan pointer yang tidak menunjuk ke mana pun.

Kode 7.1 `first_pointer.c`: mengambil alamat, mengikutinya, menulis melaluinya, mengarahkan ulang pointer, menyalinnya, dan `NULL`.

```c
/* first_pointer.c - alamat adalah nilai yang bisa disimpan, disalin, dan diikuti */
#include <stdio.h>

int main(void)
{
    int kg = 5;
    int *p = &kg;                       /* p menyimpan alamat kg */
    printf("kg = %d, *p = %d\n", kg, *p);

    *p = 7;                             /* tulis MELALUI pointer */
    printf("after *p = 7: kg = %d\n", kg);

    int other = 12;
    p = &other;                         /* arahkan p ke variabel lain */
    *p += 1;
    printf("other = %d, kg = %d\n", other, kg);
    printf("p points to other: %s\n", p == &other ? "yes" : "no");

    int *q = p;                         /* menyalin pointer: dua panah */
    *q = 20;
    printf("after *q = 20: other = %d, *p = %d\n", other, *p);

    int *nowhere = NULL;                /* pointer yang tidak menunjuk apa pun */
    if (nowhere == NULL)
        printf("nowhere is NULL: never follow it\n");
    return 0;
}
```

```
kg = 5, *p = 5
after *p = 7: kg = 7
other = 13, kg = 7
p points to other: yes
after *q = 20: other = 20, *p = 20
nowhere is NULL: never follow it
```

Ikutilah program itu pada **Gambar 7.1** baris demi baris. Setelah `int *p = &kg;` panah mengarah dari `p` ke `kg`, dan `*p` terbaca 5. Setelah `*p = 7;` nilai di `kg` menjadi 7, walaupun pernyataan itu tidak pernah menyebut `kg`: itulah intinya. Lalu `p = &other;` memindahkan panah (tanpa bintang, penugasan mengubah `p` sendiri, bukan yang ditunjuknya), dan `*p += 1` membuat `other` menjadi

13 sementara `kg` tetap 7. Menyalin `q = p` membuat panah kedua ke variabel yang sama; menulis melalui `q` mengubah `other`, dan membaca melalui `p` melihat perubahan itu. Dua pointer ke satu variabel seperti dua label bernomor rak yang sama: siapa pun yang memakai label mana pun mengerjakan kantong yang sama.

Program ini sengaja tidak mencetak alamat. `printf("%p\n", (void *) p);` akan mencetak alamat sungguhan, bilangan seperti `0x7ffd5c3a1b2c`, tetapi bilangan itu akan berbeda di komputer Anda dan pada setiap kali jalan (sistem modern menempatkan variabel di alamat acak sebagai langkah pengamanan), sedangkan buku ini hanya mencetak keluaran yang dapat Anda hasilkan ulang dengan persis. Cobalah sekali sendiri: nilainya tidak bermakna, dan yang penting hanyalah hubungannya dengan alamat lain, seperti diperlihatkan Subbab 7.6.

Kernighan dan Ritchie, perancang pointer C, mengingatkan bahwa pointer "telah disamakan dengan pernyataan goto sebagai cara jitu untuk membuat program yang mustahil dipahami", dan bahwa dengan disiplin pointer justru dapat dipakai untuk mencapai kejelasan dan kesederhanaan [4]. Disiplin itulah yang diajarkan sisa bab ini.

Python tidak memiliki pointer, dan itu disengaja: program Python tidak dapat mengambil alamat sebuah variabel atau mengikutinya [5]. Yang dimiliki Python adalah *referensi*. Setiap nama di Python merujuk ke sebuah objek, dan dua nama dapat merujuk ke objek yang sama. Ketika objek itu dapat diubah, seperti list, perubahan melalui satu nama terlihat melalui nama lainnya. **Kode 7.2** memakai sifat ini untuk meniru program C: list berisi satu elemen berperan sebagai variabel di memori, dan nama kedua untuk list yang sama berperan sebagai pointer.

Kode 7.2 `first_pointer.py`: list satu elemen sebagai kotak yang dapat dipakai bersama dua nama, yaitu bagian pointer yang dimiliki Python.

```
# first_pointer.py - Python tak punya pointer; list satu elemen menjadi kotak
# yang bisa dipakai bersama dua nama, bagian pointer yang dimiliki Python.


def main():
    kg = [5]                            # kotak yang berisi satu bilangan
    p = kg                              # p merujuk ke kotak yang SAMA
    print(f"kg = {kg[0]}, *p = {p[0]}")

    p[0] = 7                            # tulis MELALUI kotak bersama
    print(f"after *p = 7: kg = {kg[0]}")

    other = [12]
    p = other                           # p kini merujuk ke kotak lain
    p[0] += 1
    print(f"other = {other[0]}, kg = {kg[0]}")
    print(f"p points to other: {'yes' if p is other else 'no'}")

    q = p                               # menyalin referensi: dua panah
    q[0] = 20
    print(f"after *q = 20: other = {other[0]}, *p = {p[0]}")

    nowhere = None                      # NULL-nya Python
    if nowhere is None:
        print("nowhere is NULL: never follow it")


if __name__ == "__main__":
    main()
```

Keluarannya sama persis, dan tiruan itu mengajarkan gagasan pokoknya: setelah `p = kg`, nama `p` dan `kg` merujuk ke *satu* kotak, dan `p is kg` bernilai `True`. Yang tidak dapat ditiru Python adalah aritmetika pada alamat, pokok bahasan Subbab 7.6.

7.4 Pass-by-Reference: Mengubah Variabel Pemanggil

Sekarang raknya. **Kode 7.3** berisi dua fungsi. `swap_copies` menerima dua nilai `int`, cara lama, dan menukar salinannya sendiri. `swap` menerima dua *alamat*, `int *a` dan `int *b`, dan menukar apa yang ditunjuk keduanya. Pemanggilannya meneruskan alamat dengan `&`: `swap(&shelf_left, &shelf_right)`.

Kode 7.3 `swap.c`: menukar salinan tidak mengubah apa pun di `main`; menukar melalui pointer mengubah variabel milik `main`. Tiga langkah banding-tukar lalu mengurutkan tiga bilangan.

```c
/* swap.c - mengapa fungsi butuh alamat untuk mengubah variabel pemanggil */
#include <stdio.h>

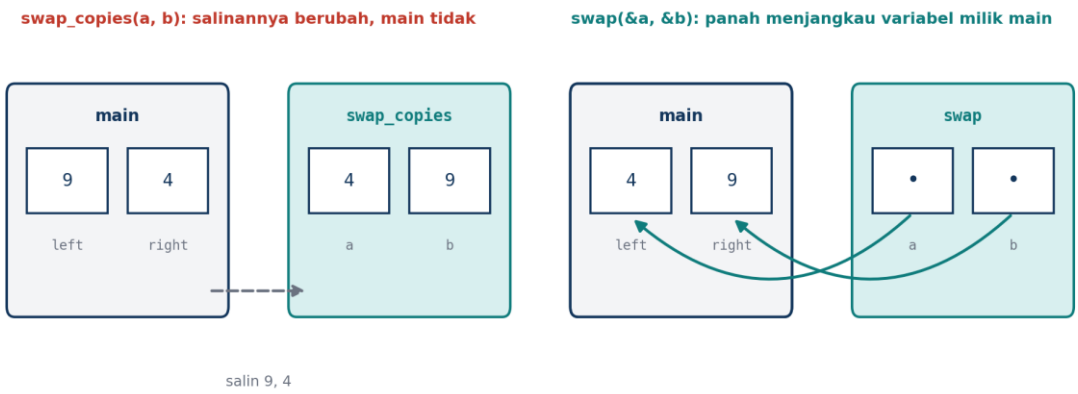
void swap_copies(int a, int b)          /* hanya menukar salinannya sendiri */
{
    int t = a;
    a = b;
    b = t;
}

void swap(int *a, int *b)               /* menukar apa yang ditunjuk panah */
{
    int t = *a;
    *a = *b;
    *b = t;
}

int main(void)
{
    int shelf_left = 9, shelf_right = 4;     /* kg dua kantong di rak */
    swap_copies(shelf_left, shelf_right);
    printf("after swap_copies: left %d, right %d\n", shelf_left, shelf_right);
    swap(&shelf_left, &shelf_right);
    printf("after swap:        left %d, right %d\n", shelf_left, shelf_right);

    int bag[3] = {7, 3, 5};
    if (bag[0] > bag[1]) swap(&bag[0], &bag[1]);   /* tiga banding-tukar */
    if (bag[1] > bag[2]) swap(&bag[1], &bag[2]);   /* urutkan tiga bilangan */
    if (bag[0] > bag[1]) swap(&bag[0], &bag[1]);
    printf("sorted bags: %d %d %d\n", bag[0], bag[1], bag[2]);
    return 0;
}
```

Gambar 7.2 menjajarkan kedua pemanggilan. Di kiri, `swap_copies` memiliki kotak sendiri, `a` dan `b`, yang diisi salinan 9 dan 4; fungsi itu menukarnya menjadi 4 dan 9, dan ketika kembali, kotak-kotaknya lenyap dan `main` masih memegang 9 dan 4. Di kanan, kotak-kotak milik `swap` berisi panah ke dalam `main`. `int t = *a;` membaca `left` milik `main` (9) ke `t`; `*a = *b;` menyalin `right` milik `main` (4) ke `left` milik `main`; `*b = t;` menaruh 9 ke `right` milik `main`. Ketika `swap` kembali, kotak-kotaknya lenyap, tetapi pekerjaan sudah dilakukan pada variabel milik `main`, dan hasilnya tetap.



Gambar 7.2 Kiri: `swap_copies` bekerja pada salinan, dan `main` tetap memegang 9 dan 4. Kanan: `swap` menerima alamat variabel milik `main` dan menukar nilai-nilai di sana.

```
after swap_copies: left 9, right 4
after swap:        left 4, right 9
sorted bags: 3 5 7
```

Sekarang `scanf` tidak lagi menyimpan misteri. Ketika Anda menulis `scanf("%d", &kg)`, Anda menyerahkan alamat `kg` kepada `scanf`, dan di dalam pustaka sebuah pernyataan yang setara dengan `*p = number;` menaruh bilangan yang diketik ke dalam variabel Anda. Lupa menulis `&`, yaitu menulis `scanf("%d", kg)`, berarti menyerahkan *nilai* `kg`, bilangan acak, yang lalu dipakai `scanf` sebagai alamat; bilangan yang diketik tertulis ke tempat acak di memori. `gcc -Wall` menangkap kesalahan khusus ini, karena ia tahu apa yang diharapkan `scanf`.

Cara pengiriman ini, ketika fungsi menerima alamat sehingga dapat mengubah variabel pemanggil, disebut **pass-by-reference** (pengiriman referensi). Tepatnya, C tetap mengirim nilai: fungsi menerima *salinan alamat*. Tetapi salinan alamat membawa ke variabel yang sama, sebagaimana fotokopi label rak membawa ke rak yang sama, dan akibatnya adalah referensi yang diinginkan pemanggil.

Baris-baris terakhir `main` mengurutkan tiga kantong dengan tiga pemanggilan semacam `if (bag[0] > bag[1]) swap(&bag[0], &bag[1]);`. Alamat elemen array diambil dengan `&` seperti alamat lainnya. Setelah dua langkah pertama, berat terbesar berada di ujung; langkah ketiga mengurutkan dua sisanya. *Jaringan pengurutan* kecil ini adalah bubble sort untuk tepat tiga elemen yang dituliskan satu per satu.

Periksalah ketiga langkah itu dengan tangan pada `{7, 3, 5}`. Langkah pertama membandingkan 7 dan 3, lalu menukar: `{3, 7, 5}`. Langkah kedua membandingkan 7 dan 5, lalu menukar: `{3, 5, 7}`, dan yang terbesar berada di ujung. Langkah ketiga membandingkan 3 dan 5, tanpa tukar. Keluarannya berbunyi `3 5 7`. Cobalah lima urutan lain dari tiga bilangan di atas kertas; semuanya berakhir terurut, bukti kecil bahwa tiga langkah itu sudah cukup.

Python tidak dapat meneruskan alamat sebuah variabel, sehingga `swap_copies` versi Python gagal persis seperti di C, dan Python menawarkan dua cara lain, keduanya diperlihatkan di **Kode 7.4**. Untuk dua variabel terpisah, cara Python yang wajar adalah penugasan tuple `left, right = right, left`, yang dilakukan di pihak pemanggil. Untuk elemen list, fungsi dapat mengubah list itu sendiri, karena fungsi menerima referensi ke list yang sama: `swap_items(bag, 0, 1)`. FAQ Python membahas persis pengganti parameter keluaran semacam ini [6].

Kode 7.4 `swap.py`: Python tidak dapat menukar dua variabel milik pemanggilnya; Python menukar dengan penugasan tuple di pihak pemanggil, atau mengubah list yang dipakai bersama.

```python
# swap.py - Python mengoper referensi objek, tetapi mengikat ulang parameter
# tak pernah sampai ke pemanggil; kembalikan nilai baru atau ubah list bersama.


def swap_copies(a, b):                  # hanya mengikat ulang namanya sendiri
    a, b = b, a


def swap_items(a, i, j):                # mengubah list pemanggil di tempat
    a[i], a[j] = a[j], a[i]


def main():
    shelf_left, shelf_right = 9, 4
    swap_copies(shelf_left, shelf_right)
    print(f"after swap_copies: left {shelf_left}, right {shelf_right}")
    shelf_left, shelf_right = shelf_right, shelf_left   # cara Python
    print(f"after swap:        left {shelf_left}, right {shelf_right}")

    bag = [7, 3, 5]
    if bag[0] > bag[1]:
        swap_items(bag, 0, 1)
    if bag[1] > bag[2]:
        swap_items(bag, 1, 2)
    if bag[0] > bag[1]:
        swap_items(bag, 0, 1)
    print(f"sorted bags: {bag[0]} {bag[1]} {bag[2]}")


if __name__ == "__main__":
    main()
```

7.5 Parameter Keluaran: Satu Panggilan, Beberapa Jawaban

Dengan pointer, sebuah fungsi dapat memberikan hasil sebanyak yang diperlukan: pemanggil meneruskan alamat variabel-variabel yang akan menerima hasil, dan fungsi menulis ke dalamnya. Parameter seperti itu disebut **parameter keluaran** (*output parameter*). **Kode 7.5** memiliki dua fungsi jenis ini.

`split_weight` mengubah berat dalam kilogram menjadi kilogram bulat dan gram, untuk nota pelanggan yang lebih suka “3 kg 450 g” daripada “3.45 kg”. Fungsi ini mula-mula mengubahnya menjadi gram bulat, dibulatkan ke gram terdekat, lalu membagi. Dengan w berat dalam kilogram dan g banyaknya gram bulat,

$$g = \operatorname{round}(1000\,w), \qquad \text{kg} = \lfloor g/1000 \rfloor, \qquad \text{gram} = g \bmod 1000 \tag{7.2}$$

Pembulatan dalam Persamaan (7.2) bukan hiasan. Bilangan 3,45 tidak dapat disimpan persis dalam biner; `double` yang paling dekat dengannya sedikit lebih besar, dan 1000 kali bilangan itu adalah 3450.0000000000005. Mengubahnya langsung menjadi bilangan bulat kebetulan menghasilkan 3450, tetapi untuk berat lain galatnya jatuh di sisi sebaliknya: 0,29 kg dikali 1000 adalah 289.99999999999994, yang oleh konversi biasa diubah menjadi 289 gram. `lround` dari `<math.h>` membulatkan ke bilangan bulat terdekat dan menyingkirkan masalah itu.

`day_stats` menelusuri tagihan sekali dan mengisi tiga jawaban: total, rata-rata, dan tagihan terbesar. Fungsi ini juga *mengembalikan* nilai, 1 atau 0, yang menyatakan apakah jawabannya sah. Untuk hari tanpa pesanan tidak ada rata-rata (akan terjadi pembagian dengan nol) dan tidak ada tagihan terbesar, sehingga fungsi mengembalikan 0 dan tidak menyentuh jawaban-jawabannya. Pemanggil menguji status itu sebelum memakai jawabannya, seperti diperlihatkan `main`.

Kode 7.5 `out_params.c`: `split_weight` memberikan dua jawaban melalui pointer; `day_stats` memberikan tiga jawaban dan mengembalikan apakah jawaban itu sah.

```c
/* out_params.c - satu panggilan, beberapa jawaban: parameter keluaran */
#include <stdio.h>
#include <math.h>

void split_weight(double weight, int *kg, int *grams)
{
    long total_grams = lround(weight * 1000);
    *kg = (int) (total_grams / 1000);
    *grams = (int) (total_grams % 1000);
}

/* mengembalikan 1 dan mengisi tiga jawaban; 0 untuk hari kosong */
int day_stats(const long *bill, int n, long *total, double *average,
              long *biggest)
{
    if (n <= 0)
        return 0;
    *total = 0;
    *biggest = bill[0];
    for (int i = 0; i < n; i++) {
        *total += bill[i];
        if (bill[i] > *biggest)
            *biggest = bill[i];
    }
    *average = (double) *total / n;
    return 1;
}

int main(void)
{
    int kg, grams;
    split_weight(3.45, &kg, &grams);
    printf("3.45 kg is %d kg and %d g\n", kg, grams);

    long bill[5] = {28000, 30000, 30000, 70000, 136500};
    long total, biggest;
    double average;
    if (day_stats(bill, 5, &total, &average, &biggest))
        printf("total %ld, average %.2f, biggest %ld\n", total, average, biggest);
    if (!day_stats(bill, 0, &total, &average, &biggest))
        printf("an empty day has no statistics\n");
    return 0;
}
```

```
3.45 kg is 3 kg and 450 g
total 294500, average 58900.00, biggest 136500
an empty day has no statistics
```

Telusurilah pemanggilan `day_stats(bill, 5, &total, &average, &biggest)` melalui memori sekali saja. `main` memiliki lima variabel yang penting di sini: array `bill` dan tiga variabel jawaban, yang berisi sampah karena belum pernah diberi nilai. Pemanggilan itu membuat parameter-parameter `day_stats`: `bill` menerima alamat array milik `main`, `n` menerima 5, dan `total`, `average`, serta `biggest` menerima *alamat* tiga variabel jawaban milik `main`. Di dalam fungsi, nama `total`, `average`, dan `biggest` adalah pointer, dan setiap pernyataan yang menyimpan jawaban melakukannya melalui bintang: `*total = 0;` menulis nol ke `total` milik `main`, `*total += bill[i];` menambah ke dalamnya, `*biggest = bill[i];` mengganti `biggest` milik `main`. Ketika fungsi mengembalikan 1, parameter-parameternya lenyap, dan `main` mendapati ketiga variabelnya sudah terisi. Tidak ada yang dikembalikan kecuali status; jawaban-jawabannya diantar sampai ke depan pintu `main`.

Perhatikan nama-namanya. Di dalam `day_stats`, parameternya bernama `total` dan berupa pointer; di dalam `main`, variabelnya bernama `total` dan bertipe `long`. Keduanya variabel yang berbeda yang kebetulan bernama sama, dalam cakupan yang berbeda (Bab 5). Sebagian pemrogram menandai parameter keluaran dengan awalan seperti `out_total` agar perbedaannya terlihat; gaya mana pun yang Anda pilih, pakailah secara konsisten.

Pola "hasil melalui pointer, keberhasilan sebagai nilai kembalian" ada di mana-mana dalam C. `scanf` sendiri mengikutinya: ia menulis nilai yang dibacanya melalui pointer yang diterimanya, dan mengembalikan banyaknya nilai yang berhasil dibaca, itulah sebabnya buku ini selalu menulis `scanf(...) == 1` di dalam perulangannya.

Python, di **Kode 7.6**, tidak memerlukan parameter keluaran, karena fungsi dapat mengembalikan *tuple*, beberapa nilai sekaligus, dan pemanggil membongkarnya: `kg, grams = split_weight(3.45)`. Untuk "tidak ada jawaban sah", Python mengembalikan `None`, versi `NULL` milik Python, dan pemanggil menguji `answer is not None`. Program ini juga mendefinisikan `c_lround`, karena `round` milik Python membulatkan setengah ke bilangan genap (Bab 5), sedangkan `lround` milik C membulatkannya menjauhi nol.

Kode 7.6 `out_params.py`: cara Python: kembalikan tuple, atau `None` bila tidak ada jawaban yang sah.

```
# out_params.py - satu panggilan, beberapa jawaban: Python mengembalikan tuple


def split_weight(weight):
    total_grams = c_lround(weight * 1000)
    return total_grams // 1000, total_grams % 1000


def c_lround(x):                        # lround C: setengah menjauhi nol
    return int(x + 0.5) if x >= 0 else -int(-x + 0.5)


def day_stats(bill):
    """Kembalikan (total, rata-rata, terbesar), atau None untuk hari kosong."""
    if not bill:
        return None
    total, biggest = 0, bill[0]
    for b in bill:
        total += b
        if b > biggest:
            biggest = b
    return total, total / len(bill), biggest


def main():
    kg, grams = split_weight(3.45)
    print(f"3.45 kg is {kg} kg and {grams} g")

    bill = [28000, 30000, 30000, 70000, 136500]
    answer = day_stats(bill)
    if answer is not None:
        total, average, biggest = answer
        print(f"total {total}, average {average:.2f}, biggest {biggest}")
    if day_stats(bill[:0]) is None:
        print("an empty day has no statistics")


if __name__ == "__main__":
    main()
```

7.6 Array dan Pointer: Dua Notasi, Satu Memori

7.6.1 Nama array adalah alamat elemen pertamanya

Bab 6 menyatakan bahwa array yang diteruskan ke fungsi tiba sebagai alamat elemen pertamanya. Inilah kebenaran selengkapnya. Di hampir setiap ekspresi, nama sebuah array diubah menjadi pointer ke elemen pertamanya: `bill` berarti `&bill[0]`. Pengecualiannya adalah `sizeof bill`, yang memberikan ukuran seluruh array, dan `&bill`. Konversi ini disebut *array-to-pointer decay* (peluruhan array menjadi pointer), dan itulah sebabnya `long *p = bill;` tidak memerlukan `&`.

Pengecualian untuk `sizeof` adalah tempat array dan pointer terlihat berpisah jalan. Dengan `long bill[5]; long *p = bill;`, ekspresi `sizeof bill` adalah ukuran seluruh array, 40 bita di komputer yang menyimpan `long` dalam 8 bita, sedangkan `sizeof p` adalah ukuran sebuah alamat, 8 bita di komputer 64-bit, apa pun yang ditunjuknya. Array *adalah* elemen-elemennya; pointer hanya *tahu di mana* elemen-elemen itu berada. Keduanya berperilaku sama ketika diindeks atau ditelusuri, dan berbeda ketika ditanya seberapa besar atau ketika hendak diisi (`bill = p;` tidak dapat dikompilasi, `p = bill;` dapat).

7.6.2 Aritmetika pointer

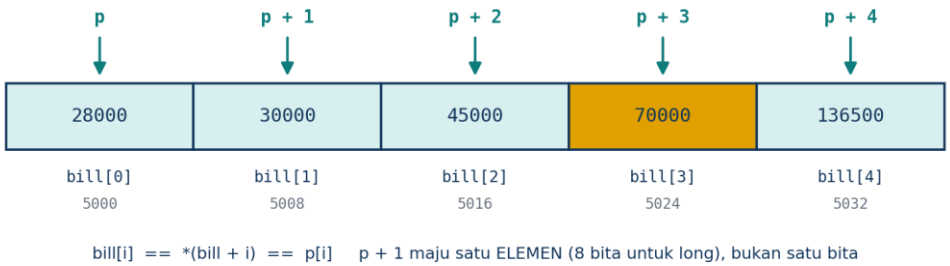
Menambahkan bilangan bulat k pada pointer memindahkannya k *elemen*, bukan k bita. Jika `p` menunjuk `bill[0]`, maka `p + 1` menunjuk `bill[1]`, `p + 3` menunjuk `bill[3]`. Kompilator mengalikan dengan ukuran elemen untuk Anda:

$$\text{alamat}(p + k) = \text{alamat}(p) + k \cdot \text{sizeof}(*p) \tag{7.3}$$

Persamaan (7.3) adalah rumus alamat Subbab 6.2.2 dalam bentuk baru. Rumus ini langsung membawa ke identitas pokok C: untuk array atau pointer `a` dan indeks `i`,

$$a[i] \equiv *(a + i) \tag{7.4}$$

Standar C bahkan *mendefinisikan* kurung siku dengan Persamaan (7.4) [1]: `a[i]` adalah singkatan dari "maju `i` elemen dari `a` lalu ikuti". **Gambar 7.3** menggambarkannya pada lima tagihan. Karena `p` menyimpan alamat yang sama dengan `bill`, `p[2]` bekerja sama baiknya dengan `bill[2]`; kurung siku tidak peduli apakah diberi nama array atau pointer.



Gambar 7.3 Aritmetika pointer: `p + k` menunjuk `k` elemen lebih jauh, berapa pun ukuran elemennya, sehingga `bill[i]`, `*(bill + i)`, dan `p[i]` menamai kotak yang sama.

Kerjakan satu contoh dengan bilangan-bilangan pada **Gambar 7.3**. Array dimulai di 5000 dan sebuah `long` memakan 8 bita. `p` menyimpan 5000; menurut Persamaan (7.3), `p + 3` menyimpan $5000 + 3 \cdot 8 = 5024$, alamat `bill[3]`, dan `*(p + 3)` bernilai 70000. Seandainya arraynya bertipe

`int`, empat bita per elemen, `p + 3` akan menyimpan 5012, lagi-lagi alamat elemen 3. Program menulis `p + 3` yang sama dalam kedua kasus, dan kompilator menyediakan pengali yang tepat; itulah sebabnya kode pointer ditulis dalam satuan elemen, tidak pernah dalam bita.

Mengurangkan dua pointer ke dalam array yang sama menghasilkan banyaknya elemen di antara keduanya, kebalikan dari Persamaan (7.3). Jika `p` menunjuk `bill[0]` dan `last` menunjuk `bill[4]`, maka `last - p` bernilai 4, berapa pun ukuran sebuah `long`. Tipe hasilnya adalah `ptrdiff_t`, dicetak dengan `%td`. Membandingkan dua pointer ke dalam array yang sama dengan `<` menunjukkan mana yang lebih dulu. Aritmetika dan perbandingan itu hanya terdefinisi di dalam satu array (dan satu posisi sesudah ujungnya); mengurangkan pointer ke dua array yang berbeda tidak terdefinisi [1].

7.6.3 Berjalan dengan pointer

Kode 7.7 memperlihatkan semua ini dalam praktik. Program ini membaca `bill[2]` dengan tiga cara, menjumlahkan array dengan menggerakkan pointer `q` dari elemen pertama sampai satu sesudah elemen terakhir, mengukur jarak antara dua pointer, memajukan pointer tiga elemen lalu membaca elemen *sebelumnya* dengan `p[-1]`, dan akhirnya membalik sebuah kata di tempat dengan dua pointer yang berjalan saling mendekat, gagasan dua indeks yang sama dengan Soal Latihan 6.3.

Kode 7.7 `pointer_walk.c`: tiga nama untuk satu elemen, penelusuran dengan pointer, selisih pointer, indeks negatif dari pointer yang sudah dimajukan, dan kata yang dibalik dengan dua pointer.

```c
/* pointer_walk.c - array dan pointer: dua notasi, satu memori */
#include <stdio.h>
#include <string.h>

int main(void)
{
    long bill[5] = {28000, 30000, 45000, 70000, 136500};
    long *p = bill;                      /* sama dengan &bill[0] */

    printf("bill[2] = %ld, *(bill + 2) = %ld, p[2] = %ld\n",
           bill[2], *(bill + 2), p[2]);

    long sum = 0;
    for (long *q = bill; q < bill + 5; q++)   /* telusuri array dengan pointer */
        sum += *q;
    printf("sum by walking: %ld\n", sum);

    long *last = &bill[4];
    printf("last - p = %td elements apart\n", last - p);

    p += 3;                              /* maju tiga ELEMEN, bukan bita */
    printf("after p += 3, *p = %ld and p[-1] = %ld\n", *p, p[-1]);

    char word[] = "laundry";             /* string adalah array char */
    char *left = word;
    char *right = word + strlen(word) - 1;
    while (left < right) {               /* dua pointer berjalan saling mendekat */
        char t = *left;
        *left = *right;
        *right = t;
        left++;
        right--;
    }
    printf("reversed: %s\n", word);
    return 0;
}
```

Perulangan `for (long *q = bill; q < bill + 5; q++)` layak dicermati, karena inilah idiom C yang klasik. `bill + 5` adalah alamat tepat sesudah elemen terakhir: pointer boleh menunjuk ke sana (begitulah ujung ditandai), tetapi tidak boleh diikuti. Perulangan berlanjut selama `q` masih sebelum ujung, dan `q++` maju satu elemen setiap kali. Badannya membaca elemen yang sedang ditunjuk dengan `*q`. Tidak ada indeks sama sekali.

Kata itu disimpan sebagai `char word[] = "laundry";`, array delapan karakter: tujuh huruf dan karakter nol `'\0'` yang menandai akhir setiap string di C. `strlen(word)` menghitung huruf sebelum nol itu, 7, sehingga `word + strlen(word) - 1` menunjuk huruf terakhir, `y`. Kedua pointer menukar karakter yang ditunjuknya lalu melangkah saling mendekat sampai bertemu di tengah. Bab 8 memakai persis penelusuran ini untuk memutuskan apakah sebuah kata palindrom, dan Bab 9 seluruhnya membahas string.

```
bill[2] = 45000, *(bill + 2) = 45000, p[2] = 45000
sum by walking: 309500
last - p = 4 elements apart
after p += 3, *p = 70000 and p[-1] = 45000
reversed: yrdnual
```

Python tidak memiliki aritmetika pointer; programnya memakai indeks, seperti yang dilakukan **Kode 7.8**. String Python tidak dapat diubah di tempat, sehingga kata itu lebih dulu diubah menjadi list karakter, dibalik, lalu disambung kembali menjadi string.

Kode 7.8 `pointer_walk.py`: penelusuran yang sama dengan indeks, cara Python menyatakan apa yang dikerjakan C dengan pointer.

```python
# pointer_walk.py - yang dilakukan C dengan pointer, Python lakukan dengan indeks


def main():
    bill = [28000, 30000, 45000, 70000, 136500]
    p = 0                                   # indeks berperan sebagai pointer
    print(f"bill[2] = {bill[2]}, *(bill + 2) = {bill[0 + 2]}, p[2] = {bill[p + 2]}")

    total = 0
    for q in range(len(bill)):              # berjalan dengan indeks
        total += bill[q]
    print(f"sum by walking: {total}")

    last = 4
    print(f"last - p = {last - p} elements apart")

    p += 3
    print(f"after p += 3, *p = {bill[p]} and p[-1] = {bill[p - 1]}")

    word = list("laundry")                  # string Python tak bisa diubah: pakai list
    left, right = 0, len(word) - 1
    while left < right:                     # dua indeks berjalan saling mendekat
        word[left], word[right] = word[right], word[left]
        left += 1
        right -= 1
    print(f"reversed: {''.join(word)}")


if __name__ == "__main__":
    main()
```

7.6.4 Mengapa pointer, bukan hanya indeks?

Jika `a[i]` dan `*(a + i)` sama, mengapa perlu keduanya? Sebagian karena sejarah: kompilator C generasi awal menghasilkan kode yang lebih cepat untuk penelusuran dengan pointer daripada dengan indeks, dan satu generasi kode C ditulis dengan gaya itu; kompilator masa kini menghasilkan kode yang sama untuk keduanya. Alasan yang bertahan berbeda. Pointer dapat menunjuk ke *tengah* sebuah array dan diserahkan ke fungsi seolah-olah array itu dimulai di sana: `total(bill + 3, 2)` menjumlahkan `bill[3]` dan `bill[4]` dengan fungsi Bab 6 yang tidak diubah. Pointer dapat menunjuk sebuah variabel tunggal semudah menunjuk elemen array, yang dibutuhkan `swap`. Dan di Bab 10 pointer menghubungkan potongan-potongan memori yang terpisah menjadi struktur, seperti linked list, yang tidak dapat dinyatakan dengan array.

7.7 `const` dan Pointer

Sebuah pointer melibatkan dua hal yang mungkin dilindungi: pointer itu sendiri dan variabel yang ditunjuknya. Letak `const` menentukan yang mana, seperti diperlihatkan **Tabel 7.1**. Bacalah deklarasinya dari kanan ke kiri: `const long *p` adalah “p adalah pointer ke long yang const”.

Tabel 7.1 Letak `const` dalam deklarasi pointer menentukan apa yang dilindungi: variabel yang ditunjuk, pointernya, atau keduanya.

Deklarasi	Boleh mengubah *p?	Boleh mengubah p?	Pemakaian lazim
`long *p`	ya	ya	parameter keluaran; penelusuran yang menulis
`const long *p`	tidak	ya	parameter hanya-baca; penelusuran yang hanya membaca
`long *const p`	ya	tidak	pointer yang terpaku di satu tempat
`const long *const p`	tidak	tidak	pandangan tetap yang hanya-baca

Baris kedua adalah yang paling sering Anda tulis. Parameter `const long *bill` milik `day_stats` berjanji bahwa fungsi itu hanya membaca tagihan; penelusuran `for (const long *p = bill; ...)` di dalamnya menggerakkan pointer (boleh) tetapi tidak pernah menulis melaluinya (tidak boleh). Parameter `const long a[]` di Bab 6 adalah hal yang persis sama yang ditulis dengan kurung siku: dalam daftar parameter, `long a[]` dan `long *a` bermakna sama.

7.8 Pointer ke Fungsi: Perilaku sebagai Parameter

Mbak Rina kadang ingin tagihan diurutkan dari yang termurah ke yang termahal, kadang sebaliknya, dan akuntannya ingin tagihan yang berupa ribuan bulat di depan, karena tagihan itu kemungkinan besar dibayar tunai tanpa kembalian. Tiga pengurutan yang hanya berbeda pada satu perbandingan berarti tiga fungsi yang hampir sama. Rancangan yang lebih baik menulis pengurutan sekali dan menyerahkan perbandingannya sebagai parameter.

Fungsi juga tinggal di sebuah alamat di memori, dan C memungkinkan program menyimpan dan meneruskan alamat itu. Deklarasi `int (*before)(long, long)` menjadikan `before` sebuah **pointer ke fungsi** yang menerima dua nilai `long` dan mengembalikan `int`. Tanda kurung di sekitar `*before` wajib; tanpanya, `int *before(long, long)` akan mendeklarasikan fungsi yang mengembalikan pointer. Nama fungsi, seperti nama array, berubah menjadi alamatnya, sehingga `sort_by(a, 6, cheaper_first)` meneruskan fungsi `cheaper_first`, dan di dalam `sort_by` pemanggilan `before(a[i + 1], a[i])` memanggil fungsi mana pun yang diteruskan.

Kode 7.9 menulis ketiga aturan sebagai fungsi kecil, menyimpannya dalam *array pointer fungsi*, `rule[3]`, dan mengurutkan enam tagihan yang sama tiga kali, sekali dengan setiap aturan. Kemudian program itu mengurutkannya dengan `qsort` dari pustaka standar [7].

Kode 7.9 `sort_by.c`: satu bubble sort yang bertanya kepada sebuah fungsi mana dari dua nilai yang lebih dulu, tiga fungsi semacam itu dalam sebuah array, dan `qsort` dari pustaka dengan fungsi pembandingnya.

```c
/* sort_by.c - mengoper perilaku sebagai parameter: pointer ke fungsi */
#include <stdio.h>
#include <stdlib.h>

int cheaper_first(long a, long b) { return a < b; }
int dearer_first(long a, long b) { return a > b; }
int round_thousands_first(long a, long b)    /* ribuan bulat mendahului yang lain */
{
    return (a % 1000 == 0) && (b % 1000 != 0);
}

/* bubble sort yang bertanya "before(x, y)?" alih-alih memakai < sendiri */
void sort_by(long a[], int n, int (*before)(long, long))
{
    for (int pass = 1; pass < n; pass++)
        for (int i = 0; i < n - pass; i++)
            if (before(a[i + 1], a[i])) {
                long t = a[i];
                a[i] = a[i + 1];
                a[i + 1] = t;
            }
}

int compare_long(const void *x, const void *y)  /* bentuk yang diminta qsort */
{
    long a = *(const long *) x, b = *(const long *) y;
    return (a > b) - (a < b);            /* -1, 0, atau 1; jangan pernah a - b */
}

void print_all(const char *title, const long a[], int n)
{
    printf("%-18s", title);
    for (int i = 0; i < n; i++)
        printf(" %6ld", a[i]);
    printf("\n");
}

int main(void)
{
    long start[6] = {45500, 21000, 70000, 28500, 136500, 30000};
    long a[6];
    int (*rule[3])(long, long) = {cheaper_first, dearer_first,
                                  round_thousands_first};
    const char *name[3] = {"cheaper first:", "dearer first:", "round first:"};
```

Kode 7.9 (Lanjutan) *sort_by.c: satu bubble sort yang bertanya kepada sebuah fungsi mana dari dua nilai yang lebih dulu, tiga fungsi semacam itu dalam sebuah array, dan qsort dari pustaka dengan fungsi pembandingnya.*

```c
    print_all("as taken:", start, 6);
    for (int r = 0; r < 3; r++) {
        for (int i = 0; i < 6; i++)
            a[i] = start[i];
        sort_by(a, 6, rule[r]);
        print_all(name[r], a, 6);
    }
    for (int i = 0; i < 6; i++)
        a[i] = start[i];
    qsort(a, 6, sizeof a[0], compare_long);
    print_all("qsort:", a, 6);
    return 0;
}
```

```
as taken:           45500  21000  70000  28500 136500  30000
cheaper first:      21000  28500  30000  45500  70000 136500
dearer first:      136500  70000  45500  30000  28500  21000
round first:        21000  70000  30000  45500  28500 136500
qsort:              21000  28500  30000  45500  70000 136500
```

Aturan "ribuan bulat lebih dulu" memberikan hasil yang menarik: 21000, 70000, dan 30000 berada di depan, *dalam urutan saat diterima*, lalu 45500, 28500, dan 136500, juga dalam urutan aslinya. Bubble sort dalam kode ini hanya menukar ketika `before(a[i + 1], a[i])` bernilai benar, yaitu hanya ketika nilai di kanan *harus* didahulukan, sehingga nilai-nilai yang dianggap setara oleh aturan itu tidak pernah bertukar tempat. Pengurutannya **stabil**, seperti didefinisikan Bab 6, dan urutan yang stabil itulah yang memberi aturan ini maknanya.

Pada kasir sungguhan, aturan itu dipilih oleh pengguna. Dengan array pointer fungsi, pilihan itu hanyalah sebuah indeks: baca bilangan `r` dari menu, periksa bahwa `0 <= r && r < 3` (indeks dari papan ketik diperiksa sebelum dipakai, seperti diajarkan Bab 6), lalu panggil `sort_by(a, n, rule[r])`. Menambahkan aturan keempat berarti menulis satu fungsi kecil dan menambahkannya ke array; baik pengurutan maupun kode menunya tidak berubah. Ini adalah prinsip "satu aturan, satu tempat" dari Bab 5, yang diterapkan pada perilaku, bukan pada harga.

`qsort` memiliki antarmuka yang lebih umum, karena ia mengurutkan array bertipe *apa saja*. Ia menerima alamat array, banyaknya elemen, ukuran satu elemen, dan fungsi pembanding yang parameternya bertipe `const void *`, pointer ke "sesuatu" yang harus diubah fungsi pembanding ke tipe sebenarnya. Pembanding mengembalikan bilangan negatif ketika elemen pertama harus lebih dulu, nol ketika keduanya setara, dan bilangan positif selain itu. Untuk dua bilangan a dan b, itu adalah *tanda* dari selisihnya:

$$\operatorname{cmp}(a, b) = [a > b] - [a < b] \in \{-1,\ 0,\ 1\} \tag{7.5}$$

dengan $[\cdot]$ bernilai 1 ketika syaratnya terpenuhi dan 0 selain itu, persis yang dihasilkan operator perbandingan C. Jalan pintas yang menggoda, `return a - b;`, menghasilkan tanda yang sama ketika bilangannya kecil, dan gagal ketika bilangannya besar: untuk nilai `int` di sekitar dua miliar, $a - b$ meluap dan tandanya menjadi salah, dan mengembalikan selisih `long` sebagai `int` memotongnya. Persamaan (7.5) tidak pernah meluap. Perhatikan pula bahwa `qsort` tidak diwajibkan stabil [7].

Bagaimana `qsort` memakai fungsi pembanding? `qsort` tidak mengetahui tipe elemennya, hanya ukurannya, sehingga ia memindahkan elemen sebagai blok-blok bita, dan setiap kali perlu tahu mana dari dua elemen yang lebih dulu, ia memanggil fungsi pembanding dengan alamat kedua elemen itu. `compare_long` mengubah setiap `const void *` kembali menjadi `const long *`, mengikutinya untuk mendapatkan kedua bilangan, lalu menjawab dengan Persamaan (7.5). Karena itu `qsort` yang sama dapat mengurutkan array `int`, `double`, string, atau struktur di Bab 10: hanya fungsi pembandingnya yang berganti. Ini gagasan yang sama dengan `sort_by`, algoritma umum dengan keputusan khusus yang diserahkan dari luar, dan begitulah pustaka standar kebanyakan bahasa pemrograman dirancang.

Di Python, fungsi adalah nilai seperti nilai lainnya: `rule = [cheaper_first, dearer_first, round_thousands_first]` adalah list biasa, dan `sort_by(a, rule[r])` meneruskan fungsi tanpa sintaks khusus apa pun, seperti diperlihatkan **Kode 7.10**. Pengurutan pustakanya adalah `sorted(a)`, yang stabil; untuk mengurutkan menurut sebuah aturan, `sorted` menerima fungsi `key` (`sorted(a, reverse=True)` untuk yang termahal lebih dulu, atau `sorted(a, key=lambda b: b %`

`1000 != 0)` untuk ribuan bulat lebih dulu), dan pembanding gaya C dapat disesuaikan dengan `functools.cmp_to_key` [8].

Kode 7.10 `sort_by.py`: fungsi yang diteruskan sebagai nilai di Python, dan `sorted` dari pustaka.

```
# sort_by.py - mengoper perilaku sebagai parameter: fungsi adalah nilai


def cheaper_first(a, b):
    return a < b


def dearer_first(a, b):
    return a > b


def round_thousands_first(a, b):         # ribuan bulat mendahului yang lain
    return a % 1000 == 0 and b % 1000 != 0


def sort_by(a, before):
    """Bubble sort yang bertanya before(x, y) alih-alih memakai < sendiri."""
    n = len(a)
    for pass_no in range(1, n):
        for i in range(n - pass_no):
            if before(a[i + 1], a[i]):
                a[i], a[i + 1] = a[i + 1], a[i]


def print_all(title, a):
    print(f"{title:<18}" + "".join(f" {x:6d}" for x in a))


def main():
    start = [45500, 21000, 70000, 28500, 136500, 30000]
    rule = [cheaper_first, dearer_first, round_thousands_first]
    name = ["cheaper first:", "dearer first:", "round first:"]

    print_all("as taken:", start)
    for r in range(3):
        a = list(start)
        sort_by(a, rule[r])
        print_all(name[r], a)
    print_all("qsort:", sorted(start))   # pengurutan pustaka, seperti qsort di C


if __name__ == "__main__":
    main()
```

7.9 Membangun Kasir Versi 6

Ringkasan tutup toko versi 6 menyatukan bab ini. Modul baru, `stats`, memuat statistik dengan parameter keluaran. **Kode 7.11** adalah header-nya.

Kode 7.11 `stats.h`: janji-janji modul statistik.

```
/* stats.h - statistik saat tutup dengan parameter keluaran (Bab 7) */
#ifndef STATS_H
#define STATS_H

void split_weight(double weight, int *kg, int *grams);
int day_stats(const long *bill, int n, long *total, double *average,
              long *smallest, long *biggest);
int compare_long(const void *x, const void *y);

#endif
```

Di **Kode 7.12**, `day_stats` memberikan empat jawaban, yaitu total, rata-rata, tagihan terkecil, dan tagihan terbesar, dari satu penelusuran dengan pointer `const`. Fungsi ini menolak pointer null sebagaimana menolak hari tanpa pesanan. Dan ia menulis jawabannya hanya di akhir, setelah seluruh penelusuran: sampai saat itu ia bekerja di variabel lokal. Rincian ini punya alasan. Seandainya pemanggil kebetulan meneruskan alamat variabel yang sama untuk dua jawaban, atau seandainya penelusuran terputus, jawaban yang ditulis lebih awal bisa setengah jadi atau saling menimpa; menghitung di variabel lokal lalu menulis sekali di akhir berarti pemanggil melihat keempat jawaban sekaligus atau tidak sama sekali.

Kode 7.12 `stats.c`: `split_weight`, `day_stats` dengan empat jawaban dan penelusuran pointer, serta fungsi pembanding untuk `qsort`.

```
/* stats.c - statistik saat tutup dengan parameter keluaran */
#include <math.h>
#include <stddef.h>
#include "stats.h"

void split_weight(double weight, int *kg, int *grams)
{
    long total_grams = lround(weight * 1000);
    *kg = (int) (total_grams / 1000);
    *grams = (int) (total_grams % 1000);
}

int day_stats(const long *bill, int n, long *total, double *average,
              long *smallest, long *biggest)
{
    if (bill == NULL || n <= 0)
        return 0;                       /* tak ada laporan: jawaban tak disentuh */
    long sum = 0, lo = bill[0], hi = bill[0];
    for (const long *p = bill; p < bill + n; p++) {
        sum += *p;
        if (*p < lo)
            lo = *p;
        if (*p > hi)
            hi = *p;
    }
    *total = sum;                       /* tulis jawaban hanya di akhir */
    *average = (double) sum / n;
    *smallest = lo;
    *biggest = hi;
    return 1;
}

int compare_long(const void *x, const void *y)
{
    long a = *(const long *) x, b = *(const long *) y;
    return (a > b) - (a < b);
}
```

Programnya sendiri, **Kode 7.13**, membaca pesanan dengan fungsi `read_orders` yang mengisi dua array melalui parameter pointer dan mengembalikan banyaknya pesanan yang disimpan. Lalu

satu pemanggilan `day_stats` menghasilkan keempat statistik, `split_weight` mengubah berat total menjadi kilogram dan gram, dan `qsort` mengurutkan tagihan dari yang termurah ke yang termahal, yang dicetak dengan penelusuran pointer dalam ribuan rupiah. Program ini memakai ulang `laundry.c` dari Bab 5 tanpa perubahan.

Kode 7.13 `closing.c`: kasir versi 6: ringkasan tutup toko dari satu pemanggilan `day_stats`.

```c
/* closing.c - kasir Profitina Laundry versi 6: ringkasan saat tutup.
   Masukan: satu pesanan per baris, "service weight express", diakhiri 0. */
#include <stdio.h>
#include <stdlib.h>
#include "laundry.h"
#include "stats.h"

#define MAX_ORDERS 100

int read_orders(long *bill, double *weight, int max)   /* mengembalikan banyaknya */
{
    int n = 0, s, express;
    double w;
    while (n < max && scanf("%d", &s) == 1 && s != 0) {
        if (scanf("%lf %d", &w, &express) != 2)
            break;
        if (price_per_kg(s) == 0 || w <= 0 || w > MAX_KG)
            continue;
        bill[n] = bill_for(s, w, express);
        weight[n] = w;
        n++;
    }
    return n;
}

int main(void)
{
    long bill[MAX_ORDERS], total = 0, smallest = 0, biggest = 0;
    double weight[MAX_ORDERS], average = 0, kg_sum = 0;
    int n = read_orders(bill, weight, MAX_ORDERS);

    printf("PROFITINA LAUNDRY - closing summary (v6)\n");
    if (!day_stats(bill, n, &total, &average, &smallest, &biggest)) {
        printf("no orders today\n");
        return 0;
    }
    for (int i = 0; i < n; i++)
        kg_sum += weight[i];
    int kg, grams;
    split_weight(kg_sum, &kg, &grams);
```

Kode 7.13 (Lanjutan) *closing.c: kasir versi 6: ringkasan tutup toko dari satu pemanggilan day_stats.*

```
    printf("orders   : %d\n", n);
    printf("washed   : %d kg %d g\n", kg, grams);
    printf("revenue  : ");
    print_rupiah(total);
    printf("\naverage  : ");
    print_rupiah((long) (average + 0.5));
    printf("\nsmallest : ");
    print_rupiah(smallest);
    printf("\nbiggest  : ");
    print_rupiah(biggest);

    qsort(bill, (size_t) n, sizeof bill[0], compare_long);
    printf("\ncheapest first:");
    for (long *p = bill; p < bill + n; p++)
        printf(" %ld", *p / 1000);
    printf(" (thousands)\n");
    return 0;
}
```

Perhatikan kepala `read_orders`: `int read_orders(long *bill, double *weight, int max)`. Dua parameter pertama ditulis sebagai pointer, padahal Bab 6 akan menulis `long bill[]` dan `double weight[]`. Dalam daftar parameter kedua ejaan itu bermakna persis sama, dan pilihannya adalah pesan untuk pembaca: kurung siku berkata “ini array”, bintang berkata “ini alamat”. Keduanya benar di sini. Di dalamnya, `bill[n] = ...` memakai kurung siku pada pointer, sesuai Persamaan (7.4). Parameter ketiga, `max`, adalah kapasitasnya, yang diperlukan fungsi karena pointer tidak dapat mengetahui berapa elemen yang ada di belakangnya; tanpa `max`, hari yang ramai akan membuat `read_orders` menulis melewati ujung array milik `main`, dan tidak ada kompilator yang akan menyadarinya.

Modul Python, **Kode 7.14**, mengembalikan tuple; `day_stats` mengembalikan `None` untuk hari tanpa pesanan atau untuk `None` sebagai ganti list.

Kode 7.14 `stats.py`: modul statistik dalam Python, yang mengembalikan tuple.

```
# stats.py - statistik saat tutup; Python mengembalikan beberapa nilai sekaligus


def c_lround(x):                          # lround C: setengah menjauhi nol
    return int(x + 0.5) if x >= 0 else -int(-x + 0.5)


def split_weight(weight):
    total_grams = c_lround(weight * 1000)
    return total_grams // 1000, total_grams % 1000


def day_stats(bill):
    """Kembalikan (total, rata-rata, terkecil, terbesar), atau None bila kosong."""
    if not bill:
        return None
    total, lo, hi = 0, bill[0], bill[0]
    for b in bill:
        total += b
        lo = min(lo, b)
        hi = max(hi, b)
    return total, total / len(bill), lo, hi
```

Kode 7.15 `closing.py`: kasir versi 6 dalam Python.

```
# closing.py - kasir Profitina Laundry versi 6: ringkasan saat tutup.
# Masukan: satu pesanan per baris, "service weight express", diakhiri 0.
import sys
from laundry import MAX_KG, price_per_kg, bill_for, rupiah
from stats import split_weight, day_stats

MAX_ORDERS = 100


def read_orders(words, limit):
    """Kembalikan (tagihan, berat) dari pesanan yang sah."""
    bill, weight, pos = [], [], 0
    while (len(bill) < limit and pos < len(words) and int(words[pos]) != 0
           and pos + 2 < len(words)):
        s, w = int(words[pos]), float(words[pos + 1])
        express = int(words[pos + 2])
        pos += 3
        if price_per_kg(s) == 0 or w <= 0 or w > MAX_KG:
            continue
        bill.append(bill_for(s, w, express))
        weight.append(w)
    return bill, weight


def main():
    bill, weight = read_orders(sys.stdin.read().split(), MAX_ORDERS)
    print("PROFITINA LAUNDRY - closing summary (v6)")
    answer = day_stats(bill)
    if answer is None:
        print("no orders today")
        return
    total, average, smallest, biggest = answer
    kg_sum = 0.0
    for w in weight:
        kg_sum += w
    kg, grams = split_weight(kg_sum)

    print(f"orders   : {len(bill)}")
    print(f"washed   : {kg} kg {grams} g")
    print(f"revenue  : {rupiah(total)}")
    print(f"average  : {rupiah(int(average + 0.5))}")
    print(f"smallest : {rupiah(smallest)}")
    print(f"biggest  : {rupiah(biggest)}")
    print("cheapest first:" + "".join(f" {b // 1000}" for b in sorted(bill))
          + " (thousands)")
```

Kode 7.15 (Lanjutan) *closing.py: kasir versi 6 dalam Python.*

```
if __name__ == "__main__":
    main()
```

Kode 7.15 adalah program Python-nya; `read_orders` mengembalikan kedua list sebagai tuple, dan `sorted(bill)` memberikan salinan terurut, sementara `bill` tetap dalam urutan nota.

7.10 Menjalankannya

Hari Sabtu yang sama dengan Bab 6:

```
1 3.4 0
2 1.2 1
3 5 0
2 6.5 0
1 12.75 1
1 4.2 0
3 2.5 0
2 3 1
1 7.9 0
7 3 0
3 1 0
1 5.5 1
0
```

```
PROFITINA LAUNDRY - closing summary (v6)
orders   : 11
washed   : 52 kg 950 g
revenue  : Rp 523.500
average  : Rp 47.591
smallest : Rp 12.000
biggest  : Rp 136.500
cheapest first: 12 18 28 30 30 35 45 56 63 70 136 (thousands)
```

Sebelas pesanan sah beratnya 33,75 + 10,7 + 8,5 = 52,95 kg, ditampilkan sebagai 52 kg 950 g. Pendapatannya, Rp523.500, sama dengan di Bab 6, sebagaimana mestinya. Rata-ratanya kini Rp47.591, sedangkan Bab 6 mencetak Rp47.590: rata-rata tepatnya adalah 47590,909..., dan versi ini membulatkannya ke rupiah terdekat dengan `(long) (average + 0.5)`, sedangkan Bab 6 membuang pecahannya dengan pembagian bilangan bulat. Tidak ada yang salah, tetapi sebuah toko sebaiknya memilih satu aturan dan memegangnya, atau dua laporan untuk hari yang sama akan berselisih satu rupiah dan menghabiskan satu malam untuk mencari sebabnya. Tagihan terkecil adalah Rp12.000, pesanan setrika saja seberat 1 kg yang ditagih dengan minimum 2 kg, dan yang terbesar Rp136.500. Baris terakhir mendaftar tagihan yang diurutkan `qsort`, dalam ribuan.

Cocokkan baris yang terurut itu dengan daftar yang dihitung tangan di Subbab 6.10, dibaca dari ujung yang lain: 12, 18, 28, 30, 30, 35, 45, 56, 63, 70, dan 136 ribu. Dua angka 30 adalah pesanan cuci setrika 1,2 kg ekspres dan setrika saja 5 kg; `qsort` tidak menjanjikan mana yang lebih dulu, dan untuk laporan ini hal itu tidak penting, karena yang dicetak hanya jumlah uangnya. Pembagian bilangan bulat dengan 1000 mencetak 136500 sebagai 136, itulah sebabnya baris itu bertuliskan "(thousands)": label yang memberi tahu pembaca apa yang dipotong adalah bagian dari laporan yang jujur.

7.11 Mengujinya

Kode 7.16 menguji tepian modul ini. `split_weight(0.9996, ...)` harus membulatkan ke atas menjadi 1 kg dan 0 g, bukan melaporkan 0 kg dan 1000 g. `day_stats` harus memberikan empat jawaban yang benar untuk empat tagihan kecil, dan harus mengembalikan 0 untuk hari tanpa pesanan *serta membiarkan jawabannya tak tersentuh*: uji itu lebih dulu mengisi `total` dengan nilai yang benar, memanggil `day_stats` dengan `n` = 0, lalu memeriksa bahwa `total` masih memegang nilai itu. Pointer null harus ditolak. Uji diakhiri dengan `qsort` dan `compare_long`.

Kode 7.16 `test_stats.c`: empat belas pemeriksaan, termasuk pembulatan di batas kilogram, hari tanpa pesanan, pointer null, dan janji bahwa pemanggilan yang ditolak membiarkan jawaban tak tersentuh.

```c
/* test_stats.c - menguji split_weight dan day_stats, termasuk kasus tepi */
#include <stdio.h>
#include <stdlib.h>
#include "stats.h"

static int checks = 0, failures = 0;

static void check(long got, long expected, int line)
{
    checks++;
    if (got != expected) {
        failures++;
        printf("FAILED on line %d: got %ld, expected %ld\n", line, got, expected);
    }
}

int main(void)
{
    int kg = -1, g = -1;
    split_weight(3.45, &kg, &g);
    check(kg, 3, __LINE__);
    check(g, 450, __LINE__);
    split_weight(0.9996, &kg, &g);          /* dibulatkan ke atas ke kg bulat */
    check(kg, 1, __LINE__);
    check(g, 0, __LINE__);

    long bill[4] = {30, 10, 50, 20}, total = -1, lo = -1, hi = -1;
    double avg = -1;
    check(day_stats(bill, 4, &total, &avg, &lo, &hi), 1, __LINE__);
    check(total, 110, __LINE__);
    check((long) (avg * 100), 2750, __LINE__);
    check(lo, 10, __LINE__);
    check(hi, 50, __LINE__);
    check(day_stats(bill, 0, &total, &avg, &lo, &hi), 0, __LINE__);
    check(total, 110, __LINE__);            /* tak tersentuh setelah penolakan */
    check(day_stats(NULL, 3, &total, &avg, &lo, &hi), 0, __LINE__);

    qsort(bill, 4, sizeof bill[0], compare_long);
    check(bill[0], 10, __LINE__);
    check(bill[3], 50, __LINE__);
    printf("%d checks, %d failures\n", checks, failures);
    return failures != 0;
}
```

Rata-rata diperiksa sebagai `(long) (avg * 100)` terhadap 2750, karena membandingkan dua nilai `double` untuk kesamaan persis itu rapuh; rata-rata 27,5 tepat dalam biner, tetapi pada umumnya uji terhadap `double` hasil hitungan sebaiknya membandingkan dengan toleransi kecil. Uji Python, **Kode 7.17**, memeriksa fakta-fakta yang sama; di sana, “tak tersentuh” berarti variabel milik pemanggil tidak berubah ketika `day_stats` mengembalikan `None`.

Kode 7.17 `test_stats.py`: pemeriksaan yang sama dalam Python.

```
# test_stats.py - menguji split_weight dan day_stats, termasuk kasus tepi
import inspect
import sys
from stats import split_weight, day_stats

checks = 0
failures = 0


def check(got, expected):
    global checks, failures
    checks += 1
    if got != expected:
        failures += 1
        line = inspect.stack()[1].lineno
        print(f"FAILED on line {line}: got {got}, expected {expected}")


def main():
    kg, g = split_weight(3.45)
    check(kg, 3)
    check(g, 450)
    kg, g = split_weight(0.9996)         # dibulatkan ke atas ke kg bulat
    check(kg, 1)
    check(g, 0)

    bill = [30, 10, 50, 20]
    answer = day_stats(bill)
    check(answer is not None, True)
    total, avg, lo, hi = answer
    check(total, 110)
    check(int(avg * 100), 2750)
    check(lo, 10)
    check(hi, 50)
    check(day_stats([]), None)
    check(total, 110)                    # variabel kita sendiri tak tersentuh
    check(day_stats(None), None)

    bill.sort()
    check(bill[0], 10)
    check(bill[3], 50)
    print(f"{checks} checks, {failures} failures")
    return 1 if failures else 0
```

Kode 7.17 (Lanjutan) *test_stats.py: pemeriksaan yang sama dalam Python.*

```
if __name__ == "__main__":
    sys.exit(main())
```

```
14 checks, 0 failures
```

7.12 Membuatnya Lebih Baik: Keselamatan Pointer

Pointer memberi program kekuatan untuk menjangkau ke mana saja di dalam memorinya, termasuk tempat yang seharusnya tidak dijangkaunya. Empat aturan menjaga kekuatan itu tetap terkendali.

Inisialisasi setiap pointer. `int *p;` di dalam fungsi menyimpan alamat acak; `*p = 5;` lalu menulis ke tempat acak. Berikan setiap pointer nilai saat itu juga, alamat variabel sungguhan atau `NULL`.

Uji sebelum mengikuti. Fungsi yang menerima pointer dari luar, seperti `day_stats`, sebaiknya menentukan apa arti `NULL` dan mengujinya. Aturan SEI CERT C EXP34-C, "Do not dereference null pointers", ada karena satu baris pengujian ini begitu sering terlupa [2].

Jangan pernah mengembalikan alamat variabel lokal. Variabel lokal sebuah fungsi lenyap ketika fungsi itu kembali (Bab 5). Pointer ke salah satunya menunjuk memori yang akan dipakai ulang oleh pemanggilan berikutnya, sebuah *dangling pointer* (pointer menggantung):

```
long *biggest_bill_wrong(const long *bill, int n)
{
    long best = bill[0];
    for (int i = 1; i < n; i++)
        if (bill[i] > best)
            best = bill[i];
    return &best;          /* WRONG: best disappears at the return */
}
```

`gcc` memperingatkan *function returns address of local variable*, dan aturan SEI CERT DCL30-C melarangnya [9]. Kembalikan nilainya, atau pointer ke dalam array milik pemanggil (`return &bill[k];` boleh, karena array pemanggil masih ada), atau tulis jawabannya melalui parameter keluaran.

Ketahui siapa pemilik memorinya. Setiap pointer menunjuk ke memori milik seseorang: variabel lokal `main`, array milik pemanggil, literal string, atau (mulai Bab 10) blok yang diminta program. Pointer aman diikuti hanya selama pemiliknya masih ada dan masih mengizinkannya. Fungsi yang menerima pointer *meminjam* memori itu selama pemanggilan berlangsung; fungsi itu tidak boleh menyimpan pointer tersebut di variabel global untuk dipakai belakangan, karena pada saat itu pemiliknya mungkin sudah tiada. Bertanya "memori siapa ini, dan berapa lama ia hidup?" untuk setiap pointer yang Anda tulis mencegah kebanyakan kesalahan pointer sebelum sempat diketik.

Kembangkan program dengan sanitizer. Setiap program buku ini juga dikompilasi dengan `-fsanitize=address,undefined`, yang membuat program yang berjalan memeriksa setiap akses memori dan berhenti dengan laporan yang menyebut baris tempat terjadinya dereferensi null, akses di luar batas, atau pemakaian dangling pointer [10]. Kesalahan yang diam-diam merusak memori pada build biasa berubah menjadi pesan galat yang lantang dan tepat. Pakailah opsi ini setiap kali Anda menulis kode dengan pointer, dan lepaskan hanya untuk versi yang Anda serahkan kepada pengguna.

7.13 Memakainya Sungguhan

Bentuk-bentuk dalam bab ini muncul di setiap program yang mengurus pembukuan, apa pun usahanya.

- **Parameter keluaran dengan status.** Timbangan sebuah lapak pasar membaca berat: `int read_weight(double *kg)` mengembalikan 1 dan mengisi `*kg`, atau mengembalikan 0 ketika timbangan tidak mengirim apa-apa. Pemanggil tidak pernah memakai berat yang tidak terbaca.
- **Beberapa statistik dari satu penelusuran.** Kebun buah dan durian menghitung, dari satu penelusuran atas panen seminggu, total, rata-rata per pohon, pohon terlemah, dan pohon terkuat, dengan pola `day_stats` tanpa perubahan.
- **Mengurutkan menurut aturan yang dipilih saat program berjalan.** Kios fotokopi mengurutkan pesanannya menurut harga untuk pemilik, menurut waktu untuk antrean, dan menurut nama pelanggan untuk tagihan bulanan: satu pengurutan, tiga fungsi pembanding, dan aturannya dipilih dari menu dengan array pointer fungsi.

- **Pointer ke tengah.** Penjualan sebulan sebuah warung ada dalam satu array 31 hari; `total(sales + 7, 7)` adalah minggu kedua, tanpa menyalin apa pun.
- **Dua pointer yang saling mendekat.** Penjahit yang memeriksa apakah sehelai kain bermotif simetris, atau petugas yang memeriksa apakah sebuah pelat nomor terbaca sama dari kedua ujung, memakai penelusuran dua pointer dari `pointer_walk.c`; Bab 8 mengubahnya menjadi pemeriksa palindrom.
- **Memecah besaran.** `split_weight` menjadi `split_duration(int minutes, int *hours, int *mins)` untuk penyewaan motor, atau `split_money(long rupiah, long *notes_100k, long *rest)` untuk laci uang (Soal Latihan 7.4).

7.14 Kesalahan yang Sering Terjadi, dan Cara Menghindarinya

- **Lupa `&` pada pemanggilan.** `swap(left, right)` meneruskan nilai di tempat yang mengharapkan alamat; `gcc` memperingatkan *passing argument 1 makes pointer from integer without a cast*. Perlakukan peringatan itu sebagai galat.
- **Lupa `*` di dalam fungsi.** `a = b;` di dalam `swap` menukar salinan lokal alamatnya dan tidak mengubah apa pun; `*a = *b;` mengubah variabelnya.
- **Pointer yang tidak diinisialisasi.** `int *p; *p = 5;` menulis ke alamat acak. Inisialisasi saat deklarasi.
- **Mengikuti `NULL`.** Ujilah pointer yang mungkin null sebelum memakainya.
- **Mengembalikan alamat variabel lokal.** Variabelnya mati saat fungsi kembali; pointernya menggantung. Kembalikan nilai atau pakai parameter keluaran.
- **Mencampuradukkan dua bintang.** `int *p = &x;` mendeklarasikan dan menginisialisasi `p`; sesudahnya, `*p = 7;` menulis ke `x`. `p = 7;` akan menaruh bilangan 7 ke dalam pointer (dan memancing peringatan).
- **Aritmetika pointer dalam bita.** `p + 1` maju satu elemen, bukan satu bita; jangan mengalikan dengan `sizeof` sendiri.
- **Berjalan melewati ujung.** Pointer boleh menunjuk satu sesudah elemen terakhir, tetapi tidak boleh diikuti di sana. Berulanglah dengan `q < a + n`.
- **`sizeof` pada pointer.** `sizeof p` adalah ukuran sebuah alamat, bukan ukuran array yang ditunjuknya; panjangnya harus dibawa sebagai parameter terpisah.
- **`a - b` dalam fungsi pembanding.** Meluap untuk nilai besar; pakailah Persamaan (7.5).
- **Mendeklarasikan dua pointer dalam satu baris.** `long *a, b;` hanya menjadikan `a` pointer. Tulislah `long *a; long *b;`.
- **Membandingkan pointer alih-alih nilai.** `if (p == q)` bertanya apakah dua pointer menunjuk tempat yang sama; `if (*p == *q)` bertanya apakah kedua nilainya sama. Keduanya berguna, dan keduanya bukan pertanyaan yang sama.
- **Mengira Python mengubah bilangan milik pemanggil.** Mengikat ulang parameter tidak pernah sampai ke pemanggil. Kembalikan nilai baru, kembalikan tuple, atau ubah list yang dipakai bersama.

7.15 Ringkasan Bab

- Setiap variabel punya alamat; `&x` memberikannya, dan pointer adalah variabel yang menyimpannya. `*p` mengikuti pointer ke variabelnya, sehingga `*(&x)` adalah `x`, Persamaan (7.1). `NULL` tidak menunjuk ke mana pun dan tidak boleh diikuti.

- Meneruskan alamat memungkinkan fungsi mengubah variabel pemanggilnya (pass-by-reference, yang dibangun dari pass-by-value atas sebuah alamat), seperti dilakukan `swap` dan `scanf`.
- Parameter keluaran memberi fungsi beberapa jawaban; status yang dikembalikan menyatakan apakah jawabannya sah. Python mengembalikan tuple dan `None` sebagai gantinya.
- Nama array berubah menjadi alamat elemen pertamanya. Aritmetika pointer bergerak per elemen, Persamaan (7.3), dan `a[i]` didefinisikan sebagai `*(a + i)`, Persamaan (7.4). Mengurangkan pointer ke dalam satu array menghitung elemen di antara keduanya.
- `const` sebelum tipe melindungi variabel yang ditunjuk; `const` sesudah bintang melindungi pointernya, **Tabel 7.1**.
- Pointer ke fungsi meneruskan perilaku sebagai parameter; `qsort` mengurutkan array apa pun dengan fungsi pembanding yang mengembalikan tanda perbandingan, Persamaan (7.5), bukan selisih mentah.
- Kasir versi 6 mendapatkan empat statistik dari satu pemanggilan, memecah berat hari itu dengan Persamaan (7.2), dan mengurutkan tagihan dengan `qsort`.
- Inisialisasi pointer, uji terhadap `NULL`, jangan pernah mengembalikan alamat variabel lokal, dan kembangkan program dengan `-fsanitize=address,undefined`.

7.16 Soal Latihan

1. **Gambarlah memorinya.** Untuk `int a = 3, b = 8; int *p = &a, *q = &b; *p = *q + 1; p = q; *p = 20;`, gambarlah kotak dan panahnya setelah setiap pernyataan, lalu sebutkan nilai `a`, `b`, `*p`, dan `*q` di akhir.
2. **Minimum dan maksimum.** Tulislah `void min_max(const int *a, int n, int *lo, int *hi)` yang menemukan keduanya dalam satu penelusuran dengan pointer. Apa yang sebaiknya dilakukan fungsi itu ketika `n` bernilai 0, dan bagaimana pemanggil mengetahuinya?
3. **Panjang dan pencacahan dengan pointer.** Tulislah `size_t my_strlen(const char *s)` yang berjalan sampai `'\0'` penutup dan mengembalikan jaraknya, serta `int count_char(const char *s, char c)`. Ujilah pada string kosong, pada `"laundry"`, dan pada `"cuci setrika express"`.
4. **Kembalian dalam lembaran uang.** Seorang pelanggan membayar Rp200.000 untuk tagihan Rp136.500. Tulislah `int take_notes(long *remaining, long note)` yang mengembalikan berapa lembar uang bernilai `note` yang muat dalam `*remaining` dan mengurangi `*remaining` sesuai jumlah itu, lalu pakailah untuk lembaran Rp100.000, 50.000, 20.000, 10.000, 5.000, 2.000, dan 1.000.
5. **Mana yang sah?** Dengan `long bill[5]; long *p = bill; const long *c = bill; long *const k = bill;`, mana di antara berikut yang dapat dikompilasi, dan apa yang dilakukan masing-masing: `p++`, `c++`, `k++`, `*p = 1`, `*c = 1`, `*k = 1`, `p = c`?
6. **Mengurutkan dengan aturan baru.** Tambahkan fungsi pembanding yang menaruh tagihan Rp50.000 atau lebih di depan (pesanan besar, untuk merencanakan pengantaran) dan selain itu mempertahankan urutan asli. Mengapa pengurutan yang Anda berikan kepadanya harus stabil?
7. **Jelaskan kegagalannya.** Sebuah program mendeklarasikan `char *name;` lalu memanggil `scanf("%s", name);`. Program itu berhenti mendadak di sebagian komputer dan tampak berjalan di komputer lain. Jelaskan, lalu perbaiki dengan array.

7.17 Penyelesaian Soal Latihan

1. Setelah `*p = *q + 1;`, `a` bernilai 9 (8 + 1) dan `b` bernilai 8. Setelah `p = q;`, kedua pointer menunjuk `b`, dan `a` tidak lagi terjangkau melalui pointer. Setelah `*p = 20;`, `b` bernilai 20. Hasil akhir: `a` = 9, `b` = 20, `*p` = 20, `*q` = 20.

2. Berjalanlah dengan `const int *p` dari `a` sampai `a + n`, dengan kedua jawaban dimulai dari `a[0]`. Untuk `n` = 0 tidak ada minimum; seperti `day_stats`, kembalikan status `int` sebagai ganti `void`: `int min_max(...)` mengembalikan 0 dan membiarkan `*lo` serta `*hi` tak tersentuh, dan pemanggil menguji statusnya. Bab 8 menyelesaikan masalah yang sangat mirip secara lengkap.

3. **Kode 7.18** dan **Kode 7.19** menyelesaikannya; versi C menggerakkan pointer `p` sampai ke nol lalu mengembalikan `p - s`, banyaknya karakter yang dilewati, dengan selisih pointer dari Subbab 7.6.2. `count_char` menggerakkan salinan parameter `s` miliknya sendiri; pointer milik pemanggil tidak terpengaruh, karena pointer itu sendiri diteruskan sebagai nilai.

Kode 7.18 `ex7_strlen.c`: panjang string dengan selisih pointer, dan pencacahan sebuah karakter dengan penelusuran pointer.

```c
/* ex7_strlen.c - Latihan 7.3: panjang string dan pencacahan, dengan pointer */
#include <stdio.h>

size_t my_strlen(const char *s)
{
    const char *p = s;
    while (*p != '\0')                  /* berjalan sampai nol penutup */
        p++;
    return (size_t) (p - s);            /* jarak = banyaknya karakter */
}

int count_char(const char *s, char c)
{
    int count = 0;
    for (; *s; s++)                     /* s adalah salinan alamat milik kita */
        if (*s == c)
            count++;
    return count;
}

int main(void)
{
    const char *words[3] = {"", "laundry", "cuci setrika express"};
    for (int i = 0; i < 3; i++)
        printf("\"%s\": length %zu, %d letter e, %d spaces\n", words[i],
               my_strlen(words[i]), count_char(words[i], 'e'),
               count_char(words[i], ' '));
    return 0;
}
```

Kode 7.19 `ex7_strlen.py`: hitungan yang sama dalam Python, yang string-nya tidak membawa nol penutup.

```python
# ex7_strlen.py - Latihan 7.3: panjang string dan pencacahan, dengan indeks


def my_strlen(s):
    n = 0
    for _ in s:                          # string Python tidak membawa '\0'
        n += 1
    return n


def count_char(s, c):
    count = 0
    for ch in s:
        if ch == c:
            count += 1
    return count


def main():
    for w in ["", "laundry", "cuci setrika express"]:
        print(f'"{w}": length {my_strlen(w)}, {count_char(w, "e")} letter e, '
              f'{count_char(w, " ")} spaces')


if __name__ == "__main__":
    main()
```

```
"": length 0, 0 letter e, 0 spaces
"laundry": length 7, 0 letter e, 0 spaces
"cuci setrika express": length 20, 3 letter e, 2 spaces
```

4. **Kode 7.20** meneruskan alamat `change` ke setiap pemanggilan, sehingga setiap pemanggilan mengambil lembarannya dari sisa yang ditinggalkan pemanggilan sebelumnya. Dalam Python, **Kode 7.21**, fungsinya mengembalikan banyaknya lembar dan sisa yang baru sebagai tuple.

Kode 7.20 `ex7_change.c`: kembalian dalam lembaran uang, setiap pemanggilan mengurangi sisa jumlah milik pemanggil melalui pointer.

```c
/* ex7_change.c - Latihan 7.4: uang kembalian; tiap panggilan mengambil
   sebisanya dari sisa jumlah, lewat pointer */
#include <stdio.h>

int take_notes(long *remaining, long note)
{
    int count = (int) (*remaining / note);
    *remaining -= count * note;          /* jumlah milik pemanggil menyusut */
    return count;
}

int main(void)
{
    long notes[7] = {100000, 50000, 20000, 10000, 5000, 2000, 1000};
    long paid = 200000, bill = 136500;
    long change = paid - bill;
    printf("change for Rp %ld: Rp %ld\n", bill, change);
    for (int i = 0; i < 7; i++) {
        int k = take_notes(&change, notes[i]);
        if (k > 0)
            printf("  %d x %ld\n", k, notes[i]);
    }
    printf("left as coins: Rp %ld\n", change);
    return 0;
}
```

Kode 7.21 `ex7_change.py`: hal yang sama dengan tuple yang dikembalikan.

```python
# ex7_change.py - Latihan 7.4: uang kembalian; tiap panggilan mengembalikan
# banyaknya DAN sisa jumlah baru, karena Python tak punya parameter keluaran


def take_notes(remaining, note):
    count = remaining // note
    return count, remaining - count * note


def main():
    notes = [100000, 50000, 20000, 10000, 5000, 2000, 1000]
    paid, bill = 200000, 136500
    change = paid - bill
    print(f"change for Rp {bill}: Rp {change}")
    for note in notes:
        k, change = take_notes(change, note)
        if k > 0:
            print(f"  {k} x {note}")
    print(f"left as coins: Rp {change}")


if __name__ == "__main__":
    main()
```

```
change for Rp 136500: Rp 63500
  1 x 50000
  1 x 10000
  1 x 2000
  1 x 1000
left as coins: Rp 500
```

Kembalian Rp63.500 terdiri atas satu lembar 50.000, satu lembar 10.000, satu lembar 2.000, dan satu lembar 1.000, dengan sisa Rp500 untuk uang logam. Cara rakus (*greedy*) ini, yang selalu mengambil lembaran terbesar yang muat, memberikan jumlah lembaran paling sedikit untuk pecahan rupiah, walaupun tidak untuk setiap kemungkinan susunan nilai lembaran.

5. `p++` dapat dikompilasi dan memindahkan `p` ke `bill[1]`. `c++` dapat dikompilasi (pointernya tidak const). `k++` tidak dapat dikompilasi (pointernya const). `*p = 1` dapat dikompilasi dan menulis `bill[0]`. `*c = 1` tidak dapat dikompilasi (nilai yang ditunjuk bersifat const melalui `c`). `*k = 1` dapat dikompilasi. `p = c` memancing peringatan (*discards const qualifier*): penugasan itu akan membolehkan `p` menulis di tempat yang dijanjikan `c` tidak akan ditulisi; C++ menolaknya mentah-mentah.

6. `int big_first(long a, long b) { return a >= 50000 && b < 50000; }` menyatakan bahwa `a` harus mendahului `b` hanya ketika `a` besar dan `b` tidak. Di antara dua tagihan besar, atau dua tagihan kecil, aturan itu tidak berkata apa-apa, dan hanya pengurutan yang stabil yang mempertahankannya dalam urutan nota; pengurutan yang tidak stabil seperti `qsort` bisa mengacaknya, dan daftar pengantaran akan berubah dari satu kali jalan ke kali berikutnya.

7. `char *name;` adalah pointer yang tidak diinisialisasi: ia menyimpan alamat acak, dan `scanf` menulis karakter yang diketik ke sana. Di tempat alamat itu kebetulan berada di memori yang boleh ditulisi, program tampak bekerja sambil merusak data lain; di tempat lain program berhenti mendadak. Perbaikannya adalah memberi `scanf` memori sungguhan: `char name[50]; scanf("%49s", name);` (nama array berubah menjadi alamat elemen pertamanya, sehingga `&` tidak diperlukan, dan lebar 49 menyisakan tempat untuk `'\0'`).

Daftar Pustaka

[1] ISO/IEC 9899:2018, *Information technology — Programming languages — C*. Geneva, Switzerland: International Organization for Standardization, 2018.

[2] Carnegie Mellon University Software Engineering Institute, "EXP34-C. Do not dereference null pointers," in *SEI CERT C Coding Standard*. [Online]. Available: https://wiki.sei.cmu.edu/confluence/display/c/EXP34-C.+Do+not+dereference+null+pointers

[3] C. A. R. Hoare, "Null references: the billion dollar mistake," presented at QCon London, 2009. [Online]. Available: https://www.infoq.com/presentations/Null-References-The-Billion-Dollar-Mistake-Tony-Hoare/

[4] B. W. Kernighan and D. M. Ritchie, *The C Programming Language*, 2nd ed. Englewood Cliffs, NJ, USA: Prentice Hall, 1988.

[5] Python Software Foundation, "Data model: objects, values and types," in *The Python Language Reference*, Python 3.14 documentation. [Online]. Available: https://docs.python.org/3/reference/datamodel.html#objects-values-and-types

[6] Python Software Foundation, "How do I write a function with output parameters (call by reference)?," in *Programming FAQ*, Python 3.14 documentation. [Online]. Available: https://docs.python.org/3/faq/programming.html#how-do-i-write-a-function-with-output-parameters-call-by-reference

[7] cppreference.com, "qsort, qsort_s." [Online]. Available: https://en.cppreference.com/w/c/algorithm/qsort

[8] Python Software Foundation, "functools — Higher-order functions and operations on callable objects," Python 3.14 documentation. [Online]. Available: https://docs.python.org/3/library/functools.html#functools.cmp_to_key

[9] Carnegie Mellon University Software Engineering Institute, "DCL30-C. Declare objects with appropriate storage durations," in *SEI CERT C Coding Standard*. [Online]. Available: https://wiki.sei.cmu.edu/confluence/display/c/DCL30-C.+Declare+objects+with+appropriate+storage+durations

[10] Free Software Foundation, "Program instrumentation options," in *Using the GNU Compiler Collection (GCC)*. [Online]. Available: https://gcc.gnu.org/onlinedocs/gcc/Instrumentation-Options.html

Bab 8

Evaluasi dan Pembahasan II: Fungsi, Array, dan Pointer, Diselesaikan Langkah demi Langkah

Bab ini tidak menambahkan fitur baru C. Bab ini mengambil tiga alat dari Bab 5 sampai 7 (fungsi, array, dan pointer) lalu memakainya untuk enam masalah utuh, masing-masing diselesaikan pelan-pelan dan lengkap. Empat di antaranya masalah klasik bagi siapa pun yang baru belajar memprogram: lima fungsi statistik yang dilengkapi di dalam kerangka yang tidak boleh diubah; bilangan Fibonacci yang dihitung dua kali, dengan perulangan dan dengan rekursi, lalu kedua cara itu diukur satu sama lain; memutuskan apakah sebuah kata palindrom, dengan tiga cara berbeda yang harus sepakat; serta determinan matriks 3×3 beserta pertanyaan apakah matriks itu punya invers. Dua lagi masalah baru yang sekali lagi menggabungkan alat-alat itu: mencari nomor nota dengan pencarian biner rekursif, dan menggabungkan tagihan terurut dari dua sif dengan dua pointer yang berjalan. Untuk setiap masalah Anda akan melihat pernyataannya dengan kata-kata sederhana, penyelesaian yang dikerjakan dengan tangan sebelum sebaris kode pun ditulis, rancangan di atas kertas, program lengkap dalam C dan Python beserta keluarannya yang sungguhan, *solusi salah yang menggoda* yang tampak benar padahal tidak, yang dibongkar untuk memperlihatkan dengan tepat di mana ia gagal, serta pertanyaan "periksa diri" beserta jawabannya. Di akhir bab ini Anda akan mampu menghadapi masalah baru dengan delapan langkah yang sama, menemukan kasus uji yang menyingkap program yang salah, dan menjelaskan mengapa program yang benar itu benar.

8.1 Titik Berangkat Kita

Sebelum menyelesaikan apa pun, mari kita gelar kotak peralatannya, karena menyebut alat yang diperlukan sebuah masalah sudah setengah dari penyelesaiannya.

Dari **Bab 5** Anda memiliki **fungsi**. Fungsi punya nama, parameter, dan nilai kembalian; argumennya disalin ke parameternya; prototipe menjanjikan bentuknya kepada kompilator; pustaka standar menyediakan `sqrt`, `ceil`, dan kerabatnya dari `<math.h>`, yang ditautkan dengan `-lm`; cakupan dan masa hidup sebuah variabel menentukan di mana ia terlihat dan berapa lama ia hidup; dan fungsi rekursif menyelesaikan masalah dengan memanggil dirinya sendiri pada contoh yang lebih kecil, dengan kasus dasar yang menghentikannya. Anda juga melihat bahwa rekursi ada harganya: `fib(30)` melakukan 2.692.537 pemanggilan.

Dari **Bab 6** Anda memiliki **array**: deretan elemen bertipe sama dengan satu nama, diindeks dari 0 sampai $n - 1$; perulangan yang menelusurinya dengan `i < n`; penghitung yang diindeks dengan data; array yang diteruskan ke fungsi sebagai alamat elemen pertamanya, beserta panjangnya; bubble sort, insertion sort, pencarian linear, dan pencarian biner; serta array dua dimensi yang disimpan baris demi baris.

Dari **Bab 7** Anda memiliki **pointer**: `&` mengambil alamat dan `*` mengikutinya; fungsi yang menerima alamat dapat mengubah variabel pemanggilnya dan mengembalikan beberapa

jawaban melalui parameter keluaran; `a[i]` adalah `*(a + i)`, sehingga pointer dapat menelusuri array dan dua pointer dapat berjalan saling mendekat; `const` melindungi yang ditunjuk pointer; dan fungsi dapat diteruskan sebagai parameter.

Tabel 8.1 mendaftar keenam masalah dan alat yang paling banyak dilatih masing-masing.

Tabel 8.1 Enam masalah dalam bab ini dan alat dari Bab 5 sampai 7 yang paling banyak dilatih masing-masing.

Soal	Tugas	Alat utama
8.1	Lima fungsi statistik dalam kerangka yang diberikan	parameter array, perulangan, cast, `sqrt`
8.2	Fibonacci dengan perulangan dan rekursi, serta jumlahnya	parameter keluaran, rekursi, menghitung pemanggilan, pengukuran waktu
8.3	Apakah sebuah kata palindrom?	string sebagai array, dua pointer, rekursi, XOR
8.4	Determinan 3 × 3 dan keterbalikan	array dua dimensi, satu rumus yang disalin persis
8.5	Mencari nomor nota	pencarian biner rekursif pada array terurut
8.6	Menggabungkan tagihan dua sif	dua pointer yang berjalan, array keluaran

8.2 Cara Menghadapi Sebuah Masalah

8.2.1 Delapan langkah

Matematikawan George Pólya menguraikan pemecahan masalah dalam empat tahap: pahami masalahnya, susun rencana, jalankan rencana, tinjau kembali [1]. Untuk program, tahap-tahap yang sama terurai menjadi delapan langkah, dan setiap masalah dalam bab ini mengikutinya secara berurutan.

1. **Nyatakan ulang masalahnya dengan kata-kata sendiri**, dan tuliskan setiap keputusan yang dibiarkan terbuka oleh pernyataannya: apa persisnya yang dibaca, apa persisnya yang dicetak, apa yang terjadi dengan masukan yang janggal.
2. **Selesaikan sebuah contoh kecil dengan tangan**, di atas kertas, sampai digit terakhir. Jika Anda tidak sanggup mengerjakannya dengan tangan, Anda tidak dapat memberi tahu komputer cara mengerjakannya, dan Anda tidak akan punya pembanding untuk keluaran program.
3. **Rancang di atas kertas**: fungsi apa, dengan parameter dan nilai kembalian apa; array apa; perulangan atau rekursi apa. Tuliskan rancangannya sebagai beberapa baris pseudokode atau sebagai prototipe fungsi.
4. **Perkirakan biayanya**: berapa langkah untuk masukan berukuran n? Rancangan yang memerlukan 2^n langkah tidak berguna untuk $n = 60$, apa pun bahasanya.
5. **Tulis kodenya**, satu fungsi setiap kali, mengompilasi setelah setiap fungsi dengan `-Wall -Wextra`, dan perbaiki setiap peringatan.
6. **Uji**: contoh dari langkah 2, kasus-kasus tepi (masukan terkecil, masukan kosong, masukan terbesar, nilai-nilai yang sama), dan masukan yang akan salah dijawab program yang keliru.
7. **Carilah solusi salah yang menggoda.** Tanyakan apa yang akan ditulis pemrogram yang tergesa-gesa, lalu temukan masukan yang membuatnya gagal. Jika program Anda sendiri lolos pada masukan itu, Anda memahami masalahnya.
8. **Tinjau kembali**: bisakah lebih sederhana, lebih cepat, atau lebih jelas? Apa yang berubah jika masalahnya sedikit berubah?

> **WAWASAN KUNCI:** Langkah 2 adalah langkah yang dilewati pemula dan tidak pernah dilewati para ahli. Contoh yang dikerjakan dengan tangan adalah satu-satunya hal yang memberi tahu Anda *seperti apa* keluaran yang benar; tanpanya, program yang mencetak angka-angka masuk akal tampak benar, dan kesalahannya baru muncul ketika orang lain memeriksa.

8.2.2 Susunan setiap masalah, dan letak kodenya

Setiap subbab masalah memiliki enam bagian yang sama: masalahnya, memahaminya dengan tangan, rancangan di atas kertas, penyelesaian dalam C, penyelesaian dalam Python, dan solusi salah yang menggoda disusul pertanyaan "periksa diri". Setiap program lengkap dan telah dikompilasi dengan `gcc -std=c17 -O2 -Wall -Wextra` dan, untuk kedua kalinya, dengan `-fsanitize=address,undefined`, dan setiap keluaran yang tercetak di sini adalah keluaran yang benar-benar dihasilkan program itu. Versi Python setiap program mencetak teks yang persis sama. Semua berkas ada di folder `Chapter08` pada kode buku ini, dengan masukan di `tests` dan skrip `run_tests.py` yang mengompilasi setiap program C, menjalankannya bersama kembaran Python-nya, lalu membandingkan kedua keluaran dengan keluaran yang diharapkan.

> **PERINGATAN:** Bacalah setiap masalah dan cobalah langkah 1 sampai 3 sendiri *sebelum* membaca penyelesaiannya. Mengikuti sebuah penyelesaian itu mudah; menghasilkannya itulah keterampilan yang dilatih. Penyelesaian di sini ada untuk dibandingkan, bukan untuk menggantikan usaha Anda.

8.3 Soal 8.1: Fungsi-Fungsi Statistik

8.3.1 Masalahnya

> **SOAL 8.1.** Anda diberi fungsi `main` yang lengkap, sepuluh bilangan bulat dalam sebuah array, dan prototipe lima fungsi: `max`, `min`, `sum`, `average`, dan `sDeviation`, yang masing-masing menerima pointer ke array dan ukurannya sebagai `unsigned int`. Lengkapi kelima badan fungsi itu, tanpa mengubah `main`, nama-nama, atau prototipenya, sehingga program mencetak nilai terbesar, nilai terkecil, jumlah, rata-rata, dan simpangan baku, dua yang terakhir dengan tiga desimal.

Kerangka itu sudah menentukan banyak hal, dan salah satu ketentuannya adalah bagian dari masalah: kode yang diberikan bersifat tetap. Dalam tim sungguhan hal ini lumrah; prototipe adalah kesepakatan dengan kode milik orang lain, dan "memperbaiki" sebuah nama (`Max`, atau `int max(int *a, int n)`) melanggar kesepakatan itu walaupun programnya masih jalan. Kesepuluh bilangan, dalam versi kita, adalah kilogram cucian yang dibawa satu pelanggan, sebuah rumah kos kecil, pada sepuluh kunjungan: 12, 7, 15, 9, 20, 11, 8, 14, 10, dan 18. Simpangan bakunya adalah simpangan baku *populasi*, yang membagi dengan banyaknya nilai n; Subbab 8.3.6 kembali ke pilihan ini.

8.3.2 Memahaminya dengan tangan

Nilai terbesar 20 dan terkecil 7, cukup dengan melihat. Jumlahnya $12 + 7 + 15 + 9 + 20 + 11 + 8 + 14 + 10 + 18 = 124$. Rata-rata, yaitu rerata aritmetika, adalah

$$\bar{x} = \frac{1}{n}\sum_{i=0}^{n-1} x_i = \frac{124}{10} = 12.4 \tag{8.1}$$

Simpangan baku (*standard deviation*) mengukur seberapa jauh nilai-nilai lazimnya terletak dari rata-rata. Untuk setiap nilai hitung jaraknya dari rata-rata, $x_i - \bar{x}$, kuadratkan (supaya jarak di bawah dan di atas rata-rata tidak saling meniadakan, dan jarak besar lebih diperhitungkan), ambil rata-rata kuadratnya, lalu akar kuadratnya, yang mengembalikan satuannya ke kilogram:

$$\sigma = \sqrt{\frac{1}{n}\sum_{i=0}^{n-1}(x_i - \bar{x})^2} \tag{8.2}$$

Tabel 8.2 mengerjakan Persamaan (8.2) dengan tangan untuk kesepuluh nilai itu.

Tabel 8.2 Simpangan baku kesepuluh nilai yang dihitung dengan tangan: simpangan-simpangannya berjumlah nol, dan kuadratnya berjumlah 166,40.

x_i	$x_i - x$	$(x_i - x)^2$
12	−0,4	0,16
7	−5,4	29,16
15	2,6	6,76
9	−3,4	11,56
20	7,6	57,76
11	−1,4	1,96
8	−4,4	19,36
14	1,6	2,56
10	−2,4	5,76
18	5,6	31,36
jumlah 124	**0,0**	**166,40**

Kolom simpangan berjumlah tepat nol, sebagaimana selalu harus demikian (rata-rata adalah titik keseimbangan data), dan ini pemeriksaan hitungan yang berguna. Rata-rata kuadratnya adalah $166.40/10 = 16.64$, dan $\sigma = \sqrt{16.64} \approx 4.0792$, yang dicetak dengan tiga desimal sebagai 4.079. Jadi program harus mencetak 20, 7, 124, 12.400, dan 4.079. Dengan kata lain: pelanggan ini membawa sekitar 12,4 kg per kunjungan, kurang atau lebih sekitar 4 kg.

8.3.3 Merancangnya di atas kertas

Kerangka sudah menetapkan kelima prototipe, sehingga rancangannya adalah badan masing-masing fungsi.

- `max`: mulai dengan `INT_MIN`, `int` terkecil yang ada (dari `<limits.h>`), dan biarkan setiap elemen yang lebih besar menggantikannya. Mulai dari `INT_MIN` berarti elemen pertama pasti menggantikannya, sehingga tidak perlu kasus khusus untuk elemen pertama. `min` mencerminkannya dengan `INT_MAX` dan `<`.
- `sum`: penampung yang mulai dari 0.
- `average`: satu baris, `(double) sum(a, aSize) / aSize`. Cast harus mengubah sebuah operan *sebelum* pembagian; ini pelajaran cast dari Bab 2 sekali lagi.
- `sDeviation`: hitung rata-ratanya sekali, lalu kumpulkan kuadrat simpangan dalam `double`, bagi dengan `aSize`, dan ambil `sqrt`. Fungsi ini memakai ulang `average`, yang memakai ulang `sum`: bagi dan taklukkan, Bab 5.

Parameter ukurannya bertipe `unsigned int`. Penghitung perulangan yang dideklarasikan `int i` dan dibandingkan dengan `i < aSize` mencampur nilai bertanda dan tak bertanda, dan `-Wextra` memperingatkannya (*comparison of integer expressions of different signedness*). Mendeklarasikan penghitung sebagai `unsigned int i` membuat build tetap senyap dan merupakan tipe yang jujur: indeks tidak pernah negatif. Biayanya: setiap fungsi menelusuri array sekali, sehingga kelimanya bersama-sama memerlukan beberapa puluh langkah untuk sepuluh bilangan; `sDeviation` menelusurinya tiga kali (dua kali di dalam `average` dan `sum`, sekali oleh dirinya sendiri), tetap linear dalam n.

8.3.4 Penyelesaian dalam C

Kode 8.1 adalah kerangka yang sudah dilengkapi. `main` dan kelima prototipe persis seperti yang diberikan; hanya badan-badan fungsinya yang milik kita.

Kode 8.1 `stats.c`: kerangka yang diberikan dengan kelima badan fungsi yang dilengkapi; `INT_MIN` dan `INT_MAX` memulai pencarian, cast datang sebelum pembagian, dan simpangan memakai `sqrt`.

```c
/* stats.c - Soal 8.1: lima fungsi statistik yang dilengkapi di dalam
   kerangka yang diberikan (main dan prototipe persis seperti diberikan) */
#include <stdio.h>
#include <limits.h>
#include <math.h>

int max(int *a, unsigned int aSize);
int min(int *a, unsigned int aSize);
int sum(int *a, unsigned int aSize);
double average(int *a, unsigned int aSize);
double sDeviation(int *a, unsigned int aSize);

int main(void)
{
    int kg[10] = {12, 7, 15, 9, 20, 11, 8, 14, 10, 18};
    unsigned int n = 10;

    printf("Maximum value      : %d\n", max(kg, n));
    printf("Minimum value      : %d\n", min(kg, n));
    printf("Sum                : %d\n", sum(kg, n));
    printf("Average            : %.3f\n", average(kg, n));
    printf("Standard deviation : %.3f\n", sDeviation(kg, n));
    return 0;
}

int max(int *a, unsigned int aSize)
{
    int result = INT_MIN;                /* lebih kecil dari nilai nyata mana pun */
    for (unsigned int i = 0; i < aSize; i++)
        if (a[i] > result)
            result = a[i];
    return result;
}

int min(int *a, unsigned int aSize)
{
    int result = INT_MAX;                /* lebih besar dari nilai nyata mana pun */
    for (unsigned int i = 0; i < aSize; i++)
        if (a[i] < result)
            result = a[i];
    return result;
}
```

Kode 8.1 (Lanjutan) *stats.c: kerangka yang diberikan dengan kelima badan fungsi yang dilengkapi; INT_MIN dan INT_MAX memulai pencarian, cast datang sebelum pembagian, dan simpangan memakai sqrt.*

```c
int sum(int *a, unsigned int aSize)
{
    int result = 0;
    for (unsigned int i = 0; i < aSize; i++)
        result += a[i];
    return result;
}

double average(int *a, unsigned int aSize)
{
    return (double) sum(a, aSize) / aSize;   /* cast SEBELUM membagi */
}

double sDeviation(int *a, unsigned int aSize)
{
    double mean = average(a, aSize);
    double squares = 0.0;
    for (unsigned int i = 0; i < aSize; i++) {
        double d = a[i] - mean;          /* jarak dari rata-rata */
        squares += d * d;
    }
    return sqrt(squares / aSize);        /* simpangan baku populasi */
}
```

```
Maximum value      : 20
Minimum value      : 7
Sum                : 124
Average            : 12.400
Standard deviation : 4.079
```

Kelima baris itu cocok dengan hitungan tangan. Bangunlah dengan `-lm` di akhir perintah di Linux, karena `sqrt` (Bab 5):

```
gcc -std=c17 -Wall -Wextra -o stats stats.c -lm
```

8.3.5 Penyelesaian dalam Python

Python sudah memiliki fungsi bawaan bernama `max`, `min`, dan `sum`; mendefinisikan fungsi sendiri dengan nama-nama itu akan menyembunyikan fungsi bawaan tersebut di sisa berkas. Karena itu **Kode 8.2** menamainya `max_`, `min_`, dan `sum_`, kebiasaan Python yang lazim untuk nama yang sebenarnya akan bertabrakan. Python tidak memiliki `INT_MIN`, karena bilangan bulatnya tidak berbatas; `-math.inf` dan `math.inf`, minus dan plus tak hingga, memainkan peran yang sama.

Kode 8.2 `stats.py`: kelima fungsi yang sama dalam Python, tempat `/` selalu menghasilkan pecahan.

```python
# stats.py - Soal 8.1: lima fungsi statistik yang sama dalam Python
import math


def max_(a):                                # max_ : Python sudah punya max()
    result = -math.inf                      # lebih kecil dari nilai nyata mana pun
    for x in a:
        if x > result:
            result = x
    return result


def min_(a):
    result = math.inf                       # lebih besar dari nilai nyata mana pun
    for x in a:
        if x < result:
            result = x
    return result


def sum_(a):
    result = 0
    for x in a:
        result += x
    return result


def average(a):
    return sum_(a) / len(a)                 # / selalu menghasilkan float di Python


def s_deviation(a):
    mean = average(a)
    squares = 0.0
    for x in a:
        d = x - mean                        # jarak dari rata-rata
        squares += d * d
    return math.sqrt(squares / len(a))      # simpangan baku populasi
```

Kode 8.2 (Lanjutan) *stats.py: kelima fungsi yang sama dalam Python, tempat / selalu menghasilkan pecahan.*

```python
def main():
    kg = [12, 7, 15, 9, 20, 11, 8, 14, 10, 18]
    print(f"Maximum value      : {max_(kg)}")
    print(f"Minimum value      : {min_(kg)}")
    print(f"Sum                : {sum_(kg)}")
    print(f"Average            : {average(kg):.3f}")
    print(f"Standard deviation : {s_deviation(kg):.3f}")


if __name__ == "__main__":
    main()
```

Keluarannya sama persis. Modul pustaka standar `statistics` menyediakan `mean` dan `pstdev`, yang menghitung dua bilangan yang sama [2]; menulisnya sendiri sekali adalah cara untuk mengetahui apa yang dikerjakan fungsi-fungsi pustaka itu.

8.3.6 Solusi salah yang menggoda

Dua jalan pintas menggoda semua orang. Yang pertama memulai `min` dari 0 “karena penghitung dimulai dari 0”. Yang kedua menulis rata-rata sebagai `sum(a, aSize) / aSize`, yang tampak

persis seperti rumusnya. **Kode 8.3** melakukan kedua kesalahan itu, dan **Kode 8.4** mengulanginya dalam Python.

Kode 8.3 `wrong_stats.c`: solusi salah yang menggoda, yang memulai minimum dari 0 dan membagi dua bilangan bulat.

```c
/* wrong_stats.c - solusi salah yang menggoda untuk Soal 8.1 (jangan ditiru) */
#include <stdio.h>
#include <math.h>

int min(int *a, unsigned int aSize)
{
    int result = 0;                      /* SALAH: 0 lebih kecil dari semua nilai */
    for (unsigned int i = 0; i < aSize; i++)
        if (a[i] < result)
            result = a[i];
    return result;
}

int sum(int *a, unsigned int aSize)
{
    int result = 0;
    for (unsigned int i = 0; i < aSize; i++)
        result += a[i];
    return result;
}

double average(int *a, unsigned int aSize)
{
    return sum(a, aSize) / aSize;        /* SALAH: dibagi bulat lebih dulu */
}

double sDeviation(int *a, unsigned int aSize)
{
    double mean = average(a, aSize);
    double squares = 0.0;
    for (unsigned int i = 0; i < aSize; i++)
        squares += (a[i] - mean) * (a[i] - mean);
    return sqrt(squares / aSize);
}

int main(void)
{
    int kg[10] = {12, 7, 15, 9, 20, 11, 8, 14, 10, 18};
    unsigned int n = 10;

    printf("Minimum value      : %d\n", min(kg, n));
    printf("Average            : %.3f\n", average(kg, n));
    printf("Standard deviation : %.3f\n", sDeviation(kg, n));
    return 0;
}
```

Kode 8.4 `wrong_stats.py`: dua kesalahan yang sama dalam Python, tempat pembagian bilangan bulat ditulis `//`.

```python
# wrong_stats.py - solusi salah yang menggoda untuk Soal 8.1 (jangan ditiru)
import math


def min_(a):
    result = 0                          # SALAH: 0 lebih kecil dari semua nilai
    for x in a:
        if x < result:
            result = x
    return result


def average(a):
    return sum(a) // len(a)             # SALAH: // adalah pembagian bilangan bulat


def s_deviation(a):
    mean = average(a)
    return math.sqrt(sum((x - mean) ** 2 for x in a) / len(a))


def main():
    kg = [12, 7, 15, 9, 20, 11, 8, 14, 10, 18]
    print(f"Minimum value      : {min_(kg)}")
    print(f"Average            : {average(kg):.3f}")
    print(f"Standard deviation : {s_deviation(kg):.3f}")


if __name__ == "__main__":
    main()
```

```
Minimum value      : 0
Average            : 12.000
Standard deviation : 4.099
```

Minimumnya 0, nilai yang sama sekali tidak ada di dalam data: setiap nilai dibandingkan dengan 0, tidak ada yang lebih kecil, dan 0 bertahan. Rata-ratanya 12.000 karena `124 / 10` dengan dua bilangan bulat bernilai 12; pecahannya dibuang *sebelum* hasilnya diubah menjadi `double` untuk dikembalikan. Dan simpangan bakunya, 4.099, salah dengan cara yang halus: nilai itu dihitung dengan benar *dari rata-rata yang salah*. Dengan 12 sebagai ganti 12,4, kuadrat simpangannya berjumlah 168 alih-alih 166,40, dan $\sqrt{16.8} \approx 4.099$. Angka salah yang dekat dengan angka yang benar adalah jenis yang paling berbahaya, karena lolos dari pandangan sekilas; hanya hitungan tangan di Subbab 8.3.2 yang menyingkapnya.

Mengapa minimum yang salah begitu sering tidak ketahuan? Karena banyak array uji memuat 0 atau bilangan negatif, dan dalam kasus itu memulai dari 0 kebetulan memberi jawaban yang benar. Uji yang menyingkapnya adalah array yang berisi nilai positif saja, seperti milik kita. Memulai dari `INT_MAX`, atau dari elemen pertama `a[0]`, benar untuk setiap masukan.

> **PERIKSA DIRI.** Sebuah program statistik memberi 4.300 alih-alih 4.079 untuk sepuluh nilai yang sama, dan pemrogram yang menulisnya bersikeras bahwa program itu benar. Apa yang dilakukan program itu secara berbeda, dan kapan pilihan itu tepat?
>
> *Jawaban.* Program itu membagi dengan $n - 1 = 9$ alih-alih $n = 10$: $\sqrt{166.40/9} = \sqrt{18.49} \approx 4.300$. Itulah simpangan baku *sampel*, pilihan yang tepat ketika kesepuluh nilai merupakan sampel untuk memperkirakan sebaran populasi yang lebih besar (`statistics.stdev` di Python); membagi dengan n tepat ketika kesepuluh nilai itu adalah seluruh populasi yang diperhatikan (`statistics.pstdev`) [2]. Keduanya bukan kesalahan; kesalahannya adalah tidak menyatakan mana yang dihitung sebuah program.

8.4 Soal 8.2: Fibonacci dengan Dua Cara

8.4.1 Masalahnya

> **SOAL 8.2.** Bacalah sebuah bilangan bulat n. Hitunglah bilangan Fibonacci ke-n, $F(n)$, dimulai dengan $F(0) = 0$ dan $F(1) = 1$, serta jumlah barisan dari $F(0)$ sampai $F(n)$, dengan dua cara: dengan perulangan, dan dengan rekursi. Kemudian bandingkan besarnya kerja yang diperlukan kedua cara itu untuk nilai n di atas 30. Untuk $n = 9$ barisannya adalah 0 1 1 2 3 5 8 13 21 34, sehingga $F(9) = 34$ dan jumlahnya 88.

Contoh itu menetapkan kesepakatannya, dan kesepakatan itu harus ditetapkan lebih dulu: sebagian buku memulai barisan dengan 1, 1, sehingga setiap bilangan bergeser satu tempat. Sisanya kita tentukan sendiri. Program mencetak enam belas bilangan pertama barisan (atau semuanya jika n lebih kecil), dua hasil dari setiap cara, lalu tabel kerja yang diperlukan rekursi untuk n = 20, 25, 30, dan 32. Nilai n di atas 90 ditolak, karena jumlah sampai $F(90)$ adalah yang terbesar yang muat dalam `long long` 64 bit; untuk n di atas 40 cara rekursif dilewati, dengan alasan yang akan menjadi jelas dari tabel itu.

8.4.2 Memahaminya dengan tangan

Untuk $n = 9$: 0, 1, 1, 2, 3, 5, 8, 13, 21, 34. Jumlahnya $0 + 1 + 1 + 2 + 3 + 5 + 8 + 13 + 21 + 34 = 88$. Ada jalan pintas yang layak diketahui. Lanjutkan barisan dua tempat lagi: $F(10) = 55$, $F(11) = 89$. Dan $89 - 1 = 88$. Secara umum

$$\sum_{i=0}^{n} F\,(i) = F(n+2) - 1 \tag{8.3}$$

Pembuktiannya dengan induksi, kembaran matematis rekursi. Untuk $n = 0$: ruas kiri $F(0) = 0$ dan ruas kanan $F(2) - 1 = 1 - 1 = 0$. Jika Persamaan (8.3) berlaku untuk n, maka menambahkan $F(n+1)$ pada kedua ruas memberikan $F(n+2) - 1 + F(n+1) = F(n+3) - 1$ di ruas kanan, yaitu Persamaan (8.3) untuk $n + 1$ [3]. Cara rekursif memakai identitas ini untuk jumlahnya, sehingga tidak perlu memanggil `fib` sekali untuk setiap suku.

Biaya cara rekursif sudah dihitung di Bab 5: `fib(n)` melakukan $C(n) = 2F(n+1) - 1$ pemanggilan. Untuk $n = 32$ itu $2 \cdot 3524578 - 1 = 7049155$ pemanggilan. Perulangan memerlukan n langkah.

8.4.3 Merancangnya di atas kertas

Cara iteratif menyimpan dua variabel yang bergulir, `prev` dan `curr`, yang selalu memegang dua tetangga dalam barisan, serta `sum` yang terus bertambah. Masalah ini menginginkan dua hasil dari satu fungsi, sehingga fungsi mengembalikan $F(n)$ dan memberikan jumlahnya melalui parameter keluaran, `long long *sum`, pola dari Bab 7. Dalam pseudokode:

Pseudokode 8.1 Fibonacci iteratif: dua tetangga dalam barisan bergerak maju n kali, dan jumlahnya mengumpulkan setiap nilai yang pernah dipegang `prev`.

```
FIB-ITER(n)
1  prev = 0, curr = 1, sum = 0          // F(0) dan F(1)
2  repeat n times
3      sum = sum + prev
4      next = prev + curr
5      prev = curr, curr = next
6  sum = sum + prev                     // prev kini F(n)
7  return prev and sum
```

Pseudokode 8.1 dapat ditelusuri untuk $n = 2$: sebelum perulangan `prev` = 0, `curr` = 1; putaran pertama menambahkan 0 ke jumlah dan bergerak ke 1, 1; putaran kedua menambahkan 1 dan bergerak ke 1, 2; setelah perulangan `prev` = 1 = $F(2)$, dan menambahkannya membuat jumlahnya $0 + 1 + 1 = 2$, yaitu $F(4) - 1 = 3 - 1$. Cara rekursif adalah fungsi tiga baris dari Bab 5 dengan penghitung pemanggilan global, ditambah `fib_rec_sum(n)`, yang mengembalikan `fib_rec(n + 2) - 1` menurut Persamaan (8.3).

8.4.4 Penyelesaian dalam C

Kode 8.5 mengikuti rancangan itu baris demi baris; `main` membaca n, memeriksa jangkauannya, mencetak barisan, menjalankan kedua cara, dan menyusun tabel kerja.

Kode 8.5 `fibonacci.c`: cara iteratif dengan jumlah melalui parameter keluaran, cara rekursif dengan penghitung pemanggilan, dan tabel kerja untuk n yang lebih besar.

```c
/* fibonacci.c - Soal 8.2: bilangan Fibonacci ke-n dan jumlah
   barisan sampai suku itu, iteratif dan rekursif, serta biaya masing-masing.
   Masukan: n (0..90). Jalankan dengan --time untuk mengukur detik juga. */
#include <stdio.h>
#include <string.h>
#include <time.h>

long long calls = 0;

long long fib_iter(int n, long long *sum)
{
    long long prev = 0, curr = 1;        /* F(0) dan F(1) */
    *sum = 0;
    for (int i = 0; i < n; i++) {
        *sum += prev;
        long long next = prev + curr;
        prev = curr;
        curr = next;
    }
    *sum += prev;                        /* prev kini F(n) */
    return prev;
}

long long fib_rec(int n)
{
    calls++;
    if (n < 2)
        return n;                        /* F(0) = 0, F(1) = 1 */
    return fib_rec(n - 1) + fib_rec(n - 2);
}

long long fib_rec_sum(int n)             /* F(0) + ... + F(n) = F(n+2) - 1 */
{
    return fib_rec(n + 2) - 1;
}
```

Kode 8.5 (Lanjutan) *fibonacci.c: cara iteratif dengan jumlah melalui parameter keluaran, cara rekursif dengan penghitung pemanggilan, dan tabel kerja untuk n yang lebih besar.*

```c
int main(int argc, char *argv[])
{
    int n;
    int timing = argc > 1 && strcmp(argv[1], "--time") == 0;
    if (scanf("%d", &n) != 1 || n < 0 || n > 90) {
        printf("please give n from 0 to 90\n");
        return 1;
    }
    long long s;
    printf("sequence:");
    for (int i = 0; i <= n && i <= 15; i++)
        printf(" %lld", fib_iter(i, &s));
    printf(n > 15 ? " ...\n" : "\n");

    long long f = fib_iter(n, &s);
    printf("iterative: F(%d) = %lld, sum = %lld\n", n, f, s);
    if (n <= 40) {
        f = fib_rec(n);
        printf("recursive: F(%d) = %lld, sum = %lld\n", n, f, fib_rec_sum(n));
    }

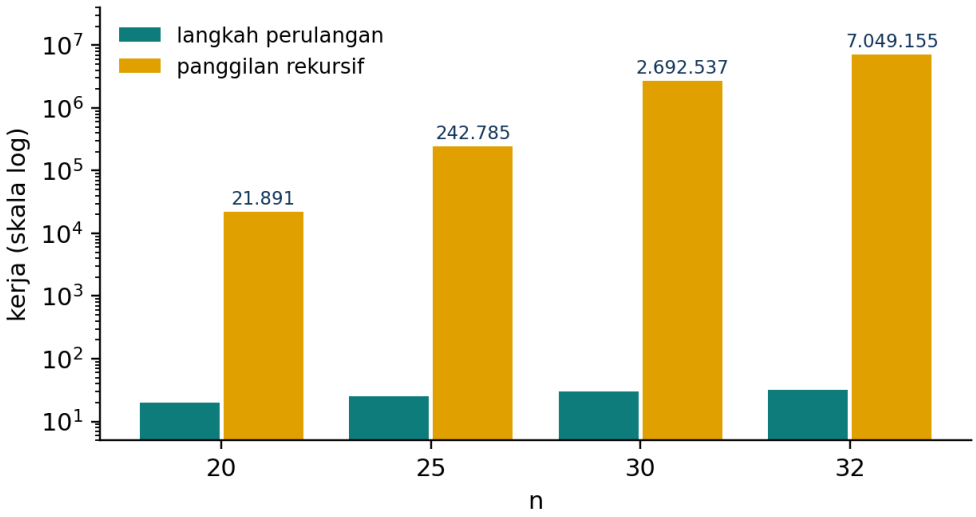
    int sizes[4] = {20, 25, 30, 32};
    printf("   n   loop steps   recursive calls\n");
    for (int k = 0; k < 4; k++) {
        calls = 0;
        clock_t t0 = clock();
        fib_rec(sizes[k]);
        double secs = (double) (clock() - t0) / CLOCKS_PER_SEC;
        printf("%4d %12d %17lld", sizes[k], sizes[k], calls);
        if (timing)
            printf("   %.3f seconds", secs);
        printf("\n");
    }
    return 0;
}
```

Masukannya satu bilangan:

```
9
```

```
sequence: 0 1 1 2 3 5 8 13 21 34
iterative: F(9) = 34, sum = 88
recursive: F(9) = 34, sum = 88
   n   loop steps   recursive calls
  20           20             21891
  25           25            242785
  30           30           2692537
  32           32           7049155
```

Kedua cara sesuai dengan contoh, 34 dan 88. Tabelnya adalah pelajaran masalah ini: sementara perulangan memerlukan 32 langkah untuk $n = 32$, rekursi memerlukan lebih dari tujuh juta pemanggilan, dan setiap kenaikan n sebesar 5 mengalikan pemanggilan dengan sekitar 11. **Gambar 8.1** menggambar kedua kolom itu dengan skala logaritmik, tempat setiap garis kisi sepuluh kali garis di bawahnya; batang perulangan hampir tidak naik, sedangkan batang rekursi naik sepuluh kali lipat setiap dua atau tiga langkah n.



Gambar 8.1 Kerja kedua cara, seperti dihitung oleh `fibonacci.c`: langkah perulangan tumbuh seperti n, pemanggilan rekursif seperti 1.618^n (skala logaritmik).

8.4.5 Penyelesaian dalam Python

Pada **Kode 8.6**, fungsi iteratif mengembalikan tuple berisi dua nilai alih-alih memakai parameter keluaran, dan langkah bergulirnya adalah penugasan tuple `prev, curr = curr, prev + curr`, yang tidak memerlukan variabel sementara.

Kode 8.6 `fibonacci.py`: kedua cara yang sama dalam Python; cara iteratif mengembalikan tuple.

```python
# fibonacci.py - Soal 8.2: bilangan Fibonacci ke-n dan jumlah
# barisan sampai suku itu, iteratif dan rekursif, serta biaya masing-masing.
# Masukan: n (0..90). Jalankan dengan --time untuk mengukur detik juga.
import sys
import time

calls = 0


def fib_iter(n):
    """Kembalikan (F(n), F(0) + ... + F(n))."""
    prev, curr, total = 0, 1, 0          # F(0) dan F(1)
    for _ in range(n):
        total += prev
        prev, curr = curr, prev + curr
    return prev, total + prev            # prev kini F(n)


def fib_rec(n):
    global calls
    calls += 1
    if n < 2:
        return n                         # F(0) = 0, F(1) = 1
    return fib_rec(n - 1) + fib_rec(n - 2)


def fib_rec_sum(n):                      # F(0) + ... + F(n) = F(n+2) - 1
    return fib_rec(n + 2) - 1


def main():
    global calls
    timing = len(sys.argv) > 1 and sys.argv[1] == "--time"
    words = sys.stdin.read().split()
    n = int(words[0]) if words and words[0].lstrip("-").isdigit() else -1
    if not 0 <= n <= 90:
        print("please give n from 0 to 90")
        return 1
    seq = " ".join(str(fib_iter(i)[0]) for i in range(min(n, 15) + 1))
    print(f"sequence: {seq}" + (" ..." if n > 15 else ""))
```

Kode 8.6 (Lanjutan) *fibonacci.py: kedua cara yang sama dalam Python; cara iteratif mengembalikan tuple.*

```python
    f, s = fib_iter(n)
    print(f"iterative: F({n}) = {f}, sum = {s}")
    if n <= 40:
        f = fib_rec(n)
        print(f"recursive: F({n}) = {f}, sum = {fib_rec_sum(n)}")

    print("   n   loop steps   recursive calls")
    for size in [20, 25, 30, 32]:
        calls = 0
        t0 = time.process_time()
        fib_rec(size)
        secs = time.process_time() - t0
        line = f"{size:4d} {size:12d} {calls:17d}"
        if timing:
            line += f"   {secs:.3f} seconds"
        print(line)
    return 0


if __name__ == "__main__":
    sys.exit(main())
```

Keluarannya sama persis. Bilangan bulat Python tidak pernah meluap, sehingga program Python sebenarnya dapat melampaui $n = 90$; batas itu dipertahankan supaya kedua program menerima masukan yang sama.

8.4.6 Mengukur perbedaannya

Masalah ini meminta *membandingkan kecepatan*, dan menghitung pemanggilan baru separuhnya. Kedua program menerima `--time`, yang menambahkan kolom detik yang diukur dengan `clock()` di C dan `time.process_time()` di Python; detiknya berbeda dari satu komputer ke komputer lain dan dari satu kali jalan ke kali berikutnya, itulah sebabnya detik tidak menjadi bagian keluaran biasa. **Tabel 8.3** memperlihatkan pengukuran pada komputer yang dipakai untuk buku ini (prosesor Intel Xeon 2,1 GHz), untuk $F(35)$ dan $F(40)$ secara rekursif, serta untuk perulangan.

Tabel 8.3 Waktu jalan cara rekursif dan iteratif yang terukur; waktu rekursif tumbuh sekitar sebelas kali dari $n = 35$ ke $n = 40$, seperti diramalkan $1.618^5 \approx 11$.

Cara	$F(35)$	$F(40)$
rekursi C, `gcc -O0` (tanpa optimasi)	0,069 s	0,807 s
rekursi C, `gcc -O2`	0,014 s	0,162 s
rekursi Python	1,05 s	11,3 s
perulangan, C atau Python	di bawah 0,0001 s	di bawah 0,0001 s

Tabel ini memuat tiga pelajaran. Pertama, pertumbuhannya: dari 35 ke 40, setiap waktu rekursif tumbuh sekitar sebelas kali lipat, persis seperti diramalkan hitungan pemanggilan, apa pun bahasa atau kompilatornya. Kedua, pengoptimal: `gcc -O2` membuat rekursi C yang sama lima kali lebih cepat daripada `-O0`, dengan menata ulang pemanggilan secara cerdik, tetapi tidak dapat mengubah pertumbuhan eksponensialnya. Ketiga, bahasanya: Python menjalankan rekursi yang sama sekitar tujuh puluh kali lebih lambat daripada C yang dioptimalkan, karena setiap pemanggilan fungsi Python mengerjakan jauh lebih banyak hal. Dan perulangan mengalahkan semuanya dengan selisih ribuan kali lipat, dalam kedua bahasa. Algoritma yang lebih baik mengalahkan bahasa yang lebih cepat dan kompilator yang lebih baik, setiap kali masukannya cukup besar [4].

8.4.7 Solusi salah yang menggoda

Kesalahan yang paling sering adalah kesepakatannya. **Kode 8.7** memulai barisan dengan 1, 1, seperti yang dilakukan banyak buku dan situs web, dan **Kode 8.8** melakukan hal yang sama dalam Python.

Kode 8.7 `wrong_fibonacci.c`: solusi salah yang menggoda, yang memulai barisan dari 1, 1.

```c
/* wrong_fibonacci.c - solusi salah yang menggoda untuk Soal 8.2 (jangan ditiru):
   barisan dimulai dari 1, 1, bukan 0, 1 */
#include <stdio.h>

int main(void)
{
    int n;
    if (scanf("%d", &n) != 1)
        return 1;
    long long a = 1, b = 1, sum = 0, last = 0;   /* SALAH: F(0) adalah 0, bukan 1 */
    printf("sequence:");
    for (int i = 0; i <= n; i++) {
        printf(" %lld", a);
        sum += a;
        last = a;
        long long next = a + b;
        a = b;
        b = next;
    }
    printf("\nF(%d) = %lld, sum = %lld\n", n, last, sum);
    return 0;
}
```

Kode 8.8 `wrong_fibonacci.py`: kesalahan yang sama dalam Python.

```python
# wrong_fibonacci.py - solusi salah yang menggoda untuk Soal 8.2 (jangan ditiru):
# barisan dimulai dari 1, 1, bukan 0, 1
import sys


def main():
    n = int(sys.stdin.read().split()[0])
    a, b, total, last = 1, 1, 0, 0      # SALAH: F(0) adalah 0, bukan 1
    seq = []
    for _ in range(n + 1):
        seq.append(str(a))
        total += a
        last = a
        a, b = b, a + b
    print("sequence: " + " ".join(seq))
    print(f"F({n}) = {last}, sum = {total}")


if __name__ == "__main__":
    main()
```

```
sequence: 1 1 2 3 5 8 13 21 34 55
F(9) = 55, sum = 143
```

Program mencetak sepuluh bilangan, seperti diminta, dan semuanya bilangan Fibonacci, itulah sebabnya kesalahan ini lolos dari pandangan sekilas. Tetapi kesepuluhnya bergeser satu tempat: $F(9)$ keluar sebagai 55 alih-alih 34, dan jumlahnya 143 alih-alih 88. Contoh dalam pernyataan masalah justru ada untuk menangkap hal ini, dan pelajarannya berlaku umum: sebelum menulis sebaris kode, hitunglah contohnya dengan tangan dan tuliskan keluaran yang diharapkan di samping kode Anda.

Kesalahan klasik kedua adalah luapan. Dengan `int` alih-alih `long long`, jumlahnya meluap pada $n = 45$ ($F(47) - 1$ melampaui $2^{31} - 1$) dan program mencetak bilangan negatif; luapan bilangan bertanda adalah perilaku tak terdefinisi di C, dan sanitizer melaporkannya.

> **PERIKSA DIRI.** Berapa pemanggilan yang dilakukan `fib_rec_sum(9)`, dan mengapa lebih banyak daripada `fib_rec(9)`?

Jawaban. `fib_rec_sum(9)` memanggil `fib_rec(11)`, yang melakukan $C(11) = 2F(12) - 1 = 2 \cdot 144 - 1 = 287$ pemanggilan, sedangkan `fib_rec(9)` melakukan $2F(10) - 1 = 109$. Menghitung jumlah melalui Persamaan (8.3) berarti satu rekursi dua tempat lebih jauh, sekitar $1.618^2 \approx 2.6$ kali kerja $F(9)$ saja. Menjumlahkan `fib_rec(i)` satu per satu untuk setiap i dari 0 sampai 9 tidak lebih baik: $\sum_{i=0}^{9}(2F(i+1) - 1) = 2 \cdot 143 - 10 = 276$ pemanggilan, hampir sama.

8.5 Soal 8.3: Palindrom

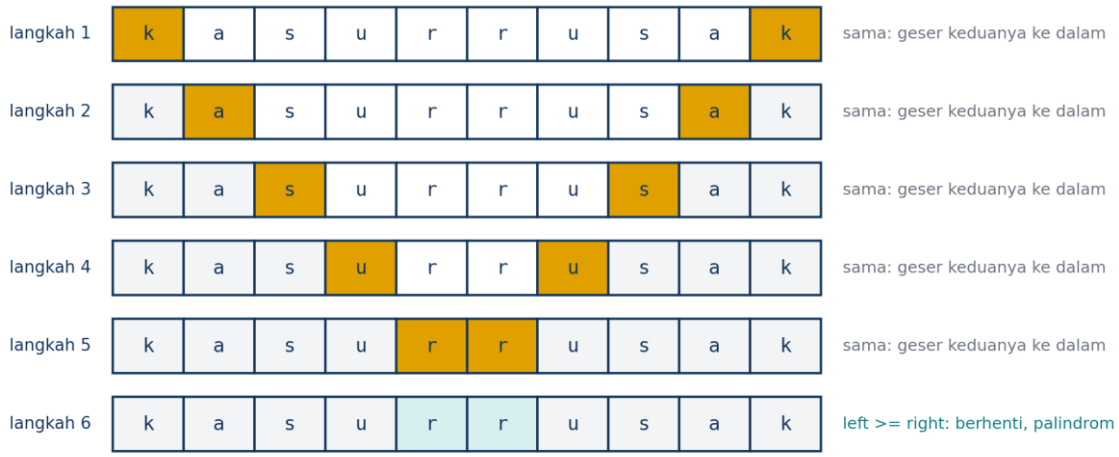
8.5.1 Masalahnya

SOAL 8.3. Bacalah kata-kata dan laporkan, untuk masing-masing, apakah kata itu *palindrom*, yaitu kata yang terbaca sama dari kiri ke kanan maupun dari kanan ke kiri, seperti `racecar`, `level`, atau `amanaplanacanalpanama` ("a man, a plan, a canal: Panama" tanpa spasi). Kata-katanya ditulis dalam huruf kecil tanpa spasi. Sebagai tantangan tambahan yang boleh dikerjakan, putuskan juga dengan cara yang memakai lebih dari sekadar perulangan dan perbandingan biasa, misalnya rekursi atau operasi bit.

Rinciannya kita tentukan sendiri. Program membaca kata sampai masukan habis, masing-masing sampai 99 huruf, lalu mencetak setiap kata beserta putusannya dan, di akhir, berapa yang palindrom. Program memutuskan setiap kata dengan tiga cara, yaitu pemindaian dua pointer yang baku, rekursi, dan pemindaian dengan operator XOR, serta memperingatkan jika ketiganya pernah tidak sepakat. Masukan uji memuat enam belas palindrom, di antaranya kata-kata Indonesia `katak`, `malam`, `kodok`, dan frasa `kasurrusak` ("kasur rusak" tanpa spasi), serta empat kata yang bukan palindrom.

8.5.2 Memahaminya dengan tangan

Gambar 8.2 memeriksa `kasurrusak` dengan dua jari, satu di huruf pertama dan satu di huruf terakhir. Keduanya menunjuk `k` dan `k`: sama, jadi kedua jari bergeser satu huruf ke dalam. Lalu `a` dan `a`, `s` dan `s`, `u` dan `u`, `r` dan `r`: semuanya sama. Sekarang jari kiri sudah melewati jari kanan, tidak ada lagi yang dibandingkan, dan kata itu palindrom. Lima perbandingan untuk sepuluh huruf: pemindaian membandingkan setiap pasangan sekali dan berhenti di tengah.



Gambar 8.2 Pemindaian dua pointer pada `kasurrusak`: huruf berwarna kuning sedang dibandingkan, yang abu-abu sudah selesai; setelah lima pasangan yang sama, kedua pointer bersilang dan kata itu palindrom.

Untuk `laundry`, perbandingan pertama, `l` melawan `y`, langsung gagal, dan jawabannya seketika "bukan palindrom". Ketidakcocokan pertama menentukan; tidak perlu melihat lebih jauh. Kata satu huruf, `x`, tidak punya pasangan sama sekali, dan merupakan palindrom. Untuk kata n huruf, pemindaian ini melakukan paling banyak $\lfloor n/2 \rfloor$ perbandingan.

8.5.3 Merancangnya di atas kertas

Dua pointer. `left` mulai di karakter pertama, `right` di karakter terakhir, yaitu `s + strlen(s) - 1`. Selama `left < right`: jika `*left != *right`, jawab 0; jika tidak, majukan `left` dan mundurkan `right`. Ketika perulangan berakhir, jawab 1. Ini persis penelusuran pembalikan dari Bab 7, dengan perbandingan sebagai ganti tukaran.

Rekursi. Sebuah kata adalah palindrom ketika huruf pertama dan terakhirnya sama *dan* bagian di antara keduanya palindrom. Kasus dasarnya adalah bagian berisi nol atau satu huruf, yang selalu palindrom. Fungsinya menerima kedua pointer: `is_pal_rec(left, right)`.

XOR. Operator bit `^` (*exclusive or*) membandingkan dua bilangan bit demi bit dan menghasilkan 1 di setiap posisi tempat keduanya berbeda, sehingga `x ^ y` bernilai 0 tepat ketika `x == y`. Menggabungkan XOR setiap pasangan dengan OR mengumpulkan bit 1 di mana pun ada pasangan yang berbeda; kata itu palindrom tepat ketika kumpulannya bernilai 0 di akhir. Pemindaian ini sama sekali tidak memiliki `if` di dalam perulangannya, sesuai semangat tantangan itu; ia selalu melihat setiap pasangan alih-alih berhenti pada ketidakcocokan pertama.

Membaca kata perlu kehati-hatian: `scanf("%99s", word)` ke dalam `char word[100]`, *tanpa* `&`, karena nama array sudah merupakan alamat elemen pertamanya (Bab 7), dan dengan lebar 99, yang mencegah `scanf` menulis melewati ujung array ketika seseorang mengetik kata yang lebih panjang, dan menyisakan tempat untuk `'\0'` penutup.

8.5.4 Penyelesaian dalam C

Kode 8.9 memuat ketiga fungsi pemutus dan `main` yang membaca kata sampai masukan habis.

Kode 8.9 `palindrome.c`: tiga cara memutuskan palindrom yang harus sepakat: dua pointer, rekursi pada dua pointer, dan XOR atas pasangan-pasangan.

```c
/* palindrome.c - Soal 8.3: apakah sebuah kata terbaca sama dari dua arah?
   Tiga cara yang harus sepakat: dua pointer, rekursi, dan XOR.
   Masukan: kata dipisah spasi atau baris baru, masing-masing maks. 99 huruf. */
#include <stdio.h>
#include <string.h>

int is_palindrome(const char *s)         /* pemindaian dua pointer yang baku */
{
    const char *left = s;
    const char *right = s + strlen(s) - 1;
    while (left < right) {               /* berhenti di tengah */
        if (*left != *right)
            return 0;                    /* ketidakcocokan pertama menentukan */
        left++;
        right--;
    }
    return 1;
}

int is_pal_rec(const char *left, const char *right)
{
    if (left >= right)                   /* tersisa nol atau satu huruf */
        return 1;
    return *left == *right && is_pal_rec(left + 1, right - 1);
}

int is_pal_xor(const char *s)            /* huruf yang sama di-XOR menjadi nol */
{
    size_t n = strlen(s);
    unsigned diff = 0;
    for (size_t i = 0; i < n / 2; i++)
        diff |= (unsigned char) s[i] ^ (unsigned char) s[n - 1 - i];
    return diff == 0;
}

int main(void)
{
    char word[100];                      /* 99 huruf dan '\0' */
    int yes = 0, total = 0;
    while (scanf("%99s", word) == 1) {   /* tanpa & : word sudah berupa alamat */
        int a = is_palindrome(word);
        int b = is_pal_rec(word, word + strlen(word) - 1);
        int c = is_pal_xor(word);
        printf("%-22s %s%s\n", word, a ? "palindrome" : "not a palindrome",
               (a == b && b == c) ? "" : "  (METHODS DISAGREE)");
        yes += a;
        total++;
    }
    printf("%d of %d words are palindromes\n", yes, total);
    return 0;
}
```

Kode 8.9 (Lanjutan) *palindrome.c: tiga cara memutuskan palindrom yang harus sepakat: dua pointer, rekursi pada dua pointer, dan XOR atas pasangan-pasangan.*

Masukannya dua puluh kata:

```
amanaplanacanalpanama neveroddoreven racecar level
kayak rotator civic madam
refer stats noon malam
katak kasurrusak kodok
x ab laundry profitina
abcdefghijklmnlkjihgfedcba
```

```
amanaplanacanalpanama  palindrome
neveroddoreven         palindrome
racecar                palindrome
level                  palindrome
kayak                  palindrome
rotator                palindrome
civic                  palindrome
madam                  palindrome
refer                  palindrome
stats                  palindrome
noon                   palindrome
malam                  palindrome
katak                  palindrome
kasurrusak             palindrome
kodok                  palindrome
x                      palindrome
ab                     not a palindrome
laundry                not a palindrome
profitina              not a palindrome
abcdefghijklmnlkjihgfedcba not a palindrome
16 of 20 words are palindromes
```

Keenam belas palindrom dikenali, dan ketiga cara tidak pernah berselisih (jika berselisih, barisnya akan diakhiri peringatan). Perhatikan kata terakhir, `abcdefghijklmnlkjihgfedcba`: kata itu palindrom kecuali dua huruf di tengahnya, `m` dan `n`. Pemindaian dua pointer membandingkan dua belas pasangan yang sama sebelum menemukan perbedaannya; uji seperti ini, yang baru gagal pada perbandingan terakhir yang mungkin, adalah uji yang menyingkap pemindaian yang berhenti satu langkah terlalu cepat.

8.5.5 Penyelesaian dalam Python

Kode 8.10 mengikuti ketiga cara yang sama dengan indeks sebagai ganti pointer. XOR dikenakan pada `ord(c)`, kode karakternya, karena `^` di Python bekerja pada bilangan bulat, bukan pada karakter.

Kode 8.10 `palindrome.py`: ketiga cara yang sama dalam Python, dengan indeks.

```
# palindrome.py - Soal 8.3: apakah sebuah kata terbaca sama dari dua arah?
# Tiga cara yang harus sepakat: dua indeks, rekursi, dan XOR.
# Masukan: kata-kata dipisah spasi atau baris baru.
import sys


def is_palindrome(s):                       # pemindaian dua indeks yang baku
    left, right = 0, len(s) - 1
    while left < right:                     # berhenti di tengah
        if s[left] != s[right]:
            return False                    # ketidakcocokan pertama menentukan
        left += 1
        right -= 1
    return True


def is_pal_rec(s, left, right):
    if left >= right:                       # tersisa nol atau satu huruf
        return True
    return s[left] == s[right] and is_pal_rec(s, left + 1, right - 1)


def is_pal_xor(s):                          # huruf yang sama di-XOR menjadi nol
    diff = 0
    n = len(s)
    for i in range(n // 2):
        diff |= ord(s[i]) ^ ord(s[n - 1 - i])
    return diff == 0


def main():
    yes = total = 0
    for word in sys.stdin.read().split():
        word = word[:99]                    # program C membaca paling banyak 99
        a = is_palindrome(word)
        b = is_pal_rec(word, 0, len(word) - 1)
        c = is_pal_xor(word)
        verdict = "palindrome" if a else "not a palindrome"
        warn = "" if a == b == c else "  (METHODS DISAGREE)"
        print(f"{word:<22} {verdict}{warn}")
        yes += a
        total += 1
    print(f"{yes} of {total} words are palindromes")
    # jalan pintas Python, sebagai pembanding: word == word[::-1]


if __name__ == "__main__":
    main()
```

Kode 8.10 (Lanjutan) *palindrome.py: ketiga cara yang sama dalam Python, dengan indeks.*

Komentar terakhir dalam program itu memperlihatkan apa yang sebenarnya akan ditulis pemrogram Python yang berpengalaman: `word == word[::-1]`, yang membandingkan kata dengan kebalikannya, dibuat dengan *slice* berlangkah -1. Cara itu benar dan singkat, tetapi membangun salinan terbalik kata itu dan membandingkan seluruh n huruf, sedangkan pemindaian dua indeks membandingkan paling banyak $n/2$ dan tidak membangun apa pun; untuk satu kata perbedaan itu tidak penting, dan mengetahui harga bentuk singkat itu adalah bagian dari memahaminya.

8.5.6 Solusi salah yang menggoda

Pemindaian dua pointer sering ditulis dengan satu indeks `i` dan pasangannya dihitung dari indeks itu. Pasangan posisi `i` dalam kata sepanjang `n` adalah posisi `n - 1 - i`: pasangan huruf pertama, 0, adalah huruf terakhir, `n - 1`. Kesalahan yang menggoda melupakan `- 1`, seperti pada **Kode 8.11** dan kembaran Python-nya **Kode 8.12**.

Kode 8.11 `wrong_palindrome.c`: solusi salah yang menggoda, yang indeks pasangannya `n - i` alih-alih `n - 1 - i`.

```c
/* wrong_palindrome.c - solusi salah yang menggoda untuk Soal 8.3 (jangan ditiru):
   pasangan s[i] ditulis sebagai s[n - i] */
#include <stdio.h>
#include <string.h>

int is_palindrome(const char *s)
{
    size_t n = strlen(s);
    for (size_t i = 0; i < n / 2; i++)
        if (s[i] != s[n - i])            /* SALAH: s[n] adalah '\0' */
            return 0;
    return 1;
}

int main(void)
{
    char word[100];
    while (scanf("%99s", word) == 1)
        printf("%-22s %s\n", word, is_palindrome(word) ? "palindrome"
                                                       : "not a palindrome");
    return 0;
}
```

Kode 8.12 `wrong_palindrome.py`: kesalahan yang sama dalam Python, dengan karakter nol eksplisit untuk meniru C.

```python
# wrong_palindrome.py - solusi salah yang menggoda untuk Soal 8.3 (jangan ditiru):
# pasangan s[i] ditulis sebagai s[n - i]
import sys


def is_palindrome(s):
    n = len(s)
    padded = s + "\0"                    # tiru nol penutup milik C
    for i in range(n // 2):
        if padded[i] != padded[n - i]:   # SALAH: s[n] adalah '\0'
            return False
    return True


def main():
    for word in sys.stdin.read().split():
        verdict = "palindrome" if is_palindrome(word) else "not a palindrome"
        print(f"{word:<22} {verdict}")


if __name__ == "__main__":
    main()
```

```
amanaplanacanalpanama  not a palindrome
neveroddoreven         not a palindrome
racecar                not a palindrome
level                  not a palindrome
kayak                  not a palindrome
rotator                not a palindrome
civic                  not a palindrome
madam                  not a palindrome
refer                  not a palindrome
stats                  not a palindrome
noon                   not a palindrome
malam                  not a palindrome
katak                  not a palindrome
kasurrusak             not a palindrome
kodok                  not a palindrome
x                      palindrome
ab                     not a palindrome
laundry                not a palindrome
profitina              not a palindrome
abcdefghijklmnlkjihgfedcba not a palindrome
```

Setiap palindrom yang berhuruf dua atau lebih ditolak. Perbandingan pertama, dengan `i` = 0, membandingkan `s[0]` dengan `s[n]`, dan `s[n]` bukan huruf terakhir melainkan `'\0'` penutup string, yang tidak sama dengan huruf mana pun. Hanya `x`, dengan $n/2 = 0$ perbandingan, yang lolos sebagai palindrom. Kesalahan ini mendidik karena di C ia tidak membuat program berhenti mendadak: `s[n]` masih berada di dalam array (yaitu nolnya), sehingga sanitizer pun diam. Di Python, tanpa nol eksplisit, `s[n]` akan melempar `IndexError`, itulah sebabnya versi Python menambahkan `"\0"` untuk meniru C dengan setia.

> **PERIKSA DIRI.** Ubahlah program itu agar juga menerima frasa dengan huruf kapital dan spasi, seperti “Kasur rusak” atau “Never odd or even”. Dua hal apa yang harus berubah, dan mengapa pembacaan dengan `%99s` menjadi penghalang?
>
> *Jawaban.* Perbandingan harus melewati setiap karakter yang bukan huruf dan membandingkan huruf tanpa memedulikan besar-kecilnya: majukan `left` selama `*left` bukan huruf (`isalpha` dari `<ctype.h>`), mundurkan `right` dengan cara yang sama, lalu bandingkan `tolower(*left)` dengan `tolower(*right)`. Dan masukannya harus dibaca per baris, bukan per kata, karena `%99s` berhenti pada spasi pertama dan akan memecah “Kasur rusak” menjadi dua kata; Bab 9 memperkenalkan `fgets` untuk keperluan ini.

8.6 Soal 8.4: Determinan Matriks 3 × 3

8.6.1 Masalahnya

> **SOAL 8.4.** Bacalah matriks 3 × 3 berisi bilangan bulat, hitung determinannya, dan laporkan apakah matriks itu punya invers (*invertible*): matriks itu punya invers tepat ketika determinannya tidak nol.

Matriks adalah tabel bilangan, dan matriks 3 × 3 adalah array dua dimensi dengan tiga baris dan tiga kolom. Mengapa laundry perlu peduli? Inilah pertanyaan sungguhan. Mbak Rina membeli deterjen, pelembut, dan pelicin setrika dari satu toko, dan harga-harganya berubah tanpa daftar harga. Ia memegang tiga nota: dua bungkus deterjen, satu pelembut, dan satu pelicin berharga Rp60.000; satu deterjen, tiga pelembut, dan dua pelicin berharga Rp72.000; dan satu deterjen saja berharga Rp20.000. Dapatkah ia menghitung harga setiap barang? Jumlah-jumlah barang itu membentuk matriks

$$A = \begin{pmatrix} 2 & 1 & 1 \\ 1 & 3 & 2 \\ 1 & 0 & 0 \end{pmatrix} \tag{8.4}$$

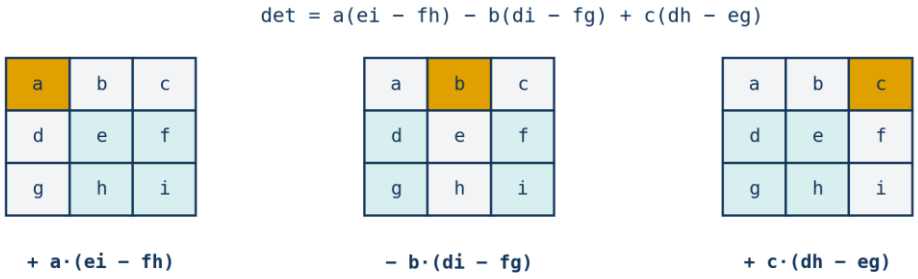
dan harga-harganya dapat ditemukan kembali tepat ketika A punya invers, yaitu ketika determinannya tidak nol [5]. Seandainya dua nota memuat jumlah barang yang sama, keduanya mengatakan hal yang sama dua kali, matriksnya memiliki dua baris yang sama, determinannya 0, dan hitungan sebanyak apa pun tidak dapat memisahkan ketiga harga itu. Program kita membaca beberapa matriks, kelompok sembilan bilangan satu demi satu, baris demi baris, lalu mencetak masing-masing beserta determinan dan putusannya.

8.6.2 Memahaminya dengan tangan

Namai kesembilan entri dengan huruf, baris demi baris: a, b, c di baris pertama, d, e, f di baris kedua, g, h, i di baris ketiga. Dengan ekspansi sepanjang baris pertama, determinannya adalah

$$\det A = a(ei - fh) - b(di - fg) + c(dh - eg) \tag{8.5}$$

Gambar 8.3 memperlihatkan asal setiap suku. Setiap entri baris pertama dikalikan dengan determinan matriks 2×2 yang tersisa ketika baris dan kolomnya dicoret (sel berwarna hijau kebiruan), dan ketiga hasil kali itu dijumlahkan dengan tanda $+, -, +$. Tanda minus pada suku tengah itulah bagian yang mudah terlupa.



Gambar 8.3 Ekspansi sepanjang baris pertama: setiap entri baris pertama (kuning) mengalikan determinan 2×2 dari sel-sel yang tersisa ketika baris dan kolomnya dihapus (hijau kebiruan), dengan tanda $+, -, +$.

Kerjakan satu contoh sampai tuntas. Untuk matriks dengan baris $(2, -3, 1)$, $(2, 0, -1)$, dan $(1, 4, 5)$:

- $a(ei - fh) = 2 \cdot (0 \cdot 5 - (-1) \cdot 4) = 2 \cdot 4 = 8$;
- $-b(di - fg) = -(-3) \cdot (2 \cdot 5 - (-1) \cdot 1) = 3 \cdot 11 = 33$;
- $c(dh - eg) = 1 \cdot (2 \cdot 4 - 0 \cdot 1) = 8$;

sehingga $\det = 8 + 33 + 8 = 49$, tidak nol: punya invers. Untuk nota-nota Mbak Rina, Persamaan (8.4): $2(3 \cdot 0 - 2 \cdot 0) - 1(1 \cdot 0 - 2 \cdot 1) + 1(1 \cdot 0 - 3 \cdot 1) = 0 + 2 - 3 = -1$. Tidak nol, sehingga harga-harganya dapat ditemukan; Soal Latihan 8.5 mengerjakannya. Dua uji cepat akan berguna: *matriks identitas*, dengan 1 di diagonal dan 0 di tempat lain, berdeterminan 1, dan setiap matriks dengan dua baris yang sama berdeterminan 0.

8.6.3 Merancangnya di atas kertas

Matriksnya tinggal di `long long m[3][3]`; `long long` karena hasil kali tiga entri dapat membesar dengan cepat. Pembacaannya memakai dua perulangan bersarang, baris dan kolom, bukan

sembilan pemanggilan `scanf` terpisah, dalam fungsi `read_matrix` yang mengembalikan 0 ketika masukan habis, sehingga `main` dapat berulang untuk sebanyak apa pun matriks yang ada. Determinannya satu fungsi, `det3`, yang menyalin Persamaan (8.5) dengan `m[0][0]` untuk a, `m[0][1]` untuk b, dan seterusnya; huruf-hurufnya ditulis dalam komentar di atasnya, sehingga setiap suku kode dapat dicocokkan dengan hurufnya. Parameternya `const long long m[3][3]`: fungsi itu hanya membaca, dan banyaknya kolom, 3, harus diketahui, seperti dijelaskan Bab 6.

8.6.4 Penyelesaian dalam C

Kode 8.13 menyimpan susunan huruf sebagai komentar di atas `det3`, sehingga setiap dari ketiga suku dapat dicocokkan dengan Persamaan (8.5).

Kode 8.13 `determinant.c`: membaca matriks 3×3 dengan perulangan bersarang dan menghitung setiap determinan dengan ekspansi sepanjang baris pertama.

```c
/* determinant.c - Soal 8.4: determinan matriks 3x3, dan apakah
   matriks itu punya invers. Masukan: sembilan bilangan bulat, per baris. */
#include <stdio.h>

/*  | a b c |
    | d e f |   det = a(ei - fh) - b(di - fg) + c(dh - eg)
    | g h i |                                              */
long long det3(const long long m[3][3])
{
    return m[0][0] * (m[1][1] * m[2][2] - m[1][2] * m[2][1])
         - m[0][1] * (m[1][0] * m[2][2] - m[1][2] * m[2][0])
         + m[0][2] * (m[1][0] * m[2][1] - m[1][1] * m[2][0]);
}

int read_matrix(long long m[3][3])
{
    for (int r = 0; r < 3; r++)
        for (int c = 0; c < 3; c++)
            if (scanf("%lld", &m[r][c]) != 1)
                return 0;
    return 1;
}

int main(void)
{
    long long m[3][3];
    int number = 0;
    while (read_matrix(m)) {
        long long d = det3(m);
        printf("matrix %d:\n", ++number);
        for (int r = 0; r < 3; r++)
            printf("  %5lld %5lld %5lld\n", m[r][0], m[r][1], m[r][2]);
        printf("  determinant = %lld, %s\n", d,
               d != 0 ? "invertible" : "not invertible");
    }
    return 0;
}
```

Masukan uji memuat empat matriks: identitas, contoh yang dikerjakan di atas, matriks yang baris ketiganya mengulang baris pertama, dan nota-nota Mbak Rina.

```
1 0 0
0 1 0
0 0 1

2 -3 1
2 0 -1
1 4 5

1 2 3
4 5 6
1 2 3

2 1 1
1 3 2
1 0 0
```

```
matrix 1:
      1     0     0
      0     1     0
      0     0     1
  determinant = 1, invertible
matrix 2:
      2    -3     1
      2     0    -1
      1     4     5
  determinant = 49, invertible
matrix 3:
      1     2     3
      4     5     6
      1     2     3
  determinant = 0, not invertible
matrix 4:
      2     1     1
      1     3     2
      1     0     0
  determinant = -1, invertible
```

Keempatnya sesuai dengan hitungan tangan: 1, 49, 0, dan -1.

8.6.5 Penyelesaian dalam Python

Pada **Kode 8.14**, sembilan bilangan setiap kelompok dipotong menjadi tiga baris dengan slice, `flat[0:3]`, `flat[3:6]`, dan `flat[6:9]`, yang membangun list berisi list dari Bab 6.

Kode 8.14 `determinant.py`: determinan yang sama dalam Python, dengan matriks yang dibangun dari slice.

```python
# determinant.py - Soal 8.4: determinan matriks 3x3, dan apakah
# matriks itu punya invers. Masukan: sembilan bilangan bulat, per baris.
import sys

#  | a b c |
#  | d e f |   det = a(ei - fh) - b(di - fg) + c(dh - eg)
#  | g h i |


def det3(m):
    return (m[0][0] * (m[1][1] * m[2][2] - m[1][2] * m[2][1])
            - m[0][1] * (m[1][0] * m[2][2] - m[1][2] * m[2][0])
            + m[0][2] * (m[1][0] * m[2][1] - m[1][1] * m[2][0]))


def main():
    numbers = [int(w) for w in sys.stdin.read().split()]
    number = 0
    for start in range(0, len(numbers) - 8, 9):
        flat = numbers[start:start + 9]
        m = [flat[0:3], flat[3:6], flat[6:9]]
        d = det3(m)
        number += 1
        print(f"matrix {number}:")
        for row in m:
            print(f"  {row[0]:5d} {row[1]:5d} {row[2]:5d}")
        print(f"  determinant = {d}, "
              + ("invertible" if d != 0 else "not invertible"))


if __name__ == "__main__":
    main()
```

8.6.6 Solusi salah yang menggoda

Kesalahan penyalinan yang klasik membalik tanda suku tengah, seperti pada **Kode 8.15** dan **Kode 8.16**: ketiga suku dijumlahkan.

Kode 8.15 `wrong_determinant.c`: solusi salah yang menggoda, yang menjumlahkan suku tengah alih-alih mengurangkannya.

```c
/* wrong_determinant.c - solusi salah yang menggoda untuk Soal 8.4 (jangan ditiru):
   suku tengah ditambahkan, bukan dikurangkan */
#include <stdio.h>

long long det3(const long long m[3][3])
{
    return m[0][0] * (m[1][1] * m[2][2] - m[1][2] * m[2][1])
         + m[0][1] * (m[1][0] * m[2][2] - m[1][2] * m[2][0])   /* tanda SALAH */
         + m[0][2] * (m[1][0] * m[2][1] - m[1][1] * m[2][0]);
}

int main(void)
{
    long long m[3][3];
    int number = 0;
    for (;;) {
        for (int k = 0; k < 9; k++)
            if (scanf("%lld", &m[k / 3][k % 3]) != 1)
                return 0;
        printf("matrix %d: determinant = %lld\n", ++number, det3(m));
    }
}
```

Kode 8.16 `wrong_determinant.py`: kesalahan yang sama dalam Python.

```
# wrong_determinant.py - solusi salah yang menggoda untuk Soal 8.4 (jangan ditiru):
# suku tengah ditambahkan, bukan dikurangkan
import sys


def det3(m):
    return (m[0][0] * (m[1][1] * m[2][2] - m[1][2] * m[2][1])
            + m[0][1] * (m[1][0] * m[2][2] - m[1][2] * m[2][0])  # tanda SALAH
            + m[0][2] * (m[1][0] * m[2][1] - m[1][1] * m[2][0]))


def main():
    numbers = [int(w) for w in sys.stdin.read().split()]
    for k, start in enumerate(range(0, len(numbers) - 8, 9), 1):
        f = numbers[start:start + 9]
        print(f"matrix {k}: determinant = {det3([f[0:3], f[3:6], f[6:9]])}")


if __name__ == "__main__":
    main()
```

```
matrix 1: determinant = 1
matrix 2: determinant = -17
matrix 3: determinant = 24
matrix 4: determinant = -5
```

Perhatikan baris pertama. Program yang salah memberi jawaban yang benar, 1, untuk matriks identitas, karena suku tengahnya dikalikan dengan $b = 0$ sehingga tandanya tidak berpengaruh. Matriks identitas, uji yang dicoba semua orang pertama kali, tidak dapat menangkap kesalahan ini. Matriks dengan dua baris yang sama dapat: determinannya harus 0, dan program yang salah menjawab 24. Contoh yang dikerjakan dengan tangan juga menangkapnya (-17 alih-alih 49), begitu pula nota-nota Mbak Rina (-5 alih-alih -1). Pelajaran umum pengujian terlihat di sini: sebuah uji hanya berharga jika kesalahan yang Anda khawatirkan akan mengubah hasilnya. Pilihlah masukan uji yang membuat setiap bagian rumus berperan; masukan yang penuh nol hanya menguji sedikit [6].

> **PERIKSA DIRI.** Tanpa menghitung, berapakah determinan matriks yang baris keduanya dua kali baris pertamanya, misalnya baris (1,2,3), (2,4,6), dan (0,1,5)? Periksalah jawaban Anda dengan Persamaan (8.5).
>
> *Jawaban.* Nol: baris yang merupakan kelipatan baris lain tidak menambah informasi baru, seperti nota yang berulang, sehingga matriksnya tidak punya invers. Menurut Persamaan (8.5): $1(4 \cdot 5 - 6 \cdot 1) - 2(2 \cdot 5 - 6 \cdot 0) + 3(2 \cdot 1 - 4 \cdot 0) = 14 - 20 + 6 = 0$.

8.7 Soal 8.5: Mencari Sebuah Nota

8.7.1 Masalahnya

> **SOAL 8.5.** Nomor-nomor nota pesanan laundry disimpan dalam array yang terurut. Tulislah fungsi rekursif yang mencari sebuah nomor nota dan mengembalikan indeksnya, atau -1 jika nomor itu tidak ada di dalam array, serta cetak setiap langkah pencariannya.

Pelanggan kembali untuk mengambil cuciannya dengan membawa nota, dan kadang hanya dengan nomornya, yang dibacakan lewat telepon. Array itu memuat dua belas nomor nota dari 1003 sampai 1066, terurut, dengan celah di tempat pesanan yang dibatalkan. Program mencari tiga nomor: satu di bagian tengah, yang pertama, dan satu yang tidak ada.

8.7.2 Memahaminya dengan tangan, dan rancangannya

Inilah pencarian biner dari Bab 6, ditulis sebagai rekursi. Bagian yang masih dipertanyakan membentang dari indeks `low` sampai indeks `high`. Jika bagian itu kosong (`low > high`), nomornya tidak ada: kasus dasar, jawab -1. Jika tidak, lihat elemen tengahnya, `mid = low + (high - low) / 2`. Jika itulah nomornya, jawab `mid`: kasus dasar kedua. Jika nomornya lebih kecil, jawabannya adalah jawaban untuk bagian kiri, `low` sampai `mid - 1`; jika tidak, untuk bagian kanan, `mid + 1` sampai `high`. Setiap pemanggilan bekerja pada bagian yang paling besar separuhnya, sehingga rekursinya paling dalam $\lfloor \log_2 n \rfloor + 1$ pemanggilan: empat untuk dua belas nota.

Dengan tangan, untuk 1047 dalam array 1003, 1008, 1015, 1021, 1022, 1030, 1034, 1041, 1047, 1052, 1060, 1066: pemanggilan pertama mencakup 0 sampai 11, tengahnya indeks 5, 1030, lebih kecil daripada 1047, jadi lanjutkan dengan 6 sampai 11; tengahnya indeks 8, 1047: ketemu. Dua pemanggilan. Fungsinya menerima pointer ke array, kedua batas, dan nomornya, ditambah `depth` yang hanya dipakai untuk mengindentasi langkah yang dicetak, supaya kedalaman rekursi terlihat di layar.

8.7.3 Penyelesaiannya

Kode 8.17 dan **Kode 8.18** menerapkan rancangan itu; `main` mencari ketiga nomor secara bergiliran.

Kode 8.17 `find_receipt.c`: pencarian biner rekursif atas nomor nota yang terurut, mencetak setiap pemanggilan dengan indentasi sesuai kedalamannya.

```c
/* find_receipt.c - Soal 8.5: cari nomor nota dalam array terurut dengan
   pencarian biner rekursif, dan laporkan setiap langkah */
#include <stdio.h>

int find(const int *a, int low, int high, int key, int depth)
{
    if (low > high)
        return -1;                          /* bagian kosong: tidak ada */
    int mid = low + (high - low) / 2;
    printf("%*scall %d: [%d..%d], middle a[%d] = %d\n", 2 * depth, "",
           depth + 1, low, high, mid, a[mid]);
    if (a[mid] == key)
        return mid;
    if (key < a[mid])
        return find(a, low, mid - 1, key, depth + 1);
    return find(a, mid + 1, high, key, depth + 1);
}

int main(void)
{
    int receipt[12] = {1003, 1008, 1015, 1021, 1022, 1030,
                       1034, 1041, 1047, 1052, 1060, 1066};
    int keys[3] = {1047, 1003, 1050};
    for (int k = 0; k < 3; k++) {
        printf("looking for %d\n", keys[k]);
        int at = find(receipt, 0, 11, keys[k], 0);
        if (at >= 0)
            printf("found at index %d\n", at);
        else
            printf("not in the book\n");
    }
    return 0;
}
```

Kode 8.18 `find_receipt.py`: rekursi yang sama dalam Python; `f"{'':{2 * depth}}"` mencetak indentasinya.

```
# find_receipt.py - Soal 8.5: cari nomor nota dalam list terurut dengan
# pencarian biner rekursif, dan laporkan setiap langkah


def find(a, low, high, key, depth):
    if low > high:
        return -1                       # bagian kosong: tidak ada
    mid = low + (high - low) // 2
    print(f"{'':{2 * depth}}call {depth + 1}: [{low}..{high}], "
          f"middle a[{mid}] = {a[mid]}")
    if a[mid] == key:
        return mid
    if key < a[mid]:
        return find(a, low, mid - 1, key, depth + 1)
    return find(a, mid + 1, high, key, depth + 1)


def main():
    receipt = [1003, 1008, 1015, 1021, 1022, 1030,
               1034, 1041, 1047, 1052, 1060, 1066]
    for key in [1047, 1003, 1050]:
        print(f"looking for {key}")
        at = find(receipt, 0, len(receipt) - 1, key, 0)
        print(f"found at index {at}" if at >= 0 else "not in the book")


if __name__ == "__main__":
    main()
```

```
looking for 1047
call 1: [0..11], middle a[5] = 1030
  call 2: [6..11], middle a[8] = 1047
found at index 8
looking for 1003
call 1: [0..11], middle a[5] = 1030
  call 2: [0..4], middle a[2] = 1015
    call 3: [0..1], middle a[0] = 1003
found at index 0
looking for 1050
call 1: [0..11], middle a[5] = 1030
  call 2: [6..11], middle a[8] = 1047
    call 3: [9..11], middle a[10] = 1060
      call 4: [9..9], middle a[9] = 1052
not in the book
```

Nomor 1050 tidak ada di dalam array; pemanggilan keempat mempersempit bagiannya menjadi indeks 9 saja, melihat 1052, dan pemanggilan berikutnya akan mendapat bagian yang kosong. Empat pemanggilan, seperti diramalkan.

8.7.4 Solusi salah yang menggoda

Jalan pintas yang menggoda menulis pemanggilan rekursifnya dengan `mid` alih-alih `mid - 1` atau `mid + 1`: `find(a, low, mid, ...)` dan `find(a, mid, high, ...)`. Tampaknya tidak berbahaya, karena `mid` sudah diperiksa. Tetapi ketika bagiannya berisi dua elemen, misalnya `low = 8` dan `high = 9`, `mid` bernilai 8, dan jika nomornya lebih besar, pemanggilan berikutnya mendapat `low = 8, high = 9` lagi: bagiannya tidak pernah menyusut, rekursinya tidak pernah mencapai kasus dasar, dan program berakhir dengan stack overflow. Aturan yang mencegahnya adalah aturan setiap rekursi: *setiap pemanggilan harus bekerja pada masalah yang benar-benar lebih kecil*. Mengecualikan `mid`, yang sudah diperiksa, menjaminnya.

PERIKSA DIRI. Mengapa versi rekursif memakai memori tumpukan panggilan sebanyak pemanggilan yang dilakukannya, dan mengapa hal itu tidak berbahaya di sini, padahal berbahaya untuk `fib`?

Jawaban. Setiap pemanggilan yang aktif menyimpan bingkainya di tumpukan sampai kembali, dan di sini setiap pemanggilan menunggu pemanggilan yang dilakukannya, sehingga tumpukannya sedalam banyaknya pemanggilan, paling banyak $\lfloor \log_2 n \rfloor + 1$: 20 bingkai untuk sejuta nota, 31 untuk dua miliar. `fib` berbahaya bukan karena kedalamannya, yang hanya n, melainkan karena jumlah seluruh pemanggilannya; pencarian biner hanya melakukan satu pemanggilan rekursif per tingkat, sehingga jumlah dan kedalamannya sama-sama bilangan kecil yang sama.

8.8 Soal 8.6: Menggabungkan Dua Sif

8.8.1 Masalahnya

SOAL 8.6. Sif pagi dan sif sore masing-masing menyimpan tagihannya terurut dari yang terkecil sampai yang terbesar. Tulislah fungsi yang menggabungkan kedua array terurut itu menjadi satu array terurut, dengan menelusuri keduanya memakai pointer, dan mengembalikan banyaknya tagihan dalam hasilnya. Fungsi itu juga harus bekerja ketika salah satu sif tidak mendapat pesanan.

8.8.2 Memahaminya dengan tangan, dan rancangannya

Bayangkan dua tumpukan nota yang menghadap ke atas, masing-masing terurut dengan yang terkecil di atas. Untuk membuat satu tumpukan terurut, bandingkan kedua nota teratas, ambil yang lebih kecil, lalu ulangi. Ketika satu tumpukan habis, taruh sisa tumpukan yang lain ke hasil apa adanya: sisa itu sudah terurut, dan semua isinya paling sedikit sebesar semua yang sudah diambil. Dengan pagi 12, 28, 30, 45, 136,5 dan sore 18, 30, 56, 70 (dalam ribuan): 12 melawan 18, ambil 12; 28 melawan 18, ambil 18; 28 melawan 30, ambil 28; 30 melawan 30, ambil 30 pagi; 45 melawan 30, ambil 30 sore; 45 melawan 56, ambil 45; 136,5 melawan 56, ambil 56; 136,5 melawan 70, ambil 70; tumpukan sore habis, jadi taruh 136,5 ke hasil.

Tiga pointer melaksanakannya: `pa` menelusuri array pagi, `pb` array sore, dan `po` keluarannya. Intinya satu baris, `*po++ = (*pa <= *pb) ? *pa++ : *pb++;`, yang menyalin tagihan depan yang lebih kecil ke keluaran dan memajukan dua pointer yang terlibat. Perbandingan `<=` mengambil tagihan pagi ketika keduanya sama, sehingga tagihan yang sama tetap dalam urutan sif: penggabungannya stabil. Jika array-array itu berisi n_a dan n_b elemen, setiap langkah mengeluarkan satu elemen, sehingga penggabungan memerlukan tepat $n_a + n_b$ langkah, dan melakukan paling banyak

$$n_a + n_b - 1 \tag{8.6}$$

perbandingan, karena setelah perbandingan terakhir paling sedikit satu elemen disalin tanpa perbandingan. Langkah ini adalah jantung *merge sort*, yang mengurutkan dengan membelah array menjadi dua, mengurutkan setiap belahan secara rekursif, lalu menggabungkan kedua belahan, dengan sekitar $n \log_2 n$ perbandingan [7].

8.8.3 Penyelesaiannya

Kode 8.19 mengembalikan banyaknya tagihan yang ditulis, yang dihitung sebagai selisih pointer `po - out` dari Bab 7; **Kode 8.20** membangun list baru sebagai gantinya.

Kode 8.19 `merge_shifts.c`: menggabungkan dua array terurut dengan tiga pointer yang berjalan, termasuk kasus sif sore yang kosong.

```c
/* merge_shifts.c - Soal 8.6: gabungkan tagihan terurut dari sif pagi dan
   sif sore menjadi satu daftar terurut, menelusuri keduanya dengan pointer */
#include <stdio.h>

int merge(const long *a, int na, const long *b, int nb, long *out)
{
    const long *pa = a, *pb = b;
    const long *enda = a + na, *endb = b + nb;
    long *po = out;
    while (pa < enda && pb < endb)       /* ambil tagihan depan yang lebih kecil */
        *po++ = (*pa <= *pb) ? *pa++ : *pb++;
    while (pa < enda)                    /* satu sif sudah habis: */
        *po++ = *pa++;                   /* salin sisa sif yang lain */
    while (pb < endb)
        *po++ = *pb++;
    return (int) (po - out);
}

void print_all(const char *title, const long *a, int n)
{
    printf("%-9s", title);
    for (int i = 0; i < n; i++)
        printf(" %ld", a[i] / 1000);
    printf("\n");
}

int main(void)
{
    long morning[5] = {12000, 28000, 30000, 45000, 136500};
    long evening[4] = {18000, 30000, 56000, 70000};
    long day[9];
    int n = merge(morning, 5, evening, 4, day);
    print_all("morning:", morning, 5);
    print_all("evening:", evening, 4);
    print_all("day:", day, n);
    n = merge(morning, 5, evening, 0, day);   /* sif sore yang tutup */
    print_all("no eve:", day, n);
    return 0;
}
```

Kode 8.20 `merge_shifts.py`: penggabungan yang sama dalam Python dengan dua indeks; `extend` menyalin sisa list yang belum habis.

```python
# merge_shifts.py - Soal 8.6: gabungkan tagihan terurut dari sif pagi dan
# sif sore menjadi satu daftar terurut, menelusuri keduanya dengan indeks


def merge(a, b):
    out = []
    i = j = 0
    while i < len(a) and j < len(b):     # ambil tagihan depan yang lebih kecil
        if a[i] <= b[j]:
            out.append(a[i])
            i += 1
        else:
            out.append(b[j])
            j += 1
    out.extend(a[i:])                    # satu sif sudah habis:
    out.extend(b[j:])                    # salin sisa sif yang lain
    return out


def print_all(title, a):
    print(f"{title:<9}" + "".join(f" {x // 1000}" for x in a))


def main():
    morning = [12000, 28000, 30000, 45000, 136500]
    evening = [18000, 30000, 56000, 70000]
    print_all("morning:", morning)
    print_all("evening:", evening)
    print_all("day:", merge(morning, evening))
    print_all("no eve:", merge(morning, []))   # sif sore yang tutup


if __name__ == "__main__":
    main()
```

```
morning:  12 28 30 45 136
evening:  18 30 56 70
day:      12 18 28 30 30 45 56 70 136
no eve:   12 28 30 45 136
```

Hari yang digabungkan itu terurut, memuat kesembilan tagihan, dan sif sore yang kosong cukup mengembalikan tagihan pagi. Untuk sembilan tagihan dalam contoh, Persamaan (8.6) membolehkan paling banyak 8 perbandingan; penelusuran tangan di atas melakukan tepat 8.

8.8.4 Solusi salah yang menggoda

Versi yang menggoda hanya punya perulangan pertama: selama kedua tumpukan masih berisi nota, ambil yang lebih kecil. Versi itu lolos pada uji yang kedua tumpukannya kebetulan habis bersamaan, dan kehilangan tagihan dalam setiap kasus lain: dengan data contoh, 136,5 tidak akan pernah tersalin, dan dengan sif sore yang kosong hasilnya akan kosong. Dua perulangan kecil sesudah perulangan utama, yang menyalin sisa tumpukan mana pun yang tertinggal, adalah bagian yang dilupakan versi yang menggoda itu. Uji dengan array kosong, yang dituntut pernyataan masalah, adalah persis uji yang menyingkapnya.

> **PERIKSA DIRI.** Apa yang salah jika perbandingannya ditulis `*pa < *pb` alih-alih `*pa <= *pb`?
>
> *Jawaban.* Keluarannya tetap terurut dan lengkap, sehingga setiap uji yang hanya memeriksa bilangannya lolos. Yang berubah adalah urutan tagihan yang sama: ketika dua tagihan depan sama besar, tagihan sore diambil lebih dulu. Penggabungannya tidak lagi stabil. Untuk tagihan saja tidak ada yang bisa melihat bedanya; untuk tagihan yang membawa nomor nota atau

> nama sif (Bab 10), tagihan yang sama akan keluar dengan pesanan sore mendahului pesanan pagi.

8.9 Keenam Masalah Berdampingan

Tabel 8.4 mengumpulkan apa yang diajarkan keenam masalah: biaya setiap penyelesaian dan jebakan yang menjerat solusi salah yang menggoda, beserta uji yang menyingkapnya.

Tabel 8.4 Keenam masalah berdampingan: biaya setiap penyelesaian, jebakan solusi salah yang menggoda, dan masukan uji yang menyingkap jebakan itu.

Soal	Biaya	Jebakannya	Uji yang menyingkapnya
8.1 Statistik	n langkah per fungsi	`min` dimulai dari 0; rata-rata bilangan bulat	nilai positif saja; rata-rata yang tidak bulat
8.2 Fibonacci	perulangan n; rekursi $2F(n+1)-1$	barisan dimulai dari 1, 1	contoh yang diberikan, $n = 9$
8.3 Palindrom	paling banyak $n/2$ perbandingan	pasangan di `n - i`	palindrom apa pun berhuruf dua atau lebih
8.4 Determinan	tetap 14 operasi	tanda suku tengah	dua baris sama (identitas tidak menangkapnya)
8.5 Nota	paling banyak $\lfloor \log_2 n \rfloor + 1$ pemanggilan	rekursi pada `mid`	bagian dua elemen, nomor di kanan
8.6 Penggabungan	$n_a + n_b$ langkah	melupakan sisanya	array kosong; panjang yang tidak sama

Bacalah kolom terakhir sekali lagi. Dalam setiap kasus, uji yang menyingkap adalah sebuah *tepi*: nilai di batas, masukan kosong, nilai yang berulang, contoh yang dihitung dengan tangan. Kernighan dan Pike menyebutnya *menguji di batas-batas* (*testing at the boundaries*), dan inilah satu-satunya kebiasaan pengujian paling produktif yang dapat dimiliki seorang pemrogram [6].

8.10 Kesalahan yang Sering Terjadi, dan Cara Menghindarinya

- **Mengubah kode yang diberikan.** Ketika prototipe dan `main` diberikan, keduanya adalah kesepakatan. Lengkapi badan fungsinya; jangan mengganti nama atau "memperbaiki" yang sudah diberikan.
- **Meng-cast hasil alih-alih operan.** `(double) (sum / n)` melakukan cast setelah pembagian bilangan bulat membuang pecahannya. Tulislah `(double) sum / n`.
- **Memulai minimum atau maksimum dari 0.** Mulailah dari `INT_MAX`/`INT_MIN` atau dari elemen pertama.
- **Mencampur penghitung bertanda dan tak bertanda.** Bandingkan ukuran `unsigned int` dengan penghitung `unsigned int`; `-Wextra` memberi tahu di mana.
- **Melewatkan contoh.** Hitunglah contoh yang diberikan dengan tangan lebih dulu; contoh itu menangkap kesalahan kesepakatan seperti awal barisan Fibonacci.
- **Rekursi pada masukan besar tanpa menghitung.** Hitunglah pemanggilannya, atau bernalarlah dengan relasi rekurensi, sebelum mempercayakan rekursi pada $n = 40$.
- **Indeks pasangan yang meleset satu.** Dalam array sepanjang `n`, pasangan `i` adalah `n - 1 - i`; `s[n]` adalah nol penutup sebuah string.
- **`&` pada array dalam `scanf`.** `scanf("%99s", word)` tidak memerlukan `&`, dan selalu berikan lebarnya.

- **Matriks uji yang lemah.** Matriks identitas menyembunyikan kesalahan tanda; ujilah dengan matriks yang setiap entrinya berperan dan dengan matriks yang determinannya harus 0.
- **Rekursi yang tidak menyusut.** Setiap pemanggilan rekursif harus bekerja pada bagian yang benar-benar lebih kecil; kecualikan yang sudah diperiksa.
- **Melupakan sisa setelah penggabungan.** Setelah perulangan utama, salin apa yang tersisa dari setiap masukan.

8.11 Ringkasan Bab

- Setiap masalah mengikuti delapan langkah: nyatakan ulang, selesaikan dengan tangan, rancang di atas kertas, perkirakan biaya, tulis kode, uji, temukan solusi salah yang menggoda, tinjau kembali. Contoh yang dikerjakan dengan tangan adalah langkah yang menentukan apa arti "benar".
- Soal 8.1 melengkapi lima fungsi statistik di dalam kerangka yang tetap; rata-rata Persamaan (8.1) memerlukan cast sebelum pembagian, dan simpangan baku Persamaan (8.2) adalah akar kuadrat rata-rata kuadrat simpangan, yang dihitung dengan tangan di **Tabel 8.2**.
- Soal 8.2 menghitung bilangan Fibonacci dengan perulangan dalam n langkah dan dengan rekursi dalam $2F(n+1)-1$ pemanggilan, memakai identitas Persamaan (8.3) untuk jumlahnya, dan mengukur perbedaannya (**Gambar 8.1**, **Tabel 8.3**): algoritma yang lebih baik mengalahkan bahasa yang lebih cepat.
- Soal 8.3 memutuskan palindrom dengan dua pointer, dengan rekursi, dan dengan XOR; pasangan posisi `i` adalah `n - 1 - i`.
- Soal 8.4 menghitung determinan 3×3 dengan Persamaan (8.5), memutuskan keterbalikannya, dan memperlihatkan bahwa matriks identitas tidak dapat menangkap kesalahan tanda.
- Soal 8.5 menulis pencarian biner sebagai rekursi yang setiap pemanggilannya bekerja pada bagian yang benar-benar lebih kecil; Soal 8.6 menggabungkan dua array terurut dengan pointer yang berjalan dalam $n_a + n_b$ langkah dan paling banyak Persamaan (8.6) perbandingan.
- Dalam setiap masalah, uji yang menyingkap solusi yang salah adalah kasus tepi: **Tabel 8.4**.

8.12 Soal Latihan

1. **Statistik, diperluas.** Tambahkan fungsi keenam, `double median(int *a, unsigned int aSize)`, ke Soal 8.1 tanpa mengubah prototipe yang diberikan, lalu hitung dengan tangan apa yang dikembalikannya untuk kesepuluh nilai itu. Fungsi itu tidak boleh mengubah urutan array milik pemanggil.
2. **Fibonacci tanpa trik jumlah.** Tulislah fungsi rekursif `long long fib_sum_rec(int n)` yang menghitung $\sum_{i=0}^{n} F(i)$ langsung dari definisinya, $S(n) = S(n-1) + F(n)$, lalu hitung pemanggilannya untuk $n = 9$. Bandingkan dengan 287 pemanggilan `fib_rec_sum(9)`.
3. **Bilangan palindrom.** Nomor nota seperti 12321 adalah palindrom. Tulislah `int is_pal_number(long n)` yang memutuskannya *tanpa* mengubah bilangan itu menjadi string, dengan membangun bilangan kebalikannya memakai `% 10` dan `/ 10`. Bilangan berakhiran 0 mana yang dapat menjadi palindrom?
4. **Sarrus.** Untuk matriks 3×3, aturan Sarrus menulis determinan sebagai $aei + bfg + cdh - ceg - bdi - afh$. Tunjukkan bahwa aturan itu sesuai dengan Persamaan (8.5) dengan mengekspansi keduanya, lalu hitung determinan nota-nota Mbak Rina, Persamaan (8.4), dengan aturan itu.

5. **Menemukan kembali harga-harganya.** Aturan Cramer menyatakan bahwa unsur tak diketahui pertama dari sistem $A\mathbf{x} = \mathbf{b}$ adalah $x_1 = \det A_1 / \det A$, dengan A_1 adalah A yang kolom pertamanya diganti $\mathbf{b}$, dan demikian pula untuk yang lain. Pakailah untuk menemukan harga deterjen, pelembut, dan pelicin dari ketiga nota di Subbab 8.6.1 (total Rp60.000, Rp72.000, dan Rp20.000).
6. **Menggabungkan tiga sif.** Sif ketiga, sif malam, ditambahkan. Gabungkan tiga array terurut menjadi satu, dengan lebih dulu menggabungkan dua lalu hasilnya dengan yang ketiga, dan hitung langkahnya untuk sif berukuran n_1, n_2, n_3. Apakah urutan penggabungannya berpengaruh?
7. **Pencarian biner iteratif dari yang rekursif.** Ubahlah `find` pada Soal 8.5 menjadi perulangan dengan mengganti setiap pemanggilan rekursif dengan pembaruan `low` atau `high`. Mengapa hal ini selalu mungkin untuk fungsi yang pemanggilan rekursifnya adalah hal terakhir yang dilakukannya?

8.13 Penyelesaian Soal Latihan

1. Salin array ke array lokal berisi 10 elemen (atau berkapasitas tetap), urutkan salinannya dengan bubble sort atau insertion sort dari Bab 6, lalu kembalikan nilai tengahnya: untuk `aSize` genap, rata-rata elemen di `aSize / 2 - 1` dan `aSize / 2`, sebagai `double`. Nilai-nilai terurutnya 7, 8, 9, 10, 11, 12, 14, 15, 18, 20, sehingga mediannya $(11 + 12)/2 = 11.5$, sedikit di bawah rata-rata 12,4 karena dua kunjungan terbesar, 18 dan 20 kg, menarik rata-rata ke atas.

2. `if (n < 0) return 0; return fib_sum_rec(n - 1) + fib_rec(n);`. Pemanggilan `fib_sum_rec` sendiri sebanyak $n + 2$ (dari n turun sampai -1), 11 untuk $n = 9$, dan setiap tingkat juga memanggil `fib_rec(i)`, dengan $2F(i + 1) - 1$ pemanggilan, untuk i = 0 sampai 9: $\sum_{i=0}^{9}(2F(i + 1) - 1) = 2 \cdot (F(1) + \cdots + F(10)) - 10 = 2 \cdot 143 - 10 = 276$. Seluruhnya 287 pemanggilan, tepat sebanyak `fib_rec_sum(9)`, dan bukan kebetulan: secara umum cara langsung ini melakukan $(n + 2) + (2F(n + 3) - n - 3) = 2F(n + 3) - 1$ pemanggilan, yaitu $C(n + 2)$, biaya `fib_rec(n + 2)`. Kedua cara itu eksponensial; hanya perulangan yang tidak.

3. Bangunlah kebalikannya: `long r = 0, m = n; while (m > 0) { r = r * 10 + m % 10; m /= 10; } return r == n;`. Untuk 12321 perulangannya membangun 1, 12, 123, 1232, 12321, sama dengan n. Bilangan berakhiran 0 harus diawali 0 untuk menjadi palindrom, dan tidak ada bilangan yang ditulis tanpa nol di depan yang demikian; satu-satunya palindrom seperti itu adalah 0 sendiri (dan bilangan negatif bukan palindrom karena tanda minusnya).

4. Mengekspansi Persamaan (8.5): $a(ei - fh) - b(di - fg) + c(dh - eg) = aei - afh - bdi + bfg + cdh - ceg$, yaitu keenam suku Sarrus dalam urutan lain. Untuk Persamaan (8.4), dengan $a, b, c = 2,1,1$; $d, e, f = 1,3,2$; $g, h, i = 1,0,0$: $aei + bfg + cdh = 0 + 2 + 0 = 2$ dan $ceg + bdi + afh = 3 + 0 + 0 = 3$, sehingga det $= 2 - 3 = -1$, seperti sebelumnya.

5. $\det A = -1$. Mengganti kolom pertama dengan (60000,72000,20000): $\det A_1 = 60000(3 \cdot 0 - 2 \cdot 0) - 1(72000 \cdot 0 - 2 \cdot 20000) + 1(72000 \cdot 0 - 3 \cdot 20000) = 0 + 40000 - 60000 = -20000$, sehingga harga deterjen \$-20000 / -1 = \$ Rp20.000, yang dikonfirmasi nota ketiga. Mengganti kolom kedua: $\det A_2 = 2(72000 \cdot 0 - 2 \cdot 20000) - 60000(1 \cdot 0 - 2 \cdot 1) + 1(1 \cdot 20000 - 72000 \cdot 1) = -80000 + 120000 - 52000 = -12000$, pelembut Rp12.000. Mengganti kolom ketiga: $\det A_3 = 2(3 \cdot 20000 - 72000 \cdot 0) - 1(1 \cdot 20000 - 72000 \cdot 1) + 60000(1 \cdot 0 - 3 \cdot 1) = 120000 + 52000 - 180000 = -8000$, pelicin Rp8.000. Periksa dengan nota pertama: $2 \cdot 20000 + 12000 + 8000 = 60000$.

6. Menggabungkan dua yang pertama memerlukan $n_1 + n_2$ langkah dan menghasilkan $n_1 + n_2$ tagihan; menggabungkannya dengan yang ketiga memerlukan $n_1 + n_2 + n_3$ langkah. Seluruhnya $2n_1 + 2n_2 + n_3$. Urutannya berpengaruh: dua array yang digabungkan lebih dulu tersalin dua kali, jadi gabungkan dua yang *terkecil* lebih dulu. Dengan ukuran 50, 40, dan 5, menggabungkan 40 dan 5 lebih dulu memerlukan $45 + 95 = 140$ langkah; menggabungkan 50 dan 40 lebih dulu memerlukan $90 + 95 = 185$.

7. Ganti `return find(a, low, mid - 1, key, depth + 1);` dengan `high = mid - 1;` dan pemanggilan lainnya dengan `low = mid + 1;`, bungkus badannya dalam `while (low <= high)`, lalu kembalikan -1 setelah perulangan; hasilnya adalah `binary_search` dari Bab 6. Hal ini selalu mungkin ketika pemanggilan rekursif adalah tindakan terakhir (*tail call*), karena tidak ada lagi yang harus dikerjakan setelah pemanggilan itu kembali: pemanggilan itu hanya mengubah argumen lalu memulai ulang, persis yang dilakukan satu putaran perulangan.

Daftar Pustaka

[1] G. Pólya, *How to Solve It: A New Aspect of Mathematical Method*, Princeton Science Library ed. Princeton, NJ, USA: Princeton University Press, 2014.

[2] Python Software Foundation, “statistics — Mathematical statistics functions,” Python 3.14 documentation. [Online]. Available: https://docs.python.org/3/library/statistics.html

[3] D. E. Knuth, *The Art of Computer Programming, Vol. 1: Fundamental Algorithms*, 3rd ed. Reading, MA, USA: Addison-Wesley, 1997.

[4] H. Abelson and G. J. Sussman, *Structure and Interpretation of Computer Programs*, 2nd ed. Cambridge, MA, USA: MIT Press, 1996.

[5] G. Strang, *Introduction to Linear Algebra*, 6th ed. Wellesley, MA, USA: Wellesley-Cambridge Press, 2023.

[6] B. W. Kernighan and R. Pike, *The Practice of Programming*. Reading, MA, USA: Addison-Wesley, 1999.

[7] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*, 4th ed. Cambridge, MA, USA: MIT Press, 2022.

Bab 9

Karakter, String, dan I/O Terformat

Hampir semua yang diterima program dari manusia berupa teks: nama yang diketik di meja kasir, nomor telepon yang disalin dari obrolan, sebuah perintah, sebaris isi berkas. Hampir semua yang dikembalikannya juga teks: struk, tabel, pesan. Bab ini mengajarkan cara C menyimpan teks, yang ternyata sangat sederhana, dan cara menanganinya tanpa kesalahan-kesalahan yang membuat penanganan string menjadi sumber *bug* paling terkenal dalam sejarah C. Di akhir bab Anda akan mampu menjelaskan apa itu string di memori dan mengapa ia diakhiri byte nol; menggolongkan dan mengubah karakter tunggal dengan `<ctype.h>`; memakai fungsi-fungsi sehari-hari `<string.h>` beserta satu aturan yang menjaga keamanan masing-masing; mengatur setiap kolom keluaran dengan *flag*, lebar, dan presisi `printf`; membaca baris utuh secara aman dengan `fgets` lalu memilahnya; serta membangun kasir Profitina Laundry versi 0.7 yang memahami perintah ketikan, menulis nama setiap pelanggan dengan cara yang sama, dan menemukan pelanggan dari sebagian namanya. Seperti biasa, setiap program diberikan dalam C dan Python, dan keduanya mencetak keluaran yang persis sama.

9.1 Posisi Kita Sekarang: Nama-nama di Buku Mbak Rina

Di akhir Bab 7, kasir Profitina Laundry sudah bisa menghitung tagihan, menyimpan pesanan sehari dalam larik (*array*), dan mengembalikan beberapa hasil dari satu fungsi lewat pointer. Bab 8 lalu memakai semua alat itu untuk sejumlah soal utuh. Namun ada satu hal yang belum bisa dilakukan kasir: menangani *nama*. Ia memberi nomor 1, 2, 3 pada pesanan, dan Mbak Rina, yang menjaga meja kasir, harus menyimpan buku catatan kertas di samping komputer untuk mengingat nomor mana milik pelanggan siapa.

Lihat satu halaman buku itu dan Anda akan paham mengapa program bisa membantu. Pelanggan tetap yang sama tertulis sebagai “andi”, “ANDI PRATAMA”, “Andi Pratama” dengan dua spasi, dan “andi p.” dalam empat tulisan tangan berbeda, karena minggu itu empat orang bergantian menjaga meja. Nomor telepon tertulis “0812-3456-7890”, “0812 3456 7890”, dan “+62 812 3456 7890”. Ketika Andi kembali hari Jumat untuk mengambil cuciannya dan menyebut “Pratama”, Mbak Rina membolak-balik tiga halaman untuk menemukannya. Dan struk yang dicetak kasir kolomnya berkelok-kelok: nama panjang mendorong harga ke kanan, nama pendek menariknya ke kiri, dan pelanggan harus memicingkan mata mencari totalnya.

Semua masalah ini adalah masalah *teks*. Merapikan nama berarti memeriksa hurufnya satu per satu dan mengubah sebagian. Membersihkan nomor telepon berarti menyimpan angkanya dan membuang sisanya. Menemukan “Pratama” di dalam “Andi Pratama” adalah pencarian sepotong teks di dalam teks lain. Meratakan struk berarti memberi tahu `printf` persis berapa kolom yang boleh dipakai setiap nilai. Dan memahami perintah yang diketik, seperti `add fold 3.5 andi pratama`, berarti memotong sebaris teks menjadi kata-kata dan menentukan arti setiap kata.

C dikenal agak canggung dengan teks, dan itu ada benarnya: tidak ada “tipe string” bawaan yang membesar dan mengecil sendiri seperti di Python. Tetapi gagasan di balik string C cukup kecil untuk dipahami sepenuhnya dalam satu subbab, dan setelah Anda memahaminya, setiap fungsi string di pustaka menjadi perulangan mudah yang bisa Anda tulis sendiri. Begitulah bab ini

berjalan: pertama gagasannya, lalu pustaka karakter dan string, kemudian keluaran terformat dan masukan yang aman, dan akhirnya kasir v0.7 yang memakai semuanya.

9.2 Gagasannya dengan Kata-kata Sederhana: Teks Adalah Bilangan

9.2.1 Karakter adalah bilangan kecil

Komputer hanya menyimpan bilangan. Untuk menyimpan huruf `A`, ia menyimpan bilangan yang telah disepakati semua orang sebagai lambang `A`. Kesepakatan yang dipakai C di hampir semua komputer adalah ASCII, *American Standard Code for Information Interchange*, yang memberikan bilangan 0 sampai 127 kepada huruf Inggris, angka, tanda baca, spasi, dan beberapa karakter kendali seperti baris baru [1], [2]. Dalam ASCII, `A` bernilai 65, `B` 66, dan seterusnya sampai `Z` yang bernilai 90; huruf kecil dimulai dari `a` yang bernilai 97; angka `0` bernilai 48, `1` bernilai 49, sampai `9` yang bernilai 57; spasi bernilai 32 dan baris baru 10.

Di C, tipe `char` menampung satu bilangan seperti itu, dan konstanta karakter yang ditulis di antara tanda petik tunggal, seperti `'A'`, hanyalah cara lain menulis bilangan 65. Anda boleh menjumlahkannya, membandingkannya, dan menguranginya seperti bilangan bulat lain. Dua fakta kecil berikut akan Anda pakai berulang kali:

- Angka-angka letaknya berurutan, sehingga *nilai* karakter angka `c` adalah `c - '0'`. Misalnya, `'7' - '0'` bernilai $55 - 48 = 7$.
- Huruf kecil berurutan dan huruf kapital berurutan, dan setiap huruf kecil terletak tepat 32 tempat di atas kapitalnya: `'a' - 'A'` bernilai $97 - 65 = 32$. Jadi mengubah huruf besar-kecil hanyalah penjumlahan atau pengurangan, meskipun, seperti ditunjukkan Subbab 9.3, pustaka melakukannya dengan lebih aman.

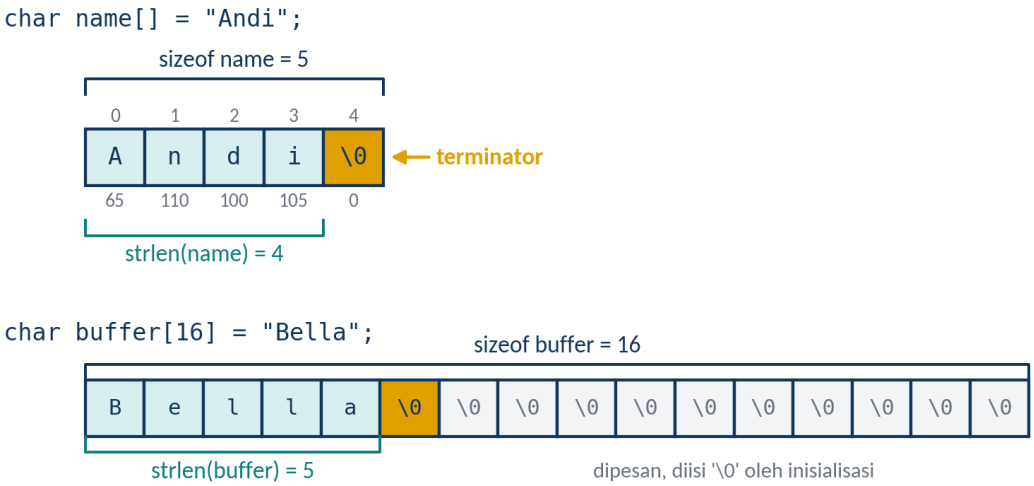
Nama seperti “Dewi Anggraini” muat dalam ASCII. Nama dengan huruf di luar abjad Inggris, seperti “Chloé”, memerlukan lebih dari satu byte per huruf dalam pengodean UTF-8 yang dipakai sistem modern. Bab ini bekerja dengan teks ASCII, yang mencakup semua yang diketik di laundry; Subbab 9.11 menjelaskan apa yang berubah untuk aksara lain.

9.2.2 String adalah larik yang diakhiri nol

String di C bukan tipe khusus. Ia adalah larik `char` yang menampung kode karakter satu demi satu, diikuti satu byte tambahan bernilai 0, ditulis `'\0'` dan disebut *null terminator* (penutup nol) atau *karakter nol*. Penutup itulah seluruh triknya: ia menandai di mana teks berakhir, sehingga panjangnya tidak perlu disimpan terpisah.

> **DEFINISI.** *String* di C adalah deretan karakter yang disimpan dalam byte-byte berurutan sebuah larik `char` dan diakhiri karakter nol `'\0'` (byte bernilai 0). *Literal string* adalah teks yang ditulis di antara tanda petik ganda di dalam program, seperti `"Andi"`; kompilator menyimpannya dengan penutup yang sudah ditambahkan. *Panjang* string adalah banyaknya karakter sebelum penutup.

Gambar 9.1 memperlihatkan dua larik dari program pertama bab ini. Deklarasi `char name[] = "Andi";` membuat larik yang pas untuk literalnya: empat huruf ditambah penutup, lima byte. Deklarasi `char buffer[16] = "Bella";` memesan enam belas byte; enam byte pertama menerima `B`, `e`, `l`, `l`, `a`, dan penutup, dan karena larik yang diinisialisasi dengan nilai lebih sedikit daripada ukurannya mendapat nol di sisanya, sepuluh byte lainnya juga bernilai nol.



Gambar 9.1 Dua string di memori: `name` menampung empat huruf dan penutup dalam tepat lima byte, sedangkan `buffer` memesan enam belas byte dan string hanya memakai enam. Bilangan abu-abu di bawah `name` adalah kode ASCII-nya.

Dua pertanyaan tentang string terdengar mirip tetapi jawabannya berbeda. Berapa byte yang ditempati larik? Itulah `sizeof`, yang diketahui kompilator dari deklarasinya: 5 untuk `name`, 16 untuk `buffer`. Berapa panjang teksnya? Itulah `strlen`, fungsi yang berjalan menyusuri larik sambil menghitung karakter sampai bertemu penutup: 4 untuk `name`, 5 untuk `buffer`. Sebagai rumus, untuk larik yang dideklarasikan dari literal berisi n karakter,

$$\text{sizeof name} = n + 1, \qquad \text{strlen(name)} = n \tag{9.1}$$

dan Persamaan (9.1) patut diingat, karena lupa "+ 1" untuk penutup adalah cara paling umum membuat larik kurang satu byte.

Setiap fungsi string di C, tanpa kecuali, adalah perulangan yang berjalan sampai bertemu `'\0'`. **Kode 9.1** menulis satu perulangan seperti itu secara manual, `my_strlen`, lalu memeriksa larik-larik pada **Gambar 9.1** byte demi byte.

Kode 9.1 `string_basics.c`: string adalah larik `char` yang diakhiri `'\0'`; `sizeof` menghitung byte, `strlen` menghitung karakter.

```c
/* string_basics.c - a C string is an array of char that ends with '\0'. */
#include <stdio.h>
#include <string.h>

/* Count the characters before the terminator, exactly as strlen does. */
static size_t my_strlen(const char *s)
{
    size_t n = 0;
    while (s[n] != '\0')
        n++;
    return n;
}

int main(void)
{
    char name[] = "Andi";            /* 5 bytes: 'A' 'n' 'd' 'i' '\0' */
    char buffer[16] = "Bella";       /* 16 bytes reserved, 6 of them used */
    const char *shop = "Profitina Laundry";  /* points at a read-only literal */

    printf("name   = \"%s\"  sizeof = %zu  strlen = %zu\n",
           name, sizeof name, strlen(name));
    printf("buffer = \"%s\"  sizeof = %zu  strlen = %zu\n",
           buffer, sizeof buffer, strlen(buffer));
    printf("shop   = \"%s\"  my_strlen = %zu\n", shop, my_strlen(shop));

    printf("bytes of name:");
    for (size_t i = 0; i < sizeof name; i++)
        printf(" %d", name[i]);
    printf("\n");

    name[0] = 'a';                   /* an array holds its own copy: we may edit */
    printf("after name[0] = 'a': %s\n", name);

    buffer[3] = '\0';                /* an earlier terminator cuts the string */
    printf("after buffer[3] = '\\0': %s (strlen %zu)\n", buffer, strlen(buffer));

    printf("'A' = %d, 'a' = %d, '0' = %d, ' ' = %d\n", 'A', 'a', '0', ' ');
    printf("'7' - '0' = %d, 'a' - 'A' = %d\n", '7' - '0', 'a' - 'A');
    return 0;
}
```

Kompilasi dan jalankan seperti di Bab 1 (`gcc -std=c17 -O2 -Wall -Wextra -o string_basics string_basics.c`, lalu `./string_basics`). Hasilnya:

```
name   = "Andi"  sizeof = 5  strlen = 4
buffer = "Bella"  sizeof = 16  strlen = 5
shop   = "Profitina Laundry"  my_strlen = 17
bytes of name: 65 110 100 105 0
after name[0] = 'a': andi
after buffer[3] = '\0': Bel (strlen 3)
'A' = 65, 'a' = 97, '0' = 48, ' ' = 32
'7' - '0' = 7, 'a' - 'A' = 32
```

Bacalah keluaran baris demi baris sambil melihat kodenya. Deretan byte `65 110 100 105 0` adalah **Gambar 9.1** dalam bentuk bilangan: empat kode dan penutupnya. Baris `name[0] = 'a'` mengubah byte pertama larik, dan mencetaknya lagi menghasilkan `andi`; ini boleh karena `name` adalah larik yang memegang *salinan sendiri* huruf-hurufnya. Menulis `buffer[3] = '\0'` menaruh penutup di tengah `Bella`: tidak ada yang terhapus sesudahnya, `l`, `a`, dan penutup lama masih ada, tetapi setiap fungsi string kini berhenti di indeks 3 dan hanya melihat `Bel`. Dua baris terakhir menegaskan aritmetika Subbab 9.2.1.

> **PERINGATAN:** `const char *shop = "Profitina Laundry";` tidak membuat larik; ia membuat *pointer* (Bab 7) ke literal yang disimpan kompilator di suatu tempat, sering kali di

memori yang tidak boleh ditulisi. `shop[0] = 'p';` lolos kompilasi tetapi perilakunya tak terdefinisi dan biasanya membuat program *crash*. Deklarasikan pointer seperti itu dengan `const`, seperti dalam kode, dan kompilator akan menolak penugasan tersebut. Bila Anda memerlukan teks yang boleh diubah, deklarasikan larik.

9.2.3 Program yang sama dalam Python

Python menyimpan string secara berbeda: string Python adalah objek yang mengetahui panjangnya sendiri dan tidak pernah memerlukan penutup, dan ia tidak dapat diubah setelah dibuat (string bersifat *immutable*). Untuk mengganti huruf pertama, Anda membangun string baru dari potongan-potongan. **Kode 9.2** mencetak baris-baris yang sama dengan program C dengan menunjukkan apa yang akan disimpan C; nilai `sizeof` dihitung sebagai panjang teks ditambah satu, persis Persamaan (9.1).

Kode 9.2 `string_basics.py`: baris-baris yang sama dalam Python, yang stringnya mengetahui panjangnya dan tidak dapat diubah di tempat.

```
# string_basics.py - Python strings know their own length; C strings end in '\0'.
# This version prints the same lines as string_basics.c by showing the bytes
# a C program would store.


def my_strlen(s):
    """Count characters up to the (imagined) terminator, like strlen."""
    n = 0
    for _ in s:
        n += 1
    return n


def main():
    name = "Andi"                   # C stores 5 bytes: 'A' 'n' 'd' 'i' '\0'
    buffer = "Bella"                # C reserved 16 bytes for it
    shop = "Profitina Laundry"

    print('name   = "%s"  sizeof = %d  strlen = %d'
          % (name, len(name) + 1, len(name)))
    print('buffer = "%s"  sizeof = %d  strlen = %d' % (buffer, 16, len(buffer)))
    print('shop   = "%s"  my_strlen = %d' % (shop, my_strlen(shop)))

    raw = name.encode("ascii") + b"\0"
    print("bytes of name:" + "".join(" %d" % b for b in raw))

    name = "a" + name[1:]           # Python strings are immutable: build a new one
    print("after name[0] = 'a': %s" % name)

    buffer = buffer[:3]             # what an early '\0' does in C
    print("after buffer[3] = '\\0': %s (strlen %d)" % (buffer, len(buffer)))

    print("'A' = %d, 'a' = %d, '0' = %d, ' ' = %d"
          % (ord("A"), ord("a"), ord("0"), ord(" ")))
    print("'7' - '0' = %d, 'a' - 'A' = %d"
          % (ord("7") - ord("0"), ord("a") - ord("A")))


if __name__ == "__main__":
    main()
```

Jalankan dengan `python3 string_basics.py` (di Windows, `python` atau `py`). Keluarannya identik dengan program C, byte demi byte; penguji di Subbab 9.9 memeriksa hal itu untuk setiap program dalam bab ini.

9.3 Satu Karakter pada Satu Waktu: `<ctype.h>`

Sebelum menangani string utuh, kita perlu alat untuk karakter tunggal. Apakah karakter ini huruf? Angka? Spasi? Apa bentuk kapitalnya? *Header* `<ctype.h>` menjawab pertanyaan-pertanyaan ini dengan sekelompok fungsi kecil yang tercantum dalam **Tabel 9.1** [2], [3]. Masing-masing menerima satu karakter dan mengembalikan jawaban benar (bukan nol) atau salah (nol), atau karakter hasil perubahan.

Tabel 9.1 Fungsi `<ctype.h>` yang paling sering dipakai. Karakter yang bukan huruf melewati `toupper` dan `tolower` tanpa berubah.

Fungsi	Benar untuk	Contoh yang benar
`isalpha(c)`	huruf	`'A'`, `'z'`
`isdigit(c)`	angka desimal	`'0'` sampai `'9'`
`isalnum(c)`	huruf atau angka	`'b'`, `'7'`
`isspace(c)`	karakter spasi putih	`' '`, `'\t'`, `'\n'`, `'\r'`
`isupper(c)` / `islower(c)`	huruf kapital / kecil	`'Q'` / `'q'`
`ispunct(c)`	tanda baca	`'!'`, `'-'`, `','`
`toupper(c)` / `tolower(c)`	(mengembalikan huruf yang diubah)	`toupper('a')` bernilai `'A'`

Ada satu detail yang tampak remeh tetapi penting. Fungsi-fungsi ini menerima `int` yang harus bernilai `EOF` atau nilai yang muat dalam `unsigned char`, yaitu 0 sampai 255. Di kebanyakan komputer `char` biasa bersifat *signed* (bertanda), sehingga byte di atas 127 (yang muncul dalam teks UTF-8) tersimpan sebagai bilangan negatif, dan memberikannya langsung kepada `isalpha` berakibat perilaku tak terdefinisi. Obatnya adalah *cast*: `isalpha((unsigned char)c)`. Setiap program dalam buku ini melakukannya, dan program Anda juga sebaiknya begitu [4].

Sekarang masalah-masalah laundry. **Kode 9.3** berisi tiga fungsi kecil. `classify` menghitung huruf, angka, spasi, dan karakter lain dalam sebuah string, pemeriksaan yang biasa dilakukan formulir sebelum menerima isian. `digits_only` menyalin hanya angka-angka nomor telepon ke larik kedua, yang mengubah segala cara menuliskan nomor menjadi satu bentuk baku, dan `valid_mobile` lalu memeriksa bahwa hasilnya tampak seperti nomor ponsel Indonesia: diawali `08` dan terdiri atas 10 sampai 13 angka. `tidy_name` menerima nama yang diketik dengan campuran huruf besar-kecil dan spasi sembarang lalu menuliskannya dalam *title case*: tanpa spasi di ujung-ujungnya, satu spasi di antara kata, huruf pertama setiap kata kapital dan sisanya kecil.

Kode 9.3 `char_classes.c`: menggolongkan karakter, menyimpan angka-angka nomor telepon, dan merapikan nama.

```c
/* char_classes.c - <ctype.h>: classify characters, clean a phone number and
 * tidy a customer name the way the Profitina Laundry counter receives them. */
#include <ctype.h>
#include <stdio.h>
#include <string.h>

/* Count letters, digits, spaces and everything else in s. */
static void classify(const char *s)
{
    int letters = 0, digits = 0, spaces = 0, others = 0;
    for (size_t i = 0; s[i] != '\0'; i++) {
        unsigned char c = (unsigned char)s[i];   /* ctype needs unsigned */
        if (isalpha(c))
            letters++;
        else if (isdigit(c))
            digits++;
        else if (isspace(c))
            spaces++;
        else
            others++;
    }
    printf("\"%s\": letters=%d digits=%d spaces=%d others=%d\n",
           s, letters, digits, spaces, others);
}

/* Copy only the digits of src into dst (size bytes); return how many. */
static size_t digits_only(const char *src, char *dst, size_t size)
{
    size_t n = 0;
    for (size_t i = 0; src[i] != '\0' && n + 1 < size; i++)
        if (isdigit((unsigned char)src[i]))
            dst[n++] = src[i];
    dst[n] = '\0';
    return n;
}

/* An Indonesian mobile number: starts with 08 and has 10 to 13 digits. */
static int valid_mobile(const char *digits)
{
    size_t n = strlen(digits);
    return n >= 10 && n <= 13 && digits[0] == '0' && digits[1] == '8';
}
```

Kode 9.3 (Lanjutan) *char_classes.c: menggolongkan karakter, menyimpan angka-angka nomor telepon, dan merapikan nama.*

```c
/* Trim, squeeze inner spaces to one, and write each word as Title case. */
static void tidy_name(const char *src, char *dst, size_t size)
{
    size_t n = 0;
    int new_word = 1;
    for (size_t i = 0; src[i] != '\0' && n + 1 < size; i++) {
        unsigned char c = (unsigned char)src[i];
        if (isspace(c)) {
            new_word = 1;
            continue;
        }
        if (new_word && n > 0 && n + 2 < size)
            dst[n++] = ' ';          /* one space between words */
        dst[n++] = (char)(new_word ? toupper(c) : tolower(c));
        new_word = 0;
    }
    dst[n] = '\0';
}

int main(void)
{
    const char *phones[] = {"0812-3456-7890", "+62 812 3456 7890", "0812 34"};
    const char *names[] = {"  aNDi   pratama ", "BUDI", "citra  dewi lestari"};
    char buf[32];

    classify("Order PL-0042: 3.5 kg!");
    for (int i = 0; i < 3; i++) {
        size_t n = digits_only(phones[i], buf, sizeof buf);
        printf("phone %-18s -> %-13s (%zu digits) %s\n", phones[i], buf, n,
               valid_mobile(buf) ? "ok" : "check with customer");
    }
    for (int i = 0; i < 3; i++) {
        tidy_name(names[i], buf, sizeof buf);
        printf("name  [%s] -> [%s]\n", names[i], buf);
    }
    return 0;
}
```

Perhatikan kedua fungsi penyalin baik-baik, karena polanya muncul di setiap rutin string yang aman dalam buku ini. Masing-masing menerima larik tujuan *beserta ukurannya*, dan syarat perulangan `n + 1 < size` memastikan selalu tersisa sedikitnya satu byte untuk penutup. Setelah perulangan, `dst[n] = '\0'` mengakhiri string baru. Perulangan penyalin yang melupakan salah satu separuhnya, pemeriksaan ukuran atau penutup, adalah *bug* yang tinggal menunggu masukan panjang pertama.

`tidy_name` memakai *variabel keadaan* kecil, `new_word`, yang mengingat apakah karakter sebelumnya spasi (atau apakah kita berada di paling awal). Setiap kali spasi datang, fungsi hanya mengatur `new_word = 1` dan tidak menyalin apa pun, sehingga deretan spasi berapa pun lenyap. Ketika karakter bukan-spasi datang sesudah spasi, fungsi lebih dulu menulis satu spasi, tetapi hanya bila sudah ada yang ditulis (`n > 0`), dan dengan cara itulah spasi di depan hilang; lalu ia menulis karakternya dalam huruf kapital bila memulai kata dan dalam huruf kecil bila tidak. Spasi di belakang tidak pernah menghasilkan apa-apa karena tidak ada karakter sesudahnya. Cara berpikir “apa yang baru saja saya lihat, dan apa artinya bagi karakter sekarang” ini adalah jantung semua pengolahan teks.

Keluarannya:

```
"Order PL-0042: 3.5 kg!": letters=9 digits=6 spaces=3 others=4
phone 0812-3456-7890     -> 081234567890  (12 digits) ok
phone +62 812 3456 7890  -> 6281234567890 (13 digits) check with customer
phone 0812 34            -> 081234        (6 digits) check with customer
name  [  aNDi   pratama ] -> [Andi Pratama]
name  [BUDI] -> [Budi]
name  [citra  dewi lestari] -> [Citra Dewi Lestari]
```

Bentuk internasional `+62 812 3456 7890` tetap memuat kode negara 62 dan tidak lolos pemeriksaan. Itu disengaja: program tidak *menebak* maksud pelanggan, ia meminta kasir memeriksanya. Soal Latihan 9.3 memperluas `digits_only` agar mengubah awalan `62` menjadi `0`.

Versi Python, **Kode 9.4**, memakai metode string `isalpha`, `isdigit`, dan `isspace`, yang berperilaku seperti sepupu C-nya untuk teks ASCII, dan membangun string baru alih-alih mengisi larik. `tidy_name` menjadi satu baris: `src.split()` tanpa argumen memisahkan pada setiap deretan spasi putih dan membuang potongan kosong, persis pemangkasan dan pemampatan yang dilakukan perulangan C secara manual.

Kode 9.4 `char_classes.py`: tiga tugas yang sama dengan metode string Python.

```
# char_classes.py - classify characters, clean a phone number and tidy a
# customer name the way the Profitina Laundry counter receives them.


def classify(s):
    """Count letters, digits, spaces and everything else in s."""
    letters = digits = spaces = others = 0
    for c in s:
        if c.isalpha():
            letters += 1
        elif c.isdigit():
            digits += 1
        elif c.isspace():
            spaces += 1
        else:
            others += 1
    print('"%s": letters=%d digits=%d spaces=%d others=%d'
          % (s, letters, digits, spaces, others))


def digits_only(src, size=32):
    """Keep only the digits (at most size - 1 of them, as in the C buffer)."""
    return "".join(c for c in src if c.isdigit())[:size - 1]


def valid_mobile(digits):
    """An Indonesian mobile number: starts with 08 and has 10 to 13 digits."""
    return 10 <= len(digits) <= 13 and digits.startswith("08")


def tidy_name(src, size=32):
    """Trim, squeeze inner spaces to one, and write each word as Title case."""
    words = [w[0].upper() + w[1:].lower() for w in src.split()]
    return " ".join(words)[:size - 1]


def main():
    phones = ["0812-3456-7890", "+62 812 3456 7890", "0812 34"]
    names = ["  aNDi   pratama ", "BUDI", "citra  dewi lestari"]
```

Kode 9.4 (Lanjutan) *char_classes.py: tiga tugas yang sama dengan metode string Python.*

```
    classify("Order PL-0042: 3.5 kg!")
    for p in phones:
        d = digits_only(p)
        verdict = "ok" if valid_mobile(d) else "check with customer"
        print("phone %-18s -> %-13s (%d digits) %s" % (p, d, len(d), verdict))
    for n in names:
        print("name  [%s] -> [%s]" % (n, tidy_name(n)))


if __name__ == "__main__":
    main()
```

9.4 Pustaka String dan Sisi Tajamnya

Header `<string.h>` menyediakan fungsi siap pakai untuk pekerjaan paling umum pada string utuh: mengukur, menyalin, menyambung, membandingkan, dan mencari [2], [3]. **Tabel 9.2** mendaftar fungsi yang paling sering Anda pakai, masing-masing dengan aturan yang menjaga keamanannya. Aturan itu ada karena tidak satu pun fungsi ini memeriksa apa-apa: semuanya percaya bahwa larik yang Anda berikan cukup besar dan setiap string diakhiri penutupnya.

Tabel 9.2 Fungsi sehari-hari `<string.h>` (dan `snprintf` dari `<stdio.h>`), masing-masing dengan aturan keamanannya.

Fungsi	Kegunaan	Aturan yang menjaga keamanannya
`strlen(s)`	banyaknya karakter sebelum `'\0'`	`s` harus berpenutup, kalau tidak penelusuran terus berjalan di memori
`strcpy(dst, src)`	menyalin `src`, termasuk penutup, ke `dst`	`dst` harus memuat `strlen(src) + 1` byte; utamakan `snprintf`
`strcat(dst, src)`	menyambungkan `src` ke ujung string di `dst`	`dst` harus memuat kedua string ditambah satu byte
`snprintf(dst, size, fmt, ...)`	memformat ke `dst`, tidak pernah menulis lebih dari `size` byte	berikan ukuran sebenarnya; nilai kembaliannya memberi tahu panjang teks utuh
`strcmp(a, b)`	negatif, nol, atau positif menurut kode karakter	nol berarti *sama*; jangan pernah membandingkan string dengan `==`
`strchr(s, c)`	pointer ke `c` pertama di `s`, atau `NULL`	periksa `NULL` sebelum memakai hasilnya
`strstr(s, part)`	pointer ke kemunculan pertama `part`, atau `NULL`	periksa `NULL` sebelum memakai hasilnya
`strtok(s, delims)`	memotong `s` menjadi potongan di karakter pemisah	ia *menulis* `'\0'` ke `s`; melewati ruas kosong; satu string pada satu waktu

Kode 9.5 mencoba masing-masing pada teks laundry.

Kode 9.5 `string_library.c`: menyalin, menyambung, memformat berbatas, membandingkan, mencari, dan memotong string.

```c
/* string_library.c - the everyday functions of <string.h>, each with the one
 * rule that keeps it safe. */
#include <stdio.h>
#include <string.h>

static int sign(int x) { return (x > 0) - (x < 0); }

int main(void)
{
    char dst[24];

    strcpy(dst, "Wash");                 /* copy: dst must be big enough */
    strcat(dst, " & fold");              /* append: same rule */
    printf("strcpy + strcat : \"%s\" (%zu of %zu bytes used)\n",
           dst, strlen(dst) + 1, sizeof dst);

    char wish[48];                       /* a longer text, built at run time */
    strcpy(wish, dst);
    strcat(wish, " plus ironing and delivery");
    int need = snprintf(dst, sizeof dst, "%s", wish);   /* never overflows */
    printf("snprintf        : \"%s\" (wanted %d chars, kept %zu)\n",
           dst, need, strlen(dst));

    printf("strcmp(\"Andi\", \"Andi\")  -> %d\n", sign(strcmp("Andi", "Andi")));
    printf("strcmp(\"Andi\", \"Budi\")  -> %d\n", sign(strcmp("Andi", "Budi")));
    printf("strcmp(\"andi\", \"Andi\")  -> %d\n", sign(strcmp("andi", "Andi")));
    printf("strcmp(\"Andi\", \"Andika\")-> %d\n", sign(strcmp("Andi", "Andika")));

    const char *code = "PL-0042";
    const char *dash = strchr(code, '-');          /* first '-' or NULL */
    printf("strchr          : number part \"%s\" at index %d\n",
           dash + 1, (int)(dash - code));

    const char *menu = "fold,wash-iron,iron";
    printf("strstr          : \"wash-iron\" %s, \"dry-clean\" %s\n",
           strstr(menu, "wash-iron") ? "found" : "missing",
           strstr(menu, "dry-clean") ? "found" : "missing");

    char row[] = "Andi Pratama,fold,3.5";      /* strtok edits this array */
    int field = 0;
    for (char *tok = strtok(row, ","); tok != NULL; tok = strtok(NULL, ","))
        printf("strtok field %d  : \"%s\"\n", ++field, tok);
    printf("row afterwards  : \"%s\" (the commas became '\\0')\n", row);
    return 0;
}
```

Hasilnya:

```
strcpy + strcat : "Wash & fold" (12 of 24 bytes used)
snprintf        : "Wash & fold plus ironin" (wanted 37 chars, kept 23)
strcmp("Andi", "Andi")  -> 0
strcmp("Andi", "Budi")  -> -1
strcmp("andi", "Andi")  -> 1
strcmp("Andi", "Andika")-> -1
strchr          : number part "0042" at index 2
strstr          : "wash-iron" found, "dry-clean" missing
strtok field 1  : "Andi Pratama"
strtok field 2  : "fold"
strtok field 3  : "3.5"
row afterwards  : "Andi Pratama" (the commas became '\0')
```

Mari telusuri hasilnya berurutan.

Menyalin dan menyambung. Setelah `strcpy` dan `strcat`, `dst` berisi `Wash & fold`: 11 karakter ditambah penutup, 12 dari 24 byte-nya. Itu aman hanya karena kita menghitung. Baris berikutnya menunjukkan alat yang membuat penghitungan tak diperlukan: `snprintf(dst, sizeof dst,`

`"%s", wish)` memformat teks 37 karakter tetapi tidak pernah menulis lebih dari `sizeof dst`, yaitu 24, byte. Ia menyimpan 23 karakter pertama dan penutup, lalu mengembalikan 37, panjang yang *seharusnya* dimiliki teks utuh, sehingga program dapat mengetahui ada yang terpotong. Catatan pelanggan yang terpotong itu menjengkelkan; menulis 37 byte ke larik 24 byte adalah bencana, seperti dijelaskan peringatan di bawah.

Membandingkan. `strcmp(a, b)` membandingkan kedua string karakter demi karakter dari kiri dan berhenti di posisi pertama tempat keduanya berbeda. Bila karakter `a` di situ berkode lebih kecil, hasilnya negatif; bila lebih besar, positif; bila kedua string sama sampai habis, nol. Standar hanya menjanjikan *tanda* hasilnya, bukan besarnya, karena itu program mencetak tandanya saja. `"Andi"` sebelum `"Budi"` bernilai negatif karena 65 lebih kecil daripada 66. `"andi"` sesudah `"Andi"` bernilai positif karena huruf kecil berkode lebih besar daripada kapital: bagi komputer, `z` datang sesudah `Z`. Dan `"Andi"` sebelum `"Andika"` bernilai negatif karena ketika satu string habis lebih dulu, penutupnya yang bernilai 0 dibandingkan dengan `k`, dan 0 lebih kecil: awalan datang lebih dulu, seperti dalam kamus.

Mencari. `strchr(code, '-')` mengembalikan *pointer* ke tanda hubung di dalam `"PL-0042"`, bukan indeks; mengurangkan awal string, `dash - code`, mengubahnya menjadi indeks 2 (aritmetika pointer dari Bab 7), dan `dash + 1` adalah pointer ke sisa teks, `"0042"`, yang dicetak `printf` seperti string lain. `strstr` mencari sepotong teks utuh. Keduanya mengembalikan `NULL` bila tidak menemukan apa-apa, dan program memeriksanya sebelum memakai hasilnya.

Memotong. `strtok` menemukan koma pertama, menulis penutup di atasnya, dan mengembalikan pointer ke potongan pertama; setiap panggilan berikutnya dengan `NULL` melanjutkan dari tempat panggilan sebelumnya berhenti. Baris terakhir keluaran menunjukkan harganya: larik `row` telah berubah, koma-komanya kini penutup, dan mencetaknya hanya memperlihatkan ruas pertama. Karena itulah `strtok` tidak boleh dipakai pada literal string, dan karena itu pula Bab 11 menulis pemisah ruasnya sendiri untuk berkas data, yang ruas kosongnya seperti `Andi,,3.5` harus dipertahankan padahal `strtok` diam-diam melewatinya.

> **PERINGATAN:** Menyalin lebih banyak byte ke larik daripada yang dapat ditampungnya tidak menghentikan program dengan pesan galat. Byte kelebihan ditulis di atas apa pun yang terletak berikutnya di memori: variabel lain, alamat kembali fungsi, data pelanggan lain. *Buffer overflow* (luapan penyangga) ini telah menjadi salah satu kesalahan pemrograman yang paling banyak dieksploitasi selama puluhan tahun. Dalam daftar 25 kelemahan perangkat lunak paling berbahaya tahun 2025 yang diterbitkan MITRE, *out-of-bounds write* (menulis di luar batas) masih di urutan kelima dan *classic buffer overflow*, penyalinan tanpa memeriksa ukuran masukan, di urutan kesebelas [5]. Aturan buku ini: setiap penyalinan ke larik menyebutkan ukuran larik itu, lewat pemeriksaan ukuran dalam perulangan Anda sendiri atau lewat `snprintf`.

Versi Python, **Kode 9.6**, menunjukkan gagasan yang sama dengan operator Python. String disambung dengan `+`, dibandingkan dengan `<`, `==`, dan `>` (perbandingannya menurut kode karakter, persis seperti `strcmp`), dicari dengan `find` dan `in`, dan dipotong dengan `split`. Python tidak mungkin meluapkan string, karena string baru yang lebih besar dibuat bila perlu; untuk mencetak baris “kept 23” yang sama, program memotong teks menjadi 23 karakter sendiri.

Kode 9.6 `string_library.py`: padanan fungsi-fungsi `<string.h>` dalam Python.

```
# string_library.py - the Python counterparts of the <string.h> functions,
# printing the same lines as string_library.c.


def sign(x):
    return (x > 0) - (x < 0)


def cmp(a, b):
    """strcmp: negative, zero or positive by character codes."""
    return (a > b) - (a < b)


def main():
    size = 24
    dst = "Wash"
    dst += " & fold"
    print('strcpy + strcat : "%s" (%d of %d bytes used)'
          % (dst, len(dst) + 1, size))

    full = dst + " plus ironing and delivery"     # a longer text
    dst = full[:size - 1]                 # what snprintf keeps
    print('snprintf        : "%s" (wanted %d chars, kept %d)'
          % (dst, len(full), len(dst)))

    print('strcmp("Andi", "Andi")  -> %d' % sign(cmp("Andi", "Andi")))
    print('strcmp("Andi", "Budi")  -> %d' % sign(cmp("Andi", "Budi")))
    print('strcmp("andi", "Andi")  -> %d' % sign(cmp("andi", "Andi")))
    print('strcmp("Andi", "Andika")-> %d' % sign(cmp("Andi", "Andika")))

    code = "PL-0042"
    i = code.find("-")                    # -1 plays the role of NULL
    print('strchr          : number part "%s" at index %d' % (code[i + 1:], i))

    menu = "fold,wash-iron,iron"
    print('strstr          : "wash-iron" %s, "dry-clean" %s'
          % ("found" if "wash-iron" in menu else "missing",
             "found" if "dry-clean" in menu else "missing"))

    row = "Andi Pratama,fold,3.5"
    for field, tok in enumerate(row.split(","), start=1):
        print('strtok field %d  : "%s"' % (field, tok))
    print('row afterwards  : "%s" (the commas became \'\\0\')' % row.split(",")[0])

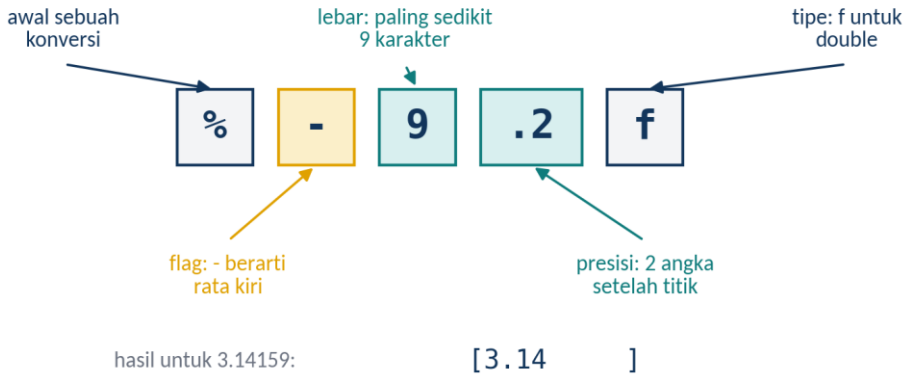

```

Kode 9.6 (Lanjutan) *string_library.py: padanan fungsi-fungsi <string.h> dalam Python.*

```
if __name__ == "__main__":
    main()
```

9.5 Keluaran Terformat: Seluruh Tombol Pengatur `printf`

Anda sudah memakai `%d`, `%f`, dan `%s` sejak Bab 1. Masing-masing adalah huruf terakhir sebuah *spesifikasi konversi*, dan sebuah spesifikasi boleh membawa sampai empat bagian lagi di antara `%` dan huruf itu. **Gambar 9.2** membedah satu contoh.



Gambar 9.2 Bagian-bagian spesifikasi konversi `%-9.2f`: *flag* `-` meratakan kiri, lebar 9 adalah banyak karakter minimum, presisi 2 memberi angka di belakang titik desimal, dan tipe `f` menyatakan bahwa yang dicetak adalah `double`.

Secara umum sebuah spesifikasi berbentuk

$$\%\ [\text{flags}]\ [\text{width}]\ [.\text{precision}]\ \text{type} \tag{9.2}$$

dengan bagian dalam kurung siku bersifat opsional [2], [6]. *Lebar* (*width*) adalah minimum: nilai yang memerlukan lebih sedikit karakter diisi spasi (di kiri, kecuali *flag* `-` meminta di kanan), dan nilai yang memerlukan lebih banyak tidak pernah dipotong, ia hanya mendorong sisa baris. *Presisi* bermakna berbeda untuk tipe berbeda: banyak angka di belakang titik untuk `f` dan `e`, banyak angka bermakna untuk `g`, dan banyak karakter *maksimum* untuk `s`. *Flag*-nya adalah `-` (rata kiri), `0` (isi bilangan dengan nol), `+` (selalu cetak tanda), spasi (spasi di tempat tanda plus), dan `#` ("bentuk alternatif", misalnya `0x` di depan heksadesimal). Lebar atau presisi yang ditulis `*` diambil dari daftar argumen, sehingga program dapat menghitung lebar kolom saat berjalan.

Kode 9.7 memutar tombol-tombol Persamaan (9.2) satu per satu. Setiap ruas dicetak di antara kurung siku, sehingga Anda dapat melihat persis di mana ruas itu mulai dan berakhir, termasuk spasinya.

Kode 9.7 `format_dial.c`: *flag*, lebar, presisi, dan tipe `printf`, satu per satu.

```c
/* format_dial.c - the conversion specification %[flags][width][.precision]type
 * of printf, turned one knob at a time. Brackets show where each field ends. */
#include <stdio.h>

int main(void)
{
    printf("[%d] [%6d] [%-6d] [%06d] [%+d]\n", 42, 42, 42, 42, 42);
    printf("[%f] [%.2f] [%9.2f] [%-9.2f]\n", 3.14159, 3.14159, 3.14159, 3.14159);
    printf("[%.2f] [%.2f] [%.1f]\n", 2.675, 1.005, 0.25);
    printf("[%e] [%.3e] [%g] [%g]\n", 1234.5, 0.000123, 1234.5, 0.0000125);
    printf("[%s] [%10s] [%-10s] [%.3s]\n", "Andi", "Andi", "Andi", "Andika");
    printf("[%c] [%3c] [%%]\n", 'A', 'B');
    printf("[%x] [%X] [%o] [%#x]\n", 255, 255, 8, 255);
    printf("[%*d] [%-*s]\n", 8, 42, 8, "left");
    int n = printf("Rp %d\n", 28000);
    printf("the line above had %d characters, newline included\n", n);
    return 0;
}
```

```
[42] [    42] [42    ] [000042] [+42]
[3.141590] [3.14] [     3.14] [3.14     ]
[2.67] [1.00] [0.2]
[1.234500e+03] [1.230e-04] [1234.5] [1.25e-05]
[Andi] [      Andi] [Andi      ] [And]
[A] [  B] [%]
[ff] [FF] [10] [0xff]
[      42] [left    ]
Rp 28000
the line above had 9 characters, newline included
```

Sebagian besar baris adalah tombol-tombol yang bekerja sesuai **Gambar 9.2**. Tiga di antaranya perlu dikomentari.

Baris ketiga adalah pelajaran tentang uang. `2.675` dicetak dengan dua desimal menghasilkan `2.67`, bukan `2.68`, dan `1.005` menghasilkan `1.00`. `printf` tidak salah: bilangan 2,675 dan 1,005 tidak dapat disimpan tepat dalam biner, dan nilai `double` terdekatnya adalah 2.67499999999999982236431605997495353221893310546875 dan 1.00499999999999989341858963598497211933135986328125, yang memang dibulatkan ke bawah [7]. `0.25` *memang* tersimpan tepat, dan di situ aturan "bulatkan setengah ke genap" menghasilkan `0.2`. Inilah sebabnya setiap program dalam buku ini menyimpan rupiah dalam bilangan bulat bertipe `int` atau `long` dan berat dalam persepuluhan kilogram utuh: bilangan bulat itu tepat, dan kasir tidak boleh kehilangan satu rupiah pun karena pembulatan.

Tipe `%g` memilih sendiri antara notasi biasa dan ilmiah, mana yang lebih pendek, dan membuang nol di belakang. Dan nilai kembalian `printf` adalah banyaknya karakter yang ditulisnya; program memakainya untuk menunjukkan bahwa `"Rp 28000\n"` terdiri atas 9 karakter, termasuk baris baru.

Di Python, operator `%` pada string memahami spesifikasi konversi yang sama, sehingga **Kode 9.8** nyaris berupa salinan. Metode `format` dan *f-string* Python memakai bahasa-mini yang berbeda (`{:<9.2f}`) dengan gagasan yang sama; buku ini memakai `%` karena persis sama dengan C.

Kode 9.8 `format_dial.py`: spesifikasi yang sama dengan operator `%` Python.

```python
# format_dial.py - Python's % operator uses the same conversion specifications
# as printf, so every line below matches format_dial.c.


def main():
    print("[%d] [%6d] [%-6d] [%06d] [%+d]" % (42, 42, 42, 42, 42))
    print("[%f] [%.2f] [%9.2f] [%-9.2f]" % (3.14159, 3.14159, 3.14159, 3.14159))
    print("[%.2f] [%.2f] [%.1f]" % (2.675, 1.005, 0.25))
    print("[%e] [%.3e] [%g] [%g]" % (1234.5, 0.000123, 1234.5, 0.0000125))
    print("[%s] [%10s] [%-10s] [%.3s]" % ("Andi", "Andi", "Andi", "Andika"))
    print("[%c] [%3c] [%%]" % ("A", "B"))
    print("[%x] [%X] [%o] [%#x]" % (255, 255, 8, 255))
    print("[%*d] [%-*s]" % (8, 42, 8, "left"))
    line = "Rp %d\n" % 28000
    print(line, end="")
    print("the line above had %d characters, newline included" % len(line))


if __name__ == "__main__":
    main()
```

9.6 Struk yang Kolomnya Rata

Sekarang kita bisa memberi Mbak Rina struk yang kolomnya tidak pernah berkelok. **Kode 9.9** mencetak satu struk untuk pesanan tiga layanan. Program ini memperkenalkan dua fungsi pembantu yang murni pekerjaan string dan terus dipakai di bagian buku selanjutnya.

`rupiah` menulis bilangan dengan titik di antara kelompok tiga angka, cara Indonesia: 1250000 menjadi `1.250.000`. Ia lebih dulu membiarkan `snprintf` mengubah bilangan menjadi angka-angkanya, lalu menyalin angka-angka itu ke hasil sambil menyisipkan titik sebelum angka ke-`i` setiap kali banyaknya angka yang masih akan datang, `len - i`, merupakan kelipatan tiga (dan `i` bukan 0). Untuk `51500`, `len` bernilai 5: sebelum indeks 2 masih tersisa tiga angka, jadi titik diletakkan di situ, menghasilkan `51.500`.

`centered` meletakkan judul di tengah baris berbingkai. Ia menghitung berapa spasi di kiri dan di kanan lalu mencetaknya dengan lebar `*`: `printf("|%*s%s%*s|\n", left, "", text, right, "")` mencetak string kosong `""` dalam ruas selebar `left` karakter, yang tak lain adalah `left` buah spasi.

Berat disimpan dalam persepuluhan kilogram sebagai `int`: 3,5 kg disimpan sebagai 35 dan dicetak dengan `%d.%d` dari `tenths / 10` dan `tenths % 10`. Jumlah untuk satu baris adalah `tenths * price / 10`; untuk 3,5 kg cuci lipat dengan harga Rp7.000 per kilogram, itu $35 \times 7000/10 = 24500$, dihitung tepat. Harga-harganya mengikuti daftar harga Bab 5: Rp7.000 per kilogram untuk cuci lipat, Rp10.000 untuk cuci setrika, Rp6.000 untuk setrika saja.

Kode 9.9 `receipt.c`: struk yang ditata dengan lebar ruas, pemusatan `%*s`, dan pemisah ribuan.

```c
/* receipt.c - a Profitina Laundry receipt laid out with printf widths. */
#include <stdio.h>
#include <string.h>

#define WIDTH 44                    /* characters per receipt line */

/* Write n with dots between thousands: 1250000 -> "1.250.000".
 * out must hold at least 32 bytes. */
static void rupiah(long n, char *out)
{
    char digits[24];
    int len = snprintf(digits, sizeof digits, "%ld", n < 0 ? -n : n);
    int k = 0;
    if (n < 0)
        out[k++] = '-';
    for (int i = 0; i < len; i++) {
        if (i > 0 && (len - i) % 3 == 0)
            out[k++] = '.';         /* a dot before each group of three */
        out[k++] = digits[i];
    }
    out[k] = '\0';
}

/* "081234567890" -> "0812-3456-7890". out must hold 24 bytes. */
static void phone_groups(const char *d, char *out)
{
    int k = 0;
    for (int i = 0; d[i] != '\0' && k < 22; i++) {
        if (i > 0 && i % 4 == 0)
            out[k++] = '-';
        out[k++] = d[i];
    }
    out[k] = '\0';
}

static void rule(char edge, char fill)
{
    putchar(edge);
    for (int i = 0; i < WIDTH - 2; i++)
        putchar(fill);
    printf("%c\n", edge);
}
```

Kode 9.9 (Lanjutan) *receipt.c: struk yang ditata dengan lebar ruas, pemusatan %*s, dan pemisah ribuan.*

```c
/* Print text in the middle of a boxed line: %*s pads with a computed width. */
static void centered(const char *text)
{
    int len = (int)strlen(text);
    int left = (WIDTH - 2 - len) / 2;
    int right = WIDTH - 2 - len - left;
    printf("|%*s%s%*s|\n", left, "", text, right, "");
}

static void money_line(const char *label, long amount)
{
    char m[32];
    rupiah(amount, m);
    printf("%-30s Rp %10s\n", label, m);
}

int main(void)
{
    const char *service[] = {"Wash & fold", "Iron only", "Wash & iron"};
    const int tenths[] = {35, 20, 15};          /* 3.5 kg, 2.0 kg, 1.5 kg */
    const int price[] = {7000, 6000, 10000};    /* rupiah per kg */
    const int items = 3;
    char phone[24], a[32], b[32];
    long subtotal = 0;

    rule('+', '-');
    centered("PROFITINA LAUNDRY");
    centered("Jl. Kenanga 12, Surabaya");
    rule('+', '-');
    printf("Order    : PL-%06d\n", 42);
    printf("Customer : %s\n", "Andi Pratama");
    phone_groups("081234567890", phone);
    printf("Phone    : %s\n\n", phone);

    printf("%-15s %8s %8s %10s\n", "Service", "Weight", "Price", "Amount");
    for (int i = 0; i < items; i++) {
        long amount = (long)tenths[i] * price[i] / 10;
        rupiah(price[i], a);
        rupiah(amount, b);
        printf("%-15s %3d.%d kg %8s %10s\n",
               service[i], tenths[i] / 10, tenths[i] % 10, a, b);
        subtotal += amount;
    }
    rule('-', '-');
    long discount = subtotal * 10 / 100;
    long total = subtotal - discount;
```

Kode 9.9 (Lanjutan) *receipt.c: struk yang ditata dengan lebar ruas, pemusatan %*s, dan pemisah ribuan.*

```c
    money_line("Subtotal", subtotal);
    money_line("Member discount 10%", -discount);
    money_line("TOTAL", total);
    money_line("Paid", 100000);
    money_line("Change", 100000 - total);
    rule('+', '-');
    centered("Thank you - ready Friday after 16.00");
    rule('+', '-');
    return 0;
}
```

```
+----------------------------------------+
|            PROFITINA LAUNDRY           |
|         Jl. Kenanga 12, Surabaya       |
+----------------------------------------+
Order    : PL-000042
Customer : Andi Pratama
Phone    : 0812-3456-7890

Service           Weight    Price     Amount
Wash & fold       3.5 kg    7.000     24.500
Iron only         2.0 kg    6.000     12.000
Wash & iron       1.5 kg   10.000     15.000
----------------------------------------
Subtotal                      Rp     51.500
Member discount 10%           Rp     -5.150
TOTAL                         Rp     46.350
Paid                          Rp    100.000
Change                        Rp     53.650
+----------------------------------------+
|   Thank you - ready Friday after 16.00   |
+----------------------------------------+
```

Setiap kolom rata karena setiap nilai dicetak dalam ruas berlebar tetap: nama layanan rata kiri dalam 15 karakter, berat rata kanan, jumlah rata kanan dalam 10. Cobalah mengganti `"Andi Pratama"` dengan nama yang jauh lebih panjang: baris pelanggan memanjang, tetapi tabelnya tidak bergeser, karena lebar-lebarnya tidak bergantung pada nama. Versi Python, **Kode 9.10**, membangun string yang sama dengan `%` dan dengan `join` untuk kelompok nomor telepon.

Kode 9.10 `receipt.py`: struk yang sama dalam Python.

```python
# receipt.py - the same Profitina Laundry receipt, laid out with % widths.

WIDTH = 44


def rupiah(n):
    """1250000 -> '1.250.000' (a dot before each group of three digits)."""
    digits = str(abs(n))
    out = "-" if n < 0 else ""
    for i, d in enumerate(digits):
        if i > 0 and (len(digits) - i) % 3 == 0:
            out += "."
        out += d
    return out


def phone_groups(d):
    """'081234567890' -> '0812-3456-7890'."""
    return "-".join(d[i:i + 4] for i in range(0, len(d), 4))


def rule(edge, fill):
    print(edge + fill * (WIDTH - 2) + edge)


def centered(text):
    left = (WIDTH - 2 - len(text)) // 2
    right = WIDTH - 2 - len(text) - left
    print("|%*s%s%*s|" % (left, "", text, right, ""))


def money_line(label, amount):
    print("%-30s Rp %10s" % (label, rupiah(amount)))


def main():
    service = ["Wash & fold", "Iron only", "Wash & iron"]
    tenths = [35, 20, 15]
    price = [7000, 6000, 10000]
    subtotal = 0
```

Kode 9.10 (Lanjutan) *receipt.py: struk yang sama dalam Python.*

```
    rule("+", "-")
    centered("PROFITINA LAUNDRY")
    centered("Jl. Kenanga 12, Surabaya")
    rule("+", "-")
    print("Order    : PL-%06d" % 42)
    print("Customer : %s" % "Andi Pratama")
    print("Phone    : %s\n" % phone_groups("081234567890"))

    print("%-15s %8s %8s %10s" % ("Service", "Weight", "Price", "Amount"))
    for i in range(len(service)):
        amount = tenths[i] * price[i] // 10
        print("%-15s %3d.%d kg %8s %10s" % (service[i], tenths[i] // 10,
              tenths[i] % 10, rupiah(price[i]), rupiah(amount)))
        subtotal += amount
    rule("-", "-")
    discount = subtotal * 10 // 100
    total = subtotal - discount
    money_line("Subtotal", subtotal)
    money_line("Member discount 10%", -discount)
    money_line("TOTAL", total)
    money_line("Paid", 100000)
    money_line("Change", 100000 - total)
    rule("+", "-")
    centered("Thank you - ready Friday after 16.00")
    rule("+", "-")


if __name__ == "__main__":
    main()
```

9.7 Membaca Teks secara Aman

Keluaran adalah separuh yang mudah. Masukanlah tempat program teks biasanya rusak, karena program tidak dapat memilih apa yang diketik orang. Bab 1 membaca bilangan dengan `scanf("%d", ...)`, dan `scanf` juga dapat membaca kata dengan `%s`. Tetapi `%s` punya dua masalah. Ia berhenti di spasi pertama, sehingga `Andi Pratama` datang sebagai dua kata terpisah. Dan `%s` polos tidak tahu seberapa besar larik tujuannya: pelanggan yang mengetik kata 40 huruf ke larik 32 byte menimbulkan luapan persis seperti di Subbab 9.4. Lebar ruas mengatasi masalah kedua (`%31s` membaca paling banyak 31 karakter ke larik 32 byte), dan *scan set* `%[^\n]`, "semua karakter kecuali baris baru", mengatasi yang pertama; `scanf(" %63[^\n]", line)` membaca satu baris utuh paling panjang 63 karakter dengan melewati spasi putih di depannya. Anda akan menemui keduanya di Bab 11.

Untuk perintah yang diketik, cara yang lebih tangguh adalah memisahkan *membaca* dari *memahami*:

1. Baca satu baris utuh ke larik dengan `fgets(buf, size, stdin)`. `fgets` tidak pernah menulis lebih dari `size` byte: ia berhenti setelah baris baru, atau ketika `size - 1` karakter sudah terbaca, dan selalu menambahkan penutup. Ia mengembalikan `NULL` di akhir masukan.
2. Periksa apa yang datang. Bila karakter terakhir adalah `'\n'`, baris utuh muat; buang baris barunya. Bila tidak ada baris baru dan larik penuh, baris itu terlalu panjang: sisanya masih menunggu di masukan dan harus dibaca lalu dibuang, kalau tidak ia akan dianggap perintah berikutnya. Berkas yang ditulis di Windows mengakhiri barisnya dengan `"\r\n"`, jadi `'\r'` di ujung juga dibuang.
3. Baru kemudian potong baris menjadi kata-kata dan tentukan artinya.

Kasir v0.7 mengemas langkah 1 dan 2 dalam fungsi `read_line` yang mengembalikan 1 untuk baris yang baik, 0 di akhir masukan, dan −1 untuk baris yang terlalu panjang. Dengan larik 80 byte, sebuah baris boleh berisi sampai 79 karakter: `fgets` membaca 79 karakter, mendapati larik penuh tanpa baris baru, dan mengintip karakter berikutnya dengan `getchar()`; bila itu baris baru, barisnya pas, kalau bukan, baris itu terlalu panjang.

Untuk langkah 3, kasir memakai pembantu `next_word(&p, out, size)` yang melewati spasi, menyalin satu kata ke `out` (memotongnya menjadi `size - 1` karakter, tetapi selalu *menghabiskan* seluruh kata), dan memindahkan pointer `p` melewati kata itu. Parameter pointer-ke-pointer `const char **p` adalah teknik Bab 7 untuk fungsi yang harus mengubah pointer milik pemanggil: setelah `next_word(&rest, cmd, sizeof cmd)`, `rest` milik pemanggil menunjuk tepat sesudah kata perintah, siap untuk panggilan berikutnya. Apa pun yang tersisa sesudah kata terakhir adalah nama pelanggan, yang boleh berisi spasi dan diproses oleh `tidy_name`.

Mengapa tidak memakai `sscanf` dengan `%15s` untuk kata-katanya? `sscanf` baik untuk ruas yang bentuknya diketahui, dan Bab 12 memakainya untuk bilangan, tetapi `%15s` pada kata 20 huruf membaca 15 huruf dan meninggalkan 5 lainnya untuk dianggap sebagai kata *berikutnya*, yang diam-diam mengubah arti baris. `next_word` memotong kata yang terlalu panjang tetapi tidak pernah membelahnya menjadi dua.

9.8 Membangun Kasir v0.7

Kini kita memiliki semua bagiannya; **Kode 9.11** dan **Kode 9.12** menyatukannya. Kasir v0.7 adalah penerjemah perintah kecil, sebuah *read–evaluate–print loop* (perulangan baca–evaluasi–cetak): ia mencetak *prompt* `>`, membaca sebaris, menjalankan perintahnya, mencetak hasilnya, lalu mulai lagi. Perintah-perintahnya:

- `add <service> <kg> <name>` mencatat pesanan; layanannya `fold` (cuci lipat), `wash-iron` (cuci setrika), atau `iron` (setrika saja), beratnya ditulis seperti `3` atau `3.5`, dan namanya boleh terdiri atas beberapa kata dengan huruf besar-kecil dan spasi sembarang.
- `find <part of a name>` mendaftar setiap pesanan yang nama pelanggannya memuat teks yang diberikan, tanpa memedulikan huruf besar-kecil.
- `list` mencetak pesanan hari itu dalam tabel beserta totalnya.
- `help` menampilkan perintah-perintah, dan `quit` menutup hari.

Pesanan disimpan seperti di Bab 6, dalam larik: larik dua dimensi `names[MAX_ORDERS][NAME_LEN]`, yaitu 50 baris masing-masing 32 karakter, satu baris per nama pelanggan, dan dua larik `int` untuk layanan dan berat dalam persepuluhan. `names[i]` adalah baris `i`, larik `char` biasa, sehingga setiap fungsi string bekerja padanya. Menyimpan tiga larik berdampingan, yang disebut *larik paralel*, memang berfungsi tetapi rapuh; Bab 10 menggantinya dengan satu larik struktur.

Tiga pembantu lagi melengkapi program. `parse_tenths` mengubah teks `3.5` menjadi bilangan 35 tanpa bilangan pecahan sama sekali: ia menerima satu sampai tiga angka, boleh diikuti titik dan tepat satu angka, dan tidak ada yang lain; `3.25`, `.5`, `3.`, dan `3kg` ditolak. `contains_ci` mencari satu string di dalam string lain tanpa memedulikan huruf besar-kecil, dengan membandingkan `tolower` kedua karakter di setiap posisi; pustaka standar tidak punya fungsi seperti itu, dan menulisnya hanya perlu sebelas baris. `rupiah` adalah pembantu dari Subbab 9.6.

Terakhir, opsi baris perintah `--echo`. Ketika Anda mengetik perintah sendiri, terminal menampilkan apa yang Anda ketik. Ketika perintah datang dari berkas (`./cashier_v7 <`

`session.txt`), tidak ada yang mengetik, dan keluaran hanya akan berisi jawaban. Dengan `--echo`, program mengulang setiap perintah sesudah *prompt*, sehingga sesi yang direkam terbaca seperti percakapan. Keluaran dalam buku ini dibuat dengan cara itu.

Kode 9.11 `cashier_v7.c`: kasir Profitina Laundry v0.7, penerjemah perintah dengan nama yang rapi dan pencarian dari sebagian nama.

```c
/* cashier_v7.c - Profitina Laundry cashier v0.7: orders are typed as text
 * commands, names are tidied, and a customer is found by part of a name.
 * Usage: cashier_v7 [--echo]    (--echo repeats each command after "> ") */
#include <ctype.h>
#include <stdio.h>
#include <string.h>

#define MAX_ORDERS 50
#define NAME_LEN 32
#define LINE_LEN 80
#define N_SERVICES 3

static const char *SERVICE[N_SERVICES] = {"fold", "wash-iron", "iron"};
static const int PRICE[N_SERVICES] = {7000, 10000, 6000};

static char names[MAX_ORDERS][NAME_LEN];   /* one row of chars per customer */
static int service_of[MAX_ORDERS];
static int tenths_of[MAX_ORDERS];          /* weight in tenths of a kg */
static int count = 0;

/* 1 = a line was read, 0 = end of input, -1 = line too long (skipped). */
static int read_line(char *buf, int size)
{
    if (fgets(buf, size, stdin) == NULL)
        return 0;
    size_t n = strlen(buf);
    if (n > 0 && buf[n - 1] == '\n') {
        buf[--n] = '\0';
    } else if (n == (size_t)size - 1) {    /* buffer full: is there more? */
        int c = getchar();
        if (c != EOF && c != '\n') {
            while (c != EOF && c != '\n')  /* throw the rest of the line away */
                c = getchar();
            return -1;
        }
    }
    if (n > 0 && buf[n - 1] == '\r')       /* a file saved on Windows */
        buf[--n] = '\0';
    return 1;
}
```

Kode 9.11 (Lanjutan) *cashier_v7.c: kasir Profitina Laundry v0.7, penerjemah perintah dengan nama yang rapi dan pencarian dari sebagian nama.*

```c
/* Copy the next word of *p into out (cut to size - 1 chars) and move *p
 * past it. Returns the word's full length; 0 means there was no word. */
static size_t next_word(const char **p, char *out, size_t size)
{
    const char *s = *p;
    size_t n = 0;
    while (isspace((unsigned char)*s))
        s++;
    while (*s != '\0' && !isspace((unsigned char)*s)) {
        if (n + 1 < size)
            out[n] = *s;
        n++;
        s++;
    }
    out[n < size ? n : size - 1] = '\0';
    *p = s;
    return n;
}

/* Trim, squeeze inner spaces to one, and write each word as Title case. */
static void tidy_name(const char *src, char *dst, size_t size)
{
    size_t n = 0;
    int new_word = 1;
    for (size_t i = 0; src[i] != '\0' && n + 1 < size; i++) {
        unsigned char c = (unsigned char)src[i];
        if (isspace(c)) {
            new_word = 1;
            continue;
        }
        if (new_word && n > 0 && n + 2 < size)
            dst[n++] = ' ';
        dst[n++] = (char)(new_word ? toupper(c) : tolower(c));
        new_word = 0;
    }
    dst[n] = '\0';
}
```

Kode 9.11 (Lanjutan) *cashier_v7.c: kasir Profitina Laundry v0.7, penerjemah perintah dengan nama yang rapi dan pencarian dari sebagian nama.*

```c
/* "3" -> 30, "3.5" -> 35. Returns 0 if s is not written like that. */
static int parse_tenths(const char *s, int *out)
{
    int whole = 0, digits = 0, tenth = 0;
    for (; isdigit((unsigned char)*s); s++) {
        if (++digits > 3)
            return 0;
        whole = whole * 10 + (*s - '0');
    }
    if (digits == 0)
        return 0;
    if (*s == '.') {
        if (!isdigit((unsigned char)s[1]))
            return 0;
        tenth = s[1] - '0';
        s += 2;
    }
    if (*s != '\0')
        return 0;
    *out = whole * 10 + tenth;
    return 1;
}

/* Does text contain part, ignoring upper and lower case? */
static int contains_ci(const char *text, const char *part)
{
    size_t n = strlen(part);
    for (size_t i = 0; text[i] != '\0'; i++) {
        size_t k = 0;
        while (k < n && text[i + k] != '\0' &&
               tolower((unsigned char)text[i + k]) ==
               tolower((unsigned char)part[k]))
            k++;
        if (k == n)
            return 1;
    }
    return 0;
}
```

Kode 9.11 (Lanjutan) *cashier_v7.c: kasir Profitina Laundry v0.7, penerjemah perintah dengan nama yang rapi dan pencarian dari sebagian nama.*

```c
static void rupiah(long n, char *out)      /* out: 32 bytes */
{
    char digits[24];
    int len = snprintf(digits, sizeof digits, "%ld", n), k = 0;
    for (int i = 0; i < len; i++) {
        if (i > 0 && (len - i) % 3 == 0)
            out[k++] = '.';
        out[k++] = digits[i];
    }
    out[k] = '\0';
}

static long amount_of(int i)
{
    return (long)tenths_of[i] * PRICE[service_of[i]] / 10;
}

static void print_order(int i)
{
    char m[32];
    rupiah(amount_of(i), m);
    printf("  #%d %s, %s, %d.%d kg = Rp %s\n", i + 1, names[i],
           SERVICE[service_of[i]], tenths_of[i] / 10, tenths_of[i] % 10, m);
}

static void cmd_add(const char *rest)
{
    char svc[16], kg[16], name[NAME_LEN];
    int s = -1, t = 0;
    if (next_word(&rest, svc, sizeof svc) == 0 ||
        next_word(&rest, kg, sizeof kg) == 0) {
        puts("  usage: add <service> <kg> <name>");
        return;
    }
    for (int i = 0; i < N_SERVICES; i++)
        if (strcmp(svc, SERVICE[i]) == 0)
            s = i;
    if (s < 0) {
        printf("  unknown service '%s' (fold, wash-iron, iron)\n", svc);
        return;
    }
    if (!parse_tenths(kg, &t) || t < 1 || t > 500) {
        puts("  weight must look like 3 or 3.5 (0.1 to 50.0 kg)");
        return;
    }
    tidy_name(rest, name, sizeof name);
```

Kode 9.11 (Lanjutan) *cashier_v7.c: kasir Profitina Laundry v0.7, penerjemah perintah dengan nama yang rapi dan pencarian dari sebagian nama.*

```
    if (name[0] == '\0') {
        puts("  the customer's name is missing");
        return;
    }
    if (count == MAX_ORDERS) {
        puts("  the order book is full");
        return;
    }
    strcpy(names[count], name);            /* safe: name fits NAME_LEN */
    service_of[count] = s;
    tenths_of[count] = t;
    print_order(count++);
}

static void cmd_find(const char *rest)
{
    char part[NAME_LEN];
    tidy_name(rest, part, sizeof part);
    if (part[0] == '\0') {
        puts("  usage: find <part of a name>");
        return;
    }
    int hits = 0;
    for (int i = 0; i < count; i++)
        if (contains_ci(names[i], part)) {
            print_order(i);
            hits++;
        }
    if (hits == 0)
        printf("  no customer matches '%s'\n", part);
}

static long total_of_day(void)
{
    long total = 0;
    for (int i = 0; i < count; i++)
        total += amount_of(i);
    return total;
}
```

Kode 9.11 (Lanjutan) *cashier_v7.c: kasir Profitina Laundry v0.7, penerjemah perintah dengan nama yang rapi dan pencarian dari sebagian nama.*

```
static void cmd_list(void)
{
    char m[32];
    printf("  %2s  %-20s  %-9s  %7s  %10s\n",
           "No", "Customer", "Service", "Weight", "Amount");
    for (int i = 0; i < count; i++) {
        rupiah(amount_of(i), m);
        printf("  %2d  %-20.20s  %-9s  %2d.%d kg  %10s\n", i + 1, names[i],
               SERVICE[service_of[i]], tenths_of[i] / 10, tenths_of[i] % 10, m);
    }
    rupiah(total_of_day(), m);
    printf("  %d order(s), total Rp %s\n", count, m);
}

static void cmd_help(void)
{
    puts("  add <service> <kg> <name>   service: fold, wash-iron, iron");
    puts("  find <part of a name>       list       help       quit");
}

int main(int argc, char *argv[])
{
    int echo = argc > 1 && strcmp(argv[1], "--echo") == 0;
    char line[LINE_LEN], cmd[16], m[32];

    puts("Profitina Laundry cashier v0.7 - type help for the commands");
    for (;;) {
        printf("> ");
        fflush(stdout);
        int r = read_line(line, sizeof line);
        if (r == 0) {
            printf("\n");
            break;
        }
        if (echo)
            printf(r < 0 ? "%.40s...\n" : "%s\n", line);
        if (r < 0) {
            puts("  line too long (at most 79 characters), ignored");
            continue;
        }
        const char *rest = line;
        if (next_word(&rest, cmd, sizeof cmd) == 0)
            continue;                       /* an empty line */
        if (strcmp(cmd, "add") == 0)
            cmd_add(rest);
        else if (strcmp(cmd, "find") == 0)
```

Kode 9.11 (Lanjutan) *cashier_v7.c: kasir Profitina Laundry v0.7, penerjemah perintah dengan nama yang rapi dan pencarian dari sebagian nama.*

```
            cmd_find(rest);
        else if (strcmp(cmd, "list") == 0)
            cmd_list();
        else if (strcmp(cmd, "help") == 0)
            cmd_help();
        else if (strcmp(cmd, "quit") == 0)
            break;
        else
            printf("  unknown command '%s' - type help\n", cmd);
    }
    rupiah(total_of_day(), m);
    printf("Bye. %d order(s), Rp %s today.\n", count, m);
    return 0;
}
```

Bacalah `main` lebih dulu: isinya hanya perulangan. Setiap perintah disalurkan dengan `strcmp(cmd, "add") == 0`, tidak pernah dengan `cmd == "add"`, yang akan membandingkan dua *alamat* (larik `cmd` dan literalnya) dan selamanya bernilai salah. Kemudian bacalah `cmd_add` dari atas ke bawah: setiap pemeriksaan yang gagal mencetak pesan yang jelas lalu kembali, sehingga pesanan hanya disimpan setelah *semua* pemeriksaan lolos, dan `strcpy` penutupnya aman karena `tidy_name` sudah memastikan `name` muat dalam `NAME_LEN` byte.

Versi Python menyimpan setiap pesanan sebagai daftar kecil `[name, service, tenths]` di dalam satu daftar `orders`, memakai `split(None, 1)` untuk memisahkan perintah dari sisa baris dan `split(None, 2)` untuk mengambil layanan dan berat dari depan argumen, serta memakai `in` pada salinan huruf kecil untuk pencarian.

Kode 9.12 `cashier_v7.py`: kasir yang sama dalam Python.

```
# cashier_v7.py - Profitina Laundry cashier v0.7: orders are typed as text
# commands, names are tidied, and a customer is found by part of a name.
# Usage: python3 cashier_v7.py [--echo]   (--echo repeats each command)
import sys

MAX_ORDERS = 50
NAME_LEN = 32
LINE_LEN = 80
SERVICE = ["fold", "wash-iron", "iron"]
PRICE = [7000, 10000, 6000]

orders = []          # each order is a list [name, service index, tenths]


def read_line():
    """Return (1, line), (0, "") at end of input, or (-1, start) if too long."""
    line = sys.stdin.readline()
    if line == "":
        return 0, ""
    line = line[:-1] if line.endswith("\n") else line
    if len(line) > LINE_LEN - 1:
        return -1, line[:LINE_LEN - 1]
    return 1, line[:-1] if line.endswith("\r") else line


def tidy_name(src, size=NAME_LEN):
    words = [w[0].upper() + w[1:].lower() for w in src.split()]
    return " ".join(words)[:size - 1]


def parse_tenths(s):
    """'3' -> 30, '3.5' -> 35, anything else -> None."""
    whole, dot, tenth = s.partition(".")
    if not (whole.isdigit() and len(whole) <= 3 and whole.isascii()):
        return None
    if dot and not (len(tenth) == 1 and tenth.isdigit()):
        return None
    return int(whole) * 10 + (int(tenth) if dot else 0)
```

Kode 9.12 (Lanjutan) *cashier_v7.py: kasir yang sama dalam Python.*

```
def rupiah(n):
    digits = str(n)
    out = ""
    for i, d in enumerate(digits):
        if i > 0 and (len(digits) - i) % 3 == 0:
            out += "."
        out += d
    return out


def amount_of(o):
    return o[2] * PRICE[o[1]] // 10


def print_order(i):
    name, s, t = orders[i]
    print("  #%d %s, %s, %d.%d kg = Rp %s"
          % (i + 1, name, SERVICE[s], t // 10, t % 10,
             rupiah(amount_of(orders[i]))))


def cmd_add(words, rest):
    if len(words) < 2:
        print("  usage: add <service> <kg> <name>")
        return
    svc, kg = words[0][:15], words[1][:15]
    if svc not in SERVICE:
        print("  unknown service '%s' (fold, wash-iron, iron)" % svc)
        return
    t = parse_tenths(kg)
    if t is None or not 1 <= t <= 500:
        print("  weight must look like 3 or 3.5 (0.1 to 50.0 kg)")
        return
    name = tidy_name(rest)
    if name == "":
        print("  the customer's name is missing")
        return
    if len(orders) == MAX_ORDERS:
        print("  the order book is full")
        return
    orders.append([name, SERVICE.index(svc), t])
    print_order(len(orders) - 1)
```

Kode 9.12 (Lanjutan) *cashier_v7.py: kasir yang sama dalam Python.*

```
def cmd_find(rest):
    part = tidy_name(rest)
    if part == "":
        print("  usage: find <part of a name>")
        return
    hits = 0
    for i, o in enumerate(orders):
        if part.lower() in o[0].lower():
            print_order(i)
            hits += 1
    if hits == 0:
        print("  no customer matches '%s'" % part)


def total_of_day():
    return sum(amount_of(o) for o in orders)


def cmd_list():
    print("  %2s  %-20s  %-9s  %7s  %10s"
          % ("No", "Customer", "Service", "Weight", "Amount"))
    for i, (name, s, t) in enumerate(orders):
        print("  %2d  %-20.20s  %-9s  %2d.%d kg  %10s"
              % (i + 1, name, SERVICE[s], t // 10, t % 10,
                 rupiah(amount_of(orders[i]))))
    print("  %d order(s), total Rp %s" % (len(orders), rupiah(total_of_day())))


def cmd_help():
    print("  add <service> <kg> <name>   service: fold, wash-iron, iron")
    print("  find <part of a name>       list       help       quit")


def main():
    echo = len(sys.argv) > 1 and sys.argv[1] == "--echo"
    print("Profitina Laundry cashier v0.7 - type help for the commands")
    while True:
        print("> ", end="", flush=True)
        r, line = read_line()
        if r == 0:
            print()
            break
        if echo:
            print(line[:40] + "..." if r < 0 else line)
        if r < 0:
            print("  line too long (at most 79 characters), ignored")
```

Kode 9.12 (Lanjutan) *cashier_v7.py: kasir yang sama dalam Python.*

```
            continue
        parts = line.split(None, 1)
        if not parts:
            continue
        cmd, rest = parts[0][:15], parts[1] if len(parts) > 1 else ""
        if cmd == "add":
            words = rest.split(None, 2)
            cmd_add(words, words[2] if len(words) > 2 else "")
        elif cmd == "find":
            cmd_find(rest)
        elif cmd == "list":
            cmd_list()
        elif cmd == "help":
            cmd_help()
        elif cmd == "quit":
            break
        else:
            print("  unknown command '%s' - type help" % cmd)
    print("Bye. %d order(s), Rp %s today." % (len(orders), rupiah(total_of_day())))


if __name__ == "__main__":
    main()
```

9.9 Menjalankan dan Mengujinya

Berkas `data/session9.txt` berisi satu pagi di meja kasir, termasuk kesalahan-kesalahan yang dibuat kasir yang lelah:

```
help
add fold 3.5 andi   pratama
add dry-clean 2 budi
add wash-iron 2 bUDI santoso
add iron 4.25 citra
add iron 4 citra dewi LESTARI

add fold 1.5
find BUDI
find ari
list
dance
add fold 2 this line is far too long because somebody kept typing a very long name
quit
```

Jalankan dengan `./cashier_v7 --echo < ../data/session9.txt` dari folder `c`, atau `python3 cashier_v7.py --echo < ../data/session9.txt` dari folder `python`. Keduanya mencetak:

```
Profitina Laundry cashier v0.7 - type help for the commands
> help
  add <service> <kg> <name>   service: fold, wash-iron, iron
  find <part of a name>       list       help       quit
> add fold 3.5 andi   pratama
  #1 Andi Pratama, fold, 3.5 kg = Rp 24.500
> add dry-clean 2 budi
  unknown service 'dry-clean' (fold, wash-iron, iron)
> add wash-iron 2 bUDI santoso
  #2 Budi Santoso, wash-iron, 2.0 kg = Rp 20.000
> add iron 4.25 citra
  weight must look like 3 or 3.5 (0.1 to 50.0 kg)
> add iron 4 citra dewi LESTARI
  #3 Citra Dewi Lestari, iron, 4.0 kg = Rp 24.000
>
> add fold 1.5
  the customer's name is missing
> find BUDI
  #2 Budi Santoso, wash-iron, 2.0 kg = Rp 20.000
> find ari
  #3 Citra Dewi Lestari, iron, 4.0 kg = Rp 24.000
> list
  No  Customer              Service     Weight      Amount
   1  Andi Pratama          fold        3.5 kg      24.500
   2  Budi Santoso          wash-iron   2.0 kg      20.000
   3  Citra Dewi Lestari    iron        4.0 kg      24.000
  3 order(s), total Rp 68.500
> dance
  unknown command 'dance' - type help
> add fold 2 this line is far too long bec...
  line too long (at most 79 characters), ignored
> quit
Bye. 3 order(s), Rp 68.500 today.
```

Periksa setiap jawaban terhadap aturannya. `andi   pratama` menjadi `Andi Pratama`; `bUDI santoso` menjadi `Budi Santoso`. Layanan tak dikenal `dry-clean`, berat `4.25` dengan dua desimal, dan pesanan tanpa nama ditolak dengan pesan, dan tidak satu nomor pesanan pun terpakai olehnya. Baris kosong tidak melakukan apa-apa. `find BUDI` menemukan Budi walaupun huruf besar-kecilnya berbeda, dan `find ari` menemukan `Citra Dewi Lestari`, karena "ari" ada di dalam "Lestari": pencarian dari sebagian nama menemukan semua yang memuat teks itu, yang biasanya memang dikehendaki Mbak Rina dan sesekali mengejutkan. Baris 82 karakter ditolak utuh alih-alih dipotong menjadi pesanan yang aneh, dan 40 karakter pertamanya diulang sehingga catatan menunjukkan apa yang terjadi.

Sebuah berkas sesi adalah sebuah *uji*: keluaran benarnya diketahui dan dapat dibandingkan secara otomatis. Setiap folder bab memiliki penguji, `tests/run_tests.py`, yang melakukannya untuk setiap program dalam bab. Penguji mengompilasi setiap program C dengan `-Wall -Wextra -Werror`, sehingga satu peringatan saja menggagalkan uji, dan dengan *sanitizer* GCC, `-fsanitize=address,undefined`, yang menambahkan pemeriksaan ke program hasil kompilasi: setiap baca atau tulis di luar larik, setiap pemakaian memori yang sudah dibebaskan, dan setiap luapan bilangan bertanda menghentikan program dengan laporan alih-alih diam-diam merusak [8]. Lalu penguji menjalankan versi C dan Python pada masukan yang sama dan memeriksa bahwa keduanya mencetak persis keluaran yang tersimpan di `tests/expected/`. Tanpa baris-baris yang melaporkan setiap kompilasi, penguji mencetak:

```
$ python3 tests/run_tests.py
ok    string_basics
ok    char_classes
ok    string_library
ok    format_dial
ok    receipt
ok    cashier_v7
ok    cashier_v7_eof
ok    word_stats
8 cases, 0 failed
```

Kasus `cashier_v7_eof` memberikan berkas yang baris terakhirnya tanpa baris baru dan tanpa `quit`, masukan yang menjebak sebagian besar pembaca baris buatan sendiri. Di Windows, *sanitizer* tidak tersedia pada GCC MSYS2; jalankan `python tests/run_tests.py --no-sanitize` di sana.

9.10 Membuatnya Lebih Baik

Versi 0.7 sudah menyelesaikan masalah-masalah Subbab 9.1: nama rapi, pencarian bekerja pada sebagian nama, dan tabel rata. Tiga kelemahan masih tersisa, dan layak disebut karena bab-bab berikutnya menghapusnya.

Pertama, pesanan tinggal di tiga larik paralel. Menambah ruas, misalnya nomor telepon, berarti larik keempat dan harus ingat mengisinya di setiap tempat pesanan dibuat. Bab 10 mengelompokkan segala sesuatu tentang satu pesanan ke dalam satu `struct`.

Kedua, buku pesanan berukuran tetap 50 dan pesanan hari itu lenyap ketika program berakhir atau listrik padam. Bab 10 menumbuhkan buku pesanan satu pesanan demi satu dengan `malloc`, dan Bab 11 menuliskannya ke berkas setelah setiap perubahan.

Ketiga, pencariannya *linear*: setiap `find` memeriksa setiap pesanan. Dengan lima puluh pesanan sehari, ini hanya perlu beberapa mikrodetik dan tidak masalah. Dengan lima puluh ribu pelanggan setahun pun hanya perlu beberapa milidetik di laptop, tetapi struktur yang lebih cerdas akan lebih cepat; Bab 12 membangun pohon, dan pelajaran Struktur Data sesudah buku ini mempelajari struktur pencarian secara mendalam.

Satu perbaikan cocok untuk bab ini, dan Soal Latihan 9.4 memintanya: ruas `phone` yang dimasukkan bersama pesanan dan dibersihkan dengan `digits_only`, sehingga pelanggan juga dapat ditemukan dari empat angka terakhir nomornya, cara banyak laundry di Indonesia benar-benar mencari pelanggannya.

9.11 Memakainya secara Nyata

Teknik-teknik bab ini cocok untuk usaha kecil mana pun yang menulis nama dan nomor dengan tangan. Seorang penjahit dapat merapikan nama pelanggan dan mencarinya dari sebagian nama persis seperti kasir ini; kebun buah dan durian yang berjualan lewat grup obrolan dapat membersihkan nomor telepon pembelinya dengan `digits_only` dan mencetak slip pengiriman dengan tata letak struk; meja depan klinik desa dapat memeriksa bahwa nomor induk kependudukan pasien terdiri atas 16 angka sebelum disimpan. Polanya selalu sama: baca satu baris utuh dengan batas ukuran, potong menjadi ruas-ruas, periksa setiap ruas dan tolak yang tidak cocok dengan pesan yang jelas, lalu simpan bentuk *baku*-nya sehingga pencarian kelak membandingkan hal yang setara.

Saat menyesuaikan kodenya, ingatlah:

- **Tentukan batasnya dan nyatakan.** `NAME_LEN` 32 dan baris 79 karakter adalah pilihan program ini; klinik mungkin memerlukan nama yang lebih panjang. Ubah konstantanya di satu tempat, dan biarkan pesannya menyebutkan batas itu.
- **Huruf di luar abjad Inggris.** Nama dalam UTF-8 memuat byte di atas 127. Kasir meneruskannya dengan aman, karena setiap panggilan `ctype` melakukan *cast* ke `unsigned char` dan byte-byte ini bukan huruf maupun spasi dalam *locale* bawaan "C". Tetapi `toupper` tidak mengubahnya, sehingga "élodie" tetap memakai é kecil, dan `NAME_LEN` menghitung byte, bukan huruf. Menangani aksara lain dengan benar memerlukan karakter lebar atau pustaka Unicode, topik di luar buku ini; `str.title()` dan `str.lower()` Python langsung menanganinya.
- **Data pribadi.** Nama dan nomor telepon adalah data pribadi yang dilindungi di Indonesia oleh Undang-Undang Nomor 27 Tahun 2022 tentang Pelindungan Data Pribadi (UU PDP). Simpan hanya yang diperlukan usaha, dan jangan mencetak nomor telepon lengkap pada struk yang mungkin dibuang pelanggan; empat angka terakhir cukup untuk mengenalinya.

9.12 Kesalahan yang Sering Terjadi, dan Cara Menghindarinya

- **Melupakan byte untuk penutup.** `char code[7]; strcpy(code, "PL-0042");` menulis 8 byte ke dalam 7. Tetapkan ukuran setiap larik sebagai teks terpanjang ditambah satu (Persamaan (9.1)), atau lebih baik, salin dengan `snprintf(code, sizeof code, "%s", text)`.
- **Membandingkan string dengan `==`.** `if (cmd == "quit")` membandingkan alamat dan tidak pernah benar. Pakailah `strcmp(cmd, "quit") == 0`. Kebalikannya adalah `if (strcmp(a, b))`, yang benar ketika kedua string *berbeda*; tulislah `== 0` atau `!= 0` secara lengkap.
- **Memakai `sizeof` pada pointer.** Di dalam fungsi yang parameternya `char *s`, `sizeof s` adalah ukuran pointer (8 pada mesin 64-bit), bukan ukuran larik milik pemanggil. Karena itulah setiap fungsi dalam bab ini yang menulis ke larik menerima ukurannya sebagai parameter tersendiri.
- **Memberikan `char` biasa kepada `<ctype.h>`.** `isalpha(c)` dengan `char` negatif tak terdefinisi. Selalu tulis `isalpha((unsigned char)c)`.
- **Menulis ke literal string.** `char *s = "andi"; s[0] = 'A';` dapat membuat program *crash*. Pakai larik, `char s[] = "andi";`, bila teksnya akan diubah, dan `const char *` bila tidak.
- **Membaca kata dengan `%s` polos.** `scanf("%s", name)` tidak punya batas ukuran. Berikan lebar satu kurang dari larik (`%31s`), atau baca barisnya dengan `fgets` lalu potong sendiri.
- **Lupa bahwa `fgets` menyimpan baris baru.** Membandingkan `"quit\n"` dengan `"quit"` gagal. Buang baris baru (dan `'\r'`) tepat setelah membaca, seperti `read_line`; baris tunggal `buf[strcspn(buf, "\r\n")] = '\0';` melakukan keduanya.
- **Memakai `strtok` pada data dengan ruas kosong atau pada dua string sekaligus.** Ia melewati ruas kosong dan mengingat posisinya dalam variabel tersembunyi. Bab 11 menulis pemisah ruas yang bebas dari kedua masalah itu.
- **Mencetak uang sebagai `double`.** `%.2f` untuk 2,675 mencetak 2.67. Simpan uang dalam bilangan bulat dan format sendiri.

9.13 Ringkasan Bab

- Karakter adalah bilangan bulat kecil; dalam ASCII, `'A'` bernilai 65, `'a'` 97, dan `'0'` 48, sehingga nilai sebuah angka adalah `c - '0'`.
- String C adalah larik `char` yang diakhiri karakter nol `'\0'`. `sizeof` memberi banyak byte larik, `strlen` banyak karakter sebelum penutup (Persamaan (9.1)).

- Larik yang diinisialisasi dari literal adalah salinan yang boleh diubah; `const char *` yang menunjuk literal tidak boleh dipakai untuk menulis.
- `<ctype.h>` menggolongkan dan mengubah karakter tunggal; lakukan *cast* argumennya ke `unsigned char`.
- `<string.h>` menyalin, menyambung, membandingkan, dan mencari string tanpa memeriksa apa pun. Beri setiap penyalinan ukuran (`snprintf`), periksa hasil `strchr` dan `strstr` terhadap `NULL`, bandingkan dengan `strcmp(...) == 0`, dan ingat bahwa `strtok` mengubah masukannya.
- Spesifikasi konversi `printf` berbentuk `%[flags][width][.precision]type` (Persamaan (9.2)); lebar tetap membuat kolom rata, `*` mengambil lebar dari argumen, dan bilangan pecahan biner bukan untuk uang.
- Masukan yang aman membaca satu baris utuh dengan `fgets`, menangani baris yang terlalu panjang, membuang akhir baris, dan baru kemudian memotong baris menjadi kata-kata.
- Kasir v0.7 adalah perulangan baca–evaluasi–cetak dengan nama yang rapi, masukan yang divalidasi, dan pencarian dari sebagian nama, dalam C dan Python dengan keluaran identik, diperiksa oleh penguji dengan *sanitizer*.

9.14 Soal Latihan

1. Tanpa menjalankannya, tuliskan apa yang dicetak oleh `printf("[%-6.3s][%06.1f][%+5d]\n", "Profitina", 3.14159, 42);`. Lalu jalankan untuk memeriksa.
2. Tulis `int count_char(const char *s, char c)` yang mengembalikan berapa kali `c` muncul di `s`, dan pakai untuk menghitung spasi dalam `"Citra Dewi Lestari"`. Mengapa parameternya `const`?
3. Perluas `digits_only` dan `valid_mobile` pada **Kode 9.3** agar nomor yang ditulis dalam bentuk internasional, `+62 812 3456 7890` atau `62812...`, diubah ke bentuk domestik `0812...` sebelum diperiksa. Berikan keluaran untuk ketiga nomor dalam program.
4. Tambahkan kolom `phone` pada kasir v0.7: `add` menerima nomor telepon sebagai kata keempat sebelum nama, menyimpannya setelah dibersihkan `digits_only`, dan `find` juga cocok bila teks pencarian sama dengan empat angka terakhir nomor telepon. Daftarlah baris-baris program yang harus Anda ubah, dan sebutkan mana yang akan menjadi tidak perlu berkat Bab 10.
5. Jelaskan, byte demi byte, apa yang terjadi di memori ketika `char buf[8]; strcpy(buf, "Profitina");` dijalankan. Berapa byte yang ditulis di luar `buf`? Tulis ulang baris itu agar aman, dan sebutkan isi `buf` sesudahnya.
6. Tulis program `word_stats` dalam C dan Python yang membaca teks dari masukan standar dan mencetak banyaknya baris, kata, dan huruf, kata terpanjang, dan huruf yang paling sering muncul (tanpa memedulikan huruf besar-kecil; bila seri, huruf yang lebih dekat ke `a`). Sebuah kata adalah deretan huruf. Ujilah pada `data/notes.txt`.

9.15 Penyelesaian Soal Latihan

1. `%-6.3s` mencetak paling banyak 3 karakter, `Pro`, rata kiri dalam 6 kolom: `[Pro   ]`. `%06.1f` membulatkan 3,14159 menjadi `3.1` dan mengisinya dengan nol sampai 6 karakter: `[0003.1]`. `%+5d` mencetak `+42` rata kanan dalam 5: `[  +42]`. Barisnya adalah `[Pro   ][0003.1][  +42]`.

2. Perulangannya berjalan sampai penutup dan menghitung yang cocok:

```
int count_char(const char *s, char c)
{
    int n = 0;
    for (; *s != '\0'; s++)
        if (*s == c)
            n++;
    return n;
}
```

Untuk `"Citra Dewi Lestari"` hasilnya 2. Parameternya `const char *` karena fungsi hanya membaca string; `const` menjanjikan hal itu kepada pemanggil, membuat fungsi dapat menerima literal string, dan membuat kompilator menolak upaya menulis tak sengaja seperti `*s = c`.

3. Setelah mengumpulkan angka ke dalam `d`, tambahkan: bila `d` diawali `62`, geser semuanya dua tempat ke kiri, taruh `'0'` di depan, dan pertahankan penutupnya, yang di C ditulis `memmove(d + 1, d + 2, strlen(d + 2) + 1); d[0] = '0';` dan di Python `d = "0" + d[2:]`. Nomor pertama tetap `081234567890 (12 digits) ok`; nomor kedua menjadi `081234567890 (12 digits) ok`; nomor ketiga tetap `081234 (6 digits) check with customer`. Perhatikan bahwa `memmove`, bukan `strcpy`, yang harus dipakai bila sumber dan tujuan bertumpang tindih.

4. Perubahannya: larik baru `char phones[MAX_ORDERS][16]`; di `cmd_add`, `next_word` ketiga untuk nomor telepon, lalu `digits_only`, lalu pemeriksaan panjang; penyalinan ke `phones[count]`; di `cmd_find`, uji kedua, `strlen(part) == 4 && strcmp(phones[i] + strlen(phones[i]) - 4, part) == 0` (setelah memastikan nomor yang tersimpan paling sedikit 4 angka); teks bantuan; dan `list` yang lebih lebar. Larik baru dan baris-baris yang menjaganya tetap selaras dengan tiga larik lain itulah yang menjadi tidak perlu berkat Bab 10: dengan `struct Order` yang memiliki anggota `phone`, satu penugasan seluruh pesanan menjaga semua ruasnya tetap bersama.

5. `"Profitina"` terdiri atas 9 karakter, jadi `strcpy` menulis 10 byte, `P r o f i t i n a \0`, mulai dari `buf[0]`. `buf` punya 8 byte, `buf[0]` sampai `buf[7]`, yang menerima `Profitin`; `a` dan penutupnya ditulis ke dua byte sesudah larik, yang milik sesuatu yang lain. Program bisa tampak berjalan, *crash* belakangan, atau merusak variabel lain; di bawah *sanitizer* ia berhenti dengan laporan *stack-buffer-overflow*. Baris yang aman adalah `snprintf(buf, sizeof buf, "%s", "Profitina");`. Ia menulis paling banyak 8 byte: `sizeof buf - 1` = 7 karakter pertama, `Profiti`, dan penutup. Ia mengembalikan 9, panjang teks utuh, sehingga program dapat mengetahui bahwa nama itu terpotong.

6. **Kode 9.13** membaca baris demi baris dengan `fgets` dan menyusuri setiap baris dengan keadaan "apakah saya di dalam kata?" yang sama seperti `tidy_name`: huruf memperpanjang kata saat ini, selain itu mengakhirinya. Menyusuri sampai *dan termasuk* penutup memastikan kata di ujung baris ikut terhitung.

Kode 9.13 `word_stats.c`: menghitung baris, kata, dan huruf, kata terpanjang, dan huruf yang paling sering muncul.

```c
/* word_stats.c - Exercise 9.6: read text from standard input and report the
 * number of lines, words and letters, the longest word and the most common
 * letter. A word is a run of letters; case is ignored when counting letters. */
#include <ctype.h>
#include <stdio.h>
#include <string.h>

#define LINE_LEN 256
#define WORD_LEN 32

int main(void)
{
    char line[LINE_LEN], word[WORD_LEN], longest[WORD_LEN] = "";
    int lines = 0, words = 0, letters = 0, freq[26] = {0};

    while (fgets(line, sizeof line, stdin) != NULL) {
        lines++;
        size_t len = 0;                          /* length of current word */
        for (size_t i = 0;; i++) {
            unsigned char c = (unsigned char)line[i];
            if (isalpha(c)) {
                if (len + 1 < sizeof word)
                    word[len] = (char)c;
                len++;
                letters++;
                freq[tolower(c) - 'a']++;
            } else {
                if (len > 0) {                   /* a word has just ended */
                    words++;
                    word[len < sizeof word ? len : sizeof word - 1] = '\0';
                    if (strlen(word) > strlen(longest))
                        strcpy(longest, word);
                    len = 0;
                }
                if (c == '\0')
                    break;
            }
        }
    }
    int best = 0;
    for (int k = 1; k < 26; k++)
        if (freq[k] > freq[best])
            best = k;
    printf("lines: %d, words: %d, letters: %d\n", lines, words, letters);
    printf("longest word: \"%s\" (%zu letters)\n", longest, strlen(longest));
    printf("most common letter: %c (%d times)\n", 'a' + best, freq[best]);
```

Kode 9.13 (Lanjutan) *word_stats.c: menghitung baris, kata, dan huruf, kata terpanjang, dan huruf yang paling sering muncul.*

```c
    return 0;
}
```

Versi Python, **Kode 9.14**, menambahkan `"\0"` ke setiap baris dengan alasan yang sama, dan memilih huruf yang paling sering dengan `max` dan kunci yang mengutamakan hitungan lebih besar dan, bila seri, indeks lebih kecil.

Kode 9.14 `word_stats.py`: statistik yang sama dalam Python.

```python
# word_stats.py - Exercise 9.6: lines, words, letters, the longest word and
# the most common letter of the text on standard input.
import sys

WORD_LEN = 32


def main():
    lines = words = letters = 0
    longest = ""
    freq = [0] * 26
    for line in sys.stdin:
        lines += 1
        word = ""
        for c in line + "\0":                # "\0" ends the last word
            if c.isascii() and c.isalpha():
                word += c
                letters += 1
                freq[ord(c.lower()) - ord("a")] += 1
            elif word:
                words += 1
                word = word[:WORD_LEN - 1]
                if len(word) > len(longest):
                    longest = word
                word = ""
    best = max(range(26), key=lambda k: (freq[k], -k))
    print("lines: %d, words: %d, letters: %d" % (lines, words, letters))
    print('longest word: "%s" (%d letters)' % (longest, len(longest)))
    print("most common letter: %c (%d times)" % (chr(ord("a") + best), freq[best]))


if __name__ == "__main__":
    main()
```

Dengan `./word_stats < ../data/notes.txt` keduanya mencetak:

```
lines: 3, words: 26, letters: 121
longest word: "separately" (10 letters)
most common letter: e (15 times)
```

Daftar Pustaka

[1] B. W. Kernighan and D. M. Ritchie, *The C Programming Language*, 2nd ed. Englewood Cliffs, NJ, USA: Prentice Hall, 1988.

[2] ISO/IEC 9899:2018, *Information technology — Programming languages — C* (C17). Geneva, Switzerland: ISO, 2018, §7.4 “Character handling <ctype.h>”, §7.21.6 “Formatted input/output functions”, §7.24 “String handling <string.h>”.

[3] cppreference.com, “Null-terminated byte strings.” [Online]. Available: https://en.cppreference.com/w/c/string/byte (accessed Sep. 2026).

[4] Software Engineering Institute, Carnegie Mellon University, *SEI CERT C Coding Standard*, rules STR31-C “Guarantee that storage for strings has sufficient space for character data and the null terminator” and STR37-C “Arguments to character-handling functions must be representable as an unsigned char.” [Online]. Available: https://wiki.sei.cmu.edu/confluence/display/c (accessed Sep. 2026).

[5] MITRE, “2025 CWE Top 25 Most Dangerous Software Weaknesses,” Dec. 2025. [Online]. Available: https://cwe.mitre.org/top25/archive/2025/2025_cwe_top25.html

[6] P. J. Deitel and H. M. Deitel, *C How to Program*, 9th ed. Hoboken, NJ, USA: Pearson, 2022, ch. 8 “Characters and Strings” and ch. 9 “Formatted Input/Output.”

[7] Python Software Foundation, “Floating-Point Arithmetic: Issues and Limitations,” *The Python Tutorial*, Python 3.14. [Online]. Available: https://docs.python.org/3/tutorial/floatingpoint.html (accessed Sep. 2026).

[8] GCC team, “Program Instrumentation Options (-fsanitize=address, -fsanitize=undefined),” *Using the GNU Compiler Collection*. [Online]. Available: https://gcc.gnu.org/onlinedocs/gcc/Instrumentation-Options.html (accessed Sep. 2026).

Bab 10

Struktur, Enum, dan Linked List Pertama

Sebuah pesanan di Profitina Laundry adalah satu hal: nomor, pelanggan, layanan, berat, jumlah tagihan, dan keadaan seperti "sedang dicuci" atau "siap diambil". Sampai sekarang program kita menyimpan fakta-fakta ini dalam larik-larik terpisah dan mengandalkan setiap baris kode untuk menjaganya tetap selaras. Bab ini memberi program cara untuk menyatakan apa *sebenarnya* pesanan itu. Di akhir bab Anda akan mampu mendefinisikan `struct` yang mengelompokkan nilai-nilai berkaitan dengan tipe berbeda, membuat, menyalin, dan membandingkan struktur, serta menjelaskan mengapa sebuah struktur bisa lebih besar daripada jumlah anggotanya; memberikan struktur kepada fungsi secara nilai (*by value*) dan lewat pointer, serta memakai operator `->`; memberi nama pada kode dengan `enum` dan pada tipe dengan `typedef`, lalu membangun tabel yang diindeks dengan nama-nama itu; meminta memori saat program berjalan dengan `malloc` dan mengembalikannya dengan `free`; serta membangun, menyusuri, menumbuhkan, menyusutkan, dan membebaskan *singly linked list* (senarai berantai tunggal), struktur data pertama dalam buku ini yang ukurannya tidak ditetapkan sejak awal. Bab ini ditutup dengan kasir v0.8, yang setiap pesanannya adalah satu struktur, pesanan di toko membentuk *linked list*, dan setiap pesanan berpindah dari *waiting* (menunggu) ke *washing* (dicuci) ke *ready* (siap) sampai pelanggan mengambilnya.

10.1 Posisi Kita Sekarang: Tiga Larik yang Harus Sepakat

Kasir v0.7 dari Bab 9 mengingat setiap pesanan dalam tiga *larik paralel*: nama di `names[i]`, layanan di `service_of[i]`, dan berat di `tenths_of[i]`. Indeks `i` adalah satu-satunya pengikat ketiganya. Selama setiap potong kode yang menyentuh pesanan menyentuh ketiga larik dengan indeks yang sama, semuanya beres.

Sekarang bayangkan Mbak Rina meminta pesanan hari itu diurutkan menurut jumlah tagihan, terbesar dulu, agar ia tahu kantong mana yang dicuci lebih dulu. Seorang pemrogram yang menulis perulangan pengurutan dengan tergesa-gesa menukar `names[i]` dan `names[j]` tetapi lupa menukar `tenths_of[i]` dan `tenths_of[j]`. Program terkompilasi, berjalan, dan mencetak tabel rapi yang menempatkan nama Andi di samping berat dan tagihan Budi. Tidak ada yang *crash*; tabelnya hanya salah, dan seorang pelanggan ditagih untuk cucian orang lain. Setiap fakta baru tentang pesanan, seperti nomor telepon atau tanggal pengambilan, menambah satu larik lagi yang harus diingat oleh setiap perulangan pengurutan, penghapusan, dan penyalinan.

Ada masalah kedua. Larik-larik itu berukuran tetap, 50 pesanan, yang dipilih ketika program ditulis. Pada hari Selasa yang sepi separuh ruang itu terbuang; pada hari Senin sesudah libur panjang, pelanggan ke-51 ditolak dengan pesan "the order book is full". Dan ada masalah ketiga: ketika pelanggan mengambil pesanannya, v0.7 tidak punya cara mengeluarkannya dari tengah larik kecuali menggeser setiap pesanan sesudahnya satu tempat ke kiri, di ketiga larik.

Jawaban C atas masalah pertama adalah **struktur**, sebuah tipe yang membungkus beberapa nilai di bawah satu nama, sehingga pesanan dipindahkan, disalin, dan diteruskan sebagai satu

kesatuan. Jawabannya atas masalah kedua dan ketiga adalah meminta memori satu pesanan demi satu saat program berjalan, lalu merangkai pesanan-pesanan itu dengan pointer menjadi **linked list**, yang tumbuh dan menyusut satu elemen demi satu dan memungkinkan pesanan keluar dari tengah hanya dengan mengubah satu pointer. Bab ini membangun keduanya, langkah demi langkah.

10.2 Gagasannya dengan Kata-kata Sederhana: Satu Rekaman, Satu Nama

10.2.1 Mendeklarasikan struktur

Deklarasi struktur menjelaskan sebuah tipe baru: nilai apa saja yang dimuatnya, yang disebut *anggota* (*member*) atau *ruas* (*field*), dan tipe masing-masing.

```
struct Order {
    int id;
    char customer[22];
    int tenths;
    long long amount;
};
```

Deklarasi ini membuat tipe bernama `struct Order`; ia belum membuat variabel apa pun. Variabel bertipe itu dideklarasikan seperti variabel lain, `struct Order a;`, dan setiap anggota dicapai dengan **operator titik**: `a.id`, `a.customer`, `a.amount`. Anggota boleh bertipe apa saja, termasuk larik seperti `customer` atau, seperti akan kita lihat, struktur lain atau pointer.

> **DEFINISI.** *Struktur* (`struct`) adalah tipe yang tersusun atas *anggota-anggota* bernama yang boleh bertipe berbeda. Variabel bertipe struktur memegang satu nilai untuk setiap anggota, tersimpan bersama dalam satu blok memori. Anggota `m` dari variabel struktur `s` ditulis `s.m`; bila `p` adalah pointer ke struktur, anggotanya adalah `(*p).m`, yang di C boleh ditulis `p->m`.

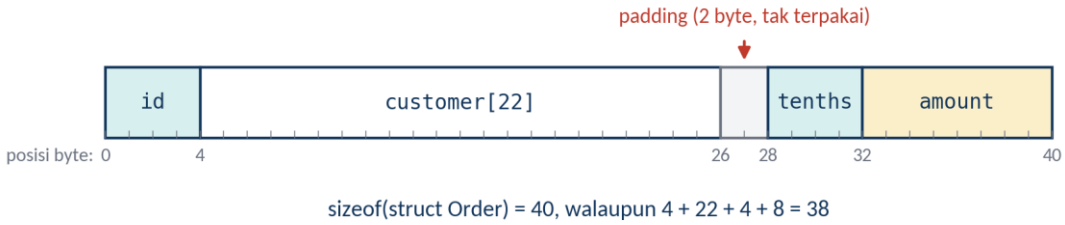
Variabel struktur dapat diinisialisasi dengan daftar nilai dalam kurung kurawal sesuai urutan anggota, atau dengan *designated initializer* (inisialisasi bernama) yang menyebut nama anggota, dalam urutan apa pun. Anggota yang tidak diberi nilai diisi nol, persis seperti pada larik:

```
struct Order b = {2, "Budi Santoso", 20, 20000};    /* in order */
struct Order c = {.customer = "Citra", .id = 3};    /* by name  */
```

10.2.2 Rupa struktur di memori

Anggota-anggota struktur disimpan berurutan sesuai urutan deklarasi. Tetapi tidak selalu dipadatkan rapat. Kebanyakan prosesor membaca `int` paling cepat bila alamatnya kelipatan 4, dan `long long` bila alamatnya kelipatan 8; syarat ini disebut *alignment* (perataan) tipe tersebut. Untuk memenuhinya, kompilator boleh menyisipkan byte tak terpakai, yang disebut *padding* (pengisi), di antara anggota dan di akhir struktur.

Gambar 10.1 memperlihatkan `struct Order` di komputer 64-bit yang umum. `id` menempati byte 0 sampai 3. `customer` menempati 22 byte berikutnya, 4 sampai 25. Anggota berikutnya, `tenths`, bertipe `int` dan harus dimulai di kelipatan 4; byte bebas berikutnya, 26, bukan kelipatan 4, jadi kompilator melewati dua byte dan memulai `tenths` di 28. `amount` lalu dimulai di 32, kelipatan 8, dan berakhir di 39. Seluruh struktur menempati 40 byte walaupun anggota-anggotanya hanya berjumlah 38.



Gambar 10.1 Tata letak `struct Order` di memori: empat anggota di posisi 0, 4, 28, dan 32, dan dua byte *padding* sesudah `customer` agar `tenths` dimulai di kelipatan empat.

Aturan yang diikuti kompilator mudah dinyatakan. Bila byte bebas berikutnya berada di posisi o dan anggota berikutnya ber-*alignment* a, anggota itu dimulai di kelipatan a pertama yang tidak lebih kecil daripada o:

$$o' = a \cdot \left\lceil \frac{o}{a} \right\rceil \tag{10.1}$$

dengan $\lceil x \rceil$ berarti "x dibulatkan ke atas menjadi bilangan bulat". Untuk `tenths`, $o = 26$ dan $a = 4$, sehingga $o' = 4 \cdot \lceil 6.5 \rceil = 4 \cdot 7 = 28$. Untuk `amount`, $o = 32$ dan $a = 8$, sehingga $o' = 8 \cdot 4 = 32$ dan tidak perlu *padding*. Di akhir, ukuran seluruh struktur dibulatkan ke atas dengan cara yang sama menjadi kelipatan *alignment* terbesarnya, agar dalam larik struktur setiap elemen juga rata. *Alignment* persisnya dipilih oleh platform, bukan oleh standar C [1], sehingga aturan praktisnya: jangan pernah menghitung ukuran struktur secara manual di dalam program; tanyakan kepada `sizeof`, dan tanyakan kepada `offsetof` (dari `<stddef.h>`) di mana sebuah anggota dimulai.

10.2.3 Program pertama

Kode 10.1 menyatukan bagian-bagian ini: mengisi struktur anggota demi anggota, menginisialisasi dua struktur lagi dengan daftar, menyalin satu, membandingkan dua, dan mengukur tata letak **Gambar 10.1**.

Kode 10.1 `order_struct.c`: mendeklarasikan, menginisialisasi, menyalin, dan membandingkan struktur, serta mengukur tata letaknya.

```c
/* order_struct.c - one struct holds everything about one laundry order. */
#include <stddef.h>
#include <stdio.h>
#include <string.h>

struct Order {
    int id;                 /* 4 bytes                                  */
    char customer[22];      /* 22 bytes                                 */
    int tenths;             /* weight in tenths of a kg; 4 bytes        */
    long long amount;       /* rupiah; 8 bytes                          */
};

int main(void)
{
    struct Order a;                             /* members still unknown */
    a.id = 1;
    strcpy(a.customer, "Andi Pratama");
    a.tenths = 35;
    a.amount = 24500;

    struct Order b = {2, "Budi Santoso", 20, 20000};     /* in order */
    struct Order c = {.customer = "Citra", .id = 3};     /* by name */

    printf("a: #%d %-14s %d.%d kg Rp %lld\n",
           a.id, a.customer, a.tenths / 10, a.tenths % 10, a.amount);
    printf("b: #%d %-14s %d.%d kg Rp %lld\n",
           b.id, b.customer, b.tenths / 10, b.tenths % 10, b.amount);
    printf("c: #%d %-14s %d.%d kg Rp %lld (members not named are 0)\n",
           c.id, c.customer, c.tenths / 10, c.tenths % 10, c.amount);

    struct Order copy = a;          /* copies every member, array included */
    copy.id = 99;
    copy.customer[0] = 'X';
    printf("copy: #%d %s | a is still #%d %s\n",
           copy.id, copy.customer, a.id, a.customer);

    int same = a.id == b.id && strcmp(a.customer, b.customer) == 0 &&
               a.tenths == b.tenths && a.amount == b.amount;
    printf("a and b hold the same order? %s\n", same ? "yes" : "no");

    printf("sizeof(struct Order) = %zu, members add up to %zu\n",
           sizeof(struct Order),
           sizeof a.id + sizeof a.customer + sizeof a.tenths + sizeof a.amount);
    printf("offsets: id %zu, customer %zu, tenths %zu, amount %zu\n",
           offsetof(struct Order, id), offsetof(struct Order, customer),
           offsetof(struct Order, tenths), offsetof(struct Order, amount));
    return 0;
}
```

Kode 10.1 (Lanjutan) *order_struct.c: mendeklarasikan, menginisialisasi, menyalin, dan membandingkan struktur, serta mengukur tata letaknya.*

```
a: #1 Andi Pratama   3.5 kg Rp 24500
b: #2 Budi Santoso   2.0 kg Rp 20000
c: #3 Citra          0.0 kg Rp 0 (members not named are 0)
copy: #99 Xndi Pratama | a is still #1 Andi Pratama
a and b hold the same order? no
sizeof(struct Order) = 40, members add up to 38
offsets: id 0, customer 4, tenths 28, amount 32
```

Tiga fakta dalam keluaran ini patut diperhatikan.

Penugasan menyalin semuanya. `struct Order copy = a;` menyalin keseluruhan 40 byte, *termasuk larik* `customer`. Mengubah `copy.customer[0]` menjadi `X` tidak menyentuh `a.customer`:

salinannya memiliki larik sendiri. Ini menarik karena Bab 6 menunjukkan bahwa larik sama sekali tidak dapat ditugaskan dengan `=`; begitu dibungkus struktur, bisa.

Perbandingan adalah tugas Anda. C tidak punya `==` untuk struktur. Dua pesanan "sama" bila Anda menyatakannya begitu, dan program menyatakannya anggota demi anggota, dengan `strcmp` untuk nama. (Membandingkan byte mentah dengan `memcmp` memang menggoda tetapi salah: byte *padding* bisa berisi apa saja.)

Ukurannya 40, bukan 38. Dua baris terakhir menegaskan **Gambar 10.1** dan Persamaan (10.1). Mengurutkan ulang anggota dari *alignment* terbesar ke terkecil, `amount` dulu, lalu dua `int`, lalu lariknya, membuat anggotanya berakhir di 38 ditambah 2 byte *padding* di akhir, tetap 40, karena ukuran harus kelipatan 8. *Padding* jarang layak dilawan; mengetahui keberadaannya menjelaskan banyak kejutan, misalnya di Bab 11 ketika struktur ditulis ke berkas.

Versi Python, **Kode 10.2**, memakai *data class*, kelas yang dideklarasikan dengan `@dataclass`, padanan terdekat struktur C di Python: ruas bernama, inisialisator otomatis yang menerima nilai berurutan atau bernama, dan `==` otomatis yang membandingkan ruas demi ruas [2]. Objek Python tidak ditata byte demi byte, jadi untuk mencetak ukuran yang sama program menggambarkan struktur C dengan modul `ctypes`, yang menerapkan aturan *alignment* C milik platform [3].

Kode 10.2 `order_struct.py`: *data class* untuk pesanan, dan `ctypes` untuk mengukur tata letak C.

```python
# order_struct.py - a class with named fields plays the role of a C struct.
# ctypes.Structure lays the fields out exactly like the C compiler does, so
# the sizes and offsets printed here match order_struct.c.
import ctypes
from dataclasses import dataclass, replace


@dataclass
class Order:
    id: int = 0
    customer: str = ""
    tenths: int = 0            # weight in tenths of a kg
    amount: int = 0            # rupiah


class COrder(ctypes.Structure):
    _fields_ = [("id", ctypes.c_int), ("customer", ctypes.c_char * 22),
                ("tenths", ctypes.c_int), ("amount", ctypes.c_longlong)]


def show(label, o, note=""):
    print("%s: #%d %-14s %d.%d kg Rp %d%s"
          % (label, o.id, o.customer, o.tenths // 10, o.tenths % 10,
             o.amount, note))


def main():
    a = Order()
    a.id = 1
    a.customer = "Andi Pratama"
    a.tenths = 35
    a.amount = 24500
    b = Order(2, "Budi Santoso", 20, 20000)
    c = Order(customer="Citra", id=3)
    show("a", a)
    show("b", b)
    show("c", c, " (members not named are 0)")

    copy = replace(a)                    # a real copy, like struct assignment
    copy.id = 99
    copy.customer = "X" + copy.customer[1:]
    print("copy: #%d %s | a is still #%d %s"
          % (copy.id, copy.customer, a.id, a.customer))
    print("a and b hold the same order? %s" % ("yes" if a == b else "no"))
```

Kode 10.2 (Lanjutan) *order_struct.py: data class untuk pesanan, dan ctypes untuk mengukur tata letak C.*

```python
    members = sum(ctypes.sizeof(t) for _, t in COrder._fields_)
    print("sizeof(struct Order) = %d, members add up to %d"
          % (ctypes.sizeof(COrder), members))
    print("offsets: id %d, customer %d, tenths %d, amount %d"
          % tuple(getattr(COrder, f).offset for f, _ in COrder._fields_))


if __name__ == "__main__":
    main()
```

Ada satu perbedaan nyata yang perlu diingat. Di Python, `b = a` *tidak* menyalin objek; ia menjadikan `b` nama kedua untuk objek yang sama, dan mengubah `b.id` mengubah `a.id`. Untuk menyalin, program memanggil `dataclasses.replace(a)`. Penugasan C menyalin; penugasan Python berbagi.

10.3 Struktur dan Fungsi

Struktur dapat diberikan kepada fungsi dan dikembalikan darinya seperti nilai lain, dan di sini perbedaan antara *by value* dan *by pointer*, yang diperkenalkan Bab 7 untuk bilangan tunggal, menjadi sangat kentara.

Diberikan **secara nilai**, fungsi menerima salinan seluruh struktur. Ia dapat membacanya dan mengubah salinannya, dan struktur milik pemanggil tidak pernah terpengaruh. Untuk pesanan 40 byte, penyalinan itu murah; untuk struktur yang memuat catatan 1.000 karakter, ia memakan 1.000 byte pada setiap panggilan.

Diberikan **lewat pointer**, fungsi hanya menerima alamat struktur milik pemanggil, 8 byte di komputer 64-bit, berapa pun ukuran strukturnya. Lewat pointer itu ia dapat membaca anggota dan mengubahnya. Karena pointer ke struktur begitu umum, C memiliki operator untuk "anggota dari struktur yang ditunjuk pointer ini": `o->amount` berarti persis `(*o).amount`. Tanda kurung dalam bentuk panjangnya perlu karena `.` mengikat lebih kuat daripada `*`; tanda panah menghematnya.

Bila fungsi hanya perlu *membaca* pesanan, parameternya dideklarasikan `const struct Order *o`: pointer menghindari penyalinan, dan `const` menjanjikan bahwa fungsi tidak akan mengubah pesanan serta membuat kompilator menegakkan janji itu. Inilah cara normal memberikan struktur di C.

Kode 10.3 menunjukkan semua ini dengan *larik struktur*, `struct Order day[4]`, yang menggantikan larik paralel Bab 9: `day[i]` adalah seluruh pesanan `i`, dan `day[i].amount` jumlah tagihannya.

Kode 10.3 `order_functions.c`: struktur yang diberikan secara nilai dan lewat pointer, fungsi yang mengembalikan pointer ke dalam larik, dan `qsort` dengan fungsi pembanding.

```c
/* order_functions.c - structs passed to functions: by value (a copy) and by
 * pointer (the original), the -> operator, and an array of structs. */
#include <stdio.h>
#include <stdlib.h>

struct Order {
    int id;
    char customer[24];
    int tenths;              /* weight in tenths of a kg */
    long amount;             /* rupiah */
};

/* const pointer: we may read the order, never change it */
static void print_order(const struct Order *o)
{
    printf("  #%d %-14s %2d.%d kg  Rp %6ld\n",
           o->id, o->customer, o->tenths / 10, o->tenths % 10, o->amount);
}

static void discount_copy(struct Order o, int percent)    /* gets a copy */
{
    o.amount -= o.amount * percent / 100;
}

static void discount(struct Order *o, int percent)        /* gets the address */
{
    o->amount -= o->amount * percent / 100;     /* same as (*o).amount */
}

static long total(const struct Order orders[], int n)
{
    long sum = 0;
    for (int i = 0; i < n; i++)
        sum += orders[i].amount;
    return sum;
}
```

Kode 10.3 (Lanjutan) *order_functions.c: struktur yang diberikan secara nilai dan lewat pointer, fungsi yang mengembalikan pointer ke dalam larik, dan qsort dengan fungsi pembanding.*

```c
/* Returns a pointer INTO the array, so the caller can change that order. */
static struct Order *largest(struct Order orders[], int n)
{
    struct Order *best = &orders[0];
    for (int i = 1; i < n; i++)
        if (orders[i].amount > best->amount)
            best = &orders[i];
    return best;
}

/* For qsort: larger amount first; equal amounts by smaller id. */
static int by_amount_desc(const void *x, const void *y)
{
    const struct Order *a = x, *b = y;
    if (a->amount != b->amount)
        return a->amount < b->amount ? 1 : -1;
    return a->id - b->id;
}

int main(void)
{
    struct Order day[] = {
        {1, "Andi Pratama", 35, 24500},
        {2, "Budi Santoso", 20, 20000},
        {3, "Citra Lestari", 40, 24000},
        {4, "Dewi Anggraini", 55, 38500},
    };
    int n = (int)(sizeof day / sizeof day[0]);

    discount_copy(day[0], 10);
    printf("after discount_copy(day[0], 10): Rp %ld (unchanged)\n", day[0].amount);
    discount(&day[0], 10);
    printf("after discount(&day[0], 10):     Rp %ld\n", day[0].amount);

    printf("orders of the day, total Rp %ld:\n", total(day, n));
    for (int i = 0; i < n; i++)
        print_order(&day[i]);

    struct Order *big = largest(day, n);
    printf("largest: #%d %s\n", big->id, big->customer);
    big->amount += 5000;                          /* a delivery fee, say */
    printf("with delivery fee: Rp %ld (day[%d] changed)\n",
           day[3].amount, (int)(big - day));
```

Kode 10.3 (Lanjutan) *order_functions.c: struktur yang diberikan secara nilai dan lewat pointer, fungsi yang mengembalikan pointer ke dalam larik, dan qsort dengan fungsi pembanding.*

```c
    qsort(day, (size_t)n, sizeof day[0], by_amount_desc);
    printf("sorted by amount, largest first:\n");
    for (int i = 0; i < n; i++)
        print_order(&day[i]);
    return 0;
}
```

```
after discount_copy(day[0], 10): Rp 24500 (unchanged)
after discount(&day[0], 10):     Rp 22050
orders of the day, total Rp 104550:
  #1 Andi Pratama    3.5 kg  Rp  22050
  #2 Budi Santoso    2.0 kg  Rp  20000
  #3 Citra Lestari   4.0 kg  Rp  24000
  #4 Dewi Anggraini  5.5 kg  Rp  38500
largest: #4 Dewi Anggraini
with delivery fee: Rp 43500 (day[3] changed)
sorted by amount, largest first:
  #4 Dewi Anggraini  5.5 kg  Rp  43500
  #3 Citra Lestari   4.0 kg  Rp  24000
  #1 Andi Pratama    3.5 kg  Rp  22050
  #2 Budi Santoso    2.0 kg  Rp  20000
```

`discount_copy` menerima salinan dan diskon 10%-nya hilang bersama salinan itu; `discount` menerima `&day[0]` dan mengubah pesanan yang sebenarnya. `largest` mengembalikan pointer *ke dalam* larik milik pemanggil, sehingga pemanggil dapat memakainya untuk mengubah pesanan yang ditemukan: menambahkan ongkos antar lewat `big` mengubah `day[3]`, dan `big - day` memberi indeksnya lewat aritmetika pointer.

Bagian terakhir mengurutkan pesanan dengan fungsi pustaka `qsort` dari `<stdlib.h>` [4]. `qsort` dapat mengurutkan larik berisi apa saja, karena ia diberi tahu banyak elemen, ukuran satu elemen, dan sebuah *fungsi pembanding* (pointer ke fungsi, Bab 7). Fungsi pembanding menerima dua pointer `const void *`, yang diubahnya menjadi pointer ke pesanan, dan mengembalikan bilangan negatif bila yang pertama harus datang lebih dulu, positif bila yang kedua, dan nol bila keduanya sama. Di sini ia menaruh jumlah yang lebih besar di depan dan, untuk jumlah yang sama, id yang lebih kecil lebih dulu. Aturan kedua itu penting: `qsort` tidak menjanjikan bahwa elemen yang sama tetap dalam urutan asalnya, jadi tanpa pemutus seri, dua pesanan dengan jumlah sama bisa keluar dalam urutan berbeda di komputer berbeda. Dan lihatlah keluarannya: seluruh pesanan berpindah sebagai satu kesatuan, sehingga tidak ada lagi nama yang bisa bersanding dengan tagihan orang lain. Itulah masalah Subbab 10.1, terselesaikan secara konstruksi.

Di Python, **Kode 10.4**, setiap argumen diberikan sebagai rujukan ke objek, sehingga fungsi selalu dapat mengubah pesanan milik pemanggil; `discount_copy` membuat salinannya sendiri agar berperilaku seperti parameter *by value* di C. Pengurutan memakai `list.sort` dengan fungsi *key*; kunci `(-o.amount, o.id)` mengurutkan menurut jumlah menurun dan id menaik, urutan yang sama dengan pembanding C.

Kode 10.4 `order_functions.py`: fungsi-fungsi yang sama dalam Python, tempat setiap argumen adalah rujukan.

```python
# order_functions.py - Python passes a reference to the object, so a function
# can change an order; to protect the caller's order we pass a copy ourselves.
from dataclasses import dataclass, replace


@dataclass
class Order:
    id: int
    customer: str
    tenths: int
    amount: int


def print_order(o):
    print("  #%d %-14s %2d.%d kg  Rp %6d"
          % (o.id, o.customer, o.tenths // 10, o.tenths % 10, o.amount))


def discount_copy(o, percent):
    o = replace(o)                      # work on a copy, like C by-value
    o.amount -= o.amount * percent // 100


def discount(o, percent):
    o.amount -= o.amount * percent // 100


def total(orders):
    return sum(o.amount for o in orders)


def largest(orders):
    best = orders[0]
    for o in orders[1:]:
        if o.amount > best.amount:
            best = o
    return best


def main():
    day = [Order(1, "Andi Pratama", 35, 24500),
           Order(2, "Budi Santoso", 20, 20000),
           Order(3, "Citra Lestari", 40, 24000),
           Order(4, "Dewi Anggraini", 55, 38500)]
```

Kode 10.4 (Lanjutan) *order_functions.py: fungsi-fungsi yang sama dalam Python, tempat setiap argumen adalah rujukan.*

```
    discount_copy(day[0], 10)
    print("after discount_copy(day[0], 10): Rp %d (unchanged)" % day[0].amount)
    discount(day[0], 10)
    print("after discount(&day[0], 10):     Rp %d" % day[0].amount)

    print("orders of the day, total Rp %d:" % total(day))
    for o in day:
        print_order(o)

    big = largest(day)
    print("largest: #%d %s" % (big.id, big.customer))
    big.amount += 5000
    print("with delivery fee: Rp %d (day[%d] changed)"
          % (day[3].amount, day.index(big)))

    day.sort(key=lambda o: (-o.amount, o.id))
    print("sorted by amount, largest first:")
    for o in day:
        print_order(o)


if __name__ == "__main__":
    main()
```

10.4 `enum` dan `typedef`: Nama Menggantikan Angka

Di bab-bab sebelumnya layanan berupa angka: 1 untuk cuci lipat, 2 untuk cuci setrika, 3 untuk setrika saja. Baris seperti `if (service == 2)` berfungsi, tetapi pembaca harus mengingat arti 2, dan tidak ada yang mencegah program menyimpan 7. Sebuah **enumerasi** memberi nama pada kode-kode itu:

```
typedef enum { FOLD, WASH_IRON, IRON_ONLY, N_SERVICES } Service;
```

Nama-nama di dalam kurung kurawal menjadi konstanta bilangan bulat: `FOLD` bernilai 0, `WASH_IRON` 1, `IRON_ONLY` 2, masing-masing satu lebih besar daripada sebelumnya kecuali nilainya ditulis tegas (`FOLD = 1`). Nama terakhir, `N_SERVICES`, adalah trik yang lazim: nilainya sama dengan banyak layanan sebenarnya, 3, sehingga tabel dapat dideklarasikan dengan `N_SERVICES` elemen dan perulangan dapat berjalan `for (s = FOLD; s < N_SERVICES; s++)`, dan keduanya tetap benar bila layanan baru ditambahkan sebelum nama itu.

Kata kunci **typedef** memberi sebuah tipe nama kedua. Tanpanya, tipe di atas bernama `enum { ... }` atau `enum Service`; dengan `typedef ... Service;` namanya cukup `Service`. Hal yang sama berlaku untuk struktur: `typedef struct { ... } ServiceInfo;` membuat kita dapat menulis `ServiceInfo` alih-alih `struct ServiceInfo`. `typedef` tidak membuat tipe baru maupun variabel, hanya nama yang lebih pendek.

Enumerasi menjadi sangat berguna bersama **designated initializer untuk larik**, yang mengisi elemen menurut indeks: `[WASH_IRON] = {"wash-iron", 10000, 48}` menaruh struktur itu di elemen 1, dalam urutan apa pun baris-barisnya ditulis. **Kode 10.5** membangun tabel layanan laundry dengan cara itu, dan memakai enumerasi kedua, `Status`, untuk riwayat hidup sebuah pesanan.

Kode 10.5 `service_enum.c`: enumerasi untuk layanan dan status, serta tabel layanan yang dibangun dengan *designated initializer*.

```c
/* service_enum.c - enum gives names to codes, typedef gives names to types,
 * and a designated initializer builds a table indexed by those names. */
#include <stdio.h>

typedef enum { FOLD, WASH_IRON, IRON_ONLY, N_SERVICES } Service;
typedef enum { WAITING, WASHING, READY, PICKED_UP } Status;

typedef struct {
    const char *name;
    int price_per_kg;        /* rupiah */
    int hours;               /* promised turnaround */
} ServiceInfo;

static const ServiceInfo SERVICES[N_SERVICES] = {
    [FOLD] = {"fold", 7000, 24},
    [WASH_IRON] = {"wash-iron", 10000, 48},
    [IRON_ONLY] = {"iron", 6000, 24},
};

static const char *status_name(Status s)
{
    switch (s) {
    case WAITING:   return "waiting";
    case WASHING:   return "washing";
    case READY:     return "ready";
    case PICKED_UP: return "picked up";
    }
    return "?";              /* only reached if s holds a bad number */
}

static Status next_status(Status s)
{
    return s == PICKED_UP ? PICKED_UP : (Status)(s + 1);
}

int main(void)
{
    printf("%-3s %-10s %6s %6s\n", "#", "service", "Rp/kg", "hours");
    for (Service s = FOLD; s < N_SERVICES; s++)
        printf("%-3d %-10s %6d %6d\n",
               (int)s, SERVICES[s].name, SERVICES[s].price_per_kg,
               SERVICES[s].hours);
```

Kode 10.5 (Lanjutan) *service_enum.c: enumerasi untuk layanan dan status, serta tabel layanan yang dibangun dengan designated initializer.*

```c
    Service pick = WASH_IRON;
    printf("\n3.0 kg %s = Rp %d, ready in %d hours\n", SERVICES[pick].name,
           3 * SERVICES[pick].price_per_kg, SERVICES[pick].hours);

    Status st = WAITING;
    printf("life of an order:");
    for (int step = 0; step < 5; step++) {
        printf(" %s%s", status_name(st), step < 4 ? " ->" : "\n");
        st = next_status(st);
    }
    printf("sizeof(Service) = %zu, READY = %d\n", sizeof(Service), (int)READY);
    return 0;
}
```

```
#   service     Rp/kg  hours
0   fold         7000     24
1   wash-iron   10000     48
2   iron         6000     24

3.0 kg wash-iron = Rp 30000, ready in 48 hours
life of an order: waiting -> washing -> ready -> picked up -> picked up
sizeof(Service) = 4, READY = 2
```

`status_name` memakai `switch` atas enumerasi. Karena `switch` itu mendaftar setiap nilai `Status` dan tidak punya `default`, `-Wall` GCC akan memberi peringatan bila seseorang menambahkan status baru, misalnya `DELIVERED`, tetapi lupa memberinya nama di sini: kompilator membantu Anda menjaga program tetap lengkap. `return "?"` sesudah `switch` hanya tercapai bila sebuah variabel memegang bilangan yang bukan salah satu nama, yang tidak dicegah oleh C: variabel `enum` pada dasarnya bilangan bulat, dan `sizeof(Service)` bernilai 4 karena disimpan sebagai `int`. Itulah batas enumerasi di C: ia membuat kode mudah dibaca, tetapi tidak membuat nilai salah menjadi mustahil, jadi masukan dari luar tetap harus diperiksa.

Modul `enum` Python menyediakan `IntEnum`, yang anggotanya adalah bilangan bulat bernama yang dapat dipakai sebagai indeks [5]. **Kode 10.6** membangun tabel yang sama sebagai kamus (*dictionary*) yang berkunci enumerasi.

Kode 10.6 `service_enum.py`: `IntEnum` dan kamus sebagai tabel layanan.

```
# service_enum.py - IntEnum gives names to codes; a dict of tuples is the table.
from enum import IntEnum


class Service(IntEnum):
    FOLD = 0
    WASH_IRON = 1
    IRON_ONLY = 2


class Status(IntEnum):
    WAITING = 0
    WASHING = 1
    READY = 2
    PICKED_UP = 3


SERVICES = {                          # name, price per kg, hours
    Service.FOLD: ("fold", 7000, 24),
    Service.WASH_IRON: ("wash-iron", 10000, 48),
    Service.IRON_ONLY: ("iron", 6000, 24),
}

STATUS_NAME = {Status.WAITING: "waiting", Status.WASHING: "washing",
               Status.READY: "ready", Status.PICKED_UP: "picked up"}


def next_status(s):
    return s if s == Status.PICKED_UP else Status(s + 1)


def main():
    print("%-3s %-10s %6s %6s" % ("#", "service", "Rp/kg", "hours"))
    for s in Service:
        name, price, hours = SERVICES[s]
        print("%-3d %-10s %6d %6d" % (s, name, price, hours))

    name, price, hours = SERVICES[Service.WASH_IRON]
    print("\n3.0 kg %s = Rp %d, ready in %d hours" % (name, 3 * price, hours))
```

Kode 10.6 (Lanjutan) *service_enum.py: IntEnum dan kamus sebagai tabel layanan.*

```
    st = Status.WAITING
    line = "life of an order:"
    for step in range(5):
        line += " %s%s" % (STATUS_NAME[st], " ->" if step < 4 else "")
        st = next_status(st)
    print(line)
    print("sizeof(Service) = %d, READY = %d" % (4, Status.READY))  # C: an int


if __name__ == "__main__":
    main()
```

10.5 Memori Sesuai Permintaan: `malloc` dan `free`

Setiap variabel yang kita pakai sejauh ini dibuat dengan salah satu dari dua cara. Variabel yang dideklarasikan di luar semua fungsi (atau dengan `static`) ada selama program berjalan. Variabel yang dideklarasikan di dalam fungsi ada selama fungsi berjalan dan lenyap ketika fungsi kembali; Bab 5 menggambarkannya sebagai penyimpanan *otomatis* di *stack* (tumpukan). Tidak satu pun dari keduanya membiarkan program memutuskan *saat berjalan* berapa banyak pesanan yang disimpan.

Jenis ketiga adalah memori **dinamis**, yang diambil dari wilayah besar bernama *heap* ketika program memintanya, dan disimpan sampai program mengembalikannya. Dua fungsi dari `<stdlib.h>` melakukannya [1], [4]:

- `malloc(n)` memesan satu blok `n` byte dan mengembalikan pointer ke byte pertamanya, bertipe `void *`, yang diubah C secara otomatis menjadi tipe pointer lain. Bila memori tidak tersedia, ia mengembalikan `NULL`. Isi blok itu *tidak* diinisialisasi.
- `free(p)` mengembalikan blok yang ditunjuk `p`, yang harus pointer hasil `malloc` dan belum dibebaskan. `free(NULL)` tidak melakukan apa-apa.

Idiom untuk membuat satu struktur di *heap* adalah

```
Node *n = malloc(sizeof *n);
if (n == NULL) { /* report and stop, or give up this order */ }
```

Menulis `sizeof *n` alih-alih `sizeof(Node)` berarti "ukuran apa pun yang ditunjuk `n`", yang tetap benar bila tipe `n` kelak diubah.

Blok itu hidup sampai `free`, *terlepas dari fungsi yang membuatnya*. Itulah pokoknya: sebuah fungsi dapat membuat pesanan, mengembalikan pointer kepadanya, dan pesanan itu tetap ada sesudah fungsi kembali. Ini juga membawa tiga kewajiban, yang akan kita sebut tiga aturan `malloc`:

1. **Periksa `NULL`.** Menulis lewat pointer nol membuat program *crash*.
2. **Bebaskan setiap blok tepat sekali.** Blok yang tidak pernah dibebaskan adalah *memory leak* (kebocoran memori): tidak berbahaya pada program yang berjalan sedetik, fatal pada kasir yang berjalan seharian dan bocor sedikit pada setiap pesanan. Blok yang dibebaskan dua kali merusak *heap*.
3. **Jangan pernah memakai blok setelah dibebaskan.** Pointer masih memegang alamat lama, tetapi memorinya mungkin sudah menjadi milik sesuatu yang lain. *Use after free* berada di urutan ketujuh dan *NULL pointer dereference* di urutan ketiga belas dalam daftar MITRE 2025 tentang kelemahan perangkat lunak paling berbahaya [6]; keduanya tak lain adalah aturan-aturan ini yang dilanggar [7].

Sanitizer yang dinyalakan oleh penguji buku ini (`-fsanitize=address`) menangkap ketiganya: ia melaporkan setiap operasi tulis lewat pointer yang buruk, setiap pemakaian memori yang sudah dibebaskan, dan, ketika program berakhir, setiap blok yang tidak pernah dibebaskan.

Python tidak punya `malloc` maupun `free`. Setiap objek dibuat di *heap* milik Python, dan penerjemah membebaskannya secara otomatis ketika tidak ada lagi variabel atau objek lain yang merujuknya, layanan yang disebut *garbage collection* (pengumpulan sampah; di CPython sebagian besar lewat penghitungan rujukan). Itulah sebabnya versi Python dalam bab ini lebih pendek dan tidak punya kode pembersihan.

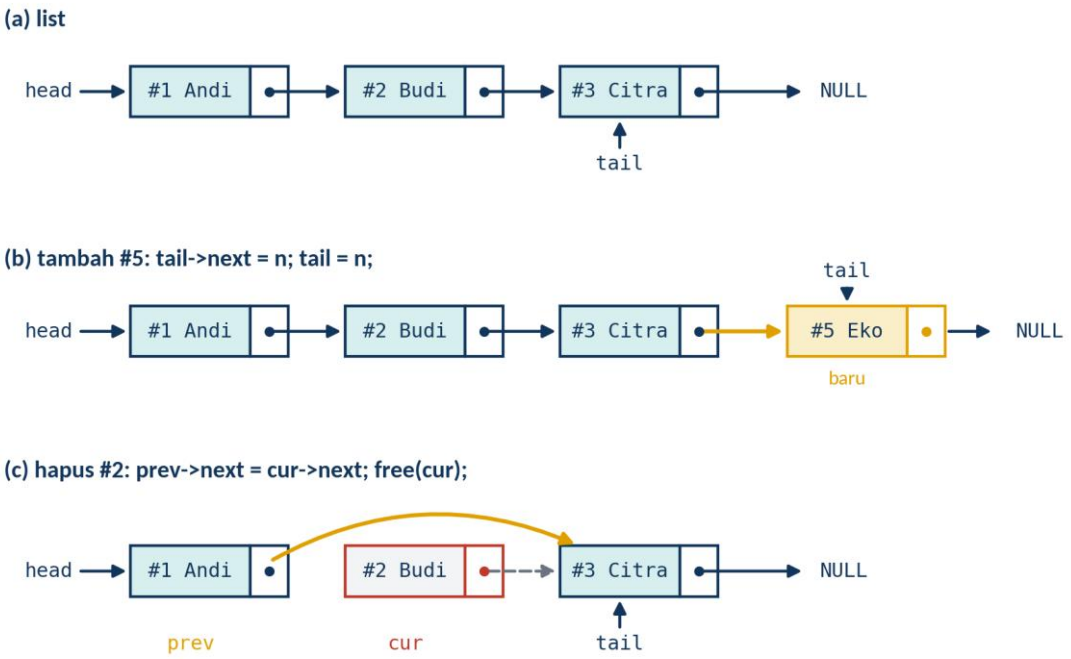
10.6 Struktur yang Menunjuk Dirinya Sendiri: Linked List

10.6.1 Gagasannya

Sebuah struktur boleh memuat pointer ke struktur *bertipe sama*:

```
typedef struct Node {
    int id;
    char customer[24];
    long amount;
    struct Node *next;
} Node;
```

Struktur seperti ini disebut *self-referential* (merujuk diri). Setiap simpul (*node*) memegang satu pesanan dan alamat simpul lain. Mulai dari satu simpul dan mengikuti pointer `next` akan mengunjungi simpul demi simpul, seperti permainan berburu harta karun yang setiap petunjuknya memberi tahu di mana petunjuk berikutnya disembunyikan. `next` simpul terakhir bernilai `NULL`, yang berarti "tidak ada berikutnya". Alamat simpul pertama disimpan dalam variabel bernama `head`; ketika senarai kosong, `head` bernilai `NULL`. Rantai ini adalah **singly linked list** [8], yang tampak pada bagian (a) **Gambar 10.2**.



Gambar 10.2 *Singly linked list* berisi pesanan: (a) `head` menunjuk simpul pertama dan setiap `next` menunjuk simpul berikutnya, yang terakhir menunjuk `NULL`; (b) menambah di akhir mengubah `next` simpul terakhir dan pointer `tail`; (c) menghapus simpul di tengah menjembatani simpul itu dengan satu penugasan.

Dibandingkan larik, *linked list* melepaskan satu hal dan mendapat tiga. Ia melepaskan *akses langsung*: tidak ada `list[5]`; untuk mencapai pesanan keenam Anda harus mengikuti lima pointer dari `head`. Ia mendapat, pertama, bahwa simpul dibuat satu demi satu dengan `malloc`, sehingga panjang senarai persis sesibuk hari itu. Kedua, simpul dapat disisipkan atau dihapus di mana saja dengan mengubah satu atau dua pointer, tanpa menggeser simpul lain. Ketiga, simpul tidak pernah berpindah di memori, sehingga pointer ke sebuah simpul tetap sah selama simpul itu ada. **Tabel 10.1** membandingkan kerja operasi-operasi umum untuk n pesanan.

Tabel 10.1 Kerja operasi-operasi umum pada n pesanan, untuk larik dan untuk *linked list* yang menyimpan pointer ke simpul pertama dan terakhirnya.

Operasi	Larik (Bab 6)	*Linked list* dengan `head` dan `tail`
mencapai pesanan ke-k	satu langkah	k langkah menyusuri `next`
menambah di akhir	satu langkah, bila masih ada ruang	satu langkah, memakai `tail`
menambah di depan	menggeser seluruh n pesanan	satu langkah
menghapus pesanan ke-k	menggeser $n - k$ pesanan sesudahnya	berjalan ke sana, lalu satu langkah
tumbuh melebihi ukuran rencana	mustahil tanpa menyalin	selalu bisa, satu `malloc` per pesanan

Untuk antrean di meja laundry, tempat pesanan ditambahkan di akhir dan keluar dari mana saja, *linked list* yang menang.

10.6.2 Menyusuri senarai

Hampir setiap operasi senarai dibangun di atas satu perulangan, yaitu *penelusuran* (*traversal*):

```
for (const Node *p = head; p != NULL; p = p->next)
    /* do something with p->id, p->customer, p->amount */
```

`p` mulai di simpul pertama; setelah setiap simpul ia pindah ke penerusnya; ketika mencapai `NULL`, penelusuran selesai. Untuk senarai kosong, `head` bernilai `NULL` dan badan perulangan berjalan nol kali, tanpa kasus khusus. Mencetak, menghitung, menjumlahkan, dan mencari semuanya adalah perulangan ini dengan badan yang berbeda.

10.6.3 Menambah simpul

Menambah di **depan** memerlukan dua penugasan, dan urutannya penting: pertama simpul baru menunjuk simpul pertama yang lama, `n->next = head;`, lalu senarai dimulai dari simpul baru, `head = n;`. Bila dibalik, senarai lama akan hilang, karena `head` sudah menunjuk `n` ketika kita mencoba membaca simpul pertama yang lama darinya.

Untuk menambah di **akhir**, kita bisa berjalan ke simpul terakhir setiap kali, tetapi itu memakan n langkah. Sebagai gantinya senarai menyimpan pointer kedua, `tail`, ke simpul terakhirnya. Menambah di akhir lalu menjadi, seperti bagian (b) **Gambar 10.2**, `tail->next = n; tail = n;`, dan satu-satunya kasus khusus adalah senarai kosong, yang tidak punya simpul terakhir sehingga simpul baru sekaligus menjadi `head`.

Kita menyimpan `head`, `tail`, dan banyak simpul bersama dalam satu struktur, `List`, sehingga fungsi dapat menerima seluruh senarai lewat satu pointer dan mengubah ketiganya.

10.6.4 Menghapus simpul

Untuk menghapus simpul, simpul *sebelumnya* harus diberi tahu agar menunjuk melewatinya, seperti bagian (c) **Gambar 10.2**: `prev->next = cur->next;`. Karena itu penelusurannya memakai dua pointer, `prev` satu langkah di belakang `cur`. Tiga kasus perlu perhatian: bila simpul itu yang pertama, tidak ada `prev`, dan `head` yang bergeser; bila yang terakhir, `tail` harus mundur ke `prev`; dan simpul baru boleh dibebaskan setelah dilepas dari rantai.

Mari kita telusuri secara manual pada senarai Dewi (#4), Andi (#1), Budi (#2), Citra (#3), dengan menghapus #2. Pada awalnya `prev` bernilai `NULL` dan `cur` adalah Dewi. Id Dewi 4, bukan 2, jadi penelusuran berlanjut: `prev` menjadi Dewi dan `cur` menjadi Andi. Id Andi 1, jadi berlanjut lagi: `prev` adalah Andi dan `cur` adalah Budi. Id Budi 2, dan penelusuran berhenti dengan `prev`

menunjuk Andi dan `cur` menunjuk Budi. Sekarang `prev` bukan `NULL`, sehingga program menjalankan `prev->next = cur->next`, yang menyalin `next` milik Budi, yaitu alamat Citra, ke `next` milik Andi: Andi kini langsung menunjuk Citra, dan Budi tidak lagi dapat dicapai dari `head`. `tail` menunjuk Citra, bukan Budi, jadi ia tetap. Baru sekarang Budi dibebaskan, dan hitungannya turun dari 4 menjadi 3. Seandainya kita mencari #9, penelusuran akan berjalan sampai ujung, `cur` akan menjadi `NULL`, dan fungsi mengembalikan 0 tanpa mengubah apa pun. Seandainya kita menghapus Dewi, penelusuran langsung berhenti dengan `prev` masih `NULL`, dan `head` berpindah ke Andi. Tiga kasus, satu perulangan, dan masing-masing dapat diperiksa dengan pensil sebelum satu baris kode pun diketik.

Satu hal lagi patut dicatat tentang biaya. Menghapus pesanan yang sudah kita temukan hanya memerlukan satu penugasan, berapa pun panjang senarainya; yang memakan waktu adalah *mencari* pesanan itu, yang dalam kasus terburuk menyusuri seluruh n simpul. Pada larik, pencarian sama lamanya, dan sesudah itu masih harus menggeser semua pesanan di belakangnya. Karena itulah senarai unggul untuk antrean yang anggotanya sering keluar dari tengah.

10.6.5 Membebaskan seluruh senarai

Di akhir, setiap simpul harus dibebaskan. Perulangan yang tampak wajar, `for (p = head; p != NULL; p = p->next) free(p);`, melanggar aturan 3: setelah `free(p)`, pembaruan `p = p->next` membaca dari simpul yang sudah tidak ada. Perulangan yang benar menyimpan penerusnya *sebelum* membebaskan:

```
while (p != NULL) { Node *next = p->next; free(p); p = next; }
```

10.6.6 Programnya

Kode 10.7 mewujudkan semua isi subbab ini dan mengujinya.

Kode 10.7 `order_list.c`: *linked list* berisi pesanan dengan `push_front`, `append`, `find`, `remove_id`, dan `free_all`.

```c
/* order_list.c - a singly linked list of laundry orders: every node is a
 * struct allocated with malloc that points to the next node. */
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

typedef struct Node {
    int id;
    char customer[24];
    long amount;
    struct Node *next;       /* the next node, or NULL after the last one */
} Node;

typedef struct {
    Node *head;              /* first node, NULL when the list is empty */
    Node *tail;              /* last node, so that appending is quick */
    int count;
} List;

static Node *new_node(int id, const char *customer, long amount)
{
    Node *n = malloc(sizeof *n);
    if (n == NULL) {
        fprintf(stderr, "out of memory\n");
        exit(EXIT_FAILURE);
    }
    n->id = id;
    snprintf(n->customer, sizeof n->customer, "%s", customer);
    n->amount = amount;
    n->next = NULL;
    return n;
}

static void push_front(List *l, Node *n)
{
    n->next = l->head;       /* 1: the new node points at the old first */
    l->head = n;             /* 2: the list starts at the new node */
    if (l->tail == NULL)
        l->tail = n;
    l->count++;
}
```

Kode 10.7 (Lanjutan) *order_list.c: linked list berisi pesanan dengan push_front, append, find, remove_id, dan free_all.*

```c
static void append(List *l, Node *n)
{
    if (l->tail == NULL)
        l->head = n;         /* empty list: n is also the first node */
    else
        l->tail->next = n;   /* the old last node points at n */
    l->tail = n;
    l->count++;
}

static void print_list(const List *l)
{
    printf("head");
    for (const Node *p = l->head; p != NULL; p = p->next)
        printf(" -> [#%d %s]", p->id, p->customer);
    printf(" -> NULL   count=%d\n", l->count);
}

static Node *find(const List *l, const char *customer)
{
    for (Node *p = l->head; p != NULL; p = p->next)
        if (strcmp(p->customer, customer) == 0)
            return p;
    return NULL;
}

/* Take the node with this id out of the list and free it; 1 if found. */
static int remove_id(List *l, int id)
{
    Node *prev = NULL, *cur = l->head;
    while (cur != NULL && cur->id != id) {
        prev = cur;
        cur = cur->next;
    }
    if (cur == NULL)
        return 0;
    if (prev == NULL)
        l->head = cur->next;            /* removing the first node */
    else
        prev->next = cur->next;         /* bridge over cur */
    if (l->tail == cur)
        l->tail = prev;                 /* removing the last node */
    free(cur);
    l->count--;
    return 1;
}
```

Kode 10.7 (Lanjutan) *order_list.c: linked list berisi pesanan dengan push_front, append, find, remove_id, dan free_all.*

```
static void free_all(List *l)
{
    Node *p = l->head;
    while (p != NULL) {
        Node *next = p->next;           /* read next BEFORE free */
        free(p);
        p = next;
    }
    l->head = l->tail = NULL;
    l->count = 0;
}

int main(void)
{
    List queue = {NULL, NULL, 0};
    print_list(&queue);
    append(&queue, new_node(1, "Andi", 24500));
    append(&queue, new_node(2, "Budi", 20000));
    append(&queue, new_node(3, "Citra", 24000));
    print_list(&queue);
    push_front(&queue, new_node(4, "Dewi", 38500));      /* a VIP customer */
    print_list(&queue);

    long sum = 0;
    for (const Node *p = queue.head; p != NULL; p = p->next)
        sum += p->amount;
    printf("total Rp %ld\n", sum);

    const char *who[] = {"Citra", "Eko"};
    for (int i = 0; i < 2; i++) {
        Node *f = find(&queue, who[i]);
        if (f != NULL)
            printf("find %s: #%d, Rp %ld\n", who[i], f->id, f->amount);
        else
            printf("find %s: not in the list\n", who[i]);
    }
```

Kode 10.7 (Lanjutan) *order_list.c: linked list berisi pesanan dengan push_front, append, find, remove_id, dan free_all.*

```
    int ids[] = {4, 2, 3, 9};
    for (int i = 0; i < 4; i++) {
        int ok = remove_id(&queue, ids[i]);
        printf("remove #%d: %s  ", ids[i], ok ? "done" : "no such order");
        print_list(&queue);
    }
    append(&queue, new_node(5, "Eko", 14000));           /* tail still right? */
    print_list(&queue);
    free_all(&queue);
    print_list(&queue);
    return 0;
}
```

```
head -> NULL   count=0
head -> [#1 Andi] -> [#2 Budi] -> [#3 Citra] -> NULL   count=3
head -> [#4 Dewi] -> [#1 Andi] -> [#2 Budi] -> [#3 Citra] -> NULL   count=4
total Rp 107000
find Citra: #3, Rp 24000
find Eko: not in the list
remove #4: done  head -> [#1 Andi] -> [#2 Budi] -> [#3 Citra] -> NULL   count=3
remove #2: done  head -> [#1 Andi] -> [#3 Citra] -> NULL   count=2
remove #3: done  head -> [#1 Andi] -> NULL   count=1
remove #9: no such order  head -> [#1 Andi] -> NULL   count=1
head -> [#1 Andi] -> [#5 Eko] -> NULL   count=2
head -> NULL   count=0
```

Ikuti keluarannya sambil memegang **Gambar 10.2**. Senarai kosong tercetak sebagai `head -> NULL`. Tiga penambahan di akhir membangun senarai bagian (a). Pelanggan VIP Dewi masuk di depan. Menghapus #4 menghapus simpul *pertama* (kasus `prev == NULL`); menghapus #2 menghapus simpul tengah, seperti bagian (c); menghapus #3 menghapus simpul *terakhir*, dan `tail` harus mundur ke Andi. Baris sesudahnya adalah buktinya: menambahkan Eko sesudah itu menempatkannya tepat sesudah Andi. Seandainya `remove_id` lupa memindahkan `tail`, Eko akan disambungkan di belakang simpul #3 yang sudah dibebaskan, dan *sanitizer* akan menghentikan program dengan laporan *heap-use-after-free*. Menghapus #9, yang tidak ada, tidak mengubah apa pun. Akhirnya `free_all` mengembalikan setiap simpul, dan pemeriksaan kebocoran oleh penguji menegaskan tidak ada yang tertinggal.

Versi Python, **Kode 10.8**, sengaja *tidak* memakai *list* bawaan Python. Ia membangun rantai objek `Node` yang sama dengan atribut `next` dan `None` sebagai pengganti `NULL`, agar Anda melihat algoritma yang sama dalam kedua bahasa. `free_all` hanya mengatur `head` dan `tail` menjadi `None`; begitu tidak ada variabel yang merujuk simpul-simpul itu, Python membebaskannya.

Kode 10.8 `order_list.py`: *linked list* yang sama dalam Python, dibangun dari objek `Node`.

```python
# order_list.py - the same singly linked list, built from Node objects that
# point to the next Node (Python's own list is NOT used, on purpose).


class Node:
    def __init__(self, id, customer, amount):
        self.id = id
        self.customer = customer[:23]
        self.amount = amount
        self.next = None             # the next node, or None after the last


class List:
    def __init__(self):
        self.head = None
        self.tail = None
        self.count = 0


def push_front(l, n):
    n.next = l.head
    l.head = n
    if l.tail is None:
        l.tail = n
    l.count += 1


def append(l, n):
    if l.tail is None:
        l.head = n
    else:
        l.tail.next = n
    l.tail = n
    l.count += 1


def print_list(l):
    s, p = "head", l.head
    while p is not None:
        s += " -> [#%d %s]" % (p.id, p.customer)
        p = p.next
    print(s + " -> NULL   count=%d" % l.count)
```

Kode 10.8 (Lanjutan) *order_list.py: linked list yang sama dalam Python, dibangun dari objek Node.*

```
def find(l, customer):
    p = l.head
    while p is not None:
        if p.customer == customer:
            return p
        p = p.next
    return None


def remove_id(l, id):
    prev, cur = None, l.head
    while cur is not None and cur.id != id:
        prev, cur = cur, cur.next
    if cur is None:
        return False
    if prev is None:
        l.head = cur.next
    else:
        prev.next = cur.next
    if l.tail is cur:
        l.tail = prev
    l.count -= 1                        # Python frees the node for us
    return True


def free_all(l):
    l.head = l.tail = None              # no node is reachable any more
    l.count = 0


def main():
    queue = List()
    print_list(queue)
    append(queue, Node(1, "Andi", 24500))
    append(queue, Node(2, "Budi", 20000))
    append(queue, Node(3, "Citra", 24000))
    print_list(queue)
    push_front(queue, Node(4, "Dewi", 38500))
    print_list(queue)

    total, p = 0, queue.head
    while p is not None:
        total += p.amount
        p = p.next
    print("total Rp %d" % total)
```

Kode 10.8 (Lanjutan) *order_list.py: linked list yang sama dalam Python, dibangun dari objek Node.*

```
    for who in ["Citra", "Eko"]:
        f = find(queue, who)
        if f is not None:
            print("find %s: #%d, Rp %d" % (who, f.id, f.amount))
        else:
            print("find %s: not in the list" % who)

    for id in [4, 2, 3, 9]:
        ok = remove_id(queue, id)
        print("remove #%d: %s  " % (id, "done" if ok else "no such order"), end="")
        print_list(queue)
    append(queue, Node(5, "Eko", 14000))
    print_list(queue)
    free_all(queue)
    print_list(queue)


if __name__ == "__main__":
    main()
```

10.7 Membangun Kasir v0.8

Kasir v0.8 menggabungkan semua bagian. Sebuah pesanan adalah satu struktur:

- `id`, nomor pesanan, diberikan toko secara menaik dan tidak pernah dipakai ulang;
- `customer`, nama yang sudah dirapikan (Bab 9);
- `service`, bertipe `Service`, dan `tenths`, beratnya;
- `amount`, dihitung sekali ketika pesanan diterima;
- `status`, bertipe `Status`: `WAITING`, `WASHING`, atau `READY`;
- `next`, pointer yang merangkai pesanan-pesanan di toko menjadi senarai.

Semua pesanan yang sedang ada di toko membentuk satu *linked list*, disimpan dalam struktur `Shop` bersama pencacah nomor pesanan berikutnya dan penerimaan hari itu. Perintah-perintahnya:

- `add <service> <kg> <name>` membuat pesanan dengan `malloc`, mengisinya, dan menambahkannya di akhir senarai, seperti di Bab 9 tetapi tanpa batas banyaknya pesanan;
- `advance <id>` memajukan pesanan satu langkah, dari menunggu ke dicuci atau dari dicuci ke siap (id boleh ditulis `3` atau `#3`);
- `pickup <id>` menyerahkan pesanan yang *siap* kepada pelanggan: pesanan dilepas dari senarai, jumlahnya ditambahkan ke penerimaan, dan memorinya dibebaskan;
- `list` menampilkan semua pesanan di toko beserta statusnya, dan `quit` menutup hari, membebaskan setiap pesanan yang tersisa, dan mencetak penerimaan.

Pembantu-pembantu teks dari Bab 9 disalin tanpa perubahan ke dalam program; Bab 11 akan memindahkannya ke berkas tersendiri agar hanya ditulis sekali. Semua bagian lain **Kode 10.9** adalah baru.

Kode 10.9 `cashier_v8.c`: kasir Profitina Laundry v0.8, dengan satu struktur per pesanan dan pesanan di toko disimpan sebagai *linked list*.

```c
/* cashier_v8.c - Profitina Laundry cashier v0.8: every order is one struct,
 * the orders in the shop form a linked list, and an order moves through
 * waiting -> washing -> ready until the customer picks it up.
 * Usage: cashier_v8 [--echo]    (--echo repeats each command after "> ") */
#include <ctype.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define NAME_LEN 32
#define LINE_LEN 80

typedef enum { FOLD, WASH_IRON, IRON_ONLY, N_SERVICES } Service;
typedef enum { WAITING, WASHING, READY } Status;

static const struct {
    const char *name;
    int price;                                  /* rupiah per kg */
} SERVICES[N_SERVICES] = {
    [FOLD] = {"fold", 7000},
    [WASH_IRON] = {"wash-iron", 10000},
    [IRON_ONLY] = {"iron", 6000},
};
static const char *STATUS_NAME[] = {
    [WAITING] = "waiting", [WASHING] = "washing", [READY] = "ready"};

typedef struct Order {
    int id;
    char customer[NAME_LEN];
    Service service;
    int tenths;                                 /* weight in tenths of a kg */
    long amount;                                /* rupiah */
    Status status;
    struct Order *next;
} Order;

typedef struct {
    Order *head, *tail;                         /* the orders in the shop */
    int count, next_id;
    int picked;                                 /* orders picked up today */
    long takings;                               /* rupiah received today */
} Shop;
```

Kode 10.9 (Lanjutan) *cashier_v8.c: kasir Profitina Laundry v0.8, dengan satu struktur per pesanan dan pesanan di toko disimpan sebagai linked list.*

```c
/* ---- text helpers from Chapter 9 ----------------------------------- */
static int read_line(char *buf, int size)
{
    if (fgets(buf, size, stdin) == NULL)
        return 0;
    size_t n = strlen(buf);
    if (n > 0 && buf[n - 1] == '\n') {
        buf[--n] = '\0';
    } else if (n == (size_t)size - 1) {
        int c = getchar();
        if (c != EOF && c != '\n') {
            while (c != EOF && c != '\n')
                c = getchar();
            return -1;
        }
    }
    if (n > 0 && buf[n - 1] == '\r')
        buf[--n] = '\0';
    return 1;
}

static size_t next_word(const char **p, char *out, size_t size)
{
    const char *s = *p;
    size_t n = 0;
    while (isspace((unsigned char)*s))
        s++;
    while (*s != '\0' && !isspace((unsigned char)*s)) {
        if (n + 1 < size)
            out[n] = *s;
        n++;
        s++;
    }
    out[n < size ? n : size - 1] = '\0';
    *p = s;
    return n;
}
```

Kode 10.9 (Lanjutan) *cashier_v8.c: kasir Profitina Laundry v0.8, dengan satu struktur per pesanan dan pesanan di toko disimpan sebagai linked list.*

```c
static void tidy_name(const char *src, char *dst, size_t size)
{
    size_t n = 0;
    int new_word = 1;
    for (size_t i = 0; src[i] != '\0' && n + 1 < size; i++) {
        unsigned char c = (unsigned char)src[i];
        if (isspace(c)) {
            new_word = 1;
            continue;
        }
        if (new_word && n > 0 && n + 2 < size)
            dst[n++] = ' ';
        dst[n++] = (char)(new_word ? toupper(c) : tolower(c));
        new_word = 0;
    }
    dst[n] = '\0';
}

static int parse_tenths(const char *s, int *out)
{
    int whole = 0, digits = 0, tenth = 0;
    for (; isdigit((unsigned char)*s); s++) {
        if (++digits > 3)
            return 0;
        whole = whole * 10 + (*s - '0');
    }
    if (digits == 0)
        return 0;
    if (*s == '.') {
        if (!isdigit((unsigned char)s[1]))
            return 0;
        tenth = s[1] - '0';
        s += 2;
    }
    if (*s != '\0')
        return 0;
    *out = whole * 10 + tenth;
    return 1;
}
```

Kode 10.9 (Lanjutan) *cashier_v8.c: kasir Profitina Laundry v0.8, dengan satu struktur per pesanan dan pesanan di toko disimpan sebagai linked list.*

```c
static void rupiah(long n, char *out)          /* out: 32 bytes */
{
    char digits[24];
    int len = snprintf(digits, sizeof digits, "%ld", n), k = 0;
    for (int i = 0; i < len; i++) {
        if (i > 0 && (len - i) % 3 == 0)
            out[k++] = '.';
        out[k++] = digits[i];
    }
    out[k] = '\0';
}

/* ---- the list of orders ------------------------------------------- */
static Order *find_id(const Shop *shop, int id)
{
    for (Order *o = shop->head; o != NULL; o = o->next)
        if (o->id == id)
            return o;
    return NULL;
}

static void append(Shop *shop, Order *o)
{
    if (shop->tail == NULL)
        shop->head = o;
    else
        shop->tail->next = o;
    shop->tail = o;
    shop->count++;
}

static Order *unlink_id(Shop *shop, int id)     /* take out, do not free */
{
    Order *prev = NULL, *cur = shop->head;
    while (cur != NULL && cur->id != id) {
        prev = cur;
        cur = cur->next;
    }
    if (cur == NULL)
        return NULL;
    if (prev == NULL)
        shop->head = cur->next;
    else
        prev->next = cur->next;
    if (shop->tail == cur)
        shop->tail = prev;
```

Kode 10.9 (Lanjutan) *cashier_v8.c: kasir Profitina Laundry v0.8, dengan satu struktur per pesanan dan pesanan di toko disimpan sebagai linked list.*

```
    cur->next = NULL;
    shop->count--;
    return cur;
}

static void free_all(Shop *shop)
{
    while (shop->head != NULL) {
        Order *next = shop->head->next;
        free(shop->head);
        shop->head = next;
    }
    shop->tail = NULL;
    shop->count = 0;
}

static void print_order(const Order *o)
{
    char m[32];
    rupiah(o->amount, m);
    printf("  #%d %s, %s, %d.%d kg = Rp %s (%s)\n", o->id, o->customer,
           SERVICES[o->service].name, o->tenths / 10, o->tenths % 10, m,
           STATUS_NAME[o->status]);
}

/* ---- commands ------------------------------------------------------- */
static void cmd_add(Shop *shop, const char *rest)
{
    char svc[16], kg[16], name[NAME_LEN];
    int s = -1, t = 0;
    if (next_word(&rest, svc, sizeof svc) == 0 ||
        next_word(&rest, kg, sizeof kg) == 0) {
        puts("  usage: add <service> <kg> <name>");
        return;
    }
    for (int i = 0; i < N_SERVICES; i++)
        if (strcmp(svc, SERVICES[i].name) == 0)
            s = i;
    if (s < 0) {
        printf("  unknown service '%s' (fold, wash-iron, iron)\n", svc);
        return;
    }
    if (!parse_tenths(kg, &t) || t < 1 || t > 500) {
        puts("  weight must look like 3 or 3.5 (0.1 to 50.0 kg)");
        return;
    }
```

Kode 10.9 (Lanjutan) *cashier_v8.c: kasir Profitina Laundry v0.8, dengan satu struktur per pesanan dan pesanan di toko disimpan sebagai linked list.*

```
    tidy_name(rest, name, sizeof name);
    if (name[0] == '\0') {
        puts("  the customer's name is missing");
        return;
    }
    Order *o = malloc(sizeof *o);
    if (o == NULL) {
        puts("  out of memory - order not taken");
        return;
    }
    o->id = shop->next_id++;
    strcpy(o->customer, name);
    o->service = (Service)s;
    o->tenths = t;
    o->amount = (long)t * SERVICES[s].price / 10;
    o->status = WAITING;
    o->next = NULL;
    append(shop, o);
    print_order(o);
}

/* Reads "#3" or "3" from rest; returns the id, or 0 if there is none. */
static int read_id(const char *rest)
{
    char word[16];
    const char *w = word;
    if (next_word(&rest, word, sizeof word) == 0)
        return 0;
    if (*w == '#')
        w++;
    int id = 0, digits = 0;
    for (; isdigit((unsigned char)*w); w++)
        if (++digits <= 6)
            id = id * 10 + (*w - '0');
    return (*w == '\0' && digits > 0 && digits <= 6) ? id : 0;
}
```

Kode 10.9 (Lanjutan) *cashier_v8.c: kasir Profitina Laundry v0.8, dengan satu struktur per pesanan dan pesanan di toko disimpan sebagai linked list.*

```c
static void cmd_advance(Shop *shop, const char *rest)
{
    int id = read_id(rest);
    Order *o = id ? find_id(shop, id) : NULL;
    if (o == NULL)
        printf("  no order #%d in the shop\n", id);
    else if (o->status == READY)
        printf("  #%d is already ready - use pickup\n", id);
    else {
        o->status = (Status)(o->status + 1);
        print_order(o);
    }
}

static void cmd_pickup(Shop *shop, const char *rest)
{
    int id = read_id(rest);
    Order *o = id ? find_id(shop, id) : NULL;
    char m[32];
    if (o == NULL) {
        printf("  no order #%d in the shop\n", id);
        return;
    }
    if (o->status != READY) {
        printf("  #%d is still %s\n", id, STATUS_NAME[o->status]);
        return;
    }
    unlink_id(shop, id);
    rupiah(o->amount, m);
    printf("  #%d %s picked up, paid Rp %s\n", o->id, o->customer, m);
    shop->picked++;
    shop->takings += o->amount;
    free(o);
}

static void cmd_list(const Shop *shop)
{
    char m[32];
    printf("  %3s  %-20s  %-9s  %7s  %8s  %s\n",
           "No", "Customer", "Service", "Weight", "Amount", "Status");
    for (const Order *o = shop->head; o != NULL; o = o->next) {
        rupiah(o->amount, m);
        printf("  %3d  %-20.20s  %-9s  %2d.%d kg  %8s  %s\n", o->id,
               o->customer, SERVICES[o->service].name, o->tenths / 10,
               o->tenths % 10, m, STATUS_NAME[o->status]);
    }
```

Kode 10.9 (Lanjutan) *cashier_v8.c: kasir Profitina Laundry v0.8, dengan satu struktur per pesanan dan pesanan di toko disimpan sebagai linked list.*

```
    printf("  %d order(s) in the shop\n", shop->count);
}

int main(int argc, char *argv[])
{
    int echo = argc > 1 && strcmp(argv[1], "--echo") == 0;
    char line[LINE_LEN], cmd[16], m[32];
    Shop shop = {NULL, NULL, 0, 1, 0, 0};

    puts("Profitina Laundry cashier v0.8 - add, advance, pickup, list, quit");
    for (;;) {
        printf("> ");
        fflush(stdout);
        int r = read_line(line, sizeof line);
        if (r == 0) {
            printf("\n");
            break;
        }
        if (echo)
            printf(r < 0 ? "%.40s...\n" : "%s\n", line);
        if (r < 0) {
            puts("  line too long (at most 79 characters), ignored");
            continue;
        }
        const char *rest = line;
        if (next_word(&rest, cmd, sizeof cmd) == 0)
            continue;
        if (strcmp(cmd, "add") == 0)
            cmd_add(&shop, rest);
        else if (strcmp(cmd, "advance") == 0)
            cmd_advance(&shop, rest);
        else if (strcmp(cmd, "pickup") == 0)
            cmd_pickup(&shop, rest);
        else if (strcmp(cmd, "list") == 0)
            cmd_list(&shop);
        else if (strcmp(cmd, "quit") == 0)
            break;
        else
            printf("  unknown command '%s'\n", cmd);
    }
    rupiah(shop.takings, m);
    printf("Bye. %d picked up (Rp %s), %d still in the shop.\n",
           shop.picked, m, shop.count);
    free_all(&shop);
    return 0;
}
```

Beberapa detail layak dibaca perlahan. Tabel layanan adalah larik bertipe struktur *anonim*, dideklarasikan dan diinisialisasi dalam satu pernyataan dengan *designated initializer*. `unlink_id` adalah `remove_id` dari Subbab 10.6 tanpa `free`: ia mengembalikan pesanan yang dilepas agar `cmd_pickup` masih dapat mencetak nama dan jumlahnya serta menambahkannya ke penerimaan *sebelum* membebaskannya; membebaskan dulu lalu mencetak adalah *use after free*. `cmd_advance` mengubah status dengan `(Status)(o->status + 1)`, yang mengandalkan status-status berurutan, dan pemeriksaan `o->status == READY` sebelumnya memastikan nilainya tidak pernah keluar dari enumerasi. Dan `main` diakhiri `free_all(&shop)`, sehingga akhir hari yang normal tidak meninggalkan pesanan yang bocor.

Versi Python, **Kode 10.10**, berstruktur sama dengan kelas `Order` dan `Shop`, `IntEnum` untuk layanan dan status, serta *linked list* yang sama.

Kode 10.10 `cashier_v8.py`: kasir v0.8 dalam Python.

```
# cashier_v8.py - Profitina Laundry cashier v0.8: every order is one object,
# the orders in the shop form a linked list, and an order moves through
# waiting -> washing -> ready until the customer picks it up.
# Usage: python3 cashier_v8.py [--echo]
import sys
from enum import IntEnum

NAME_LEN = 32
LINE_LEN = 80


class Service(IntEnum):
    FOLD = 0
    WASH_IRON = 1
    IRON_ONLY = 2


class Status(IntEnum):
    WAITING = 0
    WASHING = 1
    READY = 2


SERVICES = {Service.FOLD: ("fold", 7000), Service.WASH_IRON: ("wash-iron", 10000),
            Service.IRON_ONLY: ("iron", 6000)}
STATUS_NAME = {Status.WAITING: "waiting", Status.WASHING: "washing",
               Status.READY: "ready"}


class Order:
    def __init__(self, id, customer, service, tenths):
        self.id = id
        self.customer = customer
        self.service = service
        self.tenths = tenths
        self.amount = tenths * SERVICES[service][1] // 10
        self.status = Status.WAITING
        self.next = None
```

Kode 10.10 (Lanjutan) *cashier_v8.py: kasir v0.8 dalam Python.*

```
class Shop:
    def __init__(self):
        self.head = self.tail = None
        self.count = 0
        self.next_id = 1
        self.picked = 0
        self.takings = 0


# ---- text helpers from Chapter 9 ---------------------------------------
def read_line():
    line = sys.stdin.readline()
    if line == "":
        return 0, ""
    line = line[:-1] if line.endswith("\n") else line
    if len(line) > LINE_LEN - 1:
        return -1, line[:LINE_LEN - 1]
    return 1, line[:-1] if line.endswith("\r") else line


def tidy_name(src, size=NAME_LEN):
    return " ".join(w[0].upper() + w[1:].lower() for w in src.split())[:size - 1]


def parse_tenths(s):
    whole, dot, tenth = s.partition(".")
    if not (whole.isascii() and whole.isdigit() and len(whole) <= 3):
        return None
    if dot and not (len(tenth) == 1 and tenth.isdigit()):
        return None
    return int(whole) * 10 + (int(tenth) if dot else 0)


def rupiah(n):
    digits, out = str(n), ""
    for i, d in enumerate(digits):
        if i > 0 and (len(digits) - i) % 3 == 0:
            out += "."
        out += d
    return out
```

Kode 10.10 (Lanjutan) *cashier_v8.py: kasir v0.8 dalam Python.*

```
# ---- the list of orders --------------------------------------------------
def find_id(shop, id):
    o = shop.head
    while o is not None:
        if o.id == id:
            return o
        o = o.next
    return None


def append(shop, o):
    if shop.tail is None:
        shop.head = o
    else:
        shop.tail.next = o
    shop.tail = o
    shop.count += 1


def unlink_id(shop, id):
    prev, cur = None, shop.head
    while cur is not None and cur.id != id:
        prev, cur = cur, cur.next
    if cur is None:
        return None
    if prev is None:
        shop.head = cur.next
    else:
        prev.next = cur.next
    if shop.tail is cur:
        shop.tail = prev
    cur.next = None
    shop.count -= 1
    return cur


def print_order(o):
    print("  #%d %s, %s, %d.%d kg = Rp %s (%s)"
          % (o.id, o.customer, SERVICES[o.service][0], o.tenths // 10,
             o.tenths % 10, rupiah(o.amount), STATUS_NAME[o.status]))
```

Kode 10.10 (Lanjutan) *cashier_v8.py: kasir v0.8 dalam Python.*

```
# ---- commands --------------------------------------------------------
def cmd_add(shop, rest):
    words = rest.split(None, 2)
    if len(words) < 2:
        print("  usage: add <service> <kg> <name>")
        return
    svc, kg = words[0][:15], words[1][:15]
    names = [v[0] for v in SERVICES.values()]
    if svc not in names:
        print("  unknown service '%s' (fold, wash-iron, iron)" % svc)
        return
    t = parse_tenths(kg)
    if t is None or not 1 <= t <= 500:
        print("  weight must look like 3 or 3.5 (0.1 to 50.0 kg)")
        return
    name = tidy_name(words[2] if len(words) > 2 else "")
    if name == "":
        print("  the customer's name is missing")
        return
    o = Order(shop.next_id, name, Service(names.index(svc)), t)
    shop.next_id += 1
    append(shop, o)
    print_order(o)


def read_id(rest):
    words = rest.split()
    if not words:
        return 0
    w = words[0][:15]
    w = w[1:] if w.startswith("#") else w
    ok = w.isascii() and w.isdigit() and len(w) <= 6
    return int(w) if ok else 0


def cmd_advance(shop, rest):
    id = read_id(rest)
    o = find_id(shop, id) if id else None
    if o is None:
        print("  no order #%d in the shop" % id)
    elif o.status == Status.READY:
        print("  #%d is already ready - use pickup" % id)
    else:
        o.status = Status(o.status + 1)
        print_order(o)
```

Kode 10.10 (Lanjutan) *cashier_v8.py: kasir v0.8 dalam Python.*

```
def cmd_pickup(shop, rest):
    id = read_id(rest)
    o = find_id(shop, id) if id else None
    if o is None:
        print("  no order #%d in the shop" % id)
        return
    if o.status != Status.READY:
        print("  #%d is still %s" % (id, STATUS_NAME[o.status]))
        return
    unlink_id(shop, id)
    print("  #%d %s picked up, paid Rp %s" % (o.id, o.customer, rupiah(o.amount)))
    shop.picked += 1
    shop.takings += o.amount


def cmd_list(shop):
    print("  %3s  %-20s  %-9s  %7s  %8s  %s"
          % ("No", "Customer", "Service", "Weight", "Amount", "Status"))
    o = shop.head
    while o is not None:
        print("  %3d  %-20.20s  %-9s  %2d.%d kg  %8s  %s"
              % (o.id, o.customer, SERVICES[o.service][0], o.tenths // 10,
                 o.tenths % 10, rupiah(o.amount), STATUS_NAME[o.status]))
        o = o.next
    print("  %d order(s) in the shop" % shop.count)


def main():
    echo = len(sys.argv) > 1 and sys.argv[1] == "--echo"
    shop = Shop()
    print("Profitina Laundry cashier v0.8 - add, advance, pickup, list, quit")
    while True:
        print("> ", end="", flush=True)
        r, line = read_line()
        if r == 0:
            print()
            break
        if echo:
            print(line[:40] + "..." if r < 0 else line)
        if r < 0:
            print("  line too long (at most 79 characters), ignored")
            continue
        parts = line.split(None, 1)
        if not parts:
            continue
```

Kode 10.10 (Lanjutan) *cashier_v8.py: kasir v0.8 dalam Python.*

```
        cmd, rest = parts[0][:15], parts[1] if len(parts) > 1 else ""
        if cmd == "add":
            cmd_add(shop, rest)
        elif cmd == "advance":
            cmd_advance(shop, rest)
        elif cmd == "pickup":
            cmd_pickup(shop, rest)
        elif cmd == "list":
            cmd_list(shop)
        elif cmd == "quit":
            break
        else:
            print("  unknown command '%s'" % cmd)
    print("Bye. %d picked up (Rp %s), %d still in the shop."
          % (shop.picked, rupiah(shop.takings), shop.count))


if __name__ == "__main__":
    main()
```

10.8 Menjalankan dan Mengujinya

Sesi dalam `data/session10.txt` menerima tiga pesanan, mencoba menyerahkan pesanan yang belum siap, memajukan pesanan, menyerahkan satu, lalu menerima pesanan baru:

```
add fold 3.5 andi pratama
add wash-iron 2 budi santoso
add iron 4 citra dewi lestari
list
pickup 2
advance 2
advance #2
advance 2
pickup 2
advance 1
advance 7
list
add fold 5 dewi
list
quit
```

`./cashier_v8 --echo < ../data/session10.txt` dan `python3 cashier_v8.py --echo < ../data/session10.txt` sama-sama mencetak:

```
Profitina Laundry cashier v0.8 - add, advance, pickup, list, quit
> add fold 3.5 andi pratama
  #1 Andi Pratama, fold, 3.5 kg = Rp 24.500 (waiting)
> add wash-iron 2 budi santoso
  #2 Budi Santoso, wash-iron, 2.0 kg = Rp 20.000 (waiting)
> add iron 4 citra dewi lestari
  #3 Citra Dewi Lestari, iron, 4.0 kg = Rp 24.000 (waiting)
> list
   No  Customer              Service     Weight    Amount  Status
    1  Andi Pratama          fold        3.5 kg    24.500  waiting
    2  Budi Santoso          wash-iron   2.0 kg    20.000  waiting
    3  Citra Dewi Lestari    iron        4.0 kg    24.000  waiting
  3 order(s) in the shop
> pickup 2
  #2 is still waiting
> advance 2
  #2 Budi Santoso, wash-iron, 2.0 kg = Rp 20.000 (washing)
> advance #2
  #2 Budi Santoso, wash-iron, 2.0 kg = Rp 20.000 (ready)
> advance 2
  #2 is already ready - use pickup
> pickup 2
  #2 Budi Santoso picked up, paid Rp 20.000
> advance 1
  #1 Andi Pratama, fold, 3.5 kg = Rp 24.500 (washing)
> advance 7
  no order #7 in the shop
> list
   No  Customer              Service     Weight    Amount  Status
    1  Andi Pratama          fold        3.5 kg    24.500  washing
    3  Citra Dewi Lestari    iron        4.0 kg    24.000  waiting
  2 order(s) in the shop
> add fold 5 dewi
  #4 Dewi, fold, 5.0 kg = Rp 35.000 (waiting)
> list
   No  Customer              Service     Weight    Amount  Status
    1  Andi Pratama          fold        3.5 kg    24.500  washing
    3  Citra Dewi Lestari    iron        4.0 kg    24.000  waiting
    4  Dewi                  fold        5.0 kg    35.000  waiting
  3 order(s) in the shop
> quit
Bye. 1 picked up (Rp 20.000), 3 still in the shop.
```

Periksalah terhadap aturannya. Pesanan Budi tidak dapat diambil selagi menunggu; dua perintah `advance` membuatnya siap (sekali sebagai `2`, sekali sebagai `#2`); yang ketiga ditolak. Setelah diambil, pesanan Budi keluar dari senarai, senarai menutup celahnya sendiri, dan pesanan baru Dewi mendapat nomor 4, bukan 2, karena nomor pesanan tidak pernah dipakai ulang: struk bernomor 2 masih ada di saku Budi. Hari berakhir dengan satu pesanan diambil senilai Rp20.000 dan tiga masih di toko.

Penguji bab ini (`python3 tests/run_tests.py`) juga menjalankan sesi ini di bawah AddressSanitizer. Versi `cmd_pickup` yang membebaskan pesanan sebelum mencetaknya, atau `free_all` yang lupa menyimpan `next`, menggagalkan uji dengan laporan tepat tentang baris yang memakai memori yang sudah dibebaskan, dan begitulah kesalahan seperti itu ditemukan dalam praktik: bukan dengan menatap kode, melainkan dengan menjalankannya di bawah alat yang mengawasi setiap akses memori.

10.9 Membuatnya Lebih Baik

Jaga senarai dalam urutan yang berguna. Di meja yang sibuk, pesanan kilat harus dicuci lebih dulu. Alih-alih selalu menambah pesanan di akhir, fungsi `insert_sorted` dapat menyusuri senarai

dan menyisipkan pesanan baru *sebelum* pesanan pertama yang janji waktunya lebih lambat. Ia memakai penelusuran dua pointer seperti `remove_id`: temukan simpul yang harus mendahului simpul baru (atau `NULL` untuk posisi depan), lalu `n->next = prev->next; prev->next = n;`. Soal Latihan 10.4 memintanya.

Temukan semua pesanan pelanggan, bukan satu saja. `find` berhenti pada kecocokan pertama. Untuk pelanggan yang membawa tiga kantong, penelusuran yang mencetak setiap kecocokan lebih berguna; dengan perulangan penelusuran, caranya cukup dengan tidak kembali lebih awal.

Jangan kehilangan hari itu. Semua yang diketahui v0.8 masih berada di memori. Listrik padam pukul empat sore menghapus setiap pesanan hari itu, dan Mbak Rina kembali ke buku kertasnya. Bab 11 menyimpan senarai ke berkas setelah setiap perubahan, perbaikan yang paling penting dari semuanya.

10.10 Memakainya secara Nyata

Struktur ditambah *linked list* adalah bentuk yang tepat setiap kali sesuatu datang dan pergi dalam urutan yang tidak dapat ditebak dan jumlahnya tidak dapat diketahui sebelumnya. Bengkel sepeda motor menyimpan motor yang menunggu perbaikan sebagai senarai struktur berisi pelat nomor, nama pemilik, keluhan, dan status (*menunggu*, *diperbaiki*, *menunggu suku cadang*, *siap*); motor pergi ketika siap, bukan menurut urutan datangnya. Posyandu desa menyimpan antrean pasien hari itu dengan jam kedatangan dan status, lalu memanggil pasien berikutnya dari kepala senarai. Kebun buah dan durian yang menerima pesanan lewat obrolan menyimpan setiap pesanan dengan pembeli, jenis buah, berat, dan status seperti *dipetik*, *dikemas*, *dikirim*. Dalam setiap kasus langkahnya sama dengan bab ini: tuliskan fakta-fakta yang saling terkait dan jadikan struktur; tuliskan keadaan-keadaannya sebagai enumerasi beserta perpindahan yang diizinkan; simpan rekaman yang hidup dalam senarai; dan beri setiap rekaman nomor yang tidak pernah dipakai ulang.

Contoh konkret membuat langkah-langkah itu terlihat. Untuk bengkel sepeda motor, strukturnya akan memuat pelat nomor sebagai string 12 karakter, nama pemilik 32 karakter, keluhan 80 karakter, enumerasi `BikeStatus` dengan empat keadaan, dan hari yang dijanjikan. Perintah `advance` pada kasir menjadi "langkah berikutnya" bagi montir, `pickup` menjadi "serahkan dan terima pembayaran", dan perintah baru, `parts <plate>`, memindahkan motor ke keadaan *menunggu suku cadang* dan kembali lagi. Hampir seluruh `cashier_v8.c` dapat dipertahankan: fungsi-fungsi senarai tidak peduli apa isi sebuah pesanan.

Saat menyesuaikan kasir, ingat selalu tiga aturan `malloc`, dan periksa setiap program di bawah *sanitizer* sebelum Anda memercayainya. Kebocoran memori kecil dalam program yang dijalankan ulang setiap pagi tidak berbahaya; kebocoran yang sama dalam program yang berjalan berbulan-bulan, seperti yang sering terjadi pada perangkat lunak toko, pada akhirnya menghentikannya.

10.11 Kesalahan yang Sering Terjadi, dan Cara Menghindarinya

- **Lupa titik koma setelah deklarasi struktur.** `struct Order { ... }` harus diakhiri `};`. Tanpanya, kompilator menggabungkan deklarasi dengan baris berikutnya dan melaporkan galat membingungkan di sana.
- **Memakai `.` pada pointer atau `->` pada struktur.** Dengan `struct Order o;` tulis `o.amount`; dengan `struct Order *p;` tulis `p->amount`. Menulis `*p.amount` berarti `*(p.amount)` dan tidak lolos kompilasi.

- **Tanpa sengaja memberikan struktur besar secara nilai.** Berfungsi, tetapi menyalin struktur pada setiap panggilan, dan perubahannya hilang. Berikan `const struct X *` untuk membaca dan `struct X *` untuk mengubah.
- **Membandingkan struktur dengan `memcmp`.** Byte *padding* membuat dua struktur yang sama tampak berbeda. Bandingkan anggota demi anggota.
- **Mengira-ngira ukuran struktur.** 38 bukan 40. Pakai `sizeof` dan `offsetof`.
- **Tidak memeriksa `malloc`.** Selalu uji hasilnya terhadap `NULL` sebelum menulis lewatnya.
- **Membebaskan lalu memakai.** Baca semua yang Anda perlukan dari simpul, lepaskan dari rantai, baru bebaskan; dan simpan `p->next` sebelum `free(p)` dalam perulangan.
- **Kehilangan senarai saat menyisipkan.** Di depan, atur `n->next = head` *sebelum* `head = n`; di tengah, atur `n->next = prev->next` *sebelum* `prev->next = n`.
- **Melupakan `tail`.** Setiap fungsi yang dapat menghapus simpul terakhir atau menambah ke senarai kosong harus memperbarui `tail` di samping `head`. Ujilah penghapusan simpul terakhir lalu penambahan di akhir, seperti dilakukan `order_list.c`.
- **Memercayai variabel `enum`.** Ia dapat memegang `int` apa saja. Periksa nilai yang berasal dari masukan atau berkas sebelum memakainya sebagai indeks.

10.12 Ringkasan Bab

- `struct` mengelompokkan anggota-anggota bernama bertipe berbeda menjadi satu tipe; anggota dicapai dengan `.` pada struktur dan `->` lewat pointer. Penugasan menyalin seluruh struktur, larik di dalamnya ikut; perbandingan harus ditulis anggota demi anggota.
- Anggota disimpan berurutan, dengan *padding* yang disisipkan agar masing-masing dimulai di kelipatan *alignment*-nya (Persamaan (10.1)); `sizeof` dan `offsetof` memberi tata letak sebenarnya (**Gambar 10.1**).
- Struktur biasanya diberikan lewat pointer: `const struct X *` untuk membaca, `struct X *` untuk mengubah. Larik struktur menggantikan larik paralel dan menjaga setiap rekaman tetap utuh, bahkan melewati `qsort`.
- `enum` menamai kode bilangan bulat; `typedef` menamai tipe; *designated initializer* membangun tabel yang diindeks nama enumerasi. Enumerasi tidak mencegah nilai yang salah.
- `malloc` memberi memori yang hidup sampai `free`. Periksa `NULL`, bebaskan tepat sekali, jangan pernah memakai setelah dibebaskan.
- Struktur yang merujuk diri dengan pointer `next` membangun *singly linked list*. Dengan pointer `head` dan `tail`, menambah di salah satu ujung perlu satu langkah, dan menghapus simpul perlu satu penelusuran dan satu penugasan (**Tabel 10.1**, **Gambar 10.2**).
- Kasir v0.8 menyimpan satu struktur per pesanan dalam *linked list*, memajukan pesanan melalui keadaan *waiting*, *washing*, dan *ready*, membebaskannya ketika diambil, dan lolos uji di bawah AddressSanitizer dalam C serta dengan keluaran identik dalam Python.

10.13 Soal Latihan

1. Gambarlah `struct Item { char code; int qty; char unit; double price; };` di memori komputer 64-bit dengan memakai Persamaan (10.1), dan berikan `sizeof`-nya. Kemudian urutkan ulang anggotanya agar sekecil mungkin dan berikan ukuran barunya.
2. Tulis `long total_waiting(const Shop *shop)` untuk kasir v0.8, yang mengembalikan total jumlah tagihan pesanan yang masih `WAITING`. Mengapa parameternya pointer `const`?

3. Perulangan `for (Node *p = head; p != NULL; p = p->next) free(p);` salah. Jelaskan langkah demi langkah apa yang terjadi untuk senarai berisi dua simpul, dan sebutkan aturan Subbab 10.5 yang dilanggarnya.
4. Tulis `void insert_sorted(List *l, Node *n)` untuk senarai dalam `order_list.c`, yang menjaga senarai terurut menurut jumlah tagihan, terkecil lebih dulu. Ujilah dengan menyisipkan pesanan 24500, 20000, 38500, dan 24000 ke senarai kosong dan mencetak senarai setelah setiap penyisipan. Kasus mana di antara "senarai kosong", "simpul pertama baru", "simpul terakhir baru", dan "tengah" yang diuji oleh setiap penyisipan?
5. Tulis fungsi yang membalik *linked list di tempat*, dengan memutar balik setiap pointer `next`, tanpa membuat atau menyalin simpul apa pun, dan ujilah pada senarai 10, 20, 30, 40, 50 dalam C dan Python.
6. Tambahkan nilai `status` `DELIVERED` pada kasir v0.8 untuk pesanan yang diantar ke rumah pelanggan alih-alih diambil. Daftarlah setiap tempat di `cashier_v8.c` yang harus berubah, dan sebutkan mana yang akan ditunjukkan kompilator dengan sendirinya.

10.14 Penyelesaian Soal Latihan

1. `code` menempati byte 0. `qty` memerlukan *alignment* 4: $o = 1$, $o' = 4\lceil 1/4 \rceil = 4$, jadi byte 1 sampai 3 adalah *padding* dan `qty` menempati 4 sampai 7. `unit` menempati byte 8. `price` memerlukan *alignment* 8: $o = 9$, $o' = 8\lceil 9/8 \rceil = 16$, jadi byte 9 sampai 15 adalah *padding* dan `price` menempati 16 sampai 23. Ukurannya 24, dan hanya 14 byte di antaranya data. Dengan urutan `price, qty, code, unit`, anggota-anggotanya menempati 8 + 4 + 1 + 1 = 14 byte di posisi 0, 8, 12, dan 13, dan ukurannya dibulatkan ke atas ke kelipatan 8 berikutnya, yaitu 16.

2. Fungsinya menyusuri senarai dan menjumlahkan:

```
long total_waiting(const Shop *shop)
{
    long sum = 0;
    for (const Order *o = shop->head; o != NULL; o = o->next)
        if (o->status == WAITING)
            sum += o->amount;
    return sum;
}
```

Pointer menghindari penyalinan struktur `Shop` (yang salinannya toh berbagi simpul yang sama), dan `const` mendokumentasikan serta menegakkan bahwa menghitung tidak mengubah toko. Dalam sesi Subbab 10.8, tepat sebelum `quit`, hasilnya 24.000 + 35.000 = 59.000 (Citra dan Dewi).

3. Dengan simpul A → B: `p` adalah A; `free(A)`; lalu `p = p->next` membaca anggota `next` milik A, yang memorinya baru saja dikembalikan. Ia mungkin masih memegang alamat B, sehingga program tampak berjalan, atau berisi apa saja, sehingga perulangan melompat ke alamat acak. Ini melanggar aturan 3, *jangan pernah memakai blok setelah dibebaskan*; *sanitizer* melaporkan *heap-use-after-free* pada baris `p = p->next`. Perbaikannya menyimpan `p->next` sebelum `free`, seperti dalam `free_all`.

4. Susuri dengan `prev` dan `cur` sampai `cur` bernilai `NULL` atau `cur->amount` lebih besar daripada `n->amount`, lalu sambungkan `n` di antara keduanya:

```
void insert_sorted(List *l, Node *n)
{
    Node *prev = NULL, *cur = l->head;
    while (cur != NULL && cur->amount <= n->amount) {
        prev = cur;
        cur = cur->next;
    }
    n->next = cur;
    if (prev == NULL)
        l->head = n;
    else
        prev->next = n;
    if (cur == NULL)
        l->tail = n;
    l->count++;
}
```

24500 masuk ke senarai kosong (ia sekaligus `head` dan `tail`); 20000 menjadi simpul pertama baru; 38500 simpul terakhir baru; 24000 masuk di tengah, di antara 20000 dan 24500. Memakai `<=` dalam perulangan menempatkan pesanan baru *sesudah* pesanan yang jumlahnya sama, sehingga pesanan berjumlah sama tetap dalam urutan kedatangannya.

5. Pakai tiga pointer: bagian yang sudah dibalik dimulai di `prev`, simpul yang sedang diputar adalah `head`, dan penerusnya disimpan sebelum pointernya diputar. **Kode 10.11** menunjukkan fungsinya dan **Kode 10.12** versi Python-nya; membalik dua kali mengembalikan senarai semula, dan membalik senarai kosong mengembalikan `NULL`.

Kode 10.11 `list_reverse.c`: membalik *linked list* di tempat.

```c
/* list_reverse.c - Exercise 10.5: reverse a linked list in place by turning
 * every next pointer around; no node is copied or allocated. */
#include <stdio.h>
#include <stdlib.h>

typedef struct Node {
    int value;
    struct Node *next;
} Node;

static Node *push_front(Node *head, int value)
{
    Node *n = malloc(sizeof *n);
    if (n == NULL)
        exit(EXIT_FAILURE);
    n->value = value;
    n->next = head;
    return n;
}

static void print(const char *label, const Node *p)
{
    printf("%-9s", label);
    for (; p != NULL; p = p->next)
        printf("%d -> ", p->value);
    printf("NULL\n");
}

static Node *reverse(Node *head)
{
    Node *prev = NULL;
    while (head != NULL) {
        Node *next = head->next;   /* 1: remember the rest of the list */
        head->next = prev;         /* 2: turn this arrow around */
        prev = head;               /* 3: this node is now the new front */
        head = next;               /* 4: go on with the rest */
    }
    return prev;
}
```

Kode 10.11 (Lanjutan) *list_reverse.c: membalik linked list di tempat.*

```c
int main(void)
{
    Node *head = NULL;
    for (int v = 50; v >= 10; v -= 10)
        head = push_front(head, v);
    print("before:", head);
    head = reverse(head);
    print("after:", head);
    head = reverse(head);
    print("again:", head);
    print("empty:", reverse(NULL));
    while (head != NULL) {
        Node *next = head->next;
        free(head);
        head = next;
    }
    return 0;
}
```

Kode 10.12 `list_reverse.py`: hal yang sama dalam Python.

```python
# list_reverse.py - Exercise 10.5: reverse a linked list in place.


class Node:
    def __init__(self, value, next=None):
        self.value = value
        self.next = next


def show(label, p):
    s = "%-9s" % label
    while p is not None:
        s += "%d -> " % p.value
        p = p.next
    print(s + "NULL")


def reverse(head):
    prev = None
    while head is not None:
        nxt = head.next            # 1: remember the rest of the list
        head.next = prev           # 2: turn this arrow around
        prev = head                # 3: this node is now the new front
        head = nxt                 # 4: go on with the rest
    return prev


def main():
    head = None
    for v in range(50, 0, -10):
        head = Node(v, head)
    show("before:", head)
    head = reverse(head)
    show("after:", head)
    head = reverse(head)
    show("again:", head)
    show("empty:", reverse(None))


if __name__ == "__main__":
    main()
```

Keduanya mencetak:

```
before:  10 -> 20 -> 30 -> 40 -> 50 -> NULL
after:   50 -> 40 -> 30 -> 20 -> 10 -> NULL
again:   10 -> 20 -> 30 -> 40 -> 50 -> NULL
empty:   NULL
```

6. Tambahkan `DELIVERED` ke `Status` sesudah `READY`; tambahkan `[DELIVERED] = "delivered"` ke `STATUS_NAME`; tambahkan perintah `deliver <id>` yang, seperti `pickup`, hanya menerima pesanan `READY`, melepaskan dan membebaskannya, serta menghitungnya; dan pastikan `cmd_advance` tetap berhenti di `READY` (ujinya, `o->status == READY`, sudah melakukannya). Kompilator tidak menunjukkan satu pun dari hal ini dengan sendirinya dalam program ini, karena `STATUS_NAME` adalah larik, bukan `switch`; `switch` atas `Status` tanpa `default`, seperti dalam `service_enum.c`, akan membuat `-Wall` memperingatkan kasus yang terlewat. Itu alasan yang baik untuk mengubah nilai enumerasi menjadi nama dengan `switch`.

Daftar Pustaka

[1] ISO/IEC 9899:2018, *Information technology — Programming languages — C* (C17). Geneva, Switzerland: ISO, 2018, §6.7.2.1 “Structure and union specifiers”, §6.7.2.2 “Enumeration specifiers”, §7.22.3 “Memory management functions”.

[2] Python Software Foundation, “dataclasses — Data Classes,” *The Python Standard Library*, Python 3.14. [Online]. Available: https://docs.python.org/3/library/dataclasses.html (accessed Sep. 2026).

[3] Python Software Foundation, “ctypes — A foreign function library for Python,” *The Python Standard Library*, Python 3.14. [Online]. Available: https://docs.python.org/3/library/ctypes.html (accessed Sep. 2026).

[4] B. W. Kernighan and D. M. Ritchie, *The C Programming Language*, 2nd ed. Englewood Cliffs, NJ, USA: Prentice Hall, 1988, ch. 6 “Structures” and appendix B5.

[5] Python Software Foundation, “enum — Support for enumerations,” *The Python Standard Library*, Python 3.14. [Online]. Available: https://docs.python.org/3/library/enum.html (accessed Sep. 2026).

[6] MITRE, “2025 CWE Top 25 Most Dangerous Software Weaknesses,” Dec. 2025. [Online]. Available: https://cwe.mitre.org/top25/archive/2025/2025_cwe_top25.html

[7] Software Engineering Institute, Carnegie Mellon University, *SEI CERT C Coding Standard*, rules MEM30-C “Do not access freed memory” and MEM31-C “Free dynamically allocated memory when no longer needed.” [Online]. Available: https://wiki.sei.cmu.edu/confluence/display/c (accessed Sep. 2026).

[8] P. J. Deitel and H. M. Deitel, *C How to Program*, 9th ed. Hoboken, NJ, USA: Pearson, 2022, ch. 10 “Structures, Unions, Bit Manipulation and Enumerations” and ch. 12 “Data Structures.”

Bab 11

Pemrosesan Berkas: Data yang Bertahan Saat Listrik Padam

Segala sesuatu yang diketahui program kita sejauh ini hidup di memori kerja komputer, dan memori kerja melupakan semuanya begitu program berakhir atau listrik padam. Bab ini mengajari program untuk mengingat. Di akhir bab Anda akan mampu menjelaskan apa itu berkas, aliran (*stream*), dan `FILE *`, serta memilih mode yang tepat saat membuka berkas; menulis dan membaca berkas teks dengan `fprintf`, `fscanf`, dan `fgets`, dan menulis perulangan baca yang berhenti tepat pada saatnya; membaca berkas data CSV baris demi baris, melewati baris rusak dengan pesan yang jelas, dan membuat laporan dari sisanya; menambah isi berkas log tanpa pernah menimpanya; menyimpan rekaman berukuran tetap dalam berkas biner dan langsung melompat ke salah satunya dengan `fseek`; menyimpan data sedemikian rupa sehingga listrik padam kapan pun hanya meninggalkan berkas lama atau berkas baru yang utuh, tidak pernah berkas yang rusak; serta memecah program menjadi beberapa berkas sumber dengan *header*. Bab ini ditutup dengan kasir v0.9, yang menyimpan pesanan laundry setelah setiap perubahan dan menemukan semuanya kembali setelah listrik menyala lagi.

11.1 Posisi Kita Sekarang: Pukul Empat, dan Listrik Padam

Kasir v0.8 dari Bab 10 melakukan semua yang diminta Mbak Rina: nama yang rapi, pesanan yang berpindah dari *waiting* ke *washing* ke *ready*, dan senarai yang tumbuh sepanjang hari. Pada pukul empat di hari Sabtu yang sibuk, ia memegang tiga puluh pesanan. Lalu listrik padam. Pemadaman listrik adalah kejadian sehari-hari di banyak daerah di Indonesia, kadang diumumkan lebih dulu untuk pemeliharaan, kadang tidak. Baterai laptop sudah rusak sejak tahun lalu, layar menjadi hitam, dan ketika listrik menyala sepuluh menit kemudian dan kasir dijalankan lagi, ia menyambut Mbak Rina dengan senarai kosong. Tiga puluh pesanan, tiga puluh pelanggan yang akan kembali membawa nomor struk, dan satu-satunya catatan ada di memori program yang sudah tidak berjalan.

Alasannya bersifat fisik. Memori kerja komputer, RAM, hanya menyimpan isinya selama ada listrik, dan variabel, larik, serta *linked list* sebuah program semuanya tinggal di sana. Data yang harus bertahan lebih lama daripada programnya harus ditulis ke *penyimpanan*: *solid-state drive*, *hard disk*, atau kartu memori, yang menyimpan isinya tanpa listrik. Program memakai penyimpanan lewat **berkas** (*file*): deretan byte bernama yang dijaga sistem operasi untuk mereka.

Menulis ke berkas itu mudah; paruh kedua bab ini menunjukkan bahwa menulis ke berkas secara *aman* memerlukan sedikit pemikiran. Bila kasir menulis pesanannya ke berkas hanya ketika Mbak Rina mengetik `quit`, listrik padam pada sore hari tetap menghapus seluruh hari. Bila ia menulis ulang berkas setelah setiap pesanan, listrik padam *di tengah* proses tulis ulang dapat meninggalkan berkas yang setengah lama setengah baru, atau kosong. Tujuan bab ini adalah kasir yang tidak mungkin mengalami keduanya.

11.2 Gagasannya dengan Kata-kata Sederhana: Berkas dan Aliran

11.2.1 Aliran dan pointer `FILE`

C tidak membiarkan program menyentuh disk secara langsung. Program berbicara dengan berkas melalui **aliran** (*stream*): aliran adalah deretan byte yang mengalir masuk ke program (masukan) atau keluar darinya (keluaran), dan program membaca atau menulisnya dari depan ke belakang, potongan demi potongan, seperti air dari selang. Anda sudah memakai tiga aliran sejak Bab 1 tanpa mengetahui namanya. Setiap program C dimulai dengan `stdin`, masukan standar (biasanya papan ketik), `stdout`, keluaran standar (biasanya layar), dan `stderr`, aliran galat standar, yang juga tampil di layar tetapi dipisahkan agar pesan galat tidak tercampur dengan keluaran yang disimpan atau disalurkan ke program lain. `printf(...)` tak lain adalah `fprintf(stdout, ...)`, dan `scanf(...)` adalah `fscanf(stdin, ...)` [1], [2].

Untuk memakai berkas, program **membukanya** dengan `fopen(name, mode)`, yang menghubungkan aliran baru ke berkas dan mengembalikan pointer ke objek bertipe `FILE`. Program tidak pernah melihat isi objek `FILE`; ia hanya memberikan pointernya kepada fungsi-fungsi masukan dan keluaran, seperti Anda menunjukkan tiket tanpa membaca huruf-huruf kecilnya. Bila berkas tidak dapat dibuka, karena tidak ada, karena foldernya salah, atau karena pengguna tidak boleh membacanya, `fopen` mengembalikan `NULL`, dan program yang lalu memakai `NULL` sebagai aliran akan *crash*. Karena itu memeriksa hasil setiap `fopen` bukanlah pilihan.

Setelah selesai dengan sebuah berkas, program **menutupnya** dengan `fclose(fp)`. Menutup lebih penting daripada kelihatannya, karena aliran memakai *penyangga* (*buffer*): untuk menghindari perjalanan lambat ke disk untuk setiap karakter, `fprintf` mengumpulkan keluaran dalam penyangga di memori dan menuliskannya ke berkas dalam blok-blok besar. Blok terakhir baru sampai di berkas ketika penyangga penuh, ketika program memanggil `fflush(fp)`, atau ketika aliran ditutup. Program yang *crash* sebelum menutup berkasnya bisa meninggalkan bagian akhir keluarannya tidak tertulis.

> **DEFINISI.** *Berkas* adalah deretan byte bernama yang dijaga sistem operasi di penyimpanan. *Aliran* (*stream*) adalah sambungan program ke sebuah berkas (atau ke papan ketik atau layar), yang dibaca atau ditulis dari depan ke belakang. Di C sebuah aliran diwakili pointer bertipe `FILE *`, yang diperoleh dari `fopen` dan dilepas dengan `fclose`. Aliran *teks* membawa baris-baris karakter; aliran *biner* membawa byte persis apa adanya.

11.2.2 Mode

Argumen kedua `fopen` menyatakan apa yang hendak dilakukan program terhadap berkas. **Tabel 11.1** mendaftar mode-modenya. Huruf `b` boleh ditambahkan ke mode mana pun (`"rb"`, `"wb"`, `"r+b"`) untuk membuka berkas dalam mode biner, yang berpengaruh di Windows: di sana aliran teks mengubah setiap `'\n'` yang ditulis program menjadi dua byte `"\r\n"`, dan sebaliknya ketika membaca, sedangkan aliran biner meneruskan setiap byte tanpa perubahan. Di Linux dan macOS mode teks dan biner berperilaku sama.

Tabel 11.1 Mode-mode `fopen`. Menambahkan `b` memberi mode yang sama untuk aliran biner.

Mode	Tujuan	Bila berkas ada	Bila berkas tidak ada
`"r"`	membaca	dibaca dari awal	`fopen` mengembalikan `NULL`
`"w"`	menulis	**langsung dikosongkan**	dibuat
`"a"`	menambah	setiap operasi tulis masuk ke akhir	dibuat
`"r+"`	membaca dan menulis	dipertahankan; mulai dari awal	`fopen` mengembalikan `NULL`
`"w+"`	menulis dan membaca	**langsung dikosongkan**	dibuat
`"a+"`	membaca dan menambah	dipertahankan; operasi tulis ke akhir	dibuat

Mode yang berbahaya adalah `"w"`: ia mengosongkan berkas yang ada *pada saat dibuka*, sebelum satu byte pun ditulis. Buka berkas pesanan hari itu dengan `"w"` karena keliru, padahal maksudnya `"r"`, dan pesanan-pesanan itu hilang. Subbab 11.8 mengubah bahaya ini menjadi aturan rancangan.

Fungsi bawaan Python `open(name, mode)` memakai huruf-huruf yang sama, memunculkan eksepsi seperti `FileNotFoundError` alih-alih mengembalikan `NULL`, dan biasanya dipakai dalam pernyataan `with`, yang menutup berkas secara otomatis di akhir blok, bahkan bila terjadi galat di dalamnya [3].

11.3 Menulis dan Membaca Berkas Teks

Kode 11.1 menjalani seluruh riwayat hidup berkas teks: membukanya untuk ditulis, menulis tiga pesanan dengan `fprintf`, menutupnya, membukanya lagi untuk dibaca, membaca pesanan satu per satu, dan menutupnya lagi.

Kode 11.1 `write_read_text.c`: menulis tiga pesanan ke berkas teks dan membacanya kembali.

```c
/* write_read_text.c - the life of a text file: open, write, close, open
 * again, read until a read fails, close. */
#include <stdio.h>
#include <stdlib.h>

int main(void)
{
    FILE *fp = fopen("orders.txt", "w");          /* "w" empties the file! */
    if (fp == NULL) {
        perror("orders.txt");
        return EXIT_FAILURE;
    }
    fprintf(fp, "%d %ld %s\n", 1, 24500L, "Andi Pratama");
    fprintf(fp, "%d %ld %s\n", 2, 20000L, "Budi Santoso");
    fprintf(fp, "%d %ld %s\n", 3, 24000L, "Citra Dewi Lestari");
    fclose(fp);                    /* only now is everything on the disk */

    fp = fopen("orders.txt", "r");
    if (fp == NULL) {
        perror("orders.txt");
        return EXIT_FAILURE;
    }
    int id, n = 0;
    long amount, total = 0;
    char name[32];
    printf("%-4s %-20s %8s\n", "id", "customer", "amount");
    /* the name is last on the line, so %31[^\n] may read its spaces */
    while (fscanf(fp, "%d %ld %31[^\n]", &id, &amount, name) == 3) {
        printf("%-4d %-20s %8ld\n", id, name, amount);
        total += amount;
        n++;
    }
    fclose(fp);
    printf("%d orders, total %ld\n", n, total);
    return 0;
}
```

```
id   customer              amount
1    Andi Pratama           24500
2    Budi Santoso           20000
3    Citra Dewi Lestari     24000
3 orders, total 68500
--- orders.txt ---
1 24500 Andi Pratama
2 20000 Budi Santoso
3 24000 Citra Dewi Lestari
```

Baris-baris sesudah `--- orders.txt ---` adalah berkas itu sendiri, seperti ditampilkan penguji: tiga baris teks biasa, satu per pesanan, yang dapat dibuka penyunting teks mana pun. Itulah kekuatan besar berkas teks: manusia dapat membacanya, memeriksanya, bahkan memperbaikinya.

Dua detail perulangan baca adalah jantung subbab ini.

Pertama, format `"%d %ld %31[^\n]"`. Nomor dan jumlah dibaca dengan `%d` dan `%ld`. Nama memuat spasi, sehingga `%s` hanya akan membaca `Andi`; sebagai gantinya *scan set* `%[^\n]` membaca setiap karakter sampai akhir baris, dan lebar 31 mencegahnya menulis lebih dari 31 karakter ke larik `name` yang 32 byte (Bab 9). Ini berhasil karena nama adalah ruas *terakhir* di baris; ruas yang boleh memuat spasi selalu paling mudah dibaca bila tidak ada apa-apa sesudahnya.

Kedua, syarat perulangannya. `fscanf` mengembalikan banyaknya ruas yang berhasil dibaca dan disimpan: 3 ketika satu pesanan utuh terbaca, kurang dari itu ketika barisnya rusak, dan konstanta `EOF` (bilangan negatif, biasanya −1) ketika akhir berkas datang sebelum apa pun dapat

dibaca. Perulangan `while (fscanf(...) == 3)` karena itu berlanjut persis selama pesanan utuh datang, dan berhenti baik di akhir berkas maupun di baris rusak pertama. Inilah idiom membaca dalam C: **uji hasil pembacaan itu sendiri**.

Versi Python, **Kode 11.2**, membaca berkas dengan perulangan `for` atas objek berkas, yang memberi satu baris setiap kali dan berhenti sendiri di akhir, lalu memotong setiap baris dengan `split(" ", 2)`, yang hanya memisah pada dua spasi pertama, sehingga nama tetap berspasi.

Kode 11.2 `write_read_text.py`: berkas yang sama ditulis dan dibaca dalam Python.

```
# write_read_text.py - the life of a text file in Python. "with" closes the
# file for us at the end of the block, even if an error happens inside it.


def main():
    with open("orders.txt", "w", encoding="utf-8") as fp:   # "w" empties it!
        fp.write("%d %d %s\n" % (1, 24500, "Andi Pratama"))
        fp.write("%d %d %s\n" % (2, 20000, "Budi Santoso"))
        fp.write("%d %d %s\n" % (3, 24000, "Citra Dewi Lestari"))

    n = total = 0
    print("%-4s %-20s %8s" % ("id", "customer", "amount"))
    with open("orders.txt", encoding="utf-8") as fp:
        for line in fp:                          # one line at a time
            id, amount, name = line.rstrip("\n").split(" ", 2)
            print("%-4d %-20s %8d" % (int(id), name, int(amount)))
            total += int(amount)
            n += 1
    print("%d orders, total %d" % (n, total))


if __name__ == "__main__":
    main()
```

11.4 Jebakan `feof`

Ada fungsi `feof(fp)` yang menjawab “apakah aliran ini sudah mencapai akhir berkas?”, dan banyak pemula menulis perulangan baca sebagai `while (!feof(fp))`. Kelihatannya benar. Ternyata salah, dan **Kode 11.3** menunjukkan sebabnya.

Kode 11.3 `feof_trap.c`: perulangan `while (!feof(fp))` berjalan satu kali terlalu banyak.

```c
/* feof_trap.c - why "while (!feof(fp))" reads the last record twice.
 * Run write_read_text first: this program reads its orders.txt. */
#include <stdio.h>
#include <stdlib.h>

int main(void)
{
    FILE *fp = fopen("orders.txt", "r");
    if (fp == NULL) {
        perror("orders.txt");
        return EXIT_FAILURE;
    }
    int id = 0, rounds = 0;
    long amount = 0, total = 0;
    char name[32] = "";
    while (!feof(fp)) {      /* WRONG: EOF is known only after a read fails */
        int got = fscanf(fp, "%d %ld %31[^\n]", &id, &amount, name);
        total += amount;             /* the failed last read left the old values */
        rounds++;
        printf("round %d: fscanf gave %2d -> #%d %s %ld\n",
               rounds, got, id, name, amount);
    }
    fclose(fp);
    printf("rounds: %d, total %ld (should be 3 and 68500)\n", rounds, total);
    return 0;
}
```

Dijalankan sesudah `write_read_text`, yang meninggalkan `orders.txt`, program ini mencetak:

```
$ step 1
id   customer               amount
1    Andi Pratama            24500
2    Budi Santoso            20000
3    Citra Dewi Lestari      24000
3 orders, total 68500
$ step 2
round 1: fscanf gave  3 -> #1 Andi Pratama 24500
round 2: fscanf gave  3 -> #2 Budi Santoso 20000
round 3: fscanf gave  3 -> #3 Citra Dewi Lestari 24000
round 4: fscanf gave -1 -> #3 Citra Dewi Lestari 24000
rounds: 4, total 92500 (should be 3 and 68500)
```

Keluaran di bawah `step 2` adalah jebakannya. Perulangan berjalan empat kali untuk tiga pesanan, dan totalnya salah sebesar satu pesanan utuh. Sebabnya, `feof` tidak melihat ke depan. Ia baru melaporkan akhir berkas *setelah sebuah pembacaan mencoba melewatinya dan gagal*. Setelah pesanan ketiga terbaca, aliran berdiri di akhir berkas, tetapi belum ada pembacaan yang gagal, sehingga `feof` masih berkata "belum di akhir". Perulangan berjalan keempat kalinya; kali ini `fscanf` tidak menemukan apa-apa dan mengembalikan −1 tanpa mengubah `id`, `amount`, atau `name`; dan program menjumlahkan jumlah lama sekali lagi. Perhatikan bahwa di Ubuntu, GCC bahkan tidak mau mengompilasi versi pertama program ini dengan opsi-opsi buku ini: pustaka C di sana mendeklarasikan `fscanf` dengan atribut yang meminta pemanggil memakai hasilnya, dan mengabaikannya menghasilkan peringatan, yang diubah `-Werror` menjadi galat.

`feof` punya satu kegunaan yang jujur: *setelah* perulangan baca berhenti, ia memberi tahu apakah perulangan berhenti karena berkas habis (`feof` benar) atau karena baris rusak atau galat baca (`feof` salah). Versi Python, **Kode 11.4**, meniru perulangan C untuk menunjukkan efek yang sama; kode Python sungguhan tidak pernah punya masalah ini, karena `for line in fp` berhenti sendiri.

Kode 11.4 `feof_trap.py`: tiruan perulangan C dalam Python.

```
# feof_trap.py - imitates the C loop "while (!feof(fp))": the end of the file
# is noticed only after one read has already failed.


def main():
    with open("orders.txt", encoding="utf-8") as fp:
        lines = fp.read().split("\n")      # the last piece is "" after the last \n
    id, amount, name = 0, 0, ""
    total = rounds = 0
    for piece in lines:                    # one round per read attempt
        got = -1                           # C's EOF
        if piece:                          # a successful read changes the values
            sid, samount, name = piece.split(" ", 2)
            id, amount, got = int(sid), int(samount), 3
        total += amount                    # the failed read left the old values
        rounds += 1
        print("round %d: fscanf gave %2d -> #%d %s %d"
              % (rounds, got, id, name, amount))
    print("rounds: %d, total %d (should be 3 and 68500)" % (rounds, total))


if __name__ == "__main__":
    main()
```

11.5 Membaca Data Nyata: Berkas CSV Baris demi Baris

Sebagian besar data yang dipertukarkan usaha kecil beredar sebagai **CSV**, *comma-separated values* (nilai yang dipisah koma): berkas teks dengan satu rekaman per baris dan ruas-ruas yang dipisah koma, sering dengan baris pertama yang menamai kolom-kolomnya. Setiap program lembar kerja dapat menyimpan lembar sebagai CSV dan membukanya, sehingga CSV adalah jalan bagi pesanan seminggu yang diketik pemilik di lembar kerja untuk sampai ke program kita, dan jalan bagi laporan program kita untuk sampai ke pemilik. Formatnya dijelaskan dalam RFC 4180 [4]; bentuk sederhana yang dipakai di sini tidak memiliki koma di dalam ruas dan tidak memakai tanda kutip.

Pemilik Profitina Laundry mencatat pesanan minggu itu di lembar kerja dan menyimpannya sebagai `data/week.csv`:

```
date,customer,service,kg
2026-09-21,Andi Pratama,fold,3.5
2026-09-21,Budi Santoso,wash-iron,2
2026-09-21,Citra Dewi Lestari,iron,4
2026-09-22,Dewi Anggraini,fold,5.5
2026-09-22,Eko Wibowo,dry-clean,3
2026-09-22,Fajar Nugroho,fold,2.5
2026-09-23,Gita Permata,wash-iron,1.5
2026-09-23,Hadi Susanto,iron
2026-09-23,Indah Sari,fold,6

2026-09-24,Joko Purnomo,iron,3.5
2026-09-24,Kartika Putri,fold,4,5
2026-09-25,Lina Marlina,wash-iron,2.5
2026-09-25,Made Wirawan,fold,3.x
```

Berkas yang diketik manusia mengandung kesalahan, dan berkas ini punya beberapa: layanan tak dikenal, baris yang kurang satu ruas, baris yang kelebihan ruas, baris kosong, dan berat yang bukan bilangan. Pembaca yang baik tidak *crash*, tidak berhenti di kesalahan pertama, dan tidak diam-diam menerima isian ngawur. Ia melaporkan setiap baris rusak *beserta nomor barisnya*, agar pemilik dapat membuka lembar kerjanya dan memperbaikinya, dan ia menghitung laporan dari baris-baris yang baik.

Kode 11.5 membaca berkas dalam tiga langkah per baris: `fgets` membaca seluruh baris (sampai 255 karakter), `strcspn(line, "\r\n")` menemukan akhir baris, entah `"\n"` atau `"\r\n"` ala Windows, dan penutup ditulis di sana; lalu `split_csv` memotong baris pada koma-komanya. `split_csv` menulis penutup di atas setiap koma dan mengingat di mana setiap ruas dimulai, seperti `strtok`, tetapi, tidak seperti `strtok`, ia mempertahankan ruas kosong: `a,,b` punya tiga ruas, yang tengahnya kosong. Ia juga menghitung *semua* ruas, bahkan yang melebihi kapasitas larik, sehingga program dapat membedakan "3 ruas" dari "5 ruas". Nama berkas diambil dari baris perintah, `argv[1]`, dan ketika berkas tidak dapat dibuka, `perror` mencetak namanya beserta alasan dari sistem operasi, yang diambil dari variabel `errno`, di `stderr`.

Kode 11.5 `csv_report.c`: membaca berkas CSV baris demi baris, melewati baris rusak dengan pesan, dan menjumlahkan per layanan dan per hari.

```c
/* csv_report.c - read a week of orders from a CSV file, skip bad lines with a
 * clear message, and print totals per service and per day.
 * Usage: csv_report <file.csv>      (the file starts with a header line) */
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define N_SERVICES 3
static const char *SERVICE[N_SERVICES] = {"fold", "wash-iron", "iron"};
static const int PRICE[N_SERVICES] = {7000, 10000, 6000};

/* Split line at commas IN PLACE, keeping empty fields (strtok would skip
 * them). Returns the number of fields; at most max are stored. */
static int split_csv(char *line, char *field[], int max)
{
    int n = 0;
    char *p = line;
    for (;;) {
        if (n < max)
            field[n] = p;
        n++;
        char *comma = strchr(p, ',');
        if (comma == NULL)
            break;
        *comma = '\0';
        p = comma + 1;
    }
    return n;
}

/* "3" -> 30, "3.5" -> 35; -1 if the text is not written like that. */
static int parse_tenths(const char *s)
{
    int whole = 0, digits = 0;
    for (; *s >= '0' && *s <= '9'; s++, digits++)
        whole = whole * 10 + (*s - '0');
    if (digits == 0 || digits > 3)
        return -1;
    if (*s == '\0')
        return whole * 10;
    if (s[0] == '.' && s[1] >= '0' && s[1] <= '9' && s[2] == '\0')
        return whole * 10 + (s[1] - '0');
    return -1;
}
```

Kode 11.5 (Lanjutan) *csv_report.c: membaca berkas CSV baris demi baris, melewati baris rusak dengan pesan, dan menjumlahkan per layanan dan per hari.*

```
int main(int argc, char *argv[])
{
    if (argc != 2) {
        fprintf(stderr, "usage: %s <file.csv>\n", argv[0]);
        return EXIT_FAILURE;
    }
    FILE *fp = fopen(argv[1], "r");
    if (fp == NULL) {
        perror(argv[1]);                    /* e.g. "No such file or directory" */
        return EXIT_FAILURE;
    }
    char line[256], day[16] = "", *f[4];
    int lineno = 0, good = 0, count[N_SERVICES] = {0}, tenths[N_SERVICES] = {0};
    long amount[N_SERVICES] = {0}, day_amount = 0;
    int day_orders = 0;
    char report[16][48];                    /* one line per day, printed later */
    int days = 0;

    while (fgets(line, sizeof line, fp) != NULL) {
        lineno++;
        line[strcspn(line, "\r\n")] = '\0';     /* cut the line ending */
        if (lineno == 1 || line[0] == '\0')
            continue;                           /* header or empty line */
        int n = split_csv(line, f, 4);
        if (n != 4) {
            printf("line %d: expected 4 fields, found %d - skipped\n", lineno, n);
            continue;
        }
        int s = -1;
        for (int i = 0; i < N_SERVICES; i++)
            if (strcmp(f[2], SERVICE[i]) == 0)
                s = i;
        int t = parse_tenths(f[3]);
        if (s < 0 || t <= 0) {
            printf("line %d: bad %s '%s' - skipped\n", lineno,
                   s < 0 ? "service" : "weight", s < 0 ? f[2] : f[3]);
            continue;
        }
        if (strcmp(f[0], day) != 0) {           /* a new day starts */
            if (day[0] != '\0' && days < 16)
                snprintf(report[days++], sizeof report[0], "%-10s %6d %12ld",
                         day, day_orders, day_amount);
            snprintf(day, sizeof day, "%s", f[0]);
            day_orders = 0;
            day_amount = 0;
        }
```

Kode 11.5 (Lanjutan) *csv_report.c: membaca berkas CSV baris demi baris, melewati baris rusak dengan pesan, dan menjumlahkan per layanan dan per hari.*

```
        long a = (long)t * PRICE[s] / 10;
        count[s]++;
        tenths[s] += t;
        amount[s] += a;
        day_orders++;
        day_amount += a;
        good++;
    }
    fclose(fp);
    if (day[0] != '\0' && days < 16)
        snprintf(report[days++], sizeof report[0], "%-10s %6d %12ld",
                 day, day_orders, day_amount);

    printf("%d good orders in %s\n\n", good, argv[1]);
    printf("%-10s %6s %10s %12s\n", "service", "orders", "weight", "amount");
    long total = 0;
    for (int i = 0; i < N_SERVICES; i++) {
        printf("%-10s %6d %7d.%d kg %12ld\n", SERVICE[i], count[i],
               tenths[i] / 10, tenths[i] % 10, amount[i]);
        total += amount[i];
    }
    printf("%-10s %6d %10s %12ld\n\n", "all", good, "", total);
    printf("%-10s %6s %12s\n", "date", "orders", "amount");
    for (int i = 0; i < days; i++)
        printf("%s\n", report[i]);
    return 0;
}
```

`./csv_report ../data/week.csv` mencetak:

```
line 6: bad service 'dry-clean' - skipped
line 9: expected 4 fields, found 3 - skipped
line 13: expected 4 fields, found 5 - skipped
line 15: bad weight '3.x' - skipped
9 good orders in ../data/week.csv

service    orders     weight       amount
fold            4      17.5 kg       122500
wash-iron       3       6.0 kg        60000
iron            2       7.5 kg        45000
all             9                    227500

date       orders       amount
2026-09-21      3        68500
2026-09-22      2        56000
2026-09-23      2        57000
2026-09-24      1        21000
2026-09-25      1        25000
```

Keempat pesan menunjuk tepat ke empat baris yang rusak (baris kosong 11 cukup dilewati); ruas tambahan di baris 13, `4,5`, adalah berat yang diketik dengan koma sebagai pemisah desimal, kebiasaan Indonesia, dan karena itulah penting bahwa pembaca menghitung ruas alih-alih mengambil empat yang pertama. Total per hari mengandalkan berkas yang terurut menurut tanggal: program menutup baris satu hari ketika tanggal berganti, cara klasik melaporkan *kelompok* dari berkas terurut dalam satu kali lewat. Baris-baris per hari dikumpulkan dengan `snprintf` ke dalam larik string dan dicetak sesudah tabel layanan.

Dan ketika berkasnya tidak ada, `./csv_report ../data/nope.csv` mencetak di `stderr` dan berakhir dengan status keluar 1:

```
../data/nope.csv: No such file or directory
```

Pesan itu hasil kerja `perror`: kata-kata sistem operasi sendiri, sesudah nama berkas yang menyebabkannya. Pesan dengan bentuk "berkas mana, apa yang salah" seperti ini menghemat berjam-jam bagi orang yang harus membereskan masalahnya.

Versi Python, **Kode 11.6**, memakai `split(",")`, yang mempertahankan ruas kosong seperti `split_csv` di C, dan kamus untuk mengumpulkan hari-hari. (Modul `csv` Python menangani ruas berkutip dan koma di dalam kutipan, dan merupakan alat yang tepat untuk berkas CSV buatan program lain [3]; di sini kita memisah secara manual agar kedua bahasa melakukan hal yang sama.)

Kode 11.6 `csv_report.py`: laporan yang sama dalam Python.

```
# csv_report.py - read a week of orders from a CSV file, skip bad lines with a
# clear message, and print totals per service and per day.
# Usage: python3 csv_report.py <file.csv>
import sys

SERVICE = ["fold", "wash-iron", "iron"]
PRICE = [7000, 10000, 6000]


def parse_tenths(s):
    whole, dot, tenth = s.partition(".")
    if not (whole.isascii() and whole.isdigit() and len(whole) <= 3):
        return -1
    if not dot:
        return int(whole) * 10
    if len(tenth) == 1 and tenth.isascii() and tenth.isdigit():
        return int(whole) * 10 + int(tenth)
    return -1


def main():
    if len(sys.argv) != 2:
        print("usage: csv_report.py <file.csv>", file=sys.stderr)
        sys.exit(1)
    try:
        fp = open(sys.argv[1], encoding="utf-8")
    except OSError as e:
        print("%s: %s" % (sys.argv[1], e.strerror), file=sys.stderr)
        sys.exit(1)
    good = 0
    count, tenths, amount = [0] * 3, [0] * 3, [0] * 3
    per_day = {}                        # date -> [orders, amount], in file order
    with fp:
        for lineno, line in enumerate(fp, start=1):
            line = line.rstrip("\r\n")
            if lineno == 1 or line == "":
                continue
            f = line.split(",")         # keeps empty fields
            if len(f) != 4:
                print("line %d: expected 4 fields, found %d - skipped"
                      % (lineno, len(f)))
                continue
            s = SERVICE.index(f[2]) if f[2] in SERVICE else -1
            t = parse_tenths(f[3])
            if s < 0 or t <= 0:
                print("line %d: bad %s '%s' - skipped" % (lineno,
```

Kode 11.6 (Lanjutan) *csv_report.py: laporan yang sama dalam Python.*

```
                    "service" if s < 0 else "weight", f[2] if s < 0 else f[3]))
                continue
            a = t * PRICE[s] // 10
            count[s] += 1
            tenths[s] += t
            amount[s] += a
            day = per_day.setdefault(f[0], [0, 0])
            day[0] += 1
            day[1] += a
            good += 1

    print("%d good orders in %s\n" % (good, sys.argv[1]))
    print("%-10s %6s %10s %12s" % ("service", "orders", "weight", "amount"))
    for i in range(3):
        print("%-10s %6d %7d.%d kg %12d"
              % (SERVICE[i], count[i], tenths[i] // 10, tenths[i] % 10, amount[i]))
    print("%-10s %6d %10s %12d\n" % ("all", good, "", sum(amount)))
    print("%-10s %6s %12s" % ("date", "orders", "amount"))
    for date, (n, a) in per_day.items():
        print("%-10s %6d %12d" % (date, n, a))


if __name__ == "__main__":
    main()
```

11.6 Mode Tambah: Cara Log Bertumbuh

Mode `"a"` membuka berkas sedemikian rupa sehingga setiap operasi tulis masuk ke *akhirnya*, dan berkas tidak pernah dikosongkan. Itulah persis yang diperlukan **log** (catatan kejadian): rekaman kejadian yang hanya pernah bertambah. Setiap server, setiap mesin kasir, dan setiap bank menyimpan log, karena ketika sesuatu salah, pertanyaan pertama adalah “apa yang terjadi, dalam urutan apa?”.

Kode 11.7 menghitung baris yang sudah ada di `activity.log` (berkas yang belum ada berarti nol baris), menambahkan satu entri bernomor, dan mencetak seluruh log.

Kode 11.7 `activity_log.c`: menambahkan satu entri bernomor ke berkas log pada setiap kali dijalankan.

```c
/* activity_log.c - append mode: every run adds one line to activity.log and
 * nothing already in the file is ever overwritten.
 * Usage: activity_log [message words...] */
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char *argv[])
{
    int lines = 0, c;
    FILE *fp = fopen("activity.log", "r");
    if (fp != NULL) {                       /* no file yet is fine: 0 lines */
        while ((c = fgetc(fp)) != EOF)
            if (c == '\n')
                lines++;
        fclose(fp);
    }

    fp = fopen("activity.log", "a");        /* "a": always write at the end */
    if (fp == NULL) {
        perror("activity.log");
        return EXIT_FAILURE;
    }
    fprintf(fp, "entry %d:", lines + 1);
    if (argc == 1)
        fprintf(fp, " cashier opened");
    for (int i = 1; i < argc; i++)
        fprintf(fp, " %s", argv[i]);
    fprintf(fp, "\n");
    fclose(fp);

    fp = fopen("activity.log", "r");
    if (fp == NULL) {
        perror("activity.log");
        return EXIT_FAILURE;
    }
    char line[128];
    while (fgets(line, sizeof line, fp) != NULL)
        printf("  %s", line);
    fclose(fp);
    return 0;
}
```

Dijalankan tiga kali, yang kedua dengan pesan (`./activity_log order #1 taken`), ia mencetak:

```
$ step 1
  entry 1: cashier opened
$ step 2
  entry 1: cashier opened
  entry 2: order #1 taken
$ step 3
  entry 1: cashier opened
  entry 2: order #1 taken
  entry 3: cashier opened
```

Setiap kali berjalan, program menemukan baris-baris dari jalan-jalan sebelumnya dan menambah satu. Penghitungannya dilakukan dengan `fgetc`, yang membaca satu karakter setiap kali dan mengembalikan `EOF` di akhir, dan perulangan `while ((c = fgetc(fp)) != EOF)` adalah bentuk tingkat karakter dari idiom membaca: baca, lalu uji apa yang dikembalikan pembacaan. Perhatikan bahwa `c` bertipe `int`, bukan `char`: `EOF` adalah nilai di luar jangkauan karakter, dan menyimpannya dalam `char` dapat membuatnya tampak seperti karakter sungguhan. Versi Python, **Kode 11.8**, menangkap `FileNotFoundError` untuk berkas yang belum ada.

Kode 11.8 `activity_log.py`: log yang sama dalam Python.

```python
# activity_log.py - append mode: every run adds one line to activity.log.
# Usage: python3 activity_log.py [message words...]
import sys


def main():
    try:
        with open("activity.log", encoding="utf-8") as fp:
            lines = fp.read().count("\n")
    except FileNotFoundError:            # no file yet is fine: 0 lines
        lines = 0
    message = " ".join(sys.argv[1:]) if len(sys.argv) > 1 else "cashier opened"
    with open("activity.log", "a", encoding="utf-8") as fp:   # "a": at the end
        fp.write("entry %d: %s\n" % (lines + 1, message))
    with open("activity.log", encoding="utf-8") as fp:
        for line in fp:
            print("  " + line, end="")


if __name__ == "__main__":
    main()
```

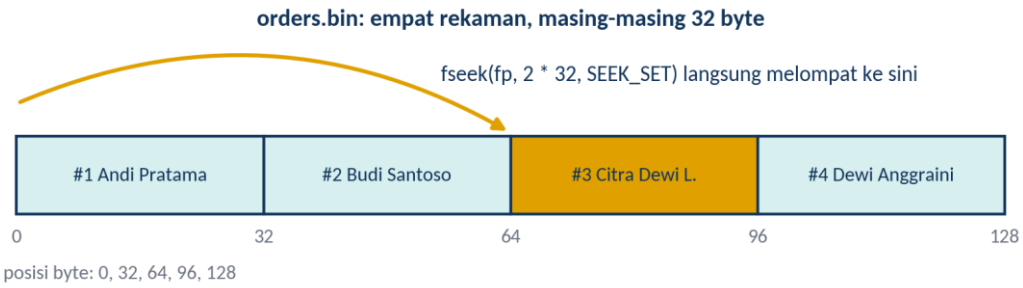
11.7 Berkas Biner dan Akses Acak

Berkas teks dibaca dari depan ke belakang. Untuk menemukan pesanan ke-5.000 dalam berkas teks, program harus membaca 4.999 pesanan sebelumnya, karena panjang baris berbeda-beda dan tidak ada cara mengetahui di mana baris ke-5.000 dimulai tanpa menghitung baris-baris barunya. Ketika setiap rekaman berukuran *sama*, ada cara yang lebih cepat.

Berkas **biner** menyimpan data sebagai byte-byte yang mewakilinya di memori, bukan sebagai teks yang dapat dibaca. `fwrite(ptr, size, count, fp)` menyalin `count` objek berukuran masing-masing `size` byte dari memori ke aliran, dan `fread(ptr, size, count, fp)` menyalinnya kembali; keduanya mengembalikan banyaknya objek utuh yang dipindahkan [1]. Bila objek-objek itu adalah struktur berukuran `s` byte, rekaman nomor i (dihitung dari 0) dimulai di byte

$$\text{offset}(i) = i \cdot s \tag{11.1}$$

dalam berkas, dan fungsi `fseek(fp, offset, SEEK_SET)` memindahkan posisi aliran langsung ke sana, tanpa membaca apa pun di jalan. Ini disebut **akses acak** (*random access*), "acak" dalam arti rekaman mana pun dapat dicapai dalam satu langkah, dalam urutan apa pun. `fseek(fp, 0, SEEK_END)` berpindah ke akhir, dan `ftell(fp)` lalu melaporkan posisinya, yang sama dengan ukuran berkas dalam byte. **Gambar 11.1** memperlihatkan berkas dari program berikutnya.



Gambar 11.1 Berkas biner berisi empat rekaman 32 byte: menurut Persamaan (11.1), rekaman 2 dimulai di byte $2 \cdot 32 = 64$, dan `fseek` langsung ke sana.

Kode 11.9 menulis empat pesanan sebagai rekaman 32 byte, mengukur berkasnya, membaca rekaman 2 secara langsung, memberinya diskon 10% dan menuliskannya kembali *di tempat*, di tengah berkas, dengan mode `"r+b"`, yang membaca dan menulis tanpa mengosongkan berkas.

Kode 11.9 `binary_records.c`: rekaman berukuran tetap yang ditulis dengan `fwrite`, dicapai dengan `fseek`, dan diubah di tempat.

```c
/* binary_records.c - fixed-size records in a binary file: write them with
 * fwrite, jump to any one with fseek, change it in place, read them back. */
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

typedef struct {
    int id;
    char customer[20];       /* the rest of the 20 bytes is filled with '\0' */
    int tenths;
    int amount;
} Record;                    /* 4 + 20 + 4 + 4 = 32 bytes, no padding */

static Record make(int id, const char *customer, int tenths, int amount)
{
    Record r;
    memset(&r, 0, sizeof r);             /* every byte defined, even unused */
    r.id = id;
    snprintf(r.customer, sizeof r.customer, "%s", customer);
    r.tenths = tenths;
    r.amount = amount;
    return r;
}

static void show(const Record *r)
{
    printf("  #%d %-19s %2d.%d kg  Rp %6d\n", r->id, r->customer,
           r->tenths / 10, r->tenths % 10, r->amount);
}

int main(void)
{
    Record day[4] = {make(1, "Andi Pratama", 35, 24500),
                     make(2, "Budi Santoso", 20, 20000),
                     make(3, "Citra Dewi Lestari", 40, 24000),
                     make(4, "Dewi Anggraini", 55, 38500)};
    FILE *fp = fopen("orders.bin", "wb");
    if (fp == NULL) {
        perror("orders.bin");
        return EXIT_FAILURE;
    }
    size_t written = fwrite(day, sizeof(Record), 4, fp);
    fclose(fp);
    printf("wrote %zu records of %zu bytes\n", written, sizeof(Record));
```

Kode 11.9 (Lanjutan) *binary_records.c: rekaman berukuran tetap yang ditulis dengan fwrite, dicapai dengan fseek, dan diubah di tempat.*

```
    fp = fopen("orders.bin", "r+b");         /* read AND write, no erasing */
    if (fp == NULL) {
        perror("orders.bin");
        return EXIT_FAILURE;
    }
    fseek(fp, 0, SEEK_END);
    long size = ftell(fp);
    printf("file size %ld bytes = %ld records\n",
           size, size / (long)sizeof(Record));

    Record r;
    fseek(fp, 2L * (long)sizeof(Record), SEEK_SET);   /* straight to record 3 */
    if (fread(&r, sizeof r, 1, fp) == 1) {
        printf("record at index 2:\n");
        show(&r);
    }

    r.amount -= r.amount / 10;                        /* a 10% discount ... */
    fseek(fp, 2L * (long)sizeof(Record), SEEK_SET);   /* ... written back in place */
    fwrite(&r, sizeof r, 1, fp);
    fclose(fp);

    fp = fopen("orders.bin", "rb");
    if (fp == NULL) {
        perror("orders.bin");
        return EXIT_FAILURE;
    }
    printf("all records after the change:\n");
    while (fread(&r, sizeof r, 1, fp) == 1)
        show(&r);
    fclose(fp);
    return 0;
}
```

```
wrote 4 records of 32 bytes
file size 128 bytes = 4 records
record at index 2:
  #3 Citra Dewi Lestari   4.0 kg  Rp  24000
all records after the change:
  #1 Andi Pratama         3.5 kg  Rp  24500
  #2 Budi Santoso         2.0 kg  Rp  20000
  #3 Citra Dewi Lestari   4.0 kg  Rp  21600
  #4 Dewi Anggraini       5.5 kg  Rp  38500
```

Rekamannya dipilih agar tidak ber-*padding*: 4 + 20 + 4 + 4 = 32 byte, semua anggota rata (Subbab 10.2.2). Fungsi `make` mengisi setiap byte rekaman dengan nol memakai `memset` sebelum mengisinya, sehingga byte tak terpakai sesudah nama yang pendek bernilai nol, bukan sisa apa pun yang kebetulan ada di memori. Itu rapi, mencegah data pelanggan dari satu rekaman bocor ke byte tak terpakai rekaman lain, dan membuat byte-byte berkas dapat diramalkan, yang diperlukan paragraf berikutnya.

Versi Python, **Kode 11.10**, menggambarkan rekaman dengan modul `struct`: format `"<i20sii"` berarti *little-endian* (`<`), bilangan bulat 4 byte, 20 byte teks, dan dua bilangan bulat lagi, total 32 byte [3]. Penguji membandingkan kedua berkas `orders.bin` byte demi byte, dan keduanya identik: program C dan program Python dapat berbagi berkas biner bila keduanya sepakat tentang tata letaknya yang persis.

Kode 11.10 `binary_records.py`: rekaman yang sama dengan modul `struct` Python.

```python
# binary_records.py - the same 32-byte records, packed with the struct module
# in the layout the C program uses: "<" little-endian, i int, 20s 20 bytes.
import os
import struct

RECORD = struct.Struct("<i20sii")        # 4 + 20 + 4 + 4 = 32 bytes


def make(id, customer, tenths, amount):
    return RECORD.pack(id, customer.encode("ascii")[:19], tenths, amount)


def show(raw):
    id, name, tenths, amount = RECORD.unpack(raw)
    name = name.split(b"\0", 1)[0].decode("ascii")
    print("  #%d %-19s %2d.%d kg  Rp %6d"
          % (id, name, tenths // 10, tenths % 10, amount))


def main():
    day = [make(1, "Andi Pratama", 35, 24500),
           make(2, "Budi Santoso", 20, 20000),
           make(3, "Citra Dewi Lestari", 40, 24000),
           make(4, "Dewi Anggraini", 55, 38500)]
    with open("orders.bin", "wb") as fp:
        for r in day:
            fp.write(r)
    print("wrote %d records of %d bytes" % (len(day), RECORD.size))

    with open("orders.bin", "r+b") as fp:          # read AND write, no erasing
        size = fp.seek(0, os.SEEK_END)
        print("file size %d bytes = %d records" % (size, size // RECORD.size))
        fp.seek(2 * RECORD.size)                   # straight to record 3
        raw = fp.read(RECORD.size)
        print("record at index 2:")
        show(raw)
        id, name, tenths, amount = RECORD.unpack(raw)
        amount -= amount // 10
        fp.seek(2 * RECORD.size)
        fp.write(RECORD.pack(id, name, tenths, amount))
```

Kode 11.10 (Lanjutan) *binary_records.py: rekaman yang sama dengan modul struct Python.*

```python
    print("all records after the change:")
    with open("orders.bin", "rb") as fp:
        while True:
            raw = fp.read(RECORD.size)
            if len(raw) < RECORD.size:
                break
            show(raw)


if __name__ == "__main__":
    main()
```

PERINGATAN: Berkas biner yang ditulis dengan `fwrite` sebuah struktur bergantung pada komputer yang menulisnya: ukuran `int` dan `long`, *padding* di dalam struktur, dan *urutan byte* (apakah byte terendah sebuah bilangan disimpan lebih dulu, seperti pada prosesor x86 dan ARM di laptop dan ponsel masa kini, atau terakhir). Berkas yang ditulis di satu platform bisa tak bermakna di platform lain. Pakailah berkas biner untuk data yang tetap di satu mesin dan memerlukan kecepatan; pakailah berkas teks seperti CSV untuk data yang dibaca manusia atau yang berpindah antarprogram dan antarmesin.

11.8 Menyimpan Tanpa Kehilangan: Tulis, Tutup, Ganti Nama

Kembali ke listrik padam di Subbab 11.1. Kasir harus menyimpan pesanannya *setelah setiap perubahan*, sehingga paling banyak hanya perintah yang sedang diketik yang hilang. Kode yang paling jelas: buka `shop.csv` dengan `"w"`, tulis setiap pesanan, tutup. Bagian (a) **Gambar 11.2** menunjukkan apa yang salah. `"w"` langsung mengosongkan berkas; selama beberapa milidetik berikutnya berkas itu kosong lalu setengah tertulis; baru setelah `fclose` berkas itu lengkap. Listrik padam dalam rentang itu, dan berkas di disk tidak berisi apa-apa, atau hanya separuh pesanan. Jarang, tetapi program yang menyimpan setelah setiap perintah menyimpan ribuan kali setahun.



Gambar 11.2 Dua cara menyimpan: (a) menimpa berkas di tempat membuatnya kosong atau setengah tertulis selama penyimpanan; (b) menulis berkas baru lalu mengganti namanya menimpa berkas lama menyisakan, pada setiap saat, satu berkas yang utuh.

Cara yang aman, bagian (b), tidak pernah menyentuh berkas lama sampai berkas baru lengkap:

1. Tulis semua pesanan ke berkas *baru*, `shop.csv.tmp`.
2. Tutup berkas itu, dan periksa bahwa penutupan berhasil: `fclose` mengembalikan nilai bukan nol bila byte-byte terakhir di penyangga tidak dapat ditulis, misalnya karena disk penuh. Periksa juga `ferror(fp)`, yang melaporkan apakah ada operasi tulis sebelumnya yang gagal.
3. Hanya bila semuanya berhasil, **ganti nama** `shop.csv.tmp` menjadi `shop.csv`, menggantikan berkas lama.

Listrik padam selama langkah 1 atau 2 membiarkan `shop.csv` lama tidak tersentuh; berkas `.tmp` yang setengah tertulis diabaikan saja dan ditimpa pada kesempatan berikutnya. Penggantian nama itu sendiri dilakukan sistem operasi dalam satu langkah: di Linux dan macOS, POSIX menjamin bahwa `rename` menggantikan sasarannya secara *atomik*, sehingga pada setiap saat nama `shop.csv` merujuk ke berkas lama yang utuh atau ke berkas baru yang utuh [5]. Windows kurang praktis: `rename` dalam pustaka C-nya menolak menggantikan berkas yang sudah ada [6], sehingga kode C yang portabel menghapus berkas lama lalu mengganti nama sekali lagi ketika penggantian nama pertama gagal, yang menyisakan celah sangat singkat tanpa `shop.csv` (tetapi dengan `shop.csv.tmp` yang utuh). `os.replace` Python melakukan penggantian atomik di setiap platform, termasuk Windows [3].

Bahkan cara ini memercayai sistem operasi telah menaruh byte-byte itu di disk. Sesudah `fclose`, data mungkin masih berada di tembolok memori sistem operasi selama beberapa detik; program yang harus selamat dari listrik padam *dengan biaya apa pun*, seperti basis data, juga memanggil fungsi sistem (`fsync` di POSIX, `FlushFileBuffers` di Windows) yang menunggu sampai disk mengonfirmasi operasi tulis. Itu berada di luar C standar dan di luar buku ini; untuk sebuah laundry, tulis–tutup–ganti nama adalah langkah besarnya.

11.9 Membangun Kasir v0.9

11.9.1 Satu program, beberapa berkas

Kasir v0.8 sudah lebih dari 300 baris, dan v0.9 menambahkan pemuatan dan penyimpanan. Sudah waktunya memecah program menjadi *modul*, masing-masing dalam berkas sumber sendiri dengan tugas yang jelas, seperti Bab 5 memisahkan aturan bisnis ke dalam `laundry.c` dan `laundry.h`:

- `shop.h` dan `shop.c`: segala sesuatu tentang pesanan, pembantu teks dari Bab 9, *linked list* dan perintah-perintah dari Bab 10. `shop.c` adalah kasir v0.8 tanpa `main` dan dengan fungsi-fungsi yang dibuat terlihat oleh berkas lain; ia ada di folder kode dan tidak diulang di sini.
- `storage.h` dan `storage.c`: memuat pesanan dari berkas CSV dan menyimpannya dengan aman.
- `cashier_v9.c`: `main`, perulangan perintah, yang kini memanggil `save_shop` setelah setiap perubahan.

Berkas **header** (`.h`) memuat *deklarasi* yang diperlukan berkas lain: tipe, konstanta, dan prototipe fungsi, tetapi tanpa badan fungsi. Berkas sumber memakainya dengan `#include "shop.h"` (tanda petik, bukan kurung sudut, untuk *header* Anda sendiri). Karena sebuah *header* bisa disertakan oleh *header* lain dan oleh berkas sumber, deklarasi yang sama bisa terbaca dua kali, yang merupakan galat untuk definisi tipe; **include guard** (pengaman penyertaan) mencegahnya [7]: badan *header* dibungkus `#ifndef SHOP_H` / `#define SHOP_H` / `#endif`, sehingga kali kedua praprosesor melihat *header* itu, `SHOP_H` sudah terdefinisi dan badannya dilewati. **Kode 11.11** adalah *header* modul toko.

Kode 11.11 `shop.h`: tipe-tipe dan fungsi-fungsi modul toko, dilindungi *include guard*.

```
/* shop.h - the orders of Profitina Laundry and the commands that change
 * them (Chapter 10's cashier v0.8, moved into a module of its own). */
#ifndef SHOP_H              /* include guard: the body is read only once */
#define SHOP_H

#include <stddef.h>

#define NAME_LEN 32

typedef enum { FOLD, WASH_IRON, IRON_ONLY, N_SERVICES } Service;
typedef enum { WAITING, WASHING, READY, N_STATUS } Status;

typedef struct Order {
    int id;
    char customer[NAME_LEN];
    Service service;
    int tenths;                 /* weight in tenths of a kg */
    long amount;                /* rupiah */
    Status status;
    struct Order *next;
} Order;

typedef struct {
    Order *head, *tail;
    int count, next_id, picked;
    long takings;
} Shop;

extern const char *const SERVICE_NAME[N_SERVICES];
extern const int SERVICE_PRICE[N_SERVICES];
extern const char *const STATUS_NAME[N_STATUS];

/* text helpers (Chapter 9) */
int read_line(char *buf, int size);
size_t next_word(const char **p, char *out, size_t size);
void tidy_name(const char *src, char *dst, size_t size);
int parse_tenths(const char *s, int *out);
void rupiah(long n, char *out);
int index_of(const char *word, const char *const names[], int n);

/* the list of orders (Chapter 10) */
Order *add_order(Shop *shop, int id, Service s, int tenths, Status st,
                 const char *customer);
Order *find_id(const Shop *shop, int id);
void free_all(Shop *shop);
```

Kode 11.11 (Lanjutan) *shop.h: tipe-tipe dan fungsi-fungsi modul toko, dilindungi include guard.*

```
/* commands; each returns 1 if it changed the shop, 0 if not */
int cmd_add(Shop *shop, const char *rest);
int cmd_advance(Shop *shop, const char *rest);
int cmd_pickup(Shop *shop, const char *rest);
void cmd_list(const Shop *shop);

#endif
```

Kata `extern` di depan ketiga tabel berarti "larik ini ada, dan didefinisikan di berkas lain": `shop.c` mendefinisikannya sekali, dan setiap berkas yang menyertakan `shop.h` boleh memakainya. Program dikompilasi dengan memberikan semua berkas sumbernya kepada `gcc` sekaligus:

```
gcc -std=c17 -O2 -Wall -Wextra -o cashier_v9 cashier_v9.c shop.c storage.c
```

11.9.2 Modul penyimpanan

Pesanan disimpan sebagai berkas CSV dengan baris judul:

```
id,service,tenths,status,customer
1,fold,35,washing,Andi Pratama
```

Pelanggan ditaruh *terakhir*, dengan alasan yang sama seperti di Subbab 11.3: bila nama pelanggan suatu hari memuat koma, pemuat hanya memotong baris pada empat koma pertama, dan segala sesuatu sesudah koma keempat adalah nama. **Kode 11.12** mendeklarasikan dua fungsi modul ini, dan **Kode 11.13** mewujudkannya.

Kode 11.12 `storage.h`: memuat dan menyimpan toko.

```c
/* storage.h - keep the shop's orders in a CSV file between runs. */
#ifndef STORAGE_H
#define STORAGE_H

#include "shop.h"

/* Adds the orders stored in path to shop; returns how many were loaded.
 * A missing file is not an error: it is the very first day. */
int load_shop(Shop *shop, const char *path);

/* Writes every order to path through a temporary file; 1 on success. */
int save_shop(const Shop *shop, const char *path);

#endif
```

Kode 11.13 `storage.c`: pemuat CSV, yang memeriksa setiap ruas setiap baris, dan penyimpan, yang menulis berkas sementara lalu mengganti namanya.

```c
/* storage.c - the shop's orders in a CSV file:
 *     id,service,tenths,status,customer
 *     1,fold,35,washing,Andi Pratama
 * The customer comes last, so a comma inside a name cannot break a line. */
#include "storage.h"

#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define HEADER "id,service,tenths,status,customer"

static int is_number(const char *s)
{
    int n = 0;
    for (; *s >= '0' && *s <= '9'; s++)
        n++;
    return *s == '\0' && n > 0 && n <= 6;
}

int load_shop(Shop *shop, const char *path)
{
    FILE *fp = fopen(path, "r");
    if (fp == NULL)
        return 0;                           /* no file yet: the first day */
    char line[128], name[NAME_LEN];
    int lineno = 0, loaded = 0;
    while (fgets(line, sizeof line, fp) != NULL) {
        lineno++;
        line[strcspn(line, "\r\n")] = '\0';
        if (lineno == 1 && strcmp(line, HEADER) == 0)
            continue;
        char *f[5];                         /* cut at the first 4 commas */
        int n = 1;
        f[0] = line;
        while (n < 5) {
            char *comma = strchr(f[n - 1], ',');
            if (comma == NULL)
                break;
            *comma = '\0';
            f[n++] = comma + 1;
        }
        name[0] = '\0';
        int s = n == 5 ? index_of(f[1], SERVICE_NAME, N_SERVICES) : -1;
        int st = n == 5 ? index_of(f[3], STATUS_NAME, N_STATUS) : -1;
        if (n == 5)
```

Kode 11.13 (Lanjutan) *storage.c: pemuat CSV, yang memeriksa setiap ruas setiap baris, dan penyimpan, yang menulis berkas sementara lalu mengganti namanya.*

```
            tidy_name(f[4], name, sizeof name);
        if (n < 5 || !is_number(f[0]) || s < 0 || !is_number(f[2]) ||
            st < 0 || name[0] == '\0' || find_id(shop, atoi(f[0])) != NULL) {
            printf("  %s line %d is damaged - skipped\n", path, lineno);
            continue;
        }
        if (add_order(shop, atoi(f[0]), (Service)s, atoi(f[2]), (Status)st,
                      name) == NULL)
            break;                          /* out of memory */
        loaded++;
    }
    fclose(fp);
    return loaded;
}

int save_shop(const Shop *shop, const char *path)
{
    char tmp[256];
    snprintf(tmp, sizeof tmp, "%s.tmp", path);
    FILE *fp = fopen(tmp, "w");
    if (fp == NULL) {
        perror(tmp);
        return 0;
    }
    fprintf(fp, "%s\n", HEADER);
    for (const Order *o = shop->head; o != NULL; o = o->next)
        fprintf(fp, "%d,%s,%d,%s,%s\n", o->id, SERVICE_NAME[o->service],
                o->tenths, STATUS_NAME[o->status], o->customer);
    int bad = ferror(fp);                   /* did any write fail? */
    if (fclose(fp) != 0)                    /* the last bytes go out here */
        bad = 1;
    if (bad) {                              /* e.g. the disk is full */
        perror(tmp);
        remove(tmp);
        return 0;
    }
    if (rename(tmp, path) != 0) {           /* Windows will not rename onto */
        remove(path);                       /* an existing file: remove it */
        if (rename(tmp, path) != 0) {       /* first and try once more */
            perror(path);
            return 0;
        }
    }
    return 1;
}
```

Pemuat memperlakukan berkas sebagai masukan yang tidak dipercaya, sebagaimana mestinya. Ia menerima baris judul hanya sebagai baris 1; ia memotong setiap baris menjadi tepat lima ruas; ia memeriksa bahwa id dan berat adalah bilangan paling banyak enam angka, bahwa layanan dan status adalah nama yang dikenalnya, bahwa nama tidak kosong setelah dirapikan, dan bahwa id itu belum terpakai. Baris yang gagal dalam pemeriksaan mana pun dilaporkan beserta nomor barisnya dan dilewati, dan sisa berkas tetap dimuat: satu baris rusak tidak boleh membuat toko kehilangan pesanan-pesanan lainnya. `add_order`, dari `shop.c`, juga menaikkan `next_id` melampaui id terbesar yang dimuat, sehingga pesanan baru tidak pernah memakai ulang nomor lama.

Penyimpan adalah Subbab 11.8 dalam C: tulis ke `shop.csv.tmp`, periksa `ferror` dan `fclose`, lalu ganti nama, dengan percobaan kedua untuk Windows.

11.9.3 Program utama

Kode 11.14 adalah `main`. Ia memuat pesanan saat mulai, dan setelah setiap perintah yang melaporkan perubahan (fungsi-fungsi perintah kini mengembalikan 1 bila mengubah toko), ia menyimpan. Bila penyimpanan gagal, ia tidak berpura-pura: ia meminta kasir menulis pesanan di kertas.

Untuk peragaan ada satu perintah tambahan, `poweroff`, yang mensimulasikan listrik padam: ia mencetak pesan lalu menghentikan program *seketika* dengan `_Exit(1)`, fungsi C standar yang mengakhiri program tanpa menjalankan pembersihan apa pun, tanpa mengosongkan penyangga aliran apa pun, dan tanpa menyimpan apa pun, sedekat mungkin dengan mencabut steker yang dapat dilakukan sebuah program. Pesannya dikosongkan secara manual lebih dulu, agar kita dapat melihatnya.

Kode 11.14 `cashier_v9.c`: kasir v0.9, yang memuat pesanan saat mulai dan menyimpan setelah setiap perubahan.

```
/* cashier_v9.c - Profitina Laundry cashier v0.9: the orders survive a power
 * cut, because every change is saved to shop.csv at once.
 * Build: gcc -std=c17 -O2 -Wall -Wextra -o cashier_v9 cashier_v9.c shop.c storage.c
 * Usage: cashier_v9 [--echo] */
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#include "shop.h"
#include "storage.h"

#define DB "shop.csv"
#define LINE_LEN 80

int main(int argc, char *argv[])
{
    int echo = argc > 1 && strcmp(argv[1], "--echo") == 0;
    char line[LINE_LEN], cmd[16], m[32];
    Shop shop = {NULL, NULL, 0, 1, 0, 0};

    int loaded = load_shop(&shop, DB);
    printf("Profitina Laundry cashier v0.9 - %d order(s) loaded from %s\n",
           loaded, DB);
    for (;;) {
        printf("> ");
        fflush(stdout);
        int r = read_line(line, sizeof line);
        if (r == 0) {
            printf("\n");
            break;
        }
        if (echo)
            printf(r < 0 ? "%.40s...\n" : "%s\n", line);
        if (r < 0) {
            puts("  line too long (at most 79 characters), ignored");
            continue;
        }
        const char *rest = line;
        int changed = 0;
        if (next_word(&rest, cmd, sizeof cmd) == 0)
            continue;
        if (strcmp(cmd, "add") == 0)
            changed = cmd_add(&shop, rest);
        else if (strcmp(cmd, "advance") == 0)
            changed = cmd_advance(&shop, rest);
        else if (strcmp(cmd, "pickup") == 0)
```

Kode 11.14 (Lanjutan) *cashier_v9.c: kasir v0.9, yang memuat pesanan saat mulai dan menyimpan setelah setiap perubahan.*

```
            changed = cmd_pickup(&shop, rest);
        else if (strcmp(cmd, "list") == 0)
            cmd_list(&shop);
        else if (strcmp(cmd, "quit") == 0)
            break;
        else if (strcmp(cmd, "poweroff") == 0) {    /* simulate a power cut */
            puts("  *** the power goes off ***");
            fflush(stdout);
            _Exit(1);          /* stop at once: no goodbye, no clean-up */
        } else
            printf("  unknown command '%s'\n", cmd);
        if (changed && !save_shop(&shop, DB))
            puts("  WARNING: not saved - write this order on paper");
    }
    rupiah(shop.takings, m);
    printf("Bye. %d picked up (Rp %s), %d still in the shop.\n",
           shop.picked, m, shop.count);
    free_all(&shop);
    return 0;
}
```

Versi Python memiliki tiga bagian yang sama: `shop.py`, kasir v0.8 tanpa `main`; **Kode 11.15**, `storage.py`, tempat `split(",", 4)` memotong pada empat koma pertama dan `os.replace` melakukan penggantian nama yang atomik; dan **Kode 11.16**, `cashier_v9.py`, tempat `os._exit(1)` memainkan peran `_Exit`.

Kode 11.15 `storage.py`: memuat dan menyimpan dalam Python.

```
# storage.py - the shop's orders in a CSV file:
#     id,service,tenths,status,customer
#     1,fold,35,washing,Andi Pratama
# The customer comes last, so a comma inside a name cannot break a line.
import os

from shop import (SERVICE_NAME, STATUS_NAME, add_order, find_id,
                  tidy_name)

HEADER = "id,service,tenths,status,customer"


def is_number(s):
    return s.isascii() and s.isdigit() and len(s) <= 6


def load_shop(shop, path):
    """Add the orders stored in path to shop; return how many were loaded."""
    try:
        fp = open(path, encoding="utf-8")
    except FileNotFoundError:
        return 0                                # no file yet: the first day
    loaded = 0
    with fp:
        for lineno, line in enumerate(fp, start=1):
            line = line.rstrip("\r\n")
            if lineno == 1 and line == HEADER:
                continue
            f = line.split(",", 4)              # cut at the first 4 commas
            ok = (len(f) == 5 and is_number(f[0]) and f[1] in SERVICE_NAME
                  and is_number(f[2]) and f[3] in STATUS_NAME
                  and tidy_name(f[4]) != "" and find_id(shop, int(f[0])) is None)
            if not ok:
                print("  %s line %d is damaged - skipped" % (path, lineno))
                continue
            add_order(shop, int(f[0]), SERVICE_NAME.index(f[1]), int(f[2]),
                      STATUS_NAME.index(f[3]), tidy_name(f[4]))
            loaded += 1
    return loaded
```

Kode 11.15 (Lanjutan) *storage.py: memuat dan menyimpan dalam Python.*

```
def save_shop(shop, path):
    """Write every order to path through a temporary file; True on success."""
    tmp = path + ".tmp"
    try:
        with open(tmp, "w", encoding="utf-8") as fp:
            fp.write(HEADER + "\n")
            o = shop.head
            while o is not None:
                fp.write("%d,%s,%d,%s,%s\n" % (o.id, SERVICE_NAME[o.service],
                         o.tenths, STATUS_NAME[o.status], o.customer))
                o = o.next
        os.replace(tmp, path)      # one step, also over an existing file
        return True
    except OSError as e:
        print("%s: %s" % (tmp, e.strerror))
        return False
```

Kode 11.16 `cashier_v9.py`: kasir v0.9 dalam Python.

```
# cashier_v9.py - Profitina Laundry cashier v0.9: the orders survive a power
# cut, because every change is saved to shop.csv at once.
# Usage: python3 cashier_v9.py [--echo]
import os
import sys

from shop import (Shop, cmd_add, cmd_advance, cmd_list, cmd_pickup,
                  read_line, rupiah)
from storage import load_shop, save_shop

DB = "shop.csv"


def main():
    echo = len(sys.argv) > 1 and sys.argv[1] == "--echo"
    shop = Shop()
    loaded = load_shop(shop, DB)
    print("Profitina Laundry cashier v0.9 - %d order(s) loaded from %s"
          % (loaded, DB))
    while True:
        print("> ", end="", flush=True)
        r, line = read_line()
        if r == 0:
            print()
            break
        if echo:
            print(line[:40] + "..." if r < 0 else line)
        if r < 0:
            print("  line too long (at most 79 characters), ignored")
            continue
        parts = line.split(None, 1)
        if not parts:
            continue
        cmd, rest = parts[0][:15], parts[1] if len(parts) > 1 else ""
        changed = False
        if cmd == "add":
            changed = cmd_add(shop, rest)
        elif cmd == "advance":
            changed = cmd_advance(shop, rest)
        elif cmd == "pickup":
            changed = cmd_pickup(shop, rest)
        elif cmd == "list":
            cmd_list(shop)
        elif cmd == "quit":
            break
        elif cmd == "poweroff":                  # simulate a power cut
```

Kode 11.16 (Lanjutan) *cashier_v9.py: kasir v0.9 dalam Python.*

```
            print("  *** the power goes off ***", flush=True)
            os._exit(1)                          # stop at once, no clean-up
        else:
            print("  unknown command '%s'" % cmd)
        if changed and not save_shop(shop, DB):
            print("  WARNING: not saved - write this order on paper")
    print("Bye. %d picked up (Rp %s), %d still in the shop."
          % (shop.picked, rupiah(shop.takings), shop.count))


if __name__ == "__main__":
    main()
```

11.10 Menjalankan dan Mengujinya

Uji untuk v0.9 menjalankan program dua kali di folder yang sama, dimulai tanpa `shop.csv` sama sekali. Sesi pertama menerima tiga pesanan, memajukan dua di antaranya, lalu listrik padam; baris terakhir berkas, yang diketik sesudah listrik padam, tidak pernah terbaca:

```
add fold 3.5 andi pratama
add wash-iron 2 budi santoso
add iron 4 citra dewi lestari
advance 1
advance 2
advance 2
poweroff
add fold 9 this order is never typed
```

Sesi kedua, setelah listrik menyala kembali, menampilkan pesanan, menyerahkan satu, dan menerima satu yang baru:

```
list
pickup 2
add fold 5 dewi anggraini
quit
```

Penguji menandai kedua jalan program sebagai `step 1` dan `step 2`, dan menampilkan `shop.csv` sebagaimana adanya di akhir:

```
$ step 1
Profitina Laundry cashier v0.9 - 0 order(s) loaded from shop.csv
> add fold 3.5 andi pratama
  #1 Andi Pratama, fold, 3.5 kg = Rp 24.500 (waiting)
> add wash-iron 2 budi santoso
  #2 Budi Santoso, wash-iron, 2.0 kg = Rp 20.000 (waiting)
> add iron 4 citra dewi lestari
  #3 Citra Dewi Lestari, iron, 4.0 kg = Rp 24.000 (waiting)
> advance 1
  #1 Andi Pratama, fold, 3.5 kg = Rp 24.500 (washing)
> advance 2
  #2 Budi Santoso, wash-iron, 2.0 kg = Rp 20.000 (washing)
> advance 2
  #2 Budi Santoso, wash-iron, 2.0 kg = Rp 20.000 (ready)
> poweroff
  *** the power goes off ***
$ step 2
Profitina Laundry cashier v0.9 - 3 order(s) loaded from shop.csv
> list
   No  Customer             Service     Weight    Amount  Status
    1  Andi Pratama         fold        3.5 kg    24.500  washing
    2  Budi Santoso         wash-iron   2.0 kg    20.000  ready
    3  Citra Dewi Lestari   iron        4.0 kg    24.000  waiting
  3 order(s) in the shop
> pickup 2
  #2 Budi Santoso picked up, paid Rp 20.000
> add fold 5 dewi anggraini
  #4 Dewi Anggraini, fold, 5.0 kg = Rp 35.000 (waiting)
> quit
Bye. 1 picked up (Rp 20.000), 3 still in the shop.
--- shop.csv ---
id,service,tenths,status,customer
1,fold,35,washing,Andi Pratama
3,iron,40,waiting,Citra Dewi Lestari
4,fold,50,waiting,Dewi Anggraini
```

Inilah seluruh bab dalam satu layar. Sesi pertama dimatikan tanpa salam perpisahan dan tanpa penyimpanan apa pun di akhir, tetapi sesi kedua menemukan ketiga pesanan, *lengkap dengan statusnya*: Andi sedang dicuci, Budi siap, Citra menunggu. Setiap perubahan sudah disimpan pada

saat terjadinya. Setelah pengambilan dan pesanan baru, berkas memuat tepat tiga pesanan yang ada di toko, dan pesanan baru diberi nomor 4, sesudah id terbesar di berkas.

Uji kedua dimulai dari `shop.csv` yang rusak, `data/shop_damaged.csv`, yang memiliki id ganda di baris 4, layanan tak dikenal di baris 5, dan baris tanpa nama di baris 6:

```
id,service,tenths,status,customer
1,fold,35,washing,Andi Pratama
3,iron,40,waiting,Citra Dewi Lestari
3,fold,20,ready,Budi Santoso
4,dry-clean,50,waiting,Eko Wibowo
5,wash-iron,15,ready
6,wash-iron,15,ready,Fajar Nugroho
```

```
  shop.csv line 4 is damaged - skipped
  shop.csv line 5 is damaged - skipped
  shop.csv line 6 is damaged - skipped
Profitina Laundry cashier v0.9 - 3 order(s) loaded from shop.csv
> list
   No  Customer              Service     Weight    Amount  Status
    1  Andi Pratama          fold        3.5 kg    24.500  washing
    3  Citra Dewi Lestari    iron        4.0 kg    24.000  waiting
    6  Fajar Nugroho         wash-iron   1.5 kg    15.000  ready
  3 order(s) in the shop
> quit
Bye. 0 picked up (Rp 0), 3 still in the shop.
```

Tiga baris dilaporkan dan dilewati, tiga pesanan dimuat, dan kasir bekerja. Mbak Rina dapat membuka berkas itu di penyunting teks, melihat baris 4 sampai 6, dan memutuskan seharusnya apa isinya. Seperti di setiap bab, penguji memeriksa semua ini untuk C dan Python, versi C di bawah AddressSanitizer, yang pemeriksaan kebocorannya menegaskan bahwa jalur muat-lalu-keluar pun membebaskan setiap pesanan.

11.11 Membuatnya Lebih Baik

Simpan catatan hari kemarin. Kasir hanya menyimpan pesanan yang *ada di toko*. Ketika pesanan diambil, ia lenyap dari `shop.csv`, sehingga tidak ada catatan usaha hari itu. Menambahkan satu baris per pengambilan ke berkas kedua, `pickups.log`, dengan mode `"a"`, berisi tanggal, id, nama, dan jumlah, membangun catatan itu dengan tepat; Soal Latihan 11.4 memintanya, dan `csv_report` dari Subbab 11.5 lalu dapat melaporkannya.

Simpan cadangan. Berkas di satu laptop hilang bersama laptopnya. Menyalin `shop.csv` dan log ke *flashdisk* atau penyimpanan awan setiap malam hanya perlu semenit.

Kenali kapan berkas tidak lagi cukup. Berkas CSV yang ditulis ulang setelah setiap perubahan sempurna untuk satu kasir, satu komputer, dan beberapa ratus pesanan. Ketika dua kasir harus mengolah pesanan yang sama sekaligus, ketika ada ratusan ribu rekaman, atau ketika laporan harus mencarinya dengan banyak cara, alat yang tepat adalah **basis data**, program yang seluruh tugasnya menyimpan rekaman dengan aman, membiarkan beberapa program mengubahnya sekaligus tanpa mencampuradukkan perubahan mereka, dan menemukannya dengan cepat. Sistem seperti itu dibangun dari bagian-bagian yang sama dengan bab ini, berkas, halaman berukuran tetap, `fseek`, log, dan penggantian nama yang hati-hati, hanya jauh lebih banyak. Semuanya menjadi pokok bahasan mata pelajaran lanjutan.

11.12 Memakainya secara Nyata

Teknik-teknik bab ini berlaku untuk program kecil apa pun yang tidak boleh lupa. Seorang penjahit menyimpan pesanan yang masih berjalan dalam berkas CSV yang dimuat program saat mulai dan disimpan setelah setiap perubahan; kebun buah dan durian menyimpan catatan panennya dalam mode tambah, satu baris per peti berisi tanggal, blok pohon, dan beratnya, lalu melaporkannya di akhir musim dengan program seperti `csv_report`; toko kecil menyimpan daftar harganya dalam berkas CSV yang disunting pemilik di lembar kerja dan dimuat program kasir setiap pagi.

Saat menyesuaikan program-programnya, ikuti aturan yang sudah mereka taati:

- **Periksa setiap `fopen`**, dan laporkan kegagalan beserta nama berkasnya dengan `perror`.
- **Baca dengan hasil pembacaan**, jangan pernah dengan `feof`.
- **Perlakukan setiap berkas sebagai masukan dari orang asing.** Periksa setiap ruas, laporkan baris rusak beserta nomornya, dan lanjutkan dengan yang baik.
- **Simpan ke berkas sementara lalu ganti nama**, jangan pernah dengan menimpa satu-satunya salinan.
- **Utamakan teks** untuk berkas yang dibaca manusia atau dipertukarkan program lain, dan nyatakan tata letaknya, seperti baris judul `shop.csv`.
- **Perhatikan data pribadi.** Berkas berisi nama dan nomor telepon adalah data pribadi menurut Undang-Undang Pelindungan Data Pribadi (UU PDP, UU Nomor 27 Tahun 2022); simpan di komputer yang berkata sandi, dan hapus yang tidak lagi diperlukan usaha.

11.13 Kesalahan yang Sering Terjadi, dan Cara Menghindarinya

- **Tidak memeriksa `fopen`.** Setiap `fopen` bisa gagal. Uji terhadap `NULL` dan laporkan nama serta alasannya dengan `perror`.
- **Membuka dengan `"w"` padahal maksudnya `"r"` atau `"a"`.** `"w"` langsung mengosongkan berkas. Periksa ulang mode setiap `fopen` yang menyentuh data berharga, dan simpan lewat berkas sementara.
- **Berulang dengan `while (!feof(fp))`.** Ia berjalan satu kali terlalu banyak (**Kode 11.3**). Berulanglah berdasarkan hasil `fscanf`, `fgets`, `fgetc`, atau `fread`.
- **Menyimpan hasil `fgetc` dalam `char`.** `EOF` tidak muat; pakai `int`.
- **Membaca nama dengan `%s`.** Ia berhenti di spasi pertama. Taruh ruas berspasi di akhir dan baca dengan *scan set* berbatas seperti `%31[^\n]`, atau baca barisnya dengan `fgets` lalu pisahkan.
- **Melupakan `'\r'` pada berkas Windows.** Akhir baris `"\r\n"` yang dibaca di Linux menyisakan `'\r'` di ujung ruas terakhir, dan `"fold\r"` bukan `"fold"`. Buang keduanya dengan `strcspn(line, "\r\n")`.
- **Memisah CSV dengan `strtok`.** Ia melewati ruas kosong, sehingga nilai yang hilang menggeser setiap ruas sesudahnya satu tempat ke kiri. Pakai pemisah yang mempertahankan ruas kosong.
- **Melupakan `fclose`, atau mengabaikan hasilnya.** Data dalam penyangga baru sampai ke berkas ketika aliran dikosongkan atau ditutup, dan disk yang penuh dilaporkan oleh `fclose`.
- **Berbagi berkas biner antarmesin.** Tata letak struktur dan urutan byte berbeda; pakai teks untuk data yang berpindah.
- **Menulis program multiberkas tanpa *include guard*.** Dua penyertaan *header* yang sama mendefinisikan ulang tipe-tipenya. Lindungi setiap *header*.

11.14 Ringkasan Bab

- RAM lupa ketika listrik padam; berkas di penyimpanan tidak. C mencapai berkas lewat aliran bertipe `FILE *`, dibuka dengan `fopen`, yang mengembalikan `NULL` bila gagal, dan ditutup dengan `fclose`, yang juga mengosongkan penyangga. `stdin`, `stdout`, dan `stderr` sudah terbuka untuk setiap program.
- Mode menentukan apa yang terjadi saat membuka (**Tabel 11.1**); `"w"` langsung mengosongkan berkas, `"a"` selalu menulis di akhir.
- Berkas teks ditulis dengan `fprintf` dan dibaca dengan `fscanf` atau `fgets`; perulangan baca menguji hasil setiap pembacaan, tidak pernah `feof`.
- Berkas data nyata dibaca baris demi baris, dipisah menjadi ruas-ruas yang boleh kosong, dan diperiksa ruas demi ruas; baris rusak dilaporkan beserta nomornya lalu dilewati.
- Log tumbuh dalam mode tambah. Berkas biner menyimpan rekaman berukuran tetap yang dicapai langsung oleh `fseek` (Persamaan (11.1)), **Gambar 11.1**), tetapi terikat pada satu platform.
- Penyimpanan yang aman menulis berkas baru, memeriksa `ferror` dan `fclose`, lalu mengganti namanya menimpa berkas lama, sehingga listrik padam selalu menyisakan berkas yang utuh (**Gambar 11.2**).
- Program yang lebih besar dipecah menjadi modul dengan *header*, deklarasi `extern`, dan *include guard*.
- Kasir v0.9 memuat pesanannya saat mulai, menyimpan setelah setiap perubahan, selamat dari simulasi listrik padam, dan memuat berkas rusak tanpa kehilangan pesanan yang baik, dalam C dan Python dengan keluaran identik.

11.15 Soal Latihan

1. Apa bedanya menjalankan `write_read_text` dua kali dan menjalankan `activity_log` dua kali? Jelaskan dengan memakai mode-mode **Tabel 11.1**.
2. Tulis berkas `bad.txt` yang membuat perulangan `while (fscanf(fp, "%d %ld %31[^\n]", ...) == 3)` dalam **Kode 11.1** berhenti *sebelum* akhir berkas. Bagaimana program dapat mengetahui sesudahnya apakah ia berhenti di akhir berkas atau di baris yang rusak?
3. Dengan Persamaan (11.1), berikan posisi rekaman terakhir dalam berkas biner berisi 250 rekaman 32 byte, dan ukuran berkasnya. Tulis panggilan `fseek` yang menuju rekaman *terakhir* tanpa mengetahui berapa banyak rekaman dalam berkas.
4. Perluas kasir v0.9 agar setiap pengambilan menambahkan satu baris `id,service,tenths,amount,customer` ke `pickups.csv` (dengan baris judul bila berkasnya baru). Mengapa mode `"a"` tepat untuk berkas ini, padahal `shop.csv` disimpan dengan tulis-lalu-ganti-nama?
5. Tulis program `find_customer` dalam C dan Python yang mencari di berkas CSV seperti `data/week.csv` pelanggan yang namanya memuat teks tertentu, tanpa memedulikan huruf besar-kecil, dan mencetak baris-baris yang cocok beserta nomor barisnya, *tanpa* memuat seluruh berkas ke memori. Kapan cara ini lebih baik daripada memuat semuanya lebih dulu, dan kapan lebih buruk?
6. Dalam `save_shop`, urutan langkahnya adalah: tulis berkas sementara, tutup, periksa, ganti nama. Apa yang bisa salah bila program mengganti nama lebih dulu lalu menulis, yaitu mengganti nama `shop.csv` lama menjadi `shop.csv.old`, lalu menulis `shop.csv` baru dengan

`"w"`? Gambarkan listrik padam yang menghilangkan data dalam rancangan itu tetapi tidak mungkin menghilangkan data dalam rancangan Subbab 11.8.

11.16 Penyelesaian Soal Latihan

1. `write_read_text` membuka `orders.txt` dengan `"w"`, yang mengosongkan berkas setiap kali, sehingga setelah dua kali jalan berkasnya memuat tiga baris yang sama seperti setelah sekali. `activity_log` membuka `activity.log` dengan `"a"`, yang mempertahankan isi lama dan menulis di akhir, sehingga setiap jalan menambah satu baris: setelah dua kali jalan berkasnya berisi dua baris.

2. Misalnya, berkas yang baris keduanya `2 twenty Budi`: pesanan pertama terbaca (3 ruas), lalu `%ld` gagal pada `twenty`, `fscanf` mengembalikan 1, dan perulangan berhenti dengan satu pesanan walaupun masih ada baris sesudahnya. Setelah perulangan, `feof(fp)` salah (akhir belum tercapai) dan `ferror(fp)` salah (tidak ada galat baca), sehingga program tahu ada baris yang rusak dan dapat melaporkannya; seandainya perulangan berhenti di akhir berkas, `feof(fp)` akan benar.

3. Rekaman terakhir berindeks 249, jadi menurut Persamaan (11.1) ia dimulai di $249 \cdot 32 = 7968$, dan berkasnya berukuran $250 \cdot 32 = 8000$ byte. Tanpa mengetahui banyaknya, bergeraklah relatif terhadap akhir: `fseek(fp, -(long)sizeof(Record), SEEK_END);`, yang mundur 32 byte dari akhir.

4. Di `cmd_pickup`, sebelum `free(o)`, buka `pickups.csv` dengan `"a"`, tulis barisnya dengan `fprintf`, lalu tutup; untuk menulis baris judul hanya pada berkas baru, coba dulu membukanya dengan `"r"`: bila gagal, berkasnya belum ada. Mode tambah tepat karena log tidak pernah ditulis ulang, hanya diperpanjang: listrik padam selama penambahan paling buruk menghilangkan atau memotong baris *terakhir*, sementara setiap baris sebelumnya tetap utuh. `shop.csv`, sebaliknya, digantikan seluruhnya pada setiap penyimpanan, dan penggantian memerlukan tulis-lalu-ganti-nama.

5. Program membaca satu baris setiap kali dengan `fgets`, menemukan ruas pelanggan di antara koma pertama dan kedua dengan `strchr`, dan membandingkan dengan pencarian yang tidak memedulikan huruf besar-kecil (`contains_ci` dari Bab 9). Hanya satu baris yang pernah berada di memori, sehingga ia bekerja pada berkas sebesar apa pun; ini lebih baik untuk satu kali pencarian di berkas besar. Memuat semuanya lebih dulu lebih baik bila program akan menjawab banyak pertanyaan tentang data yang sama, karena berkasnya lalu hanya dibaca sekali. **Kode 11.17** dan **Kode 11.18** adalah kedua versinya.

Kode 11.17 `find_customer.c`: mencari di berkas CSV baris demi baris.

```c
/* find_customer.c - Exercise 11.5: search a CSV file line by line for a
 * customer, without loading the whole file into memory.
 * Usage: find_customer <file.csv> <part of a name> */
#include <ctype.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

static int contains_ci(const char *text, const char *part)
{
    size_t n = strlen(part);
    for (size_t i = 0; text[i] != '\0'; i++) {
        size_t k = 0;
        while (k < n && text[i + k] != '\0' &&
               tolower((unsigned char)text[i + k]) ==
               tolower((unsigned char)part[k]))
            k++;
        if (k == n)
            return 1;
    }
    return 0;
}

int main(int argc, char *argv[])
{
    if (argc != 3) {
        fprintf(stderr, "usage: %s <file.csv> <part of a name>\n", argv[0]);
        return EXIT_FAILURE;
    }
    FILE *fp = fopen(argv[1], "r");
    if (fp == NULL) {
        perror(argv[1]);
        return EXIT_FAILURE;
    }
    char line[256];
    int lineno = 0, hits = 0;
    while (fgets(line, sizeof line, fp) != NULL) {  /* one line in memory */
        lineno++;
        line[strcspn(line, "\r\n")] = '\0';
        char *start = strchr(line, ',');            /* customer: 2nd field */
        if (lineno == 1 || start == NULL)
            continue;
        char *end = strchr(start + 1, ',');
        if (end != NULL)
            *end = '\0';
        if (contains_ci(start + 1, argv[2])) {
```

Kode 11.17 (Lanjutan) *find_customer.c: mencari di berkas CSV baris demi baris.*

```c
            if (end != NULL)
                *end = ',';                         /* put the comma back */
            printf("line %2d: %s\n", lineno, line);
            hits++;
        }
    }
    fclose(fp);
    printf("%d line(s) match '%s'\n", hits, argv[2]);
    return 0;
}
```

Kode 11.18 `find_customer.py`: pencarian yang sama dalam Python.

```python
# find_customer.py - Exercise 11.5: search a CSV file line by line for a
# customer, without loading the whole file into memory.
# Usage: python3 find_customer.py <file.csv> <part of a name>
import sys


def main():
    if len(sys.argv) != 3:
        print("usage: find_customer.py <file.csv> <part of a name>",
              file=sys.stderr)
        sys.exit(1)
    part = sys.argv[2].lower()
    hits = 0
    with open(sys.argv[1], encoding="utf-8") as fp:
        for lineno, line in enumerate(fp, start=1):   # one line in memory
            line = line.rstrip("\r\n")
            fields = line.split(",")
            if lineno == 1 or len(fields) < 2:
                continue
            if part in fields[1].lower():
                print("line %2d: %s" % (lineno, line))
                hits += 1
    print("%d line(s) match '%s'" % (hits, sys.argv[2]))


if __name__ == "__main__":
    main()
```

`./find_customer ../data/week.csv an` mencetak:

```
line  2: 2026-09-21,Andi Pratama,fold,3.5
line  3: 2026-09-21,Budi Santoso,wash-iron,2
line  5: 2026-09-22,Dewi Anggraini,fold,5.5
line  9: 2026-09-23,Hadi Susanto,iron
line 15: 2026-09-25,Made Wirawan,fold,3.x
5 line(s) match 'an'
```

Perhatikan bahwa “an” cocok di dalam “Santoso” dan “Susanto” selain “Andi”; dan bahwa baris 9, yang ditolak `csv_report` karena kurang satu ruas, tetap ditemukan, karena program ini hanya melihat nama.

6. Dengan ganti-nama-lalu-tulis, ada saat ketika berkas lama sudah dipindahkan namanya dan `shop.csv` baru belum lengkap. Bila listrik padam saat itu, program berikutnya menemukan `shop.csv` yang hilang atau setengah tertulis; data lengkapnya ada di `shop.csv.old`, tetapi program tidak mencarinya di sana, sehingga pesanan hari itu tampak hilang kecuali ada yang tahu untuk mengembalikan nama berkasnya. Dalam rancangan Subbab 11.8, nama `shop.csv` merujuk ke berkas yang utuh pada setiap saat sebelum penggantian nama dan pada setiap saat sesudahnya, sehingga listrik padam kapan pun menyisakan sesuatu yang utuh untuk dimuat program.

Daftar Pustaka

[1] ISO/IEC 9899:2018, *Information technology — Programming languages — C* (C17). Geneva, Switzerland: ISO, 2018, §7.21 “Input/output <stdio.h>” and §7.22.4.5 “The _Exit function”.

[2] cppreference.com, “File input/output.” [Online]. Available: https://en.cppreference.com/w/c/io (accessed Sep. 2026).

[3] Python Software Foundation, *The Python Standard Library*, Python 3.14: “Built-in Functions: open,” “struct — Interpret bytes as packed binary data,” “csv — CSV File Reading and Writing,” and “os.replace.” [Online]. Available: https://docs.python.org/3/library/ (accessed Sep. 2026).

[4] Y. Shafranovich, “Common Format and MIME Type for Comma-Separated Values (CSV) Files,” RFC 4180, IETF, Oct. 2005. [Online]. Available: https://www.rfc-editor.org/rfc/rfc4180

[5] The Open Group, “rename,” *The Open Group Base Specifications Issue 8* (IEEE Std 1003.1-2024). [Online]. Available: https://pubs.opengroup.org/onlinepubs/9799919799/functions/rename.html (accessed Sep. 2026).

[6] Microsoft, “rename, _wrename,” *C runtime library reference*. [Online]. Available: https://learn.microsoft.com/en-us/cpp/c-runtime-library/reference/rename-wrename (accessed Sep. 2026).

[7] P. J. Deitel and H. M. Deitel, *C How to Program*, 9th ed. Hoboken, NJ, USA: Pearson, 2022, ch. 11 “File Processing” and ch. 13 “The Preprocessor.”

Bab 12

Evaluasi dan Pembahasan III: Senarai, Pohon, dan Seekor Kuda Catur

Bab ini membahas tiga soal utuh yang memakai seluruh isi Bab 9 sampai 11 sekaligus: struktur yang menunjuk struktur lain, memori yang diambil dengan `malloc` dan dikembalikan dengan `free`, teks yang dibaca dari berkas lalu dipotong menjadi perintah, serta pengujian yang cermat. Soal pertama mengurutkan *singly linked list* lalu menyisipkan di ujungnya. Soal kedua menumbuhkan pohon biner dari berkas berisi perintah navigasi. Soal ketiga mencari banyak langkah paling sedikit yang diperlukan kuda catur di antara dua petak, dengan antrean yang dibangun sebagai *linked list* dan metode yang disebut *breadth-first search* (penelusuran melebar). Setiap soal dinyatakan ulang dengan kata-kata kita sendiri, dipahami secara manual, dirancang di atas kertas, diprogram dalam C dan Python dengan keluaran identik, dijalankan pada uji yang telah disiapkan, lalu dibandingkan dengan penyelesaian keliru yang menggoda, yang tampak benar padahal tidak. Di akhir bab Anda akan mampu menghadapi soal semacam ini dengan cara yang sama secara mandiri: gambar dulu, rancang kedua, ketik ketiga, dan uji sampai programnya meyakinkan Anda, bukan sebaliknya.

12.1 Posisi Kita Sekarang

Bab 9 mengajari program menangani teks, Bab 10 mengelompokkan data menjadi struktur dan merangkai struktur menjadi *linked list*, dan Bab 11 membaca serta menulis berkas. Ketiga soal dalam bab ini satu tingkat di atas program-program kasir, karena ketiganya adalah soal *struktur data*: pada masing-masing, seluruh kesulitannya terletak pada menjaga sekumpulan pointer tetap benar selagi strukturnya berubah. Ketiganya juga cicipan pertama dari bidang yang lebih luas. *Linked list*, pohon, dan *breadth-first search* [1], [2] adalah topik pembuka pelajaran struktur data, dan pembaca yang telah menyelesaikan ketiganya secara manual dan dengan program sudah selangkah lebih maju di sana.

Ketiga soal saling lepas, dan cocok untuk tim kecil: bila Anda bekerja bersama dua teman, masing-masing dapat memimpin satu soal, tetapi semuanya sebaiknya membaca dan menggambar ketiga soal lebih dulu, dan masing-masing sebaiknya menelaah program yang lain. Program yang hanya Anda baca dan tidak pernah Anda pertanyakan bukanlah program yang Anda pahami.

12.2 Cara Menghadapi Soal Struktur Data

12.2.1 Langkah-langkahnya

Setiap soal di bawah dikerjakan dengan urutan yang sama, dan urutan itu lebih penting daripada trik apa pun.

1. **Nyatakan ulang soalnya dengan kata-kata sendiri.** Tuliskan apa yang masuk, apa yang keluar, dan setiap keputusan yang dibiarkan terbuka oleh soal. Keputusan yang tidak Anda tulis akan Anda ambil dua kali, dengan cara berbeda.

2. **Kerjakan contoh kecil secara manual, dengan gambar.** Kotak untuk simpul, panah untuk pointer, satu gambar per langkah. Untuk soal pointer, langkah ini bukan kemewahan: bila Anda tidak dapat menggambar apa yang terjadi, Anda tidak dapat memprogramnya.
3. **Rancang di atas kertas.** Pilih strukturnya dan tulis algoritmanya sebagai pseudokode, baris-baris pendek bernomor tanpa detail C. Perkirakan biayanya.
4. **Programkan**, satu fungsi demi satu, dengan menguji setiap fungsi secara terpisah sebelum menggabungkannya.
5. **Uji dengan kasus yang disiapkan**: contoh dari soal, kasus tepi (kosong, satu elemen, sudah terurut, terurut terbalik, nilai kembar, masukan terbesar yang diizinkan) dan masukan yang salah. Ketahui jawaban yang diharapkan *sebelum* menjalankan.
6. **Jalankan di bawah *sanitizer*** (`-fsanitize=address,undefined`), yang melaporkan setiap pointer nyasar, setiap pemakaian memori yang sudah dibebaskan, dan setiap kebocoran. Program pointer yang belum dijalankan di bawah *sanitizer* belum diuji.
7. **Bedah penyelesaian keliru yang menggoda.** Mengetahui *mengapa* jalan pintas yang tampak jelas itu gagal adalah cara menghindarinya lain kali.
8. **Periksa diri dengan varian.** Ubah soalnya sedikit dan selesaikan lagi tanpa melihat.

12.2.2 Letak kodenya

Program-program bab ini berada di folder `Chapter12`, C di `c/` dan Python di `python/`, dengan setiap masukan uji di `data/` dan keluaran yang diharapkan di `tests/expected/`. `python3 tests/run_tests.py` mengompilasi setiap program C dengan peringatan sebagai galat dan dengan *sanitizer*, menjalankan versi C dan Python pada setiap uji, dan memeriksa bahwa keduanya mencetak persis keluaran yang diharapkan. Semua keluaran dalam bab ini adalah keluaran nyata dari jalan-jalan program itu.

12.3 Soal 12.1: Urutkan Linked List, Lalu Sisipkan di Akhir

12.3.1 Soalnya

> **SOAL 12.1.** Diberikan *singly linked list* berisi sedikitnya lima bilangan bulat, dibangun dari simpul `struct sNode { int data; struct sNode *next; };`. Urutkan senarai itu secara menaik dengan algoritma buatan sendiri (tanpa fungsi pengurutan pustaka), lalu sisipkan satu bilangan lagi di *akhir* senarai yang sudah terurut. Cetak senarai sebelum diurutkan, sesudah diurutkan, dan sesudah penyisipan.

Dengan kata lain: serangkaian simpul memuat bilangan dalam urutan tertentu; kita harus menyusunnya ulang dari kecil ke besar, lalu menyambungkan simpul baru berisi bilangan tertentu sesudah simpul terakhir, walaupun itu merusak urutannya. Soal membiarkan bentuk masukan terbuka, jadi kita memutuskan: program membaca bilangan-bilangan senarai dari baris pertama masukan standar, dan bilangan yang akan disisipkan dari baris kedua. Kita juga memutuskan bahwa program harus bekerja untuk senarai *apa pun*, termasuk senarai kosong dan senarai berisi satu bilangan, walaupun soal menjanjikan sedikitnya lima: program hidup lebih lama daripada soalnya. Terakhir kita memutuskan untuk mencetak berapa kali lintasan (*pass*) dan pertukaran (*swap*) yang diperlukan pengurutan, yang hanya memerlukan dua pencacah dan membuat uji lebih bermakna.

12.3.2 Memahaminya secara manual

Contoh utama kita adalah senarai 29, 4, 17, 4, 42, 8, 15 dan bilangan 23. Senarai terurutnya 4, 4, 8, 15, 17, 29, 42, dan setelah penyisipan di akhir, 4, 4, 8, 15, 17, 29, 42, 23. Perhatikan bilangan terakhir: "sisipkan di akhir" berarti *di akhir*, bukan "di tempat terurutnya".

Pengurutan mana? Yang kita kenal dari Bab 6 dan paling cocok untuk senarai adalah **bubble sort**, karena ia hanya membandingkan dan menukar *tetangga*, dan dalam *singly linked list* tetangga sebuah simpul tepat satu `->next` jauhnya. Ada juga pilihan tentang *apa* yang ditukar. Kita dapat menukar simpulnya, dengan menyambung ulang pointer, atau membiarkan simpul di tempatnya dan menukar bilangan di dalamnya. Menukar ruas `data` hanya perlu tiga penugasan dengan variabel sementara `int`, persis seperti di Bab 6, dan tanpa perubahan pointer sama sekali; untuk simpul yang memuat satu bilangan, jelas itu pilihan yang lebih sederhana. (Soal Latihan 12.2 menyambung ulang simpul.)

Inilah penelusurannya. Setiap lintasan menyusuri senarai dari `head`, membandingkan setiap simpul dengan penerusnya dan menukar bila urutannya salah; bilangan terbesar yang belum terurut "menggelembung" ke ujung bagian yang belum terurut.

```
start   29  4 17  4 42  8 15
pass 1   4 17  4 29  8 15 | 42     5 swaps
pass 2   4  4 17  8 15 | 29 42     3 swaps
pass 3   4  4  8 15 | 17 29 42     2 swaps
pass 4   4  4  8 | 15 17 29 42     0 swaps -> sorted, stop
```

Garis `|` menandai awal ekor yang sudah terurut: setelah lintasan ke-k, k bilangan terbesar sudah di tempat akhirnya, sehingga lintasan ke-$k+1$ dapat berhenti satu simpul lebih awal. Dan ketika satu lintasan tidak melakukan pertukaran sama sekali, setiap pasangan tetangga sudah berurutan, sehingga seluruh senarai terurut dan kita dapat berhenti lebih awal. Di sini itu terjadi pada lintasan 4, sehingga pengurutan memerlukan 4 lintasan dan $5 + 3 + 2 = 10$ pertukaran.

Sepuluh bukan kebetulan. *Inversi* adalah pasangan posisi $i < j$ yang bilangannya salah urutan, $a_i > a_j$. Setiap pertukaran dua tetangga yang salah urutan menghapus tepat satu inversi dan tidak membuat inversi baru, dan senarai terurut tidak punya inversi, sehingga

$$\text{banyak pertukaran bubble sort} = \#\{(i,j) : i < j \text{ dan } a_i > a_j\} \tag{12.1}$$

artinya banyak pertukaran sama dengan banyak inversi. Untuk senarai kita: 29 lebih besar daripada lima bilangan sesudahnya kecuali 42, jadi 5 inversi; 4 yang pertama tidak punya; 17 lebih besar daripada 4, 8, dan 15, jadi 3; 4 yang kedua tidak punya; 42 lebih besar daripada 8 dan 15, jadi 2; 8 tidak punya. Totalnya $5 + 3 + 2 = 10$, sesuai hasil penelusuran. Persamaan (12.1) memberi banyak pertukaran setiap uji sebelumnya, persis seperti yang diminta langkah 5 Subbab 12.2.

Penyisipan di akhir: susuri dari `head` selama simpul saat ini *memiliki* penerus, sehingga penelusuran berhenti *pada* simpul terakhir, bukan sesudahnya; sambungkan simpul baru di situ. Bila senarai kosong, tidak ada simpul terakhir, dan simpul baru menjadi `head`.

12.3.3 Merancangnya di atas kertas

Pseudokode 12.1 adalah *bubble sort* pada senarai, dengan dua perbaikan dari penelusuran tadi: pointer `last` menandai awal ekor yang terurut, dan penanda `swapped` menghentikan pengurutan setelah lintasan tanpa pertukaran.

Pseudokode 12.1 *Bubble sort* pada *singly linked list*, dengan menukar bilangan simpul-simpul yang bertetangga.

```
LIST-BUBBLE-SORT(head)
1  last = NIL                        // everything from last on is sorted
2  swapped = TRUE
3  while head != NIL and swapped
4      swapped = FALSE
5      p = head
6      while p.next != last
7          if p.data > p.next.data
8              exchange p.data with p.next.data
9              swapped = TRUE
10         p = p.next
11     last = p                      // p is the last node this pass looked at
```

Baris 6 adalah bagian yang halus. Pada lintasan pertama, `last` bernilai `NIL`, sehingga perulangan dalam berjalan selama `p.next` bukan `NIL`, yaitu sampai `p` adalah simpul terakhir; ia tidak pernah melihat `p.next.data` ketika `p.next` bernilai `NIL`. Pada lintasan-lintasan berikutnya ia berhenti lebih awal, di simpul sebelum ekor yang terurut. Baris 11 mencatat di mana lintasan ini berhenti.

Biayanya. Untuk n bilangan, lintasan ke-k melakukan paling banyak $n-k$ perbandingan, sehingga seluruh lintasan bersama-sama melakukan paling banyak $(n-1)+(n-2)+\cdots+1 = \frac{n(n-1)}{2}$ perbandingan, yaitu $O(n^2)$; senarai yang sudah terurut cukup satu lintasan dengan $n-1$ perbandingan. Penyisipan di akhir menyusuri n simpul, $O(n)$. Untuk senarai berisi beberapa ribu bilangan, semua ini seketika.

12.3.4 Penyelesaian dalam C

Kode 12.1 mengikuti rancangan baris demi baris. `insert_end` dipakai dua kali: untuk membangun senarai dari masukan, dan untuk menyisipkan bilangan tambahan; ia menangani senarai kosong dengan mengembalikan simpul baru sebagai `head` baru, karena itulah ia mengembalikan `head` dan pemanggil menulis `head = insert_end(head, value)`. Bilangan-bilangan di baris pertama dibaca dengan `sscanf` dan konversi `%n`, yang menyimpan berapa karakter yang sudah terpakai, sehingga perulangan dapat melangkah menyusuri baris bilangan demi bilangan.

Kode 12.1 `list_sort.c`: *bubble sort* pada *linked list* dengan menukar data, lalu penyisipan di akhir.

```c
/* list_sort.c - Problem 12.1: sort a singly linked list in ascending order
 * without any library sort, then insert one value at the end.
 * Input (standard input): line 1 = the values, line 2 = the value to insert. */
#include <stdio.h>
#include <stdlib.h>

struct sNode {
    int data;
    struct sNode *next;
};

/* Append value after the last node; works for an empty list too. */
static struct sNode *insert_end(struct sNode *head, int value)
{
    struct sNode *n = malloc(sizeof *n);
    if (n == NULL) {
        fprintf(stderr, "out of memory\n");
        exit(EXIT_FAILURE);
    }
    n->data = value;
    n->next = NULL;
    if (head == NULL)
        return n;                       /* the new node is the whole list */
    struct sNode *p = head;
    while (p->next != NULL)             /* stop ON the last node */
        p = p->next;
    p->next = n;
    return head;
}

static void print_list(const char *label, const struct sNode *p)
{
    printf("%-15s:", label);
    for (; p != NULL; p = p->next)
        printf(" %d ->", p->data);
    printf(" NULL\n");
}
```

Kode 12.1 (Lanjutan) *list_sort.c: bubble sort pada linked list dengan menukar data, lalu penyisipan di akhir.*

```c
/* Bubble sort on the list: swap the DATA of neighbours that are out of
 * order. After each pass the largest remaining value has sunk to the end,
 * so the pass stops one node earlier (at 'last') the next time. */
static void bubble_sort(struct sNode *head, int *passes, int *swaps)
{
    struct sNode *last = NULL;          /* everything from last on is sorted */
    int swapped = 1;
    *passes = *swaps = 0;
    while (head != NULL && swapped) {
        swapped = 0;
        struct sNode *p = head;
        while (p->next != last) {
            if (p->data > p->next->data) {
                int t = p->data;
                p->data = p->next->data;
                p->next->data = t;
                swapped = 1;
                (*swaps)++;
            }
            p = p->next;
        }
        last = p;                       /* p is the last node of this pass */
        (*passes)++;
    }
}

static void free_list(struct sNode *p)
{
    while (p != NULL) {
        struct sNode *next = p->next;
        free(p);
        p = next;
    }
}

int main(void)
{
    struct sNode *head = NULL;
    char line[512];
    int value, used, extra, passes, swaps;
```

Kode 12.1 (Lanjutan) *list_sort.c: bubble sort pada linked list dengan menukar data, lalu penyisipan di akhir.*

```c
    if (fgets(line, sizeof line, stdin) == NULL)
        return EXIT_FAILURE;
    for (const char *s = line; sscanf(s, "%d%n", &value, &used) == 1; s += used)
        head = insert_end(head, value);
    if (scanf("%d", &extra) != 1) {
        fprintf(stderr, "line 2 must hold the value to insert\n");
        free_list(head);
        return EXIT_FAILURE;
    }
    print_list("before sorting", head);
    bubble_sort(head, &passes, &swaps);
    print_list("after sorting", head);
    head = insert_end(head, extra);
    print_list("after insert", head);
    printf("passes: %d, swaps: %d\n", passes, swaps);
    free_list(head);
    return 0;
}
```

12.3.5 Penyelesaian dalam Python

Kode 12.2 membangun rantai objek `sNode` yang sama, bukan *list* Python, dan menukar dengan penugasan *tuple* Python `p.data, p.next.data = p.next.data, p.data`. Perbandingan `p.next is not last` memakai `is`, "objek yang sama", padanan Python untuk membandingkan dua pointer C.

Kode 12.2 `list_sort.py`: penyelesaian yang sama dalam Python.

```
# list_sort.py - Problem 12.1: sort a singly linked list in ascending order
# without any library sort, then insert one value at the end.
# Input (standard input): line 1 = the values, line 2 = the value to insert.
import sys


class sNode:
    def __init__(self, data):
        self.data = data
        self.next = None


def insert_end(head, value):
    n = sNode(value)
    if head is None:
        return n
    p = head
    while p.next is not None:          # stop ON the last node
        p = p.next
    p.next = n
    return head


def print_list(label, p):
    s = "%-15s:" % label
    while p is not None:
        s += " %d ->" % p.data
        p = p.next
    print(s + " NULL")


def bubble_sort(head):
    """Swap the data of neighbours that are out of order; return counts."""
    last, swapped, passes, swaps = None, True, 0, 0
    while head is not None and swapped:
        swapped = False
        p = head
        while p.next is not last:
            if p.data > p.next.data:
                p.data, p.next.data = p.next.data, p.data
                swapped = True
                swaps += 1
            p = p.next
        last = p
        passes += 1
    return passes, swaps
```

Kode 12.2 (Lanjutan) *list_sort.py: penyelesaian yang sama dalam Python.*

```
def main():
    lines = sys.stdin.read().split("\n", 1)
    head = None
    for word in lines[0].split():
        head = insert_end(head, int(word))
    rest = lines[1].split() if len(lines) > 1 else []
    if not rest:
        print("line 2 must hold the value to insert", file=sys.stderr)
        sys.exit(1)
    print_list("before sorting", head)
    passes, swaps = bubble_sort(head)
    print_list("after sorting", head)
    head = insert_end(head, int(rest[0]))
    print_list("after insert", head)
    print("passes: %d, swaps: %d" % (passes, swaps))


if __name__ == "__main__":
    main()
```

12.3.6 Menjalankan dan mengujinya

Contoh utamanya, `data/list1.txt`:

```
29 4 17 4 42 8 15
23
```

`./list_sort < ../data/list1.txt` mencetak:

```
before sorting : 29 -> 4 -> 17 -> 4 -> 42 -> 8 -> 15 -> NULL
after sorting  : 4 -> 4 -> 8 -> 15 -> 17 -> 29 -> 42 -> NULL
after insert   : 4 -> 4 -> 8 -> 15 -> 17 -> 29 -> 42 -> 23 -> NULL
passes: 4, swaps: 10
```

Empat lintasan dan sepuluh pertukaran, seperti diramalkan penelusuran manual dan Persamaan (12.1). Keempat uji tepi menghasilkan:

```
before sorting : 1 -> 2 -> 3 -> 4 -> 5 -> NULL
after sorting  : 1 -> 2 -> 3 -> 4 -> 5 -> NULL
after insert   : 1 -> 2 -> 3 -> 4 -> 5 -> 6 -> NULL
passes: 1, swaps: 0
before sorting : 9 -> 8 -> 7 -> 6 -> 5 -> 4 -> NULL
after sorting  : 4 -> 5 -> 6 -> 7 -> 8 -> 9 -> NULL
after insert   : 4 -> 5 -> 6 -> 7 -> 8 -> 9 -> 0 -> NULL
passes: 6, swaps: 15
before sorting : 7 -> NULL
after sorting  : 7 -> NULL
after insert   : 7 -> 3 -> NULL
passes: 1, swaps: 0
before sorting : NULL
after sorting  : NULL
after insert   : 5 -> NULL
passes: 0, swaps: 0
```

Senarai yang sudah terurut perlu satu lintasan dan tanpa pertukaran. Senarai terbalik 9, 8, 7, 6, 5, 4 memiliki setiap pasangan terbalik, $\frac{6 \cdot 5}{2} = 15$ inversi, dan memerlukan 15 pertukaran dalam 6 lintasan: kasus terburuk. Satu bilangan perlu satu lintasan, dan menyisipkan 3 sesudah 7 menunjukkan lagi bahwa penyisipan mengabaikan urutan. Senarai kosong mencetak `NULL` dua

kali, tidak perlu lintasan, dan penyisipan membuat simpul pertama. Kelimanya bersih di bawah *sanitizer*, yang juga menegaskan bahwa `free_list` mengembalikan setiap simpul.

12.3.7 Penyelesaian keliru yang menggoda

Jalan pintas yang populer melakukan *satu* lintasan pertukaran tetangga lalu berhenti. **Kode 12.3** mencobanya pada dua senarai, dan **Kode 12.4** melakukan hal yang sama dalam Python.

Kode 12.3 `wrong_list_sort.c`: satu lintasan pertukaran tetangga, yang bukan pengurutan.

```c
/* wrong_list_sort.c - a tempting wrong solution to Problem 12.1: ONE pass of
 * neighbour swaps. It moves the largest value to the end, and on a nearly
 * sorted list that is enough to look right. */
#include <stdio.h>
#include <stdlib.h>

struct sNode {
    int data;
    struct sNode *next;
};

static struct sNode *push_front(struct sNode *head, int value)
{
    struct sNode *n = malloc(sizeof *n);
    if (n == NULL)
        exit(EXIT_FAILURE);
    n->data = value;
    n->next = head;
    return n;
}

static void print_list(const char *label, const struct sNode *p)
{
    printf("%-9s:", label);
    for (; p != NULL; p = p->next)
        printf(" %d", p->data);
    printf("\n");
}

static void one_pass_sort(struct sNode *head)
{
    for (struct sNode *p = head; p != NULL && p->next != NULL; p = p->next)
        if (p->data > p->next->data) {
            int t = p->data;
            p->data = p->next->data;
            p->next->data = t;
        }
}
```

Kode 12.3 (Lanjutan) *wrong_list_sort.c: satu lintasan pertukaran tetangga, yang bukan pengurutan.*

```c
int main(void)
{
    const int tests[2][5] = {{1, 2, 4, 3, 5}, {5, 4, 3, 2, 1}};
    for (int t = 0; t < 2; t++) {
        struct sNode *head = NULL;
        for (int i = 4; i >= 0; i--)
            head = push_front(head, tests[t][i]);
        print_list("input", head);
        one_pass_sort(head);
        print_list("one pass", head);
        while (head != NULL) {
            struct sNode *next = head->next;
            free(head);
            head = next;
        }
    }
    return 0;
}
```

Kode 12.4 `wrong_list_sort.py`: kesalahan yang sama dalam Python.

```python
# wrong_list_sort.py - a tempting wrong solution to Problem 12.1: ONE pass of
# neighbour swaps, which only moves the largest value to the end.


class sNode:
    def __init__(self, data, next=None):
        self.data = data
        self.next = next


def print_list(label, p):
    s = "%-9s:" % label
    while p is not None:
        s += " %d" % p.data
        p = p.next
    print(s)


def one_pass_sort(head):
    p = head
    while p is not None and p.next is not None:
        if p.data > p.next.data:
            p.data, p.next.data = p.next.data, p.data
        p = p.next


def main():
    for test in ([1, 2, 4, 3, 5], [5, 4, 3, 2, 1]):
        head = None
        for v in reversed(test):
            head = sNode(v, head)
        print_list("input", head)
        one_pass_sort(head)
        print_list("one pass", head)


if __name__ == "__main__":
    main()
```

```
input    : 1 2 4 3 5
one pass : 1 2 3 4 5
input    : 5 4 3 2 1
one pass : 4 3 2 1 5
```

Pada senarai yang nyaris terurut 1, 2, 4, 3, 5, satu lintasan kebetulan cukup, dan pemrogram yang hanya menguji senarai itu yakin programnya benar. Pada senarai terbalik, satu lintasan membawa 5 ke ujung dan membiarkan sisanya tetap terbalik. Pelajaran umumnya: satu lintasan *bubble sort* hanya menjamin bilangan *terbesar* sampai di ujung; senarai dengan k inversi memerlukan lintasan sampai semuanya habis. Selalu uji dengan senarai terbalik.

Kesalahan klasik kedua ada pada penyisipan: `while (p->next != NULL)` dengan `p = head` membuat program *crash* pada senarai kosong, karena `head` bernilai `NULL` dan `p->next` membaca lewat pointer nol. Uji `list_sort_empty` ada justru untuk menangkap hal itu; *sanitizer* mengubah *crash* itu menjadi laporan yang jelas tentang barisnya.

12.3.8 Periksa diri

Urutkan senarai secara *menurun* dan sisipkan bilangan baru di *depan*. Satu karakter mana di **Pseudokode 12.1** yang berubah, dan fungsi mana yang menggantikan `insert_end`? (Jawaban: `>` di baris 7 menjadi `<`; fungsi barunya adalah `push_front` dua baris dari Bab 10.) Lalu ramalkan banyak pertukaran untuk 29, 4, 17, 4, 42, 8, 15 secara menurun: yaitu banyak pasangan $i < j$ dengan $a_i < a_j$, yaitu $\frac{7 \cdot 6}{2} = 21$ pasangan dikurangi 10 inversi dikurangi 1 pasangan bilangan yang sama, jadi 10.

12.4 Soal 12.2: Pohon yang Ditumbuhkan dari Berkas

12.4.1 Soalnya

> **SOAL 12.2.** Sebuah berkas teks menggambarkan pohon biner. Isinya pasangan-pasangan baris: baris perintah, lalu baris data berisi bilangan bulat. Perintah `0` membuat akar. Perintah `l` menyisipkan simpul baru sebagai anak *kiri* simpul saat ini. Perintah berbentuk `u…ur`, sejumlah huruf `u` diikuti satu `r`, lebih dulu naik dari simpul saat ini satu tingkat ke arah akar untuk setiap `u`, lalu menyisipkan simpul baru sebagai anak *kanan* simpul yang dicapai. Simpul yang baru disisipkan selalu menjadi simpul saat ini. Bangun pohon dari berkas itu dan tunjukkan bahwa bentuknya benar.

Dengan kata lain: berkas itu adalah resep untuk menggambar pohon, seperti petunjuk berjalan di dalam gedung: “mulai di sini; turun ke kiri dan taruh kotak; naik satu lantai, lalu turun ke kanan dan taruh kotak”. Kita harus mengikuti resepnya dan mencatat *di mana kita berada*. Kita memutuskan: nama berkas diberikan di baris perintah; baris kosong diabaikan; `r` polos (tanpa `u`) berarti “anak kanan simpul saat ini”; dan program berhenti dengan pesan yang jelas, menyebut barisnya, bila perintah tidak dikenal, bila baris data hilang atau bukan bilangan, bila perintah pertama bukan `0` atau muncul `0` kedua, bila pendakian akan melewati akar, atau bila tempat simpul baru sudah terisi. Untuk menunjukkan bentuknya, kita mencetak pohon dalam tiga urutan baku dan juga menyamping.

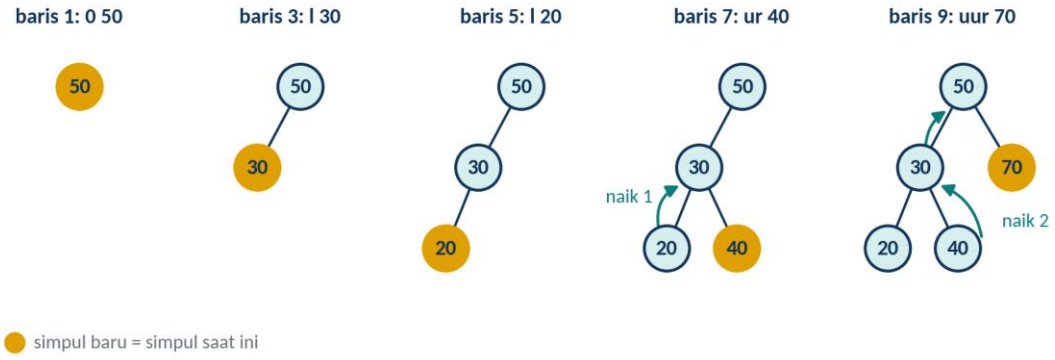
Pohon biner adalah struktur simpul-simpul yang setiap simpulnya memiliki paling banyak dua anak, anak *kiri* dan anak *kanan*, dan setiap simpul kecuali akar memiliki tepat satu *orang tua* (*parent*). Ia langkah alami berikutnya sesudah *linked list*: simpul senarai punya satu `next`, simpul pohon punya dua.

12.4.2 Memahaminya secara manual

Berkas contohnya, `data/tree5.txt`, satu butir per baris:

```
0
50
l
30
l
20
ur
40
uur
70
```

Gambar 12.1 menggambar pohon setelah setiap pasang baris. Baris 1 membuat akar 50, yang menjadi simpul saat ini. Baris 3, `l`: 30 menjadi anak kiri 50 dan kini menjadi simpul saat ini. Baris 5, `l`: 20 menjadi anak kiri 30, simpul saat ini. Baris 7, `ur`: dari simpul saat ini 20, naik *satu* tingkat, ke 30; 40 menjadi anak kanan 30, simpul saat ini. Baris 9, `uur`: dari simpul saat ini 40, naik satu tingkat ke 30 dan tingkat kedua ke 50; 70 menjadi anak kanan 50.



Gambar 12.1 Menumbuhkan pohon dari `tree5.txt`: setelah setiap perintah, simpul baru (kuning) menjadi simpul saat ini, dan perintah `u`…`ur` lebih dulu naik satu tingkat per `u` darinya.

Pengamatan kuncinya: pendakian dimulai dari simpul *saat ini*, yang baru saja disisipkan, dan mendaki memerlukan jalan ke atas. Simpul pohon biasanya hanya mengenal anak-anaknya. Jadi setiap simpul juga menyimpan pointer ke **orang tuanya**, yang diatur ketika simpul dibuat, dan "naik satu tingkat" cukup `at = at->parent`. Orang tua akar bernilai `NULL`, dan dengan cara itulah pendakian melewati akar terdeteksi.

Untuk *menunjukkan* bentuknya, kita mencetak pohon dalam tiga urutan klasik, semuanya didefinisikan secara rekursif (Bab 5): **preorder** mengunjungi simpul, lalu subpohon kirinya, lalu subpohon kanannya; **inorder** mengunjungi subpohon kiri, simpul, lalu subpohon kanan; **postorder** mengunjungi kedua subpohon lalu simpulnya. Untuk pohon kita hasil yang diharapkan: preorder 50 30 20 40 70, inorder 20 30 40 50 70, postorder 20 40 30 70 50. Dua pohon berbeda bisa memiliki inorder yang sama, tetapi tidak ada dua pohon berbeda yang preorder *dan* inorder-nya sama, sehingga mencetak keduanya membuktikan bentuknya. Cetakan menyamping membaringkan pohon di sisi kirinya: subpohon kanan di atas, subpohon kiri di bawah, dijorokkan menurut kedalaman, sehingga Anda dapat memutar halaman dan melihat pohonnya.

12.4.3 Merancangnya di atas kertas

Simpulnya adalah `struct TNode { int data; struct TNode *left, *right, *parent; }`. **Pseudokode 12.2** membangun pohonnya; `cur` adalah simpul saat ini.

Pseudokode 12.2 Membangun pohon biner dari perintah navigasi, dengan pointer orang tua di setiap simpul.

```
BUILD-TREE(file)
1  root = NIL, cur = NIL
2  for each command line cmd, followed by a data line with number x
3      if cmd = "0"
4          error if root != NIL
5          root = cur = NEW-NODE(x, parent = NIL)
6      else error if root = NIL
7      else if cmd = "l"
8          error if cur.left != NIL
9          cur.left = NEW-NODE(x, parent = cur);  cur = cur.left
10     else if cmd = u^k r   (k >= 0 letters u, then r)
11         at = cur
12         repeat k times: error if at.parent = NIL; at = at.parent
13         error if at.right != NIL
14         at.right = NEW-NODE(x, parent = at);  cur = at.right
15     else error "unknown command"
16 return root
```

Biayanya. Setiap simpul dibuat sekali; pendakian k tingkat memakan k langkah, dan k tidak pernah melebihi kedalaman pohon. Ketiga penelusuran mengunjungi setiap simpul sekali, $O(n)$. Membebaskan pohon adalah penelusuran postorder: kedua subpohon dulu, lalu simpulnya, sehingga tidak ada simpul yang dibebaskan selagi anak-anaknya masih perlu dicapai lewat simpul itu.

12.4.4 Penyelesaian dalam C

Kode 12.5 mewujudkan **Pseudokode 12.2** dalam `build`, yang mengembalikan banyak simpul yang dibuat, atau -1 setelah mencetak galat. `next_line` membaca baris tak kosong berikutnya dan menghitung nomor baris, sehingga setiap pesan dapat menyebut di mana masalahnya. `climbs` memeriksa bahwa perintah berbentuk `u…ur` dan mengembalikan banyaknya `u`. Baris data diperiksa dengan `sscanf(text, "%d %c", &data, &end) == 1`: tepat satu konversi harus berhasil, sehingga bilangan yang diikuti apa pun (yang akan dicocokkan `%c`) ditolak, begitu pula baris tanpa bilangan sama sekali.

Kode 12.5 `tree_from_file.c`: menumbuhkan pohon biner dari berkas perintah, mencetaknya dalam tiga urutan dan menyamping, lalu membebaskannya.

```c
/* tree_from_file.c - Problem 12.2: grow a binary tree from a file of
 * command lines and data lines, then print it.
 *   0      create the root (first command only)
 *   l      the new node becomes the LEFT child of the current node
 *   u...ur climb one parent per u from the current node, then the new node
 *          becomes the RIGHT child of the node reached (plain r: no climb)
 * The node just created becomes the current node.
 * Usage: tree_from_file <file> */
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

typedef struct TNode {
    int data;
    struct TNode *left, *right, *parent;
} TNode;

static TNode *new_node(int data, TNode *parent)
{
    TNode *n = malloc(sizeof *n);
    if (n == NULL) {
        fprintf(stderr, "out of memory\n");
        exit(EXIT_FAILURE);
    }
    n->data = data;
    n->left = n->right = NULL;
    n->parent = parent;
    return n;
}

static void free_tree(TNode *t)
{
    if (t == NULL)
        return;
    free_tree(t->left);
    free_tree(t->right);
    free(t);
}
```

Kode 12.5 (Lanjutan) *tree_from_file.c: menumbuhkan pohon biner dari berkas perintah, mencetaknya dalam tiga urutan dan menyamping, lalu membebaskannya.*

```c
/* Read the next non-empty line, without its line ending; 0 at the end. */
static int next_line(FILE *fp, char *buf, int size, int *lineno)
{
    while (fgets(buf, size, fp) != NULL) {
        (*lineno)++;
        buf[strcspn(buf, "\r\n")] = '\0';
        if (buf[0] != '\0')
            return 1;
    }
    return 0;
}

/* How many u's before the final r? -1 if cmd is not of the form u...ur. */
static int climbs(const char *cmd)
{
    size_t n = strlen(cmd);
    if (n == 0 || cmd[n - 1] != 'r')
        return -1;
    for (size_t i = 0; i + 1 < n; i++)
        if (cmd[i] != 'u')
            return -1;
    return (int)n - 1;
}

/* Returns the number of nodes created, or -1 after reporting an error. */
static int build(FILE *fp, TNode **root)
{
    char cmd[64], text[64];
    int lineno = 0, count = 0;
    TNode *cur = NULL;
    while (next_line(fp, cmd, sizeof cmd, &lineno)) {
        int cmd_line = lineno, data;
        char end;
        if (!next_line(fp, text, sizeof text, &lineno) ||
            sscanf(text, "%d %c", &data, &end) != 1) {
            printf("line %2d: a whole number must follow command '%s'\n",
                   cmd_line + 1, cmd);
            return -1;
        }
        if (strcmp(cmd, "0") == 0) {
            if (*root != NULL) {
                printf("line %2d: the tree already has a root\n", cmd_line);
                return -1;
            }
            cur = *root = new_node(data, NULL);
            printf("line %2d: %-5s-> %d is the root\n", cmd_line, cmd, data);
```

Kode 12.5 (Lanjutan) *tree_from_file.c: menumbuhkan pohon biner dari berkas perintah, mencetaknya dalam tiga urutan dan menyamping, lalu membebaskannya.*

```
        } else if (*root == NULL) {
            printf("line %2d: the first command must be 0\n", cmd_line);
            return -1;
        } else if (strcmp(cmd, "l") == 0) {
            if (cur->left != NULL) {
                printf("line %2d: %d already has a left child\n", cmd_line,
                       cur->data);
                return -1;
            }
            cur->left = new_node(data, cur);
            printf("line %2d: %-5s-> %d is the left child of %d\n",
                   cmd_line, cmd, data, cur->data);
            cur = cur->left;
        } else {
            int k = climbs(cmd);
            if (k < 0) {
                printf("line %2d: unknown command '%s'\n", cmd_line, cmd);
                return -1;
            }
            TNode *at = cur;
            for (int i = 0; i < k; i++) {
                if (at->parent == NULL) {
                    printf("line %2d: '%s' climbs above the root\n", cmd_line, cmd);
                    return -1;
                }
                at = at->parent;
            }
            if (at->right != NULL) {
                printf("line %2d: %d already has a right child\n", cmd_line,
                       at->data);
                return -1;
            }
            at->right = new_node(data, at);
            printf("line %2d: %-5s-> up %d to %d, %d is its right child\n",
                   cmd_line, cmd, k, at->data, data);
            cur = at->right;
        }
        count++;
    }
    return count;
}
```

Kode 12.5 (Lanjutan) *tree_from_file.c: menumbuhkan pohon biner dari berkas perintah, mencetaknya dalam tiga urutan dan menyamping, lalu membebaskannya.*

```c
static void preorder(const TNode *t)
{
    if (t == NULL)
        return;
    printf(" %d", t->data);
    preorder(t->left);
    preorder(t->right);
}

static void inorder(const TNode *t)
{
    if (t == NULL)
        return;
    inorder(t->left);
    printf(" %d", t->data);
    inorder(t->right);
}

static void postorder(const TNode *t)
{
    if (t == NULL)
        return;
    postorder(t->left);
    postorder(t->right);
    printf(" %d", t->data);
}

static int height(const TNode *t)
{
    if (t == NULL)
        return 0;
    int a = height(t->left), b = height(t->right);
    return 1 + (a > b ? a : b);
}

/* The tree lying on its side: right subtree above, left subtree below. */
static void sideways(const TNode *t, int depth)
{
    if (t == NULL)
        return;
    sideways(t->right, depth + 1);
    printf("%*s%d\n", 6 * depth, "", t->data);
    sideways(t->left, depth + 1);
}
```

Kode 12.5 (Lanjutan) *tree_from_file.c: menumbuhkan pohon biner dari berkas perintah, mencetaknya dalam tiga urutan dan menyamping, lalu membebaskannya.*

```c
int main(int argc, char *argv[])
{
    if (argc != 2) {
        fprintf(stderr, "usage: %s <file>\n", argv[0]);
        return EXIT_FAILURE;
    }
    FILE *fp = fopen(argv[1], "r");
    if (fp == NULL) {
        perror(argv[1]);
        return EXIT_FAILURE;
    }
    TNode *root = NULL;
    int n = build(fp, &root);
    fclose(fp);
    if (n <= 0) {                   /* an error was reported, or no nodes */
        if (n == 0)
            printf("%s holds no nodes\n", argv[1]);
        free_tree(root);
        return EXIT_FAILURE;
    }
    printf("nodes: %d, height: %d\n", n, height(root));
    printf("preorder :");
    preorder(root);
    printf("\ninorder  :");
    inorder(root);
    printf("\npostorder:");
    postorder(root);
    printf("\nsideways (root on the left, right children above):\n");
    sideways(root, 0);
    free_tree(root);
    return 0;
}
```

12.4.5 Penyelesaian dalam Python

Kode 12.6 memakai ekspresi reguler untuk perintahnya (`u*r` berarti “sejumlah `u`, lalu `r`”, dari modul `re` Python [3]) dan satu lagi untuk baris data. Penelusuran-penelusuran mengumpulkan nilai ke dalam *list*, yang lalu digabung untuk dicetak.

Kode 12.6 `tree_from_file.py`: pembangun pohon yang sama dalam Python.

```
# tree_from_file.py - Problem 12.2: grow a binary tree from a file of command
# lines and data lines, then print it.
#   0      create the root (first command only)
#   l      the new node becomes the LEFT child of the current node
#   u...ur climb one parent per u from the current node, then the new node
#          becomes the RIGHT child of the node reached (plain r: no climb)
# Usage: python3 tree_from_file.py <file>
import re
import sys


class TNode:
    def __init__(self, data, parent):
        self.data = data
        self.left = self.right = None
        self.parent = parent


def lines_of(path):
    """Yield (line number, text) for every non-empty line."""
    with open(path, encoding="utf-8") as fp:
        for lineno, line in enumerate(fp, start=1):
            line = line.rstrip("\r\n")
            if line:
                yield lineno, line


def climbs(cmd):
    """How many u's before the final r? -1 if cmd is not u...ur."""
    return len(cmd) - 1 if re.fullmatch(r"u*r", cmd) else -1


def build(path):
    """Return (root, number of nodes), or (root, -1) after an error."""
    root = cur = None
    count = 0
    it = lines_of(path)
    for cmd_line, cmd in it:
        nxt = next(it, None)
        if nxt is None or not re.fullmatch(r"\s*[+-]?\d+\s*", nxt[1]):
            print("line %2d: a whole number must follow command '%s'"
                  % (cmd_line + 1, cmd))
            return root, -1
        data = int(nxt[1])
        if cmd == "0":
            if root is not None:
```

Kode 12.6 (Lanjutan) *tree_from_file.py: pembangun pohon yang sama dalam Python.*

```
                print("line %2d: the tree already has a root" % cmd_line)
                return root, -1
            root = cur = TNode(data, None)
            print("line %2d: %-5s-> %d is the root" % (cmd_line, cmd, data))
        elif root is None:
            print("line %2d: the first command must be 0" % cmd_line)
            return root, -1
        elif cmd == "l":
            if cur.left is not None:
                print("line %2d: %d already has a left child"
                      % (cmd_line, cur.data))
                return root, -1
            cur.left = TNode(data, cur)
            print("line %2d: %-5s-> %d is the left child of %d"
                  % (cmd_line, cmd, data, cur.data))
            cur = cur.left
        else:
            k = climbs(cmd)
            if k < 0:
                print("line %2d: unknown command '%s'" % (cmd_line, cmd))
                return root, -1
            at = cur
            for _ in range(k):
                if at.parent is None:
                    print("line %2d: '%s' climbs above the root" % (cmd_line, cmd))
                    return root, -1
                at = at.parent
            if at.right is not None:
                print("line %2d: %d already has a right child"
                      % (cmd_line, at.data))
                return root, -1
            at.right = TNode(data, at)
            print("line %2d: %-5s-> up %d to %d, %d is its right child"
                  % (cmd_line, cmd, k, at.data, data))
            cur = at.right
        count += 1
    return root, count


def preorder(t, out):
    if t is not None:
        out.append(t.data)
        preorder(t.left, out)
        preorder(t.right, out)
    return out
```

Kode 12.6 (Lanjutan) *tree_from_file.py: pembangun pohon yang sama dalam Python.*

```
def inorder(t, out):
    if t is not None:
        inorder(t.left, out)
        out.append(t.data)
        inorder(t.right, out)
    return out


def postorder(t, out):
    if t is not None:
        postorder(t.left, out)
        postorder(t.right, out)
        out.append(t.data)
    return out


def height(t):
    return 0 if t is None else 1 + max(height(t.left), height(t.right))


def sideways(t, depth=0):
    if t is not None:
        sideways(t.right, depth + 1)
        print("%*s%d" % (6 * depth, "", t.data))
        sideways(t.left, depth + 1)


def main():
    if len(sys.argv) != 2:
        print("usage: tree_from_file.py <file>", file=sys.stderr)
        sys.exit(1)
    try:
        root, n = build(sys.argv[1])
    except OSError as e:
        print("%s: %s" % (sys.argv[1], e.strerror), file=sys.stderr)
        sys.exit(1)
    if n <= 0:
        if n == 0:
            print("%s holds no nodes" % sys.argv[1])
        sys.exit(1)
    print("nodes: %d, height: %d" % (n, height(root)))
    print("preorder :" + "".join(" %d" % v for v in preorder(root, [])))
    print("inorder  :" + "".join(" %d" % v for v in inorder(root, [])))
    print("postorder:" + "".join(" %d" % v for v in postorder(root, [])))
    print("sideways (root on the left, right children above):")
```

Kode 12.6 (Lanjutan) *tree_from_file.py: pembangun pohon yang sama dalam Python.*

```
    sideways(root)


if __name__ == "__main__":
    main()
```

12.4.6 Menjalankan dan mengujinya

`./tree_from_file ../data/tree5.txt` mencetak:

```
line  1: 0    -> 50 is the root
line  3: l    -> 30 is the left child of 50
line  5: l    -> 20 is the left child of 30
line  7: ur   -> up 1 to 30, 40 is its right child
line  9: uur  -> up 2 to 50, 70 is its right child
nodes: 5, height: 3
preorder : 50 30 20 40 70
inorder  : 20 30 40 50 70
postorder: 20 40 30 70 50
sideways (root on the left, right children above):
      70
50
            40
      30
            20
```

Setiap baris jejak cocok dengan satu panel **Gambar 12.1**, ketiga urutan cocok dengan harapan Subbab 12.4.2, dan gambar menyampingnya, bila diputar seperempat putaran ke kanan, adalah panel terakhir gambar itu.

Uji yang lebih besar, `data/tree15.txt`, membangun pohon seimbang sempurna berisi bilangan 1 sampai 15, dengan pendakian satu, dua, dan tiga tingkat:

```
line  1: 0    -> 8 is the root
line  3: l    -> 4 is the left child of 8
line  5: l    -> 2 is the left child of 4
line  7: l    -> 1 is the left child of 2
line  9: ur   -> up 1 to 2, 3 is its right child
line 11: uur  -> up 2 to 4, 6 is its right child
line 13: l    -> 5 is the left child of 6
line 15: ur   -> up 1 to 6, 7 is its right child
line 17: uuur -> up 3 to 8, 12 is its right child
line 19: l    -> 10 is the left child of 12
line 21: l    -> 9 is the left child of 10
line 23: ur   -> up 1 to 10, 11 is its right child
line 25: uur  -> up 2 to 12, 14 is its right child
line 27: l    -> 13 is the left child of 14
line 29: ur   -> up 1 to 14, 15 is its right child
nodes: 15, height: 4
preorder : 8 4 2 1 3 6 5 7 12 10 9 11 14 13 15
inorder  : 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
postorder: 1 3 2 5 7 6 4 9 11 10 13 15 14 12 8
sideways (root on the left, right children above):
                  15
            14
                  13
      12
                  11
            10
                  9
8
                  7
            6
                  5
      4
                  3
            2
                  1
```

Inorder-nya 1 sampai 15 berurutan, karena berkas ini kebetulan membangun *binary search tree* (pohon pencarian biner), yang setiap subpohon kirinya memuat bilangan lebih kecil dan setiap subpohon kanannya bilangan lebih besar; tingginya 4, yang terkecil yang mungkin untuk 15 simpul. Tiga berkas rusak memeriksa pesan-pesan galat; masing-masing mencetak simpul yang sudah dibangun lalu berhenti:

```
line  1: 0    -> 50 is the root
line  3: l    -> 30 is the left child of 50
line  5: 'uur' climbs above the root
line  1: 0    -> 50 is the root
line  3: l    -> 30 is the left child of 50
line  6: a whole number must follow command 'l'
line  1: 0    -> 50 is the root
line  3: l    -> 30 is the left child of 50
line  5: ur   -> up 1 to 50, 40 is its right child
line  7: 50 already has a right child
```

Pada yang pertama, `uur` dari 30 naik ke 50 lalu tidak menemukan orang tua. Pada yang kedua, `forty` bukan bilangan. Pada yang ketiga, `ur` kedua naik dari 40 ke 50, yang sudah punya anak kanan. Dalam ketiga kasus, program membebaskan pohon parsialnya sebelum berhenti, yang ditegaskan oleh *sanitizer*.

12.4.7 Penyelesaian keliru yang menggoda

Kesalahan paling umum adalah selisih satu (*off-by-one*) dalam pendakian: memulainya dari *orang tua* simpul saat ini, "karena `u` berarti naik, dan kita sudah satu tingkat di bawah". Ini menggoda karena membuat `ur` terdengar seperti "anak kanan orang tua saya", yang terasa wajar. Ulangi contohnya dengan aturan itu. Baris 1 sampai 6 tidak terpengaruh. Baris 7, `ur`: dari 20, aturan itu lebih dulu pergi ke orang tua 30 lalu naik satu tingkat lagi ke 50; 40 menjadi anak kanan *50*. Baris 9, `uur`: dari 40, ia pergi ke orang tua 50 lalu mencoba naik dua tingkat lagi, melewati akar, dan program melaporkan galat pada berkas yang sepenuhnya benar. Dengan berkas yang sedikit berbeda, aturan yang salah tidak menghasilkan galat sama sekali, hanya pohon yang bentuknya keliru, dan itu lebih buruk. Pertahanannya adalah yang dari Subbab 12.2: telusuri contoh dalam soal secara manual *sebelum* menulis kode, dan buat program mencetak baris jejak untuk setiap perintah, sehingga baris pertama tempat program dan kertas berbeda pendapat menunjuk kesalahannya.

Kesalahan kedua adalah melupakan pointer orang tua lalu mendaki dengan mencari di pohon, mulai dari akar, simpul yang anaknya adalah `cur`. Itu berfungsi, tetapi memakan penelusuran seluruh pohon untuk setiap pendakian, dan jauh lebih banyak kode yang harus dibuat benar.

12.4.8 Periksa diri

Tulis berkas perintah untuk pohon yang preorder-nya 8 4 2 6 12 10 14 (pohon seimbang tujuh simpul), jalankan, dan bandingkan inorder-nya dengan 2 4 6 8 10 12 14. (Satu jawaban: `0 8`, `l 4`, `l 2`, `ur 6`, `uur 12`, `l 10`, `ur 14`, setiap perintah diikuti bilangannya di baris berikutnya.)

12.5 Soal 12.3: Lintasan Terpendek Kuda Catur

12.5.1 Soalnya

> **SOAL 12.3.** Pada papan catur 8 × 8, seekor kuda berdiri di petak awal dan harus mencapai petak tujuan. Carilah banyak langkah paling sedikit yang diperlukannya. Pakailah antrean yang diwujudkan sebagai *linked list*, dengan operasi *enqueue* (tambah di belakang) dan *dequeue* (ambil dari depan). Contohnya, dari baris 7, kolom 0 ke baris 0, kolom 7 kuda memerlukan 6 langkah.

Dengan kata lain: kuda bergerak membentuk huruf "L": dua petak ke satu arah dan satu petak ke arah tegak lurus, sehingga dari petak mana pun ada paling banyak delapan langkah yang mungkin, lebih sedikit di dekat tepi. Baris dan kolom dinomori 0 sampai 7. Kita harus mencari

panjang deretan langkah terpendek dari awal ke tujuan. Kita memutuskan: program membaca pertanyaan dari masukan standar, satu per baris, `r1 c1 r2 c2`; ia mencetak banyak langkah dan juga satu lintasan terpendek, karena lintasan dapat diperiksa secara manual; pertanyaan dengan petak di luar papan dijawab dengan pesan; dan kata `board` sesudah pertanyaan mencetak jarak setiap petak dari petak awal.

12.5.2 Memahaminya secara manual

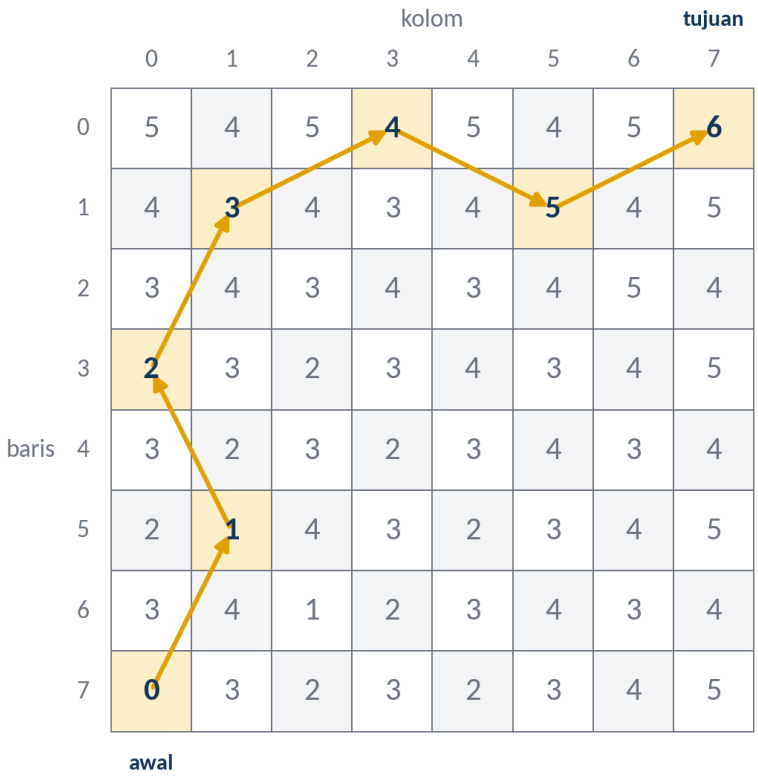
Bayangkan menjatuhkan batu ke kolam. Lingkaran gelombang pertama mencapai setiap titik berjarak 1, lingkaran kedua setiap titik berjarak 2, dan seterusnya. **Breadth-first search** (BFS) menjelajahi papan dengan cara yang sama. Mulailah dengan petak awal berjarak 0. Setiap petak yang satu langkah kuda jauhnya mendapat jarak 1. Setiap petak yang satu langkah dari petak-petak itu, dan belum tercapai, mendapat jarak 2. Teruskan lingkaran demi lingkaran sampai tujuan tercapai. Karena setiap lingkaran diselesaikan sebelum lingkaran berikutnya dimulai, pertama kali sebuah petak tercapai, ia tercapai lewat lintasan terpendek: lintasan yang lebih pendek pasti sudah mencapainya di lingkaran sebelumnya [4].

Alat yang menjaga urutan lingkaran adalah **antrean** (*queue*): petak diambil dari depan dan tetangga-tetangga barunya ditambahkan di belakang, sehingga semua petak berjarak d diambil sebelum petak mana pun yang berjarak $d + 1$. Ketika petak u yang diambil dari antrean mencapai petak baru v,

$$\text{dist}(v) = \text{dist}(u) + 1 \tag{12.2}$$

dan u dicatat sebagai petak *asal* v dicapai, sehingga di akhir lintasannya dapat diikuti mundur dari tujuan ke awal.

Gambar 12.2 menunjukkan hasilnya untuk contoh tersebut: jarak setiap petak dari (7,0). Tujuan (0,7) berjarak 6, seperti kata soal, dan petak-petak kuning adalah satu lintasan terpendek, setiap langkah satu gerakan kuda dan satu lingkaran lebih jauh.



langkah yang diperlukan dari (7,0); satu lintasan terpendek berwarna kuning

Gambar 12.2 Banyak langkah kuda yang diperlukan dari petak awal (7,0) ke setiap petak; tujuan (0,7) memerlukan 6, dan petak-petak kuning menunjukkan satu lintasan terpendek.

Lihatlah bilangan-bilangan di dekat petak awal: (5,1) dan (6,2) berjarak 1, sebagai dua langkah dari petak sudut; dan (6,1), tepat di sebelah petak awal secara diagonal, memerlukan 4 langkah. Jarak pada papan kuda hampir tidak ada hubungannya dengan jarak menurut penggaris, dan karena itulah soal ini memerlukan algoritma, bukan rumus.

Satu detail lagi menentukan apakah BFS cepat atau lambat: petak harus ditandai *sudah dikunjungi* ketika dimasukkan **ke** antrean, bukan ketika keluar. Kalau tidak, petak yang dapat dicapai dari beberapa petak di lingkaran yang sama dimasukkan ke antrean berkali-kali, dan setiap salinannya kelak menambahkan tetangga-tetangganya lagi.

12.5.3 Merancangnya di atas kertas

Antreannya adalah *linked list* dengan `head` (depan) dan `tail` (belakang), persis `List` dari Bab 10: *enqueue* adalah `append`, *dequeue* adalah "hapus simpul pertama". **Pseudokode 12.3** adalah penelusurannya.

Pseudokode 12.3 *Breadth-first search* untuk langkah kuda paling sedikit, dengan antrean berupa *linked list*.

```
KNIGHT-BFS(s, t)
1  for every square v: dist[v] = -1, visited[v] = FALSE
2  dist[s] = 0; visited[s] = TRUE; ENQUEUE(Q, s)
3  while Q is not empty
4      u = DEQUEUE(Q)
5      if u = t: stop                  // the first arrival is the shortest
6      for each of the 8 knight moves from u to a square v on the board
7          if not visited[v]
8              visited[v] = TRUE       // mark when ENQUEUED
9              dist[v] = dist[u] + 1; from[v] = u
10             ENQUEUE(Q, v)
11 return dist[t]
```

Biayanya. Setiap petak dimasukkan ke antrean paling banyak sekali berkat baris 8, sehingga paling banyak 64 petak masuk antrean dan masing-masing mencoba 8 langkah: paling banyak $64 \cdot 8 = 512$ pemeriksaan per pertanyaan. Pada papan $N \times N$ kerjanya tumbuh seperti N^2, tetap kecil untuk papan apa pun yang mungkin digambar manusia.

12.5.4 Penyelesaian dalam C

Kode 12.7 memiliki bagian-bagian soal: tipe `Cell`, tabel langkah `DR` dan `DC` (delapan langkah sebagai perubahan baris dan kolom), `is_valid`, dan fungsi-fungsi antrean `enqueue`, `dequeue`, serta `free_queue`. Penelusuran `bfs` mengikuti **Pseudokode 12.3**; parameter tambahan `late` beralih ke varian menggoda dari Subbab 12.5.7 sebagai pembanding. `print_path` mengikuti `from` mundur dari tujuan, menyimpan petak-petaknya dalam larik, dan mencetaknya terbalik. Pertanyaan dengan kata `board` menjalankan penelusuran sekali lagi dengan tujuan yang mustahil, sehingga seluruh papan terisi.

Kode 12.7 `knight.c`: *breadth-first search* dengan antrean *linked list* untuk langkah kuda paling sedikit.

```c
/* knight.c - Problem 12.3: the fewest knight moves from one square of an
 * 8 x 8 board to another, found by breadth-first search (BFS) with a queue
 * built as a linked list.
 * Input: one query per line, "r1 c1 r2 c2" (rows and columns 0..7);
 * add the word "board" to print every square's distance from the start,
 * or "late" to mark squares visited when they leave the queue instead. */
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define N 8

typedef struct {
    int r, c;
} Cell;

typedef struct QNode {
    Cell cell;
    struct QNode *next;
} QNode;

typedef struct {
    QNode *head, *tail;      /* take from the head, add at the tail */
} Queue;

static const int DR[8] = {-2, -2, -1, -1, 1, 1, 2, 2};
static const int DC[8] = {-1, 1, -2, 2, -2, 2, -1, 1};

static int is_valid(int r, int c)
{
    return r >= 0 && r < N && c >= 0 && c < N;
}

static void enqueue(Queue *q, Cell cell)
{
    QNode *n = malloc(sizeof *n);
    if (n == NULL) {
        fprintf(stderr, "out of memory\n");
        exit(EXIT_FAILURE);
    }
    n->cell = cell;
    n->next = NULL;
    if (q->tail != NULL)
        q->tail->next = n;   /* link behind the current last node */
    else
        q->head = n;         /* the queue was empty */
```

Kode 12.7 (Lanjutan) *knight.c: breadth-first search dengan antrean linked list untuk langkah kuda paling sedikit.*

```
    q->tail = n;
}

static Cell dequeue(Queue *q)    /* call only when q->head != NULL */
{
    QNode *n = q->head;
    Cell cell = n->cell;
    q->head = n->next;
    if (q->head == NULL)
        q->tail = NULL;      /* the queue is empty again */
    free(n);
    return cell;
}

static void free_queue(Queue *q)
{
    while (q->head != NULL)
        dequeue(q);
}

/* BFS from s. Fills dist (-1 = not reached) and from (the square each
 * square was first reached from). Returns how many squares were enqueued. */
static int bfs(Cell s, Cell t, int dist[N][N], Cell from[N][N], int late)
{
    int visited[N][N] = {{0}}, enqueued = 0;
    Queue q = {NULL, NULL};
    for (int r = 0; r < N; r++)
        for (int c = 0; c < N; c++)
            dist[r][c] = -1;
    dist[s.r][s.c] = 0;
    if (!late)
        visited[s.r][s.c] = 1;
    enqueue(&q, s);
    enqueued++;
    while (q.head != NULL) {
        Cell cur = dequeue(&q);
        if (late) {                      /* the tempting variant */
            if (visited[cur.r][cur.c])
                continue;
            visited[cur.r][cur.c] = 1;
        }
        if (cur.r == t.r && cur.c == t.c)
            break;                       /* the first arrival is the shortest */
        for (int k = 0; k < 8; k++) {
            int r = cur.r + DR[k], c = cur.c + DC[k];
            if (!is_valid(r, c) || visited[r][c])
```

Kode 12.7 (Lanjutan) *knight.c: breadth-first search dengan antrean linked list untuk langkah kuda paling sedikit.*

```
                continue;
            if (!late)
                visited[r][c] = 1;       /* mark it NOW, when it is enqueued */
            if (dist[r][c] < 0) {
                dist[r][c] = dist[cur.r][cur.c] + 1;
                from[r][c] = cur;
            }
            enqueue(&q, (Cell){r, c});
            enqueued++;
        }
    }
    free_queue(&q);
    return enqueued;
}

static void print_path(Cell s, Cell t, Cell from[N][N])
{
    Cell path[N * N];
    int n = 0;
    for (Cell p = t; !(p.r == s.r && p.c == s.c); p = from[p.r][p.c])
        path[n++] = p;
    path[n++] = s;
    printf("  path:");
    for (int i = n - 1; i >= 0; i--)
        printf(" (%d,%d)", path[i].r, path[i].c);
    printf("\n");
}

int main(void)
{
    char line[128], word[16] = "";
    while (fgets(line, sizeof line, stdin) != NULL) {
        Cell s, t;
        word[0] = '\0';
        int got = sscanf(line, "%d %d %d %d %15s", &s.r, &s.c, &t.r, &t.c, word);
        if (got < 4)
            continue;                    /* not a query: skip the line */
        if (!is_valid(s.r, s.c) || !is_valid(t.r, t.c)) {
            printf("(%d,%d) -> (%d,%d): off the board\n", s.r, s.c, t.r, t.c);
            continue;
        }
        int dist[N][N], late = strcmp(word, "late") == 0;
        Cell from[N][N];
        int enq = bfs(s, t, dist, from, late);
        printf("(%d,%d) -> (%d,%d): %d move(s), %d square(s) enqueued%s\n",
               s.r, s.c, t.r, t.c, dist[t.r][t.c], enq,
```

Kode 12.7 (Lanjutan) *knight.c: breadth-first search dengan antrean linked list untuk langkah kuda paling sedikit.*

```
                late ? " (marked late)" : "");
        print_path(s, t, from);
        if (strcmp(word, "board") == 0) {
            bfs(s, (Cell){-1, -1}, dist, from, 0);   /* no target: fill all */
            printf("  distance of every square from (%d,%d):\n", s.r, s.c);
            for (int r = 0; r < N; r++) {
                printf("   ");
                for (int c = 0; c < N; c++)
                    printf(" %d", dist[r][c]);
                printf("\n");
            }
        }
    }
    return 0;
}
```

Dua cabang `enqueue` adalah bagian yang paling sering salah. Bila antrean sudah memiliki simpul terakhir, simpul baru disambungkan di belakangnya; bila antrean kosong, simpul baru juga menjadi simpul pertama. `dequeue` adalah cerminannya: ketika mengambil simpul terakhir, `tail` juga harus menjadi `NULL`, kalau tidak `enqueue` berikutnya akan menyambungkan simpul baru di belakang memori yang sudah dibebaskan.

12.5.5 Penyelesaian dalam Python

Kode 12.8 membangun antrean sebagai kelas kecil dengan simpul-simpul berantainya sendiri, agar mencerminkan program C. (Dalam Python sehari-hari orang memakai `collections.deque`, yang melakukan pekerjaan yang sama [3].)

Kode 12.8 `knight.py`: penelusuran yang sama dalam Python.

```
# knight.py - Problem 12.3: the fewest knight moves between two squares of an
# 8 x 8 board, by breadth-first search with a queue built as a linked list.
# Input: one query per line, "r1 c1 r2 c2" (0..7), optionally followed by
# "board" (print all distances) or "late" (mark visited when dequeued).
import sys

N = 8
DR = [-2, -2, -1, -1, 1, 1, 2, 2]
DC = [-1, 1, -2, 2, -2, 2, -1, 1]


class QNode:
    def __init__(self, cell):
        self.cell = cell
        self.next = None


class Queue:
    def __init__(self):
        self.head = self.tail = None       # take from head, add at tail

    def enqueue(self, cell):
        n = QNode(cell)
        if self.tail is not None:
            self.tail.next = n
        else:
            self.head = n
        self.tail = n

    def dequeue(self):
        n = self.head
        self.head = n.next
        if self.head is None:
            self.tail = None
        return n.cell


def is_valid(r, c):
    return 0 <= r < N and 0 <= c < N
```

Kode 12.8 (Lanjutan) *knight.py: penelusuran yang sama dalam Python.*

```
def bfs(s, t, late=False):
    """Return (dist, from, number of squares enqueued)."""
    dist = [[-1] * N for _ in range(N)]
    frm = [[None] * N for _ in range(N)]
    visited = [[False] * N for _ in range(N)]
    dist[s[0]][s[1]] = 0
    if not late:
        visited[s[0]][s[1]] = True
    q = Queue()
    q.enqueue(s)
    enqueued = 1
    while q.head is not None:
        cur = q.dequeue()
        if late:
            if visited[cur[0]][cur[1]]:
                continue
            visited[cur[0]][cur[1]] = True
        if cur == t:
            break                           # the first arrival is the shortest
        for k in range(8):
            r, c = cur[0] + DR[k], cur[1] + DC[k]
            if not is_valid(r, c) or visited[r][c]:
                continue
            if not late:
                visited[r][c] = True        # mark it NOW, when it is enqueued
            if dist[r][c] < 0:
                dist[r][c] = dist[cur[0]][cur[1]] + 1
                frm[r][c] = cur
            q.enqueue((r, c))
            enqueued += 1
    return dist, frm, enqueued


def main():
    for line in sys.stdin:
        w = line.split()
        try:
            r1, c1, r2, c2 = (int(x) for x in w[:4])
        except ValueError:
            continue                         # not a query: skip the line
        word = w[4][:15] if len(w) > 4 else ""
        s, t = (r1, c1), (r2, c2)
        if not (is_valid(*s) and is_valid(*t)):
            print("(%d,%d) -> (%d,%d): off the board" % (r1, c1, r2, c2))
            continue
        late = word == "late"
```

Kode 12.8 (Lanjutan) *knight.py: penelusuran yang sama dalam Python.*

```
        dist, frm, enq = bfs(s, t, late)
        print("(%d,%d) -> (%d,%d): %d move(s), %d square(s) enqueued%s"
              % (r1, c1, r2, c2, dist[r2][c2], enq,
                 " (marked late)" if late else ""))
        path, p = [], t
        while p != s:
            path.append(p)
            p = frm[p[0]][p[1]]
        path.append(s)
        print("  path:" + "".join(" (%d,%d)" % x for x in reversed(path)))
        if word == "board":
            dist, _, _ = bfs(s, (-1, -1))
            print("  distance of every square from (%d,%d):" % s)
            for r in range(N):
                print("    " + "".join(" %d" % d for d in dist[r]))


if __name__ == "__main__":
    main()
```

12.5.6 Menjalankan dan mengujinya

Berkas uji `data/knight_queries.txt` memuat contoh dengan `board`, kasus tepi awal = tujuan, satu langkah, lompatan pendek yang memerlukan tiga langkah, pertanyaan sudut ke sudut, contoh sekali lagi dengan varian menggoda, dan petak di luar papan:

```
7 0 0 7 board
0 0 0 0
0 0 1 2
3 3 4 3
0 0 7 7
7 0 0 7 late
9 0 1 1
```

`./knight < ../data/knight_queries.txt` mencetak:

```
(7,0) -> (0,7): 6 move(s), 64 square(s) enqueued
  path: (7,0) (5,1) (3,0) (1,1) (0,3) (1,5) (0,7)
  distance of every square from (7,0):
    5 4 5 4 5 4 5 6
    4 3 4 3 4 5 4 5
    3 4 3 4 3 4 5 4
    2 3 2 3 4 3 4 5
    3 2 3 2 3 4 3 4
    2 1 4 3 2 3 4 5
    3 4 1 2 3 4 3 4
    0 3 2 3 2 3 4 5
(0,0) -> (0,0): 0 move(s), 1 square(s) enqueued
  path: (0,0)
(0,0) -> (1,2): 1 move(s), 3 square(s) enqueued
  path: (0,0) (1,2)
(3,3) -> (4,3): 3 move(s), 63 square(s) enqueued
  path: (3,3) (1,2) (2,4) (4,3)
(0,0) -> (7,7): 6 move(s), 64 square(s) enqueued
  path: (0,0) (1,2) (0,4) (1,6) (3,5) (5,6) (7,7)
(7,0) -> (0,7): 6 move(s), 169 square(s) enqueued (marked late)
  path: (7,0) (5,1) (3,0) (1,1) (0,3) (1,5) (0,7)
(9,0) -> (1,1): off the board
```

Contohnya memerlukan 6 langkah, dan lintasannya dapat diperiksa dengan jari pada **Gambar 12.2**: setiap langkah mengubah satu koordinat sebesar 2 dan yang lain sebesar 1. Papan yang tercetak adalah gambar itu dalam bentuk bilangan. Awal = tujuan memerlukan 0 langkah dan hanya memasukkan petak awal ke antrean; (0,0) ke (1,2) satu langkah; (3,3) ke (4,3), satu

petak lurus ke bawah, memerlukan 3 langkah; sudut ke sudut memerlukan 6. Pertanyaan di luar papan ditolak. Dan lihatlah baris bertanda `(marked late)`: jawaban yang sama, 6 langkah, tetapi 169 petak masuk antrean, bukan 64, karena petak dimasukkan lagi dan lagi sebelum salinan pertamanya keluar. Pada papan catur itu hanya memakan mikrodetik; pada peta dengan jutaan tempat, itulah beda antara jawaban cepat dan tidak ada jawaban.

12.5.7 Penyelesaian keliru yang menggoda

Rekursi dari Bab 5 menyarankan program yang lebih pendek: dari petak saat ini, coba langkah pertama; dari sana, coba langkah pertama lagi; masuk sedalam mungkin, dan mundur hanya ketika buntu. Inilah **depth-first search** (DFS, penelusuran mendalam), dan **Kode 12.9** mewujudkannya dengan larik `visited` agar tidak berputar-putar. Ia mengembalikan panjang lintasan pertama yang ditemukannya; **Kode 12.10** adalah versi Python-nya.

Kode 12.9 `knight_dfs.c`: *depth-first search* melaporkan lintasan pertama yang ditemukannya, bukan yang terpendek.

```c
/* knight_dfs.c - a tempting wrong solution to Problem 12.3: depth-first
 * search. It dives along the first move that works and reports the first
 * path it finds, which is a path, but rarely the shortest one. */
#include <stdio.h>

#define N 8

static const int DR[8] = {-2, -2, -1, -1, 1, 1, 2, 2};
static const int DC[8] = {-1, 1, -2, 2, -2, 2, -1, 1};
static int visited[N][N];

/* Returns the length of the first path found from (r,c) to (tr,tc), or -1. */
static int dfs(int r, int c, int tr, int tc, int depth)
{
    if (r == tr && c == tc)
        return depth;
    visited[r][c] = 1;
    for (int k = 0; k < 8; k++) {
        int nr = r + DR[k], nc = c + DC[k];
        if (nr >= 0 && nr < N && nc >= 0 && nc < N && !visited[nr][nc]) {
            int found = dfs(nr, nc, tr, tc, depth + 1);
            if (found >= 0)
                return found;           /* stop at the FIRST path found */
        }
    }
    return -1;
}

int main(void)
{
    const int q[3][4] = {{7, 0, 0, 7}, {0, 0, 7, 7}, {3, 3, 4, 3}};
    for (int i = 0; i < 3; i++) {
        for (int r = 0; r < N; r++)
            for (int c = 0; c < N; c++)
                visited[r][c] = 0;
        int len = dfs(q[i][0], q[i][1], q[i][2], q[i][3], 0);
        printf("(%d,%d) -> (%d,%d): DFS found a path of %d move(s)\n",
               q[i][0], q[i][1], q[i][2], q[i][3], len);
    }
    return 0;
}
```

Kode 12.10 `knight_dfs.py`: kesalahan yang sama dalam Python.

```python
# knight_dfs.py - a tempting wrong solution to Problem 12.3: depth-first
# search reports the first path it finds, which is rarely the shortest.
import sys

N = 8
DR = [-2, -2, -1, -1, 1, 1, 2, 2]
DC = [-1, 1, -2, 2, -2, 2, -1, 1]


def dfs(r, c, tr, tc, depth, visited):
    if (r, c) == (tr, tc):
        return depth
    visited[r][c] = True
    for k in range(8):
        nr, nc = r + DR[k], c + DC[k]
        if 0 <= nr < N and 0 <= nc < N and not visited[nr][nc]:
            found = dfs(nr, nc, tr, tc, depth + 1, visited)
            if found >= 0:
                return found            # stop at the FIRST path found
    return -1


def main():
    sys.setrecursionlimit(10000)
    for q in ([7, 0, 0, 7], [0, 0, 7, 7], [3, 3, 4, 3]):
        visited = [[False] * N for _ in range(N)]
        print("(%d,%d) -> (%d,%d): DFS found a path of %d move(s)"
              % (q[0], q[1], q[2], q[3], dfs(q[0], q[1], q[2], q[3], 0, visited)))


if __name__ == "__main__":
    main()
```

```
(7,0) -> (0,7): DFS found a path of 6 move(s)
(0,0) -> (7,7): DFS found a path of 52 move(s)
(3,3) -> (4,3): DFS found a path of 25 move(s)
```

Inilah keluaran paling mendidik di bab ini. Pada contoh dalam soal, DFS menjawab 6, bilangan yang benar, dan pemrogram yang hanya menguji contoh itu merasa puas. Pada pertanyaan sudut ke sudut ia menjawab 52, dan pada lompatan tiga langkah 25. DFS menemukan *sebuah* lintasan, dan dengan larik `visited` ia menemukannya dengan cepat, tetapi ia mengikuti pilihan langkah pertamanya sejauh mungkin, sehingga lintasan yang ditemukannya bergantung pada urutan tabel langkah, bukan pada jarak. BFS menemukan lintasan terpendek karena cara ia menjelajah, lingkaran demi lingkaran, bukan karena keberuntungan. Jawaban benar pada contoh yang diberikan tidak membuktikan apa-apa tentang sebuah algoritma; hanya argumen (mengapa kedatangan pertama adalah yang terpendek) dan beragam uji yang dapat membuktikannya.

12.5.8 Periksa diri

Ubah program untuk *raja*, yang bergerak satu petak ke salah satu dari delapan arah. Baris mana yang berubah? (Hanya tabel `DR` dan `DC`.) Ramalkan jawabannya untuk (7,0) ke (0,7) sebelum menjalankan: raja memerlukan langkah sebanyak yang lebih besar di antara selisih baris dan selisih kolom, $\max(7,7) = 7$.

12.6 Soal-soal Berdampingan

Tabel 12.1 menempatkan ketiga soal berdampingan. Polanya patut diingat: pada masing-masing, struktur data dipilih agar satu operasi yang penting menjadi murah, dan masing-masing

memerlukan satu pointer atau tanda tambahan (`last`, `parent`, `visited`) yang tidak disebut dalam soal.

Tabel 12.1 Ketiga soal dibandingkan: struktur, operasi kunci, gagasan yang membuatnya berhasil, biaya, dan kesalahan yang harus dihindari.

	Soal 12.1	Soal 12.2	Soal 12.3
Struktur	*singly linked list*	pohon biner dengan pointer orang tua	antrean berupa *linked list*, larik papan
Operasi kunci	menukar data tetangga	naik `k` tingkat, menyisipkan anak	*enqueue* di belakang, *dequeue* di depan
Gagasan tambahan	`last` menandai ekor terurut	pointer `parent` di setiap simpul	tandai `visited` saat dimasukkan
Biaya	$O(n^2)$ urut, $O(n)$ sisip	$O(n)$ ditambah pendakian	paling banyak 64 petak, 512 pemeriksaan
Kesalahan menggoda	hanya satu lintasan	naik dari orang tua	*depth-first search*
Uji yang menangkapnya	senarai terbalik, senarai kosong	contoh dari soal itu sendiri	pertanyaan selain contoh

12.7 Memakainya secara Nyata

Ketiga struktur ini muncul di sekitar usaha kecil begitu Anda mengenali bentuknya. *Senarai terurut dengan penyisipan* adalah daftar tunggu klinik yang diurutkan menurut jam janji temu, dengan pasien tanpa janji ditambahkan di akhir. *Pohon* adalah bagan organisasi koperasi, struktur folder di laptop, atau bagan akun usaha kecil, tempat "Beban" memiliki anak "Sewa", "Gaji", dan "Perlengkapan"; berkas yang mencantumkan pohon seperti itu dengan perintah navigasi, seperti Soal 12.2, adalah cara ringkas menyimpannya, dan pointer orang tua memungkinkan program menjawab "akun ini termasuk kelompok mana?". *Breadth-first search* menjawab setiap pertanyaan "langkah paling sedikit": transit paling sedikit di antara dua halte dalam jaringan bus kota, estafet paling sedikit untuk meneruskan pesan dalam sekelompok teman, belokan paling sedikit di gudang di antara dua rak. Untuk laundry, ia dapat menjawab pemberhentian paling sedikit yang diperlukan untuk mencapai setiap alamat antaran dengan sepeda motor dari peta jalan-jalan yang tersambung, dengan antrean yang memuat tempat-tempat yang dikunjungi berikutnya.

Ketika langkah-langkahnya tidak sama panjang, misalnya jalan-jalan dengan panjang berbeda, BFS polos tidak lagi cukup, dan pelajaran-pelajaran berikutnya mengajarkan algoritma yang memperumumnya. Tetapi antrean, tanda sudah dikunjungi, dan pointer "asal" untuk menyusun ulang lintasan tetap sama.

12.8 Kesalahan yang Sering Terjadi, dan Cara Menghindarinya

- **Menulis kode sebelum menggambar.** Setiap galat pointer dalam bab ini tampak di kertas dalam semenit dan tak tampak di layar selama sejam. Gambarlah setiap langkah dengan kotak dan panah lebih dulu.
- **Membaca `p->next->data` ketika `p->next` mungkin `NULL`.** Taruh uji `NULL` di syarat perulangan, seperti baris 6 **Pseudokode 12.1**.
- **Melupakan struktur yang kosong.** Senarai kosong di `insert_end`, antrean kosong di `dequeue`, akar yang belum ada di pohon: masing-masing memerlukan cabangnya sendiri, dan ujinya sendiri.

- **Memperbarui satu ujung antrean tetapi tidak ujung lainnya.** `enqueue` pada antrean kosong juga mengatur `head`; `dequeue` simpul terakhir mengatur `tail` menjadi `NULL`.
- **Selisih satu dalam navigasi.** Mendaki dari orang tua alih-alih dari simpul saat ini, atau menghentikan penelusuran senarai satu simpul terlambat. Telusuri contoh secara manual dan cetak baris jejak per langkah.
- **Menandai terlambat dalam BFS.** Tandai petak sudah dikunjungi ketika masuk antrean.
- **Memercayai contoh.** Kedua penyelesaian keliru dalam bab ini lolos pada contoh dari soalnya. Ujilah kasus tepi dan kasus yang sudah Anda hitung secara manual.
- **Membebaskan dalam urutan yang salah.** Bebaskan anak-anak pohon sebelum simpulnya (postorder), dan pointer penerus simpul senarai harus disimpan sebelum simpulnya dibebaskan.
- **Tidak menjalankan *sanitizer*.** Ia mengubah galat memori yang diam menjadi laporan yang tepat; pakailah pada setiap program pointer.

12.9 Ringkasan Bab

- Soal struktur data diselesaikan dengan menyatakannya ulang, menggambar contoh kecil langkah demi langkah, merancang di atas kertas, menulis kode satu fungsi demi satu, dan menguji dengan kasus yang disiapkan di bawah *sanitizer*.
- Soal 12.1: *bubble sort* pada *singly linked list* menukar data tetangga, berhenti satu simpul lebih awal setiap lintasan, dan berhenti sama sekali setelah lintasan tanpa pertukaran (**Pseudokode 12.1**); banyak pertukarannya sama dengan banyak inversi (Persamaan (12.1)). Penyisipan di akhir menyusuri sampai simpul terakhir, dengan kasus khusus untuk senarai kosong.
- Soal 12.2: pointer orang tua di setiap simpul membuat "naik k tingkat" menjadi perulangan k langkah; pohon dibangun oleh **Pseudokode 12.2** dengan pesan galat yang cermat, ditunjukkan lewat preorder, inorder, postorder, dan cetakan menyamping (**Gambar 12.1**), lalu dibebaskan secara postorder.
- Soal 12.3: *breadth-first search* dengan antrean *linked list* menjelajah lingkaran demi lingkaran, sehingga kedatangan pertama adalah yang terpendek (Persamaan (12.2), **Pseudokode 12.3**, **Gambar 12.2**); menandai petak ketika dimasukkan ke antrean menjaga antrean tetap kecil; larik "asal" menyusun ulang lintasannya.
- Penyelesaian keliru yang menggoda, yaitu satu lintasan *bubble*, pendakian dari orang tua, dan *depth-first search*, semuanya lolos pada contoh dari soalnya, dan karena itulah pengujian di luar contoh bukanlah pilihan (**Tabel 12.1**).

12.10 Soal Latihan

1. Tanpa menjalankan program, berikan banyak lintasan dan pertukaran yang diperlukan `list_sort` untuk senarai 3, 1, 2, 5, 4, dan senarainya setelah menyisipkan 0 di akhir.
2. Urutkan *linked list* dari Soal 12.1 dengan *menyambung ulang simpul* alih-alih menukar data, memakai *insertion sort*: ambil simpul-simpul dari senarai satu per satu dan taruh masing-masing di tempatnya dalam senarai terurut yang baru. Tunjukkan bahwa bilangan yang sama mempertahankan urutan asalnya.
3. Gambarlah pohon yang dibangun oleh perintah `0 1`, `l 2`, `l 3`, `r 4`, `uuur 5`, `l 6` (setiap bilangan di baris datanya sendiri), dan berikan preorder dan inorder-nya. Perintah mana yang akan membuat berkas tidak sah bila `uuur` diganti `uuuur`, dan apa yang akan dicetak program?

4. Tambahkan pada `tree_from_file` fungsi yang mencetak, untuk setiap simpul, lintasan dari akar, seperti `50 L 30 R 40`, dengan memakai pointer orang tua. Berapa biayanya untuk pohon berisi n simpul dengan tinggi h?
5. Untuk kuda, petak mana di papan yang memerlukan langkah paling banyak dari (7,0), dan berapa banyak? Bacalah jawabannya dari **Gambar 12.2**, lalu periksa dengan program.
6. Ubah `knight.c` agar ukuran papan N dibaca dari baris pertama masukan (sampai 20), dan berikan langkah paling sedikit dari satu sudut ke sudut seberangnya untuk $N = 4$, 5, dan 10.

12.11 Penyelesaian Soal Latihan

1. Inversi 3, 1, 2, 5, 4 adalah (3,1), (3,2), dan (5,4): tiga, jadi 3 pertukaran menurut Persamaan (12.1). Lintasan 1 menukar 3 dengan 1, 3 dengan 2, dan 5 dengan 4, menghasilkan 1, 2, 3, 4, 5 dengan 3 pertukaran; lintasan 2 tidak menukar apa pun dan berhenti. Dua lintasan, tiga pertukaran; setelah penyisipan: 1, 2, 3, 4, 5, 0.

2. Kode 12.11 memberi setiap simpul ruas kedua `born`, posisinya dalam masukan, agar keluaran menunjukkan simpul *mana* pergi ke mana. Perulangannya mengambil simpul pertama dari senarai masukan; bila simpul itu harus berada sebelum simpul pertama senarai terurut, ia menjadi simpul pertama yang baru; kalau tidak, penelusuran menemukan simpul terakhir yang bilangannya `<=` bilangan baru dan menyambungkan simpul baru sesudahnya, dengan dua penugasan dalam urutan aman dari Bab 10. Karena penelusuran melewati bilangan yang sama (`<=`), simpul ditaruh sesudah simpul-simpul yang sama yang sudah ada, sehingga bilangan yang sama mempertahankan urutan masukannya: *insertion sort* ini *stabil*. **Kode 12.12** adalah versi Python-nya.

Kode 12.11 `list_insertion_sort.c`: *insertion sort* dengan menyambung ulang simpul.

```c
/* list_insertion_sort.c - Exercise 12.2: sort the list by RELINKING nodes
 * (insertion sort): take the nodes off one by one and put each into its
 * place in a new, sorted list. No data field is ever changed. */
#include <stdio.h>
#include <stdlib.h>

struct sNode {
    int data;
    int born;               /* position in the input: shows which node is which */
    struct sNode *next;
};

static struct sNode *push_front(struct sNode *head, int value, int born)
{
    struct sNode *n = malloc(sizeof *n);
    if (n == NULL)
        exit(EXIT_FAILURE);
    n->data = value;
    n->born = born;
    n->next = head;
    return n;
}

static void print_list(const char *label, const struct sNode *p)
{
    printf("%-7s:", label);
    for (; p != NULL; p = p->next)
        printf(" %d(#%d)", p->data, p->born);
    printf("\n");
}

static struct sNode *insertion_sort(struct sNode *head)
{
    struct sNode *sorted = NULL;
    while (head != NULL) {
        struct sNode *n = head;          /* take the first node off */
        head = head->next;
        if (sorted == NULL || n->data < sorted->data) {
            n->next = sorted;            /* it goes to the front */
            sorted = n;
        } else {
            struct sNode *p = sorted;    /* find the last node <= n->data */
            while (p->next != NULL && p->next->data <= n->data)
                p = p->next;
            n->next = p->next;           /* 1: n points at p's successor */
            p->next = n;                 /* 2: p points at n */
```

Kode 12.11 (Lanjutan) *list_insertion_sort.c: insertion sort dengan menyambung ulang simpul.*

```
        }
    }
    return sorted;
}

int main(void)
{
    const int values[] = {29, 4, 17, 4, 42, 8, 15};
    struct sNode *head = NULL;
    for (int i = 6; i >= 0; i--)
        head = push_front(head, values[i], i + 1);
    print_list("before", head);
    head = insertion_sort(head);
    print_list("after", head);
    while (head != NULL) {
        struct sNode *next = head->next;
        free(head);
        head = next;
    }
    return 0;
}
```

Kode 12.12 `list_insertion_sort.py`: hal yang sama dalam Python.

```python
# list_insertion_sort.py - Exercise 12.2: sort the list by RELINKING nodes
# (insertion sort). No data field is ever changed.


class sNode:
    def __init__(self, data, born, next=None):
        self.data = data
        self.born = born               # position in the input
        self.next = next


def print_list(label, p):
    s = "%-7s:" % label
    while p is not None:
        s += " %d(#%d)" % (p.data, p.born)
        p = p.next
    print(s)


def insertion_sort(head):
    sorted_ = None
    while head is not None:
        n, head = head, head.next           # take the first node off
        if sorted_ is None or n.data < sorted_.data:
            n.next, sorted_ = sorted_, n    # it goes to the front
        else:
            p = sorted_                     # find the last node <= n.data
            while p.next is not None and p.next.data <= n.data:
                p = p.next
            n.next = p.next                 # 1: n points at p's successor
            p.next = n                      # 2: p points at n
    return sorted_


def main():
    values = [29, 4, 17, 4, 42, 8, 15]
    head = None
    for i in range(len(values) - 1, -1, -1):
        head = sNode(values[i], i + 1, head)
    print_list("before", head)
    head = insertion_sort(head)
    print_list("after", head)


if __name__ == "__main__":
    main()
```

```
before : 29(#1) 4(#2) 17(#3) 4(#4) 42(#5) 8(#6) 15(#7)
after  : 4(#2) 4(#4) 8(#6) 15(#7) 17(#3) 29(#1) 42(#5)
```

Simpul yang lahir kedua (4 yang pertama) tetap datang sebelum simpul yang lahir keempat (4 yang kedua), dan tidak ada ruas `data` yang diubah: simpul-simpulnyalah yang berpindah.

3. 1 adalah akar; 2 anak kirinya; 3 anak kiri 2; `r 4` membuat 4 anak kanan simpul *saat ini* 3; `uuur 5` naik dari 4 ke 3, ke 2, ke 1 dan membuat 5 anak kanan 1; `l 6` membuat 6 anak kiri 5. Preorder: 1 2 3 4 5 6; inorder: 3 4 2 1 6 5. Dengan `uuuur`, pendakian mencapai 1 setelah tiga langkah dan tidak menemukan orang tua untuk langkah keempat, sehingga program mencetak `line  9: 'uuuur' climbs above the root` (perintahnya berada di baris 9 berkas) lalu berhenti.

4. Untuk setiap simpul, ikuti `parent` naik sampai akar sambil menyimpan simpul-simpulnya dalam larik, lalu cetak dari akar ke bawah, menulis `L` atau `R` menurut apakah setiap simpul adalah anak kiri atau kanan orang tuanya. Setiap simpul memakan paling banyak h langkah, sehingga

seluruh simpul memakan $O(nh)$: $O(n\log n)$ untuk pohon seimbang, $O(n^2)$ untuk pohon yang berupa satu rantai panjang.

5. Bilangan terbesar pada **Gambar 12.2** adalah 6, yang hanya dicapai oleh (0,7), sudut seberangnya. Program dengan opsi `board` mencetak papan yang sama, dan tidak ada petak yang menunjukkan bilangan lebih besar.

6. Baca `N` lebih dulu dan pakai sebagai pengganti konstanta 8 di `is_valid`, di perulangan, dan di ukuran larik (deklarasikan larik dengan maksimum 20 dan pakai hanya `N` baris dan kolom pertama). Sudut ke sudut memerlukan 2 langkah untuk $N = 4$, misalnya (0,0), (2,1), (3,3); 4 langkah untuk $N = 5$; dan 6 langkah untuk $N = 10$; program menegaskan setiap nilai itu. Papan kecil tidak selalu lebih mudah: pada papan 3×3 petak tengah sama sekali tidak dapat dicapai, dan `dist` tetap -1, kasus yang harus dilaporkan program sebagai "tidak terjangkau".

Daftar Pustaka

[1] D. E. Knuth, *The Art of Computer Programming, Vol. 1: Fundamental Algorithms*, 3rd ed. Reading, MA, USA: Addison-Wesley, 1997, §2.2 "Linear Lists" and §2.3 "Trees."

[2] P. J. Deitel and H. M. Deitel, *C How to Program*, 9th ed. Hoboken, NJ, USA: Pearson, 2022, ch. 12 "Data Structures."

[3] Python Software Foundation, *The Python Standard Library*, Python 3.14: "re — Regular expression operations" and "collections.deque." [Online]. Available: https://docs.python.org/3/library/ (accessed Sep. 2026).

[4] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*, 4th ed. Cambridge, MA, USA: MIT Press, 2022, ch. 10 "Elementary Data Structures" and §20.2 "Breadth-first search."

Bab 13

C++ sebagai C yang Lebih Baik

Selama dua belas bab Anda menulis C: bahasa yang kecil dan jujur, tempat Anda mengurus sendiri setiap bita. Bab ini membawa Anda ke C++, bahasa yang tumbuh dari C dan masih menerima hampir setiap program C yang pernah Anda tulis, tetapi menambahkan alat-alat yang menghapus seluruh keluarga kesalahan. Di akhir bab ini Anda akan mampu mengompilasi program C++17 dengan `g++`; membaca dan menulis dengan `cin` dan `cout` alih-alih `scanf` dan `printf`; menyimpan teks dalam `std::string` dan daftar nilai dalam `std::vector` alih-alih array berukuran tetap; mengirim argumen *lewat referensi* dan menjelaskan bedanya dengan mengirim nilai atau pointer; memberi beberapa fungsi satu nama melalui *overloading* (pembebanan nama) dan membiarkan pemanggil melewatkan argumen yang punya *nilai bawaan* (*default argument*); mengalokasikan memori dengan `new` dan membebaskannya dengan `delete` yang sepadan, sekaligus mengerti mengapa Anda jarang memerlukannya; dan merangkai semuanya menjadi versi 1.0 program kasir Profitina Laundry yang ditulis ulang dalam C++ dan, seperti biasa di buku ini, juga dalam Python dengan keluaran yang persis sama.

13.1 Struk Bu Rina

Bu Rina mengelola Profitina Laundry, sebuah laundry kecil di jalan ramai dekat kampus. Mahasiswa menitipkan sekantong pakaian di pagi hari, pegawai kantor membawa kemeja untuk disetrika, dan pada hari Minggu keluarga-keluarga membawa bed cover. Setiap kantong ditimbang, berat dan jenis layanannya diketik di laptop, lalu sebuah program kecil mencetak harganya. Anda sudah mengikuti program itu sepanjang bab-bab C buku ini: ia belajar mengulang antrean pelanggan, menyimpan pesanan dalam array dan struktur, dan pada Bab 11 menyimpannya ke berkas.

Kasir versi C itu berjalan, tetapi punya riwayat kecelakaan kecil, dan masing-masing memberi pelajaran. Suatu kali seorang pelanggan menyebutkan nama lengkapnya, *Raden Ajeng Kusumawardhani Sulistyaningrum*, dan nama itu meluap dari array 30 karakter yang menampungnya; huruf-huruf yang tidak muat menimpa berat yang tersimpan tepat sesudahnya di memori, dan struk menunjukkan harga beberapa juta rupiah. Suatu kali seorang pegawai baru menambahkan pesanan, lupa menulis `&` pada pemanggilan `scanf`, dan program mogok pada pelanggan pertama hari itu. Suatu kali program lupa melakukan `free` pada memori pesanan sehari, dan setelah seminggu tanpa dinyalakan ulang, laptop menjadi lambat. Tidak satu pun kesalahan itu bodoh. Itulah kesalahan yang dibuat *semua orang* dalam C, karena C memercayakan setiap detail kepada pemrogram dan hampir tidak memeriksa apa pun.

C++ diciptakan tepat untuk keadaan seperti ini. Ia mempertahankan semua yang membuat C cepat dan dekat dengan mesin, lalu menambahkan lapisan alat yang mengerjakan pembukuan berbahaya itu untuk Anda: tipe string yang memanjang sesuai nama apa pun, tipe daftar yang bertambah panjang ketika pesanan datang dan membebaskan dirinya sendiri ketika tidak diperlukan lagi, masukan dan keluaran yang mengenal tipe setiap variabel, serta parameter yang dapat mengubah variabel milik pemanggil tanpa satu pun `&` atau `*` pada pemanggilannya. Inti bab ini adalah bahwa Anda boleh tetap berpikir seperti yang Anda pelajari di bab-bab C, dan membiarkan bahasa mengurus bagian-bagian yang dulu sering menggigit.

Siapa yang terbantu? Pertama, Bu Rina: program kasir yang tidak bisa rusak gara-gara nama panjang adalah program yang tidak membuatnya harus menelepon siapa pun. Kedua, Anda: setiap alat dalam bab ini muncul di hampir setiap program C++ yang pernah ditulis, dan Bab 14 serta 15 dibangun di atas semuanya. Ketiga, siapa saja yang kelak memelihara program itu: kode C++ yang memakai alat-alat ini lebih pendek, dan kode pendek menyisakan lebih sedikit tempat bersembunyi bagi kesalahan.

13.2 Gagasannya dalam Kata-Kata Sederhana

13.2.1 Satu keluarga, dua dialek

Pada 1979, Bjarne Stroustrup, yang bekerja di Bell Labs tempat C diciptakan beberapa tahun sebelumnya, mulai menambahkan *kelas* ke dalam C agar dapat menulis program simulasi besar dengan lebih tertata. Hasilnya ia namai "C with Classes"; pada 1983 namanya diganti menjadi C++, sebuah lelucon pemrogram, karena `++` adalah operator penambah satu dalam C: C++ berarti "C, satu langkah lebih jauh" [1]. Sejak 1998 bahasa ini dibakukan oleh International Organization for Standardization (ISO), dan edisi baru standarnya terbit setiap tiga tahun: C++11, C++14, C++17, C++20, C++23. Komite menuntaskan pekerjaan teknis edisi berikutnya, C++26, pada pertemuannya bulan Maret 2026 [2].

Buku ini memakai **C++17** di seluruh bagiannya. Versi ini didukung penuh oleh setiap kompilator yang mungkin Anda temui, memuat semua alat yang diperlukan bab ini dan dua bab berikutnya, dan masih menjadi versi yang dipakai sebagian besar kode yang sudah ada. Kompilator yang lebih baru memakai versi yang lebih baru secara bawaan (GCC 16.1, yang dirilis April 2026, mengompilasi C++20 secara bawaan [3]); karena itu kita selalu menulis `-std=c++17` secara eksplisit: perintah yang sama lalu memberi hasil yang sama di setiap mesin.

C++ *hampir* merupakan superset dari C. Hampir setiap program C juga merupakan program C++, dan kompilator yang Anda pasang di Bab 1 (`gcc` dari MSYS2 di Windows, *command-line tools* Xcode di macOS, `build-essential` di Linux) juga memuat kompilator C++ yang bernama `g++`. Ada sedikit perbedaan (misalnya, C++ tidak mengizinkan hasil `malloc` diberikan ke `int*` tanpa *cast*, dan C++ punya kata kunci tambahan seperti `class`, `new`, dan `bool`), tetapi jarang berpengaruh. Yang penting, C++ *menambahkan*, dan tambahan itu memungkinkan Anda menulis program yang sama dengan cara yang lebih aman.

13.2.2 Arti "C yang lebih baik"

Tabel 13.1 adalah seluruh bab ini dalam satu tabel: di kiri cara C yang sudah Anda kenal, di tengah cara C++ yang akan Anda pelajari, dan di kanan cara Python, bahasa kedua buku ini, untuk pekerjaan yang sama.

Tabel 13.1 Tabel migrasi: pekerjaan yang sama dalam C, dalam C++, dan dalam Python.

Pekerjaan	C (Bab 1–12)	C++ (bab ini)	Python
Mencetak	`printf("%d kg\n", w);`	`std::cout << w << " kg\n";`	`print(f"{w} kg")`
Membaca	`scanf("%d", &w);`	`std::cin >> w;`	`w = int(input())`
Teks	`char name[31];`	`std::string name;`	`name = ""`
Daftar nilai	`int grams[100];`	`std::vector<int> grams;`	`grams = []`
Mengubah variabel pemanggil	`void f(int* p)`, panggil `f(&x)`	`void f(int& p)`, panggil `f(x)`	kembalikan nilai baru
Beberapa fungsi, satu tugas	`printInt`, `printText`	*overload* `print`	satu fungsi, tipe apa pun
Argumen opsional	tidak mungkin	argumen bawaan	argumen bawaan
Alokasi / pembebasan	`malloc` / `free`	`new` / `delete` (atau `vector`)	otomatis
Benar dan salah	`int` 1 dan 0	`bool`: `true`, `false`	`True`, `False`

Baik tur singkat bahasa ini oleh penciptanya [4] maupun buku teks klasik [5] menempuh jalan yang sama dari C ke C++. Bacalah tabel ini sekali sekarang, lalu kembalilah kepadanya di akhir bab; pada saat itu setiap barisnya sudah akrab.

13.2.3 Mengompilasi program C++

Berkas sumber C++ berakhiran `.cpp`. Anda mengompilasinya dengan `g++` persis seperti mengompilasi C dengan `gcc`, dan kita menambahkan flag peringatan yang sama seperti biasa:

```
g++ -std=c++17 -O2 -Wall -Wextra hello_laundry.cpp -o hello_laundry
./hello_laundry
```

Di Windows, di terminal MSYS2 atau di PowerShell setelah folder `ucrt64\bin` milik MSYS2 masuk ke `PATH` (Bab 1), perintahnya sama dan programnya bernama `hello_laundry.exe`. Di macOS, `g++` adalah nama kedua bagi `clang++` buatan Apple, yang menerima flag yang sama. Di Visual Studio Code, ekstensi C/C++ yang Anda pasang di Bab 1 juga menangani berkas `.cpp`; *build task*-nya hanya perlu memakai `g++` alih-alih `gcc`.

Kode bab ini berada di `FP/Code_new/Chapter13`: program C++ di `cpp/`, kembarannya dalam Python di `python/`, berkas masukan di `input/`, keluaran yang diharapkan dari setiap program di `expected/`, dan program pengujian di `tests/` (Subbab 13.11). Semua perintah di bawah diketik di dalam folder `Chapter13`.

13.3 Stream: `cout` dan `cin`

Program C++ pertama kita menanyakan nama pelanggan dan berat cucian, lalu mencetak harganya. **Kode 13.1** adalah programnya secara utuh.

Kode 13.1 `hello_laundry.cpp`: membaca nama dan berat dengan `cin` serta mencetak dengan `cout`.

```cpp
// hello_laundry.cpp - Chapter 13: a first C++ program for Profitina Laundry.
// Reads a customer's name and the weight of the laundry in grams, prints the price.
#include <iostream>
#include <string>

int main() {
    const long long ratePerKg = 7000;          // wash and fold, rupiah per kg
    std::string name;                          // grows to fit any name
    long long grams = 0;

    std::cout << "Customer name: ";
    std::getline(std::cin, name);              // a whole line, spaces included
    std::cout << "Weight in grams: ";
    std::cin >> grams;                         // no format string, no &

    long long price = (grams * ratePerKg + 500) / 1000;   // rounded to 1 rupiah
    std::cout << "\nHello, " << name << "!\n";
    std::cout << grams << " g of laundry costs Rp " << price << ".\n";
    return 0;
}
```

Baris demi baris, hal-hal barunya adalah sebagai berikut.

`#include <iostream>` mendatangkan *stream* (aliran) masukan/keluaran. Stream adalah aliran karakter: `std::cout` ("*character output*") mengalir ke layar, `std::cin` ("*character input*") mengalir dari papan ketik. Nama header pustaka C++ sendiri tidak memakai `.h`; header C tetap tersedia dengan nama yang diawali `c`, seperti `<cstdio>` dan `<cmath>`.

`std::` adalah awalan *namespace* (ruang nama). Seluruh pustaka standar C++ tinggal di namespace bernama `std` ("*standard*"), sehingga nama-namanya tidak dapat bertabrakan dengan nama Anda: Anda boleh menulis fungsi sendiri bernama `sort` tanpa menyembunyikan `std::sort`. Banyak buku menulis `using namespace std;` di bagian atas lalu menghilangkan awalannya. Itu praktis di program kecil, tetapi menjadi sumber tabrakan nama yang membingungkan di program besar, jadi di buku ini kita selalu menulis awalannya.

`std::cout << "Customer name: ";` mengirim sebuah string ke layar. Operator `<<` di sini disebut operator *penyisipan* (*insertion*); bayangkan sebagai anak panah yang mendorong nilai ke dalam stream. Beberapa nilai dapat didorong dalam satu pernyataan, `std::cout << grams << " g of laundry costs Rp " << price << ".\n";`, dan setiap nilai dicetak dengan cara yang benar *sesuai tipenya*: `grams` sebagai bilangan bulat, teks sebagai teks. Tidak ada string format, jadi tidak mungkin menulis `%d` untuk sebuah `double` atau lupa satu argumen. Itulah arti "aman tipe" (*type-safe*): kompilator mengetahui tipe setiap nilai dan memilih sendiri kode pencetaknya.

`std::cin >> grams;` membaca sebuah nilai dari papan ketik ke dalam `grams`. Operator `>>` (*extraction*, pengambilan) menunjuk ke arah sebaliknya: dari stream ke variabel. Perhatikan apa yang tidak ada: tidak ada `&`. `cin` menerima variabel itu sendiri, *lewat referensi*, mekanisme yang akan Anda temui di Subbab 13.6. Seperti `scanf("%lld")`, ia melompati spasi di depan dan membaca satu bilangan.

`std::getline(std::cin, name);` membaca satu baris utuh, termasuk spasinya, ke dalam sebuah `std::string`. Operator `>>` akan berhenti pada spasi pertama dan hanya membaca "Bu"; karena itulah nama dibaca dengan `getline`.

Harga dihitung hanya dengan aritmetika bilangan bulat. Berat g gram dengan tarif r rupiah per kilogram berharga $g \cdot r/1000$ rupiah; agar dibulatkan ke rupiah terdekat alih-alih dipotong, kita menambahkan setengah dari pembaginya sebelum pembagian bulat:

$$P = \left\lfloor \frac{g \cdot r + 500}{1000} \right\rfloor \tag{13.1}$$

Di sini $\lfloor x \rfloor$ ("*floor* dari x", lantai) adalah bilangan bulat terbesar yang tidak melebihi x, tepat seperti hasil pembagian bulat C dan C++ untuk bilangan positif. Untuk $g = 3450$ dan $r = 7000$: $3450 \times 7000 = 24{,}150{,}000$; ditambah 500 lalu dibagi 1000 menghasilkan 24,150 (pecahan ,5 dibuang). Kita sengaja menghindari `double`: uang yang dihitung dengan bilangan titik mengambang bisa menjadi 24149,999999 dan tercetak 24149, padahal kasir tidak boleh selisih satu rupiah pun hanya karena cara komputer menyimpan pecahan.

Kode 13.2 adalah program yang sama dalam Python. `input(prompt)` mencetak prompt lalu membaca satu baris, dan `int(...)` mengubah teks menjadi bilangan; operator `//` adalah pembagian bulat Python, yang membulatkan ke arah minus tak hingga (sama dengan C++ untuk bilangan positif yang kita pakai).

Kode 13.2 `hello_laundry.py`: program yang sama dalam Python.

```python
# hello_laundry.py - Chapter 13: a first program for Profitina Laundry.
# Reads a customer's name and the weight of the laundry in grams, prints the price.

RATE_PER_KG = 7000                          # wash and fold, rupiah per kg

name = input("Customer name: ")
grams = int(input("Weight in grams: "))

price = (grams * RATE_PER_KG + 500) // 1000    # rounded to 1 rupiah
print(f"\nHello, {name}!")
print(f"{grams} g of laundry costs Rp {price}.")
```

Untuk menguji program tanpa mengetik, kita menaruh dua baris masukan di berkas `input/hello.txt` ("Bu Rina Wulandari" dan "3450") dan *mengalihkan* papan ketik ke berkas itu dengan `<`:

```
$ g++ -std=c++17 -O2 -Wall -Wextra cpp/hello_laundry.cpp -o hello_laundry
$ ./hello_laundry < input/hello.txt
$ python3 python/hello_laundry.py < input/hello.txt
```

Keduanya mencetak tepat seperti ini:

```
Customer name: Weight in grams:
Hello, Bu Rina Wulandari!
3450 g of laundry costs Rp 24150.
```

Kedua prompt muncul pada satu baris karena jawaban yang biasanya mengakhiri setiap baris kini datang dari berkas dan tidak tampil di layar. Bila diketik langsung, percakapannya tampak seperti yang Anda harapkan.

> **JEBAKAN:** Setelah `std::cin >> grams;`, karakter baris baru yang mengakhiri bilangan itu masih menunggu di dalam stream. `std::getline` yang langsung menyusul akan membaca sisa baris yang kosong itu dan mengembalikan string kosong. Bila Anda harus membaca satu baris setelah bilangan, lompati dulu sisa baris bilangan itu dengan `std::cin.ignore(1000, '\n');`. Dalam **Kode 13.1** masalah ini kita hindari dengan membaca barisnya lebih dulu.

13.4 `std::string`: Teks yang Selalu Muat

Dalam C, string adalah array `char` yang diakhiri karakter `'\0'`. Anda harus menentukan ukurannya di muka, menyalin dengan `strcpy`, menyambung dengan `strcat`, membandingkan dengan `strcmp`, dan selalu ingat bahwa semuanya dengan senang hati menulis melewati ujung

array. `std::string` adalah sebuah *kelas* (Bab 14 menjelaskan istilah ini) yang menampung teks sepanjang apa pun dan mengurus memorinya sendiri [6]. **Kode 13.3** menunjukkan operasi yang paling sering Anda perlukan.

Kode 13.3 `strings_demo.cpp`: operasi sehari-hari `std::string`.

```
// strings_demo.cpp - Chapter 13: std::string instead of char arrays.
#include <cctype>
#include <iostream>
#include <string>

int main() {
    std::string first = "Rina";
    std::string last = "Wulandari";
    std::string full = first + " " + last;     // + joins strings, no strcat
    std::cout << "Full name : " << full << '\n';
    std::cout << "Length    : " << full.size() << " characters\n";
    std::cout << "First char: " << full[0] << '\n';

    std::string upper = full;                  // a real copy, not a pointer
    for (char& c : upper)                      // c is a reference into upper
        c = static_cast<char>(std::toupper(static_cast<unsigned char>(c)));
    std::cout << "Upper case: " << upper << '\n';
    std::cout << "Unchanged : " << full << '\n';

    std::size_t space = full.find(' ');
    std::cout << "Space at  : " << space << '\n';
    std::cout << "Surname   : " << full.substr(space + 1) << '\n';

    std::string code = "PL-" + std::to_string(2026) + "-" + first.substr(0, 3);
    std::cout << "Order code: " << code << '\n';

    std::string a = "Ani", b = "Andi";
    std::cout << "Compare   : " << a << (a < b ? " < " : " > ") << b << '\n';
    if (full.find("Sari") == std::string::npos)
        std::cout << "Search    : no 'Sari' in the name\n";
    return 0;
}
```

Keluarannya:

```
Full name : Rina Wulandari
Length    : 14 characters
First char: R
Upper case: RINA WULANDARI
Unchanged : Rina Wulandari
Space at  : 4
Surname   : Wulandari
Order code: PL-2026-Rin
Compare   : Ani > Andi
Search    : no 'Sari' in the name
```

Telusurilah sambil mengingat versi C-nya.

- `first + " " + last` menyambung string dengan `+`. Hasilnya string baru yang panjangnya tepat sebanyak yang diperlukan; tidak ada *buffer* yang bisa meluap. `+=` menambahkan ke string yang sudah ada.
- `full.size()` adalah banyaknya karakter (14), dihitung seketika karena string mengingat panjangnya sendiri; `strlen` di C harus berjalan sampai `'\0'`.
- `full[0]` adalah karakter pertama, seperti di C. `full.at(0)` melakukan hal yang sama tetapi memeriksa indeksnya dan melaporkan kesalahan bila indeks berada di luar string.

- `std::string upper = full;` membuat *salinan* sungguhan. Di C, `char* upper = full;` hanya menyalin alamat, dan mengubah `upper` berarti mengubah `full`. Baris "Unchanged" membuktikan bahwa aslinya tidak tersentuh.
- `for (char& c : upper)` adalah *perulangan for berbasis rentang* (*range-based for loop*): ia berjalan sekali untuk setiap karakter `upper`. Tanda `&` menjadikan `c` referensi ke karakter di dalam string, sehingga memberi nilai ke `c` mengubah string itu sendiri (Subbab 13.6 menjelaskan referensi). `std::toupper` berasal dari `<cctype>`, nama C++ untuk `<ctype.h>` milik C; dua `static_cast` adalah cara cermat memanggilnya, karena karakter di atas 127 jika tidak demikian akan dikirim sebagai bilangan negatif, yang tidak diterima `toupper`.
- `full.find(' ')` mengembalikan posisi spasi pertama (4), atau nilai khusus `std::string::npos` ("*no position*") bila tidak ada. `full.substr(4 + 1)` adalah bagian dari posisi 5 sampai akhir; `first.substr(0, 3)` adalah 3 karakter mulai posisi 0.
- `std::to_string(2026)` mengubah bilangan menjadi teks, pekerjaan yang di C dilakukan `sprintf`.
- `a < b` membandingkan dua string menurut urutan kamus berdasarkan kode karakter, seperti `strcmp(a, b) < 0`. "Ani" lebih besar daripada "Andi" karena pada posisi ketiga `'i'` (kode 105) lebih besar daripada `'d'` (kode 100).

Kode 13.4 melakukan hal yang sama dalam Python, yang `str`-nya sejak dulu berperilaku seperti `std::string`. Ada satu perbedaan yang patut dicatat: string Python *immutable* (tidak dapat diubah) setelah dibuat, sehingga `full.upper()` mengembalikan string baru alih-alih mengubah karakter di tempatnya. Irisan (*slicing*), `full[space + 1:]`, berperan sebagai `substr`, dan `find` mengembalikan −1 alih-alih `npos`.

Kode 13.4 `strings_demo.py`: operasi yang sama dengan `str` milik Python.

```
# strings_demo.py - Chapter 13: the same string operations with Python's str.

first = "Rina"
last = "Wulandari"
full = first + " " + last
print(f"Full name : {full}")
print(f"Length    : {len(full)} characters")
print(f"First char: {full[0]}")

upper = full.upper()                          # str is immutable: a new string
print(f"Upper case: {upper}")
print(f"Unchanged : {full}")

space = full.find(" ")
print(f"Space at  : {space}")
print(f"Surname   : {full[space + 1:]}")

code = "PL-" + str(2026) + "-" + first[0:3]
print(f"Order code: {code}")

a, b = "Ani", "Andi"
print(f"Compare   : {a}{' < ' if a < b else ' > '}{b}")
if full.find("Sari") == -1:
    print("Search    : no 'Sari' in the name")
```

13.5 `std::vector`: Array yang Bisa Bertambah Panjang

Alat sehari-hari kedua menggantikan array berukuran tetap. Sebuah `std::vector<int>` adalah daftar `int` yang mengetahui ukurannya sendiri, dapat bertambah di ujungnya, membebaskan memorinya sendiri, dan dapat memeriksa indeksnya. Bagian di antara kurung sudut adalah tipe

elemennya: `std::vector<double>`, `std::vector<std::string>`, dan kelak `std::vector<Order>` semuanya bekerja dengan cara yang sama. **Kode 13.5** menyimpan berat pesanan sehari.

Kode 13.5 `vector_demo.cpp`: berat pesanan sehari dalam sebuah `std::vector`.

```cpp
// vector_demo.cpp - Chapter 13: std::vector instead of fixed arrays.
#include <algorithm>
#include <iostream>
#include <stdexcept>
#include <vector>

int main() {
    std::vector<int> grams = {3450, 1200, 5000, 2750};
    grams.push_back(4100);                      // the vector grows by itself
    std::cout << "Orders today : " << grams.size() << '\n';

    long long total = 0;
    for (int g : grams)                         // range-based for: every element
        total += g;
    std::cout << "Total weight : " << total << " g\n";
    std::cout << "Heaviest     : "
              << *std::max_element(grams.begin(), grams.end()) << " g\n";

    std::sort(grams.begin(), grams.end());      // the library sorts for us
    std::cout << "Sorted       :";
    for (int g : grams) std::cout << ' ' << g;
    std::cout << '\n';

    std::cout << "grams.at(2)  : " << grams.at(2) << '\n';
    try {
        std::cout << grams.at(9) << '\n';       // at() checks the index
    } catch (const std::out_of_range&) {
        std::cout << "grams.at(9)  : refused, there are only " << grams.size()
                  << " orders\n";
    }
    return 0;
}
```

```
Orders today : 5
Total weight : 16500 g
Heaviest     : 5000 g
Sorted       : 1200 2750 3450 4100 5000
grams.at(2)  : 3450
grams.at(9)  : refused, there are only 5 orders
```

`push_back` menambahkan elemen di ujung; vektor mencarikan tempat sendiri, pindah ke blok memori yang lebih besar bila perlu (ia tumbuh dengan faktor pengali, sehingga menambahkan n elemen secara keseluruhan memakan waktu sebanding dengan n). `grams.size()` adalah banyaknya elemen saat ini. Perulangan for berbasis rentang mengunjungi setiap elemen tanpa indeks, jadi tidak ada indeks yang bisa salah.

Dua fungsi berasal dari `<algorithm>`, header algoritma siap pakai. `std::max_element` mengembalikan posisi (sebuah *iterator*, pointer yang diperumum) dari elemen terbesar, dan `*` di depannya membaca nilai di posisi itu, sama seperti `*p` membaca nilai yang ditunjuk pointer. `std::sort(grams.begin(), grams.end())` mengurutkan seluruh vektor; `begin()` dan `end()` menandai elemen pertama dan posisi *tepat sesudah* elemen terakhir, konvensi "setengah terbuka" yang sama dengan perulangan C `for (i = 0; i < n; i++)`. Anda sudah menulis *bubble sort* sendiri di Bab 6, dan itu layak dilakukan sekali; mulai sekarang Anda boleh memakai `std::sort`, yang jauh lebih cepat untuk data besar.

`grams[9]` berperilaku seperti di C: tanpa pemeriksaan, dan membaca melewati ujung adalah perilaku tak terdefinisi. `grams.at(9)` memeriksa indeks dan, bila di luar rentang, *melempar eksepsi* (*throw an exception*) bertipe `std::out_of_range`. Eksepsi adalah cara sebuah fungsi mengatakan "saya tidak bisa melakukan ini" yang tidak dapat diabaikan pemanggilnya: alur normal berhenti, dan kendali melompat ke blok `catch` terdekat yang menerima jenis eksepsi tersebut. Di sini `catch` mencetak pesan yang sopan. Bab 14 memakai eksepsi untuk menolak pesanan yang tidak sah; untuk sekarang cukup diketahui bahwa `at` mengubah kesalahan memori yang diam-diam menjadi kesalahan yang jelas dan dapat ditangkap.

Kode 13.6 adalah versi Python-nya. `list` Python sejak dulu merupakan array yang dapat bertambah panjang, dan ia *selalu* memeriksa indeks, sehingga `grams[9]` memunculkan `IndexError` yang ditangkap dengan cara yang sama.

Kode 13.6 `vector_demo.py`: `list` milik Python melakukan apa yang dilakukan `std::vector`.

```python
# vector_demo.py - Chapter 13: Python's list does what std::vector does.

grams = [3450, 1200, 5000, 2750]
grams.append(4100)                          # the list grows by itself
print(f"Orders today : {len(grams)}")

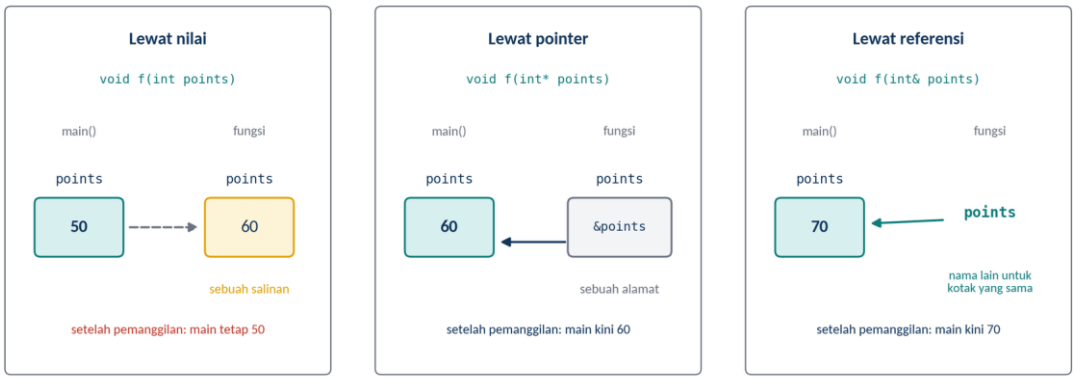
total = 0
for g in grams:
    total += g
print(f"Total weight : {total} g")
print(f"Heaviest     : {max(grams)} g")

grams.sort()
print("Sorted       :" + "".join(f" {g}" for g in grams))

print(f"grams.at(2)  : {grams[2]}")
try:
    print(grams[9])                         # Python always checks the index
except IndexError:
    print(f"grams.at(9)  : refused, there are only {len(grams)} orders")
```

13.6 Referensi: Mengubah Variabel Pemanggil Tanpa Pointer

Di Bab 7 Anda belajar bahwa fungsi C menerima *salinan* argumennya, dan bahwa agar fungsi dapat mengubah variabel pemanggil, Anda harus mengirim alamatnya dan menulis `*p` di mana-mana di dalam fungsi. C++ mempertahankan cara itu dan menambahkan cara ketiga. **Gambar 13.1** menampilkan ketiganya berdampingan.



Gambar 13.1 Tiga cara mengirim `points`: parameter nilai menerima salinan, parameter pointer menerima alamat, dan parameter referensi adalah nama lain bagi variabel milik pemanggil sendiri.

> **DEFINISI.** *Referensi* adalah nama lain bagi variabel yang sudah ada. Referensi dideklarasikan dengan `&` sesudah tipe, `int& alias = points;`, dan sejak itu `alias` dan `points` adalah kotak yang sama di memori: membaca atau menulis yang satu berarti membaca atau menulis yang lain. Referensi harus diinisialisasi saat dideklarasikan, tidak pernah dapat dialihkan ke variabel lain sesudahnya, dan tidak ada "referensi null".

Parameter referensi seperti `int& points` menjadi, pada setiap pemanggilan, nama lain bagi argumen yang dikirim pemanggil. Fungsi menulis `points += 10;` tanpa `*`, dan pemanggil menulis `addBonusByReference(points);` tanpa `&`. Di balik layar kompilator biasanya mengirim sebuah alamat, persis seperti pointer; bedanya, bahasa (bukan pemrogram) yang mengurus `&` dan `*`. **Kode 13.7** menjalankan ketiga cara itu berturut-turut pada poin loyalitas Bu Rina.

Kode 13.7 `references.cpp`: parameter nilai, pointer, dan referensi, serta fungsi dengan dua hasil.

```
// references.cpp - Chapter 13: pass by value, by pointer and by reference.
#include <iostream>
#include <vector>

void addBonusByValue(int points) { points = points + 10; }   // a copy changes
void addBonusByPointer(int* points) { *points += 10; }       // C style
void addBonusByReference(int& points) { points += 10; }      // C++ style

// Two results at once: the function fills the caller's variables.
void summarize(const std::vector<int>& grams, long long& total, int& heaviest) {
    total = 0;
    heaviest = 0;
    for (int g : grams) {
        total += g;
        if (g > heaviest) heaviest = g;
    }
}

int main() {
    int points = 50;
    addBonusByValue(points);
    std::cout << "by value     : " << points << '\n';
    addBonusByPointer(&points);
    std::cout << "by pointer   : " << points << '\n';
    addBonusByReference(points);
    std::cout << "by reference : " << points << '\n';

    int& alias = points;                        // another name for the same box
    alias = 99;
    std::cout << "alias = 99   : points is now " << points << '\n';

    std::vector<int> grams = {3450, 1200, 5000, 2750};
    long long total = 0;
    int heaviest = 0;
    summarize(grams, total, heaviest);          // const& : no copy, no change
    std::cout << "summary      : " << total << " g in total, heaviest "
              << heaviest << " g\n";
    return 0;
}
```

```
by value     : 50
by pointer   : 60
by reference : 70
alias = 99   : points is now 99
summary      : 12400 g in total, heaviest 5000 g
```

Versi nilai hanya mengubah salinannya sendiri (50 tetap 50); versi pointer dan versi referensi sama-sama mengubah variabel pemanggil. `int& alias = points;` menunjukkan bahwa referensi tidak hanya untuk parameter: sesudahnya, memberi nilai 99 ke `alias` menaruh 99 di `points`.

Fungsi `summarize` menunjukkan dua pemakaian referensi yang paling umum dalam kode nyata.

- **Parameter keluaran.** `long long& total` dan `int& heaviest` membuat satu fungsi mengembalikan dua hasil dengan mengisi variabel pemanggil. (Bab 7 melakukannya dengan pointer.)
- **Referensi const untuk masukan.** `const std::vector<int>& grams` mengirim vektor *tanpa menyalinnya*, hal penting bila vektor itu memuat sejuta pesanan, dan `const` berjanji bahwa fungsi tidak akan mengubahnya. Kompilator menegakkan janji itu: `grams.push_back(1);` di dalam `summarize` tidak akan terkompilasi. Mengirim objek besar sebagai `const&` adalah kebiasaan sehari-hari dalam C++, dan C++ Core Guidelines menganjurkannya untuk setiap parameter yang hanya dibaca [7]: `const std::string&` pada kasir di bawah adalah gagasan yang sama.

Kapan memakai pointer dan kapan referensi? Pakailah referensi bila selalu ada tepat satu objek yang dirujuk, yang hampir selalu terjadi pada parameter. Pakailah pointer bila "tidak ada objek" adalah jawaban yang bermakna (pointer boleh bernilai `nullptr`, ejaan C++ untuk `NULL`), atau bila Anda perlu menunjuk objek yang berbeda-beda dari waktu ke waktu, misalnya ketika menelusuri *linked list* seperti di Bab 10.

Python tidak punya referensi dalam arti ini, dan **Kode 13.8** menunjukkan apa yang dilakukannya sebagai gantinya. Setiap variabel Python adalah nama yang *terikat* pada sebuah objek. Mengirim argumen berarti mengikat nama parameter pada objek yang sama; bila fungsi lalu mengikat ulang nama itu (`points = points + 10`), hanya nama lokalnya yang berubah. Sebuah `int` tidak pernah dapat diubah di tempatnya, sehingga fungsi tidak dapat mengubah bilangan bulat milik pemanggil. Sebuah list *dapat* diubah di tempatnya, sehingga list berisi satu elemen berfungsi seperti kotak yang dapat dilihat kedua pihak. Jawaban khas Python untuk "dua hasil" lebih sederhana lagi: kembalikan sebuah tuple, `return total, heaviest`, dan bongkar di tempat pemanggilan.

Kode 13.8 `references.py`: apa yang dilakukan Python sebagai ganti referensi.

```python
# references.py - Chapter 13: what Python does instead of references.
# An int cannot be changed inside a function; a list (a mutable box) can.


def add_bonus_by_value(points):
    points = points + 10                        # rebinds the local name only


def add_bonus_in_box(box):
    box[0] += 10                                # changes the list the caller sees


def summarize(grams):
    """Two results at once: Python returns a tuple."""
    return sum(grams), max(grams)


points = 50
add_bonus_by_value(points)
print(f"by value     : {points}")
box = [points]
add_bonus_in_box(box)
print(f"by pointer   : {box[0]}")
add_bonus_in_box(box)
print(f"by reference : {box[0]}")

alias = box                                     # another name for the same list
alias[0] = 99
print(f"alias = 99   : points is now {box[0]}")

grams = [3450, 1200, 5000, 2750]
total, heaviest = summarize(grams)
print(f"summary      : {total} g in total, heaviest {heaviest} g")
```

13.7 Overloading dan Argumen Bawaan

Dalam C setiap fungsi memerlukan namanya sendiri. Bila Anda ingin mencetak teks, jumlah uang berlabel, dan garis pembatas, Anda menulis `printText`, `printAmount`, dan `printRule`. C++ mengizinkan ketiganya berbagi satu nama, karena kompilator dapat membedakannya dari *daftar parameternya*. Inilah yang disebut *function overloading*.

> **DEFINISI.** Dua fungsi adalah *overload* satu sama lain bila namanya sama dan daftar parameternya berbeda (banyaknya parameter berbeda, atau tipenya berbeda). Pada setiap pemanggilan, kompilator memilih overload yang parameternya paling cocok dengan argumennya; inilah *overload resolution* (resolusi overload), dan seluruhnya terjadi pada waktu kompilasi. Tipe kembalian saja tidak dapat membedakan overload.

Argumen bawaan (*default argument*) adalah nilai yang ditulis dalam deklarasi, `bool express = false`, yang diisikan kompilator bila pemanggil tidak menyertakan argumen itu. Argumen bawaan harus berada di parameter paling kanan, karena argumen dicocokkan dari kiri: `charge(3450, 7000)` berarti `charge(3450, 7000, false)`. **Kode 13.9** memakai keduanya.

Kode 13.9 `overload_default.cpp`: tiga overload `printLine` dan satu argumen bawaan pada `charge`.

```cpp
// overload_default.cpp - Chapter 13: one name, several functions, and a default.
#include <iostream>
#include <string>

// Three functions called printLine: the compiler picks one by the arguments.
void printLine(const std::string& text) { std::cout << text << '\n'; }

void printLine(const std::string& label, long long rupiah) {
    std::cout << label << ": Rp " << rupiah << '\n';
}

void printLine(char c, int count) { std::cout << std::string(count, c) << '\n'; }

// A default argument: a caller may leave out 'express'.
long long charge(int grams, long long ratePerKg, bool express = false) {
    long long price = (grams * ratePerKg + 500) / 1000;
    if (express) price += price / 2;           // express costs 50% more
    return price;
}

int main() {
    printLine('=', 30);
    printLine("PROFITINA LAUNDRY");
    printLine('=', 30);
    printLine("Wash 3450 g        ", charge(3450, 7000));
    printLine("Wash 3450 g express", charge(3450, 7000, true));
    printLine("Iron 2000 g        ", charge(2000, 5000));
    printLine('-', 30);
    return 0;
}
```

```
==============================
PROFITINA LAUNDRY
==============================
Wash 3450 g        : Rp 24150
Wash 3450 g express: Rp 36225
Iron 2000 g        : Rp 10000
------------------------------
```

Untuk setiap pemanggilan `printLine`, kompilator melihat argumennya. `printLine('=', 30)` berisi sebuah `char` dan sebuah `int`, jadi hanya overload ketiga yang cocok. `printLine("PROFITINA LAUNDRY")` berisi satu argumen, jadi hanya yang pertama yang cocok. `printLine("Wash 3450 g        ", charge(3450, 7000))` berisi teks dan sebuah `long long`, yang cocok dengan yang kedua. Overloading membuat *pemanggil* dapat memikirkan "cetak satu baris" sebagai satu gagasan.

Biaya tambahan ekspres adalah 50% dari harga reguler, dibulatkan ke bawah, sehingga harga ekspres dari harga reguler P adalah

$$P_{\text{express}} = P + \left\lfloor \frac{P}{2} \right\rfloor \tag{13.2}$$

Untuk 3450 g cucian, $P = 24{,}150$ menurut Persamaan (13.1), dan $P_{\text{express}} = 24{,}150 + 12{,}075 = 36{,}225$, jumlah kedua pada keluaran.

Bagaimana dua fungsi bisa bernama sama padahal *linker* memerlukan nama yang unik untuk masing-masing? Kompilator diam-diam *menghias* setiap nama dengan tipe parameternya (disebut *name mangling*): `printLine(char, int)` menjadi sesuatu seperti `_Z9printLineci`. Itu pula sebabnya program C++ yang memanggil fungsi hasil kompilasi C harus mendeklarasikannya dengan `extern "C"`, yang mematikan penghiasan itu.

Overloading punya satu bahaya: kompilator dapat menemukan dua overload yang sama cocoknya. Pemanggilan itu lalu *ambigu* dan program tidak terkompilasi. Bab 16 (Soal 16.2) menjumpai jebakan persis ini ketika overload dan argumen bawaan dicampur.

Python punya argumen bawaan [8], tetapi tidak punya overloading: `def printLine` yang kedua hanya akan *menggantikan* yang pertama. Karena itu **Kode 13.10** memberi tiga fungsi cetak itu tiga nama. (Fungsi Python juga dapat menerima "apa saja" lalu memeriksa tipe argumennya dengan `isinstance`, tetapi nama yang terpisah lebih jelas.)

Kode 13.10 `overload_default.py`: argumen bawaan dalam Python, dan tiga nama sebagai ganti overload.

```
# overload_default.py - Chapter 13: Python has default arguments but no
# overloading; a second 'def' with the same name would replace the first.
# So the three C++ printLine functions get three names here.


def print_text(text):
    print(text)


def print_amount(label, rupiah):
    print(f"{label}: Rp {rupiah}")


def print_rule(c, count):
    print(c * count)


def charge(grams, rate_per_kg, express=False):
    price = (grams * rate_per_kg + 500) // 1000
    if express:
        price += price // 2                    # express costs 50% more
    return price


print_rule("=", 30)
print_text("PROFITINA LAUNDRY")
print_rule("=", 30)
print_amount("Wash 3450 g        ", charge(3450, 7000))
print_amount("Wash 3450 g express", charge(3450, 7000, True))
print_amount("Iron 2000 g        ", charge(2000, 5000))
print_rule("-", 30)
```

13.8 `new` dan `delete`

C mengalokasikan memori pada waktu jalan dengan `malloc` dan membebaskannya dengan `free`. C++ punya dua operator untuk pekerjaan yang sama: `new` membuat sebuah objek (atau array objek) dan mengembalikan pointer kepadanya, dan `delete` menghancurkannya. Dibandingkan `malloc`, `new` mengenal tipenya, jadi tidak ada `sizeof` yang harus dihitung dan tidak ada *cast* yang harus ditulis, dan ia dapat langsung menginisialisasi objeknya. **Kode 13.11** menunjukkan kedua bentuknya, lalu alternatif yang lebih baik.

Kode 13.11 `new_delete.cpp`: `new[]` dengan `delete[]`, `new` dengan `delete`, dan sebuah `vector` yang tidak memerlukan keduanya.

```cpp
// new_delete.cpp - Chapter 13: new and delete, and why vector is usually better.
#include <iostream>
#include <vector>

int main() {
    int n = 4;
    int* grams = new int[n]{3450, 1200, 5000, 2750};   // typed, no sizeof, no cast
    long long total = 0;
    for (int i = 0; i < n; ++i) total += grams[i];
    std::cout << "new[]  : " << n << " orders, " << total << " g\n";
    delete[] grams;                                    // new[] pairs with delete[]
    grams = nullptr;                                   // no dangling pointer left behind

    int* one = new int(42);                            // one int, set to 42
    std::cout << "new    : *one = " << *one << '\n';
    delete one;                                        // new pairs with delete

    std::vector<int> v = {3450, 1200, 5000, 2750};     // frees itself
    v.push_back(4100);
    total = 0;
    for (int g : v) total += g;
    std::cout << "vector : " << v.size() << " orders, " << total << " g\n";
    return 0;
}
```

```
new[]  : 4 orders, 12400 g
new    : *one = 42
vector : 5 orders, 16500 g
```

Aturan yang tidak boleh dilanggar adalah bahwa bentuk-bentuk itu *berpasangan*: memori dari `new` dibebaskan dengan `delete`, memori dari `new[]` dengan `delete[]`, dan memori dari `malloc` dengan `free`. Mencampurnya (`delete` pada array, `free` pada hasil `new`) adalah perilaku tak terdefinisi; mungkin berjalan di satu mesin dan merusak memori di mesin lain. Setelah `delete`, pointer masih memegang alamat lama, sebuah *dangling pointer* (pointer menggantung); memberinya nilai `nullptr` membuat kesalahan berikutnya langsung terlihat.

Bagian terakhir program adalah pelajaran sesungguhnya: sebuah `std::vector` mengerjakan semua yang dilakukan array `new[]`, bertambah panjang bila perlu, dan membebaskan memorinya sendiri di akhir `main`. Dalam C++ modern Anda jarang menulis `new` dan `delete`; Anda memakai kontainer seperti `vector` dan `string`, dan bila memang memerlukan satu objek di *heap*, memakai *smart pointer* seperti `std::unique_ptr` yang memanggil `delete` untuk Anda. Bab 14 menunjukkan mekanisme di baliknya: *destruktor*.

Python tidak pernah meminta Anda membebaskan memori. Setiap objek dibuat saat diperlukan dan dibebaskan otomatis ketika tidak ada lagi nama yang merujuknya (**Kode 13.12**).

Kode 13.12 `new_delete.py`: Python membebaskan memori dengan sendirinya.

```python
# new_delete.py - Chapter 13: in Python every object is created on demand and
# freed automatically when nothing refers to it any more; there is no delete.

grams = [3450, 1200, 5000, 2750]
print(f"new[]  : {len(grams)} orders, {sum(grams)} g")
grams = None                                           # the list can now be freed

one = 42
print(f"new    : *one = {one}")

v = [3450, 1200, 5000, 2750]
v.append(4100)
print(f"vector : {len(v)} orders, {sum(v)} g")
```

13.9 Membangunnya: Kasir v1.0

Sekarang kita bangun ulang kasir Bu Rina dengan setiap alat dari bab ini. Kasir membaca pesanan sehari dari sebuah berkas teks, satu pesanan per baris, dengan bentuk `customer;service;grams;express`:

```
# customer;service;grams;express
Bu Rina Wulandari;wash;3450;no
Pak Budi Santoso;washiron;5200;yes
Mbak Sari Dewi;iron;2000;no
Mas Joko Prasetyo;wash;4100;yes
Bu Rina Wulandari;iron;1250;no
Pak Hendra Gunawan;washiron;2750;no
```

Baris yang diawali `#` adalah komentar. Ada tiga layanan, masing-masing dengan tarif per kilogram: `wash` (cuci lipat, Rp7.000), `washiron` (cuci setrika, Rp10.000), dan `iron` (setrika saja, Rp5.000). Program harus memeriksa setiap baris, mencetak satu baris untuk setiap pesanan yang diterima, menjelaskan setiap baris yang ditolak, dan menutup dengan total serta ringkasan per layanan. **Kode 13.13** adalah programnya secara lengkap.

Kode 13.13 `cashier.cpp`: kasir Profitina Laundry v1.0 dalam C++.

```cpp
// cashier.cpp - Chapter 13: Profitina Laundry cashier v1.0, the C cashier
// rewritten with the tools of C++: string, vector, references, overloading,
// default arguments, ifstream and iomanip.
#include <fstream>
#include <iomanip>
#include <iostream>
#include <string>
#include <vector>

struct Service {
    std::string name;
    long long ratePerKg;
};

struct Order {
    std::string customer;
    std::string service;
    int grams;
    bool express;
    long long price;
};

const std::vector<Service> SERVICES = {
    {"wash", 7000}, {"washiron", 10000}, {"iron", 5000}};

// Splits "a;b;c" into {"a", "b", "c"}.
std::vector<std::string> split(const std::string& line, char sep) {
    std::vector<std::string> parts;
    std::string current;
    for (char c : line) {
        if (c == sep) {
            parts.push_back(current);
            current.clear();
        } else {
            current += c;
        }
    }
    parts.push_back(current);
    return parts;
}
```

Kode 13.13 (Lanjutan) *cashier.cpp: kasir Profitina Laundry v1.0 dalam C++.*

```
// Finds the rate of a service; the answer comes back through a reference.
bool findRate(const std::string& service, long long& rate) {
    for (const Service& s : SERVICES)
        if (s.name == service) {
            rate = s.ratePerKg;
            return true;
        }
    return false;
}

// Accepts only 1 to 6 digits and a value from 1 to 50000 grams.
bool parseGrams(const std::string& text, int& grams) {
    if (text.empty() || text.size() > 6) return false;
    int value = 0;
    for (char c : text) {
        if (c < '0' || c > '9') return false;
        value = value * 10 + (c - '0');
    }
    if (value < 1 || value > 50000) return false;
    grams = value;
    return true;
}

long long charge(int grams, long long ratePerKg, bool express = false) {
    long long price = (grams * ratePerKg + 500) / 1000;
    if (express) price += price / 2;           // express costs 50% more
    return price;
}

// Two overloads: a weight in grams becomes "3.450 kg".
std::string kg(long long grams) {
    std::string frac = std::to_string(grams % 1000);
    return std::to_string(grams / 1000) + "." + std::string(3 - frac.size(), '0')
           + frac + " kg";
}
std::string kg(int grams) { return kg(static_cast<long long>(grams)); }
```

Kode 13.13 (Lanjutan) *cashier.cpp: kasir Profitina Laundry v1.0 dalam C++.*

```
// Reads one line; on success fills 'order', otherwise fills 'why'.
bool parseOrder(const std::string& line, Order& order, std::string& why) {
    std::vector<std::string> f = split(line, ';');
    long long rate = 0;
    if (f.size() != 4) {
        why = "expected 4 fields, found " + std::to_string(f.size());
        return false;
    }
    if (f[0].empty()) { why = "customer name is empty"; return false; }
    if (!findRate(f[1], rate)) {
        why = "unknown service '" + f[1] + "'";
        return false;
    }
    if (!parseGrams(f[2], order.grams)) {
        why = "weight '" + f[2] + "' is not 1..50000 grams";
        return false;
    }
    if (f[3] != "yes" && f[3] != "no") {
        why = "express must be yes or no, not '" + f[3] + "'";
        return false;
    }
    order.customer = f[0];
    order.service = f[1];
    order.express = (f[3] == "yes");
    order.price = charge(order.grams, rate, order.express);
    return true;
}

void printOrder(int number, const Order& o) {
    std::cout << std::right << std::setw(2) << number << ' ' << std::left
              << std::setw(20) << o.customer << ' ' << std::setw(9) << o.service
              << std::right << std::setw(10) << kg(o.grams) << "  Rp"
              << std::setw(7) << o.price << (o.express ? "  express" : "") << '\n';
}

int main(int argc, char* argv[]) {
    std::string path = argc > 1 ? argv[1] : "input/orders.txt";
    std::ifstream in(path);
    if (!in) {
        std::cout << "cannot open " << path << '\n';
        return 1;
    }
    std::cout << "PROFITINA LAUNDRY - cashier v1.0 (" << path << ")\n\n";
```

Kode 13.13 (Lanjutan) *cashier.cpp: kasir Profitina Laundry v1.0 dalam C++.*

```
    std::vector<Order> orders;
    std::string line, why;
    int lineNo = 0, refused = 0;
    while (std::getline(in, line)) {
        ++lineNo;
        if (line.empty() || line[0] == '#') continue;
        Order o;
        if (parseOrder(line, o, why)) {
            orders.push_back(o);
            printOrder(static_cast<int>(orders.size()), o);
        } else {
            ++refused;
            std::cout << "   line " << lineNo << " refused: " << why << '\n';
        }
    }

    long long totalGrams = 0, revenue = 0;
    int express = 0;
    for (const Order& o : orders) {
        totalGrams += o.grams;
        revenue += o.price;
        if (o.express) ++express;
    }
    std::cout << "\nOrders accepted : " << orders.size() << '\n'
              << "Orders refused  : " << refused << '\n'
              << "Express orders  : " << express << '\n'
              << "Total weight    : " << kg(totalGrams) << '\n'
              << "Revenue         : Rp " << revenue << "\n\nPer service\n";
    for (const Service& s : SERVICES) {
        int count = 0;
        long long grams = 0, money = 0;
        for (const Order& o : orders)
            if (o.service == s.name) {
                ++count;
                grams += o.grams;
                money += o.price;
            }
        std::cout << "  " << std::left << std::setw(9) << s.name << std::right
                  << std::setw(2) << count << " orders" << std::setw(11)
                  << kg(grams) << "  Rp" << std::setw(7) << money << '\n';
    }
    return 0;
}
```

Mari kita telusuri dari atas.

Datanya. `struct Service` dan `struct Order` adalah struktur dari Bab 10, dengan dua perbedaan C++: anggota `std::string` alih-alih array `char`, dan tidak perlu `typedef`, karena dalam C++ nama struktur sudah dengan sendirinya merupakan nama tipe. `SERVICES` adalah `const std::vector<Service>` yang diinisialisasi dengan daftar di dalam kurung kurawal; `const` berarti tidak ada bagian program yang dapat mengubah daftar harga secara tidak sengaja.

`split`. Fungsi bantu ini mengubah `"Bu Rina;wash;3450;no"` menjadi vektor berisi empat string. Ia menelusuri karakter dengan perulangan for berbasis rentang, mengumpulkannya di `current`, dan setiap kali bertemu pemisah ia mendorong `current` ke hasil lalu mengosongkannya. Kolom terakhir didorong setelah perulangan. Perhatikan apa yang *tidak* kita perlukan: tidak ada ukuran array, tidak ada `strtok` (yang mengubah string aslinya dan tidak bisa dipakai pada dua string sekaligus), dan tidak ada pemeriksaan apakah satu kolom terlalu panjang.

`findRate` dan `parseGrams`. Keduanya mengembalikan `bool` (“berhasilkah?”) dan menyerahkan hasil sebenarnya melalui parameter referensi, pola parameter keluaran dari Subbab 13.6.

`parseGrams` sengaja dibuat ketat: hanya angka `0` sampai `9`, paling banyak enam digit, dan nilai 1 sampai 50.000 gram (50 kg sudah lebih dari kantong mana pun). Kita tidak memakai `std::stoi` di sini, karena ia akan menerima `"2,5"` sebagai bilangan 2 dan diam-diam mengabaikan sisanya; kasir seharusnya menolak baris seperti itu dan mengatakan alasannya.

`charge`. Inilah Persamaan (13.1) dan Persamaan (13.2) dengan argumen bawaan, persis seperti di Subbab 13.7.

`kg`. Dua overload mengubah gram menjadi teks seperti `3.450 kg`: kilogram utuh, sebuah titik, dan gramnya diisi nol sampai tiga digit (`std::string(3 - frac.size(), '0')` adalah string berisi nol sebanyak itu). Satu overload menerima `int` (satu pesanan), yang lain `long long` (total sehari); versi `int` cukup memanggil versi lainnya.

`parseOrder`. Di sinilah semua bagian bertemu. Fungsi ini menerima baris, sebuah `Order&` untuk diisi, dan sebuah `std::string&` untuk menjelaskan kegagalan. Setiap pemeriksaan kembali pada masalah pertama dengan kalimat yang dapat dipahami orang. Hanya bila semua pemeriksaan lolos, kolom-kolom disalin ke `order` dan harganya dihitung.

`printOrder`. Pemformatan memakai *manipulator* dari `<iomanip>`. `std::setw(n)` menetapkan lebar hanya untuk nilai *berikutnya*; `std::left` dan `std::right` memilih perataan dan tetap berlaku sampai diubah. Jadi `std::left << std::setw(20) << o.customer` mencetak nama dalam kolom 20 karakter, diisi spasi di kanan. Inilah versi stream dari `printf("%-20s")`.

`main`. `int main(int argc, char* argv[])` sama seperti di C; `argv[1]`, bila ada, adalah nama berkas pesanan, dan bila tidak, dipakai `input/orders.txt`. `std::ifstream in(path);` membuka berkas untuk dibaca: `ifstream` adalah stream masukan yang terhubung ke berkas, dipakai persis seperti `cin`, dan *menutup berkasnya sendiri* ketika `in` keluar dari cakupan di akhir `main`, jadi tidak ada `fclose` yang bisa terlupa. `if (!in)` menanyakan apakah pembukaan berhasil. `while (std::getline(in, line))` membaca sampai akhir berkas. Pesanan yang diterima didorong ke `orders`, sebuah `std::vector<Order>`, sehingga tidak ada batas banyaknya pesanan dalam sehari. Total di akhir dihitung dengan perulangan berbasis rentang atas vektor itu.

Kode 13.14 adalah kasir yang sama dalam Python. Strukturnya identik; bedanya hanya pada kebiasaan khas Python: `line.split(";")` mengerjakan tugas `split` kita, sebuah pesanan berupa *dictionary*, fungsi yang mungkin gagal mengembalikan `None` atau pasangan `(order, why)` alih-alih mengisi parameter referensi, dan spesifikasi format `:<20` dan `:>7` di f-string melakukan apa yang dilakukan `setw`, `left`, dan `right` di C++.

Kode 13.14 `cashier.py`: kasir yang sama dalam Python.

```python
# cashier.py - Chapter 13: Profitina Laundry cashier v1.0 in Python, printing
# exactly what cashier.cpp prints.
import sys

SERVICES = [("wash", 7000), ("washiron", 10000), ("iron", 5000)]


def find_rate(service):
    """Returns the rate of a service, or None when it is unknown."""
    for name, rate in SERVICES:
        if name == service:
            return rate
    return None


def parse_grams(text):
    """Accepts only 1 to 6 digits and a value from 1 to 50000 grams."""
    if not text or len(text) > 6 or any(c not in "0123456789" for c in text):
        return None
    value = int(text)
    return value if 1 <= value <= 50000 else None


def charge(grams, rate_per_kg, express=False):
    price = (grams * rate_per_kg + 500) // 1000
    if express:
        price += price // 2                    # express costs 50% more
    return price


def kg(grams):
    return f"{grams // 1000}.{grams % 1000:03d} kg"


def parse_order(line):
    """Returns (order, None) on success or (None, why) on failure."""
    f = line.split(";")
    if len(f) != 4:
        return None, f"expected 4 fields, found {len(f)}"
    if f[0] == "":
        return None, "customer name is empty"
    rate = find_rate(f[1])
    if rate is None:
        return None, f"unknown service '{f[1]}'"
    grams = parse_grams(f[2])
    if grams is None:
```

Kode 13.14 (Lanjutan) *cashier.py: kasir yang sama dalam Python.*

```
        return None, f"weight '{f[2]}' is not 1..50000 grams"
    if f[3] not in ("yes", "no"):
        return None, f"express must be yes or no, not '{f[3]}'"
    express = f[3] == "yes"
    return {"customer": f[0], "service": f[1], "grams": grams,
            "express": express, "price": charge(grams, rate, express)}, None


def print_order(number, o):
    print(f"{number:>2} {o['customer']:<20} {o['service']:<9}{kg(o['grams']):>10}"
          f"  Rp{o['price']:>7}{'  express' if o['express'] else ''}")


def main():
    path = sys.argv[1] if len(sys.argv) > 1 else "input/orders.txt"
    try:
        lines = open(path, encoding="utf-8").read().splitlines()
    except OSError:
        print(f"cannot open {path}")
        return 1
    print(f"PROFITINA LAUNDRY - cashier v1.0 ({path})\n")

    orders, refused = [], 0
    for line_no, line in enumerate(lines, start=1):
        if line == "" or line.startswith("#"):
            continue
        order, why = parse_order(line)
        if order:
            orders.append(order)
            print_order(len(orders), order)
        else:
            refused += 1
            print(f"   line {line_no} refused: {why}")

    total_grams = sum(o["grams"] for o in orders)
    revenue = sum(o["price"] for o in orders)
    express = sum(1 for o in orders if o["express"])
    print(f"\nOrders accepted : {len(orders)}")
    print(f"Orders refused  : {refused}")
    print(f"Express orders  : {express}")
    print(f"Total weight    : {kg(total_grams)}")
    print(f"Revenue         : Rp {revenue}\n\nPer service")
    for name, _ in SERVICES:
        mine = [o for o in orders if o["service"] == name]
        grams = sum(o["grams"] for o in mine)
        money = sum(o["price"] for o in mine)
```

Kode 13.14 (Lanjutan) *cashier.py: kasir yang sama dalam Python.*

```
        print(f"  {name:<9}{len(mine):>2} orders{kg(grams):>11}  Rp{money:>7}")
    return 0


if __name__ == "__main__":
    sys.exit(main())
```

13.10 Menjalankannya

```
$ g++ -std=c++17 -O2 -Wall -Wextra cpp/cashier.cpp -o cashier
$ ./cashier
$ python3 python/cashier.py
```

Kedua program mencetak:

```
PROFITINA LAUNDRY - cashier v1.0 (input/orders.txt)

 1 Bu Rina Wulandari    wash       3.450 kg  Rp  24150
 2 Pak Budi Santoso     washiron   5.200 kg  Rp  78000  express
 3 Mbak Sari Dewi       iron       2.000 kg  Rp  10000
 4 Mas Joko Prasetyo    wash       4.100 kg  Rp  43050  express
 5 Bu Rina Wulandari    iron       1.250 kg  Rp   6250
 6 Pak Hendra Gunawan   washiron   2.750 kg  Rp  27500

Orders accepted : 6
Orders refused  : 0
Express orders  : 2
Total weight    : 18.750 kg
Revenue         : Rp 188950

Per service
  wash      2 orders   7.550 kg  Rp  67200
  washiron  2 orders   7.950 kg  Rp 105500
  iron      2 orders   3.250 kg  Rp  16250
```

Periksa satu baris dengan tangan. Cuci setrika Pak Budi 5.200 g dengan Rp10.000 per kg: Persamaan (13.1) memberi $(5200 \times 10000 + 500)/1000 = 52{,}000$ (tambahan $+500$ hilang dalam pembagian bulat), dan ekspres menambah setengahnya, $52{,}000 + 26{,}000 = 78{,}000$. Ringkasan per layanan menjumlahkan baris-baris setiap layanan: wash adalah $24{,}150 + 43{,}050 = 67{,}200$, dan ketiga layanan bersama-sama $67{,}200 + 105{,}500 + 16{,}250 = 188{,}950$, pendapatan hari itu. Setiap kali program mencetak total, periksalah setidaknya satu dengan tangan; itulah uji yang paling murah.

13.11 Mengujinya

Setiap program di buku ini disertai keluaran yang wajib dicetaknya, tersimpan di folder `expected/`. Program pengujian `tests/run_tests.py` mengompilasi setiap program C++ dengan `-Wall -Wextra -Werror` (satu peringatan dihitung sebagai kegagalan), menjalankannya beserta kembaran Python-nya dengan masukan yang sama, lalu membandingkan kedua keluaran dengan teks yang diharapkan, karakter demi karakter. Daftar programnya adalah berkas teks kecil, `tests/cases.txt`, satu baris per program:

```
# name | C++ sources | Python script | stdin | arguments
hello_laundry | hello_laundry.cpp | hello_laundry.py | input/hello.txt | -
strings_demo | strings_demo.cpp | strings_demo.py | - | -
vector_demo | vector_demo.cpp | vector_demo.py | - | -
references | references.cpp | references.py | - | -
overload_default | overload_default.cpp | overload_default.py | - | -
new_delete | new_delete.cpp | new_delete.py | - | -
cashier | cashier.cpp | cashier.py | - | -
cashier_bad | cashier.cpp | cashier.py | - | input/orders_bad.txt
```

Setiap baris memuat nama, sumber C++, skrip Python, berkas untuk masukan standar (`-` bila tidak ada), dan argumen baris perintah. Baris terakhir menjalankan kasir pada berkas pesanan kedua, yang dibahas di Subbab 13.12. **Kode 13.15** adalah program pengujiannya sendiri; berkas yang sama dipakai di Bab 14, 15, dan 16.

Kode 13.15 `tests/run_tests.py`: mengompilasi, menjalankan, dan membandingkan setiap program dalam satu bab.

```
#!/usr/bin/env python3
"""run_tests.py - build every C++ program of this chapter, run it and its
Python twin, and check that both print exactly the expected output in
expected/NAME.txt.

usage (from the chapter folder):  python3 tests/run_tests.py [--sanitize] [--update]
Each line of tests/cases.txt is:
    name | C++ sources | Python script | stdin | arguments
('-' means none). A C++ source list of the form '!file.cpp' must FAIL to compile.
"""
import os, shutil, subprocess, sys, tempfile

sys.dont_write_bytecode = True

HERE = os.path.dirname(os.path.dirname(os.path.abspath(__file__)))
FLAGS = ["-std=c++17", "-O2", "-Wall", "-Wextra", "-Werror"]
SAN = ["-fsanitize=address,undefined", "-fno-sanitize-recover=all"]


def run(cmd, stdin_path):
    stdin = open(os.path.join(HERE, stdin_path)) if stdin_path != "-" else None
    try:
        r = subprocess.run(cmd, cwd=HERE, stdin=stdin, capture_output=True,
                           text=True, timeout=60)
    finally:
        if stdin:
            stdin.close()
    return r.stdout + r.stderr


def main():
    sanitize = "--sanitize" in sys.argv
    update = "--update" in sys.argv
    gxx = shutil.which("g++") or "g++"
    py = sys.executable
    tmp = tempfile.mkdtemp()
    failures = 0
    for line in open(os.path.join(HERE, "tests", "cases.txt")):
        if not line.strip() or line.startswith("#"):
            continue
        name, srcs, script, stdin, args = [p.strip() for p in line.split("|")]
        args = [] if args == "-" else args.split()
        if srcs.startswith("!"):                    # a program that must not compile
            src = os.path.join(HERE, "cpp", srcs[1:])
            r = subprocess.run([gxx] + FLAGS + [src, "-o", os.path.join(tmp, name)],
                               capture_output=True, text=True)
```

Kode 13.15 (Lanjutan) *tests/run_tests.py: mengompilasi, menjalankan, dan membandingkan setiap program dalam satu bab.*

```
            ok = r.returncode != 0 and "ambiguous" in r.stderr
            verdict = "refused by g++ as expected" if ok else "FAIL"
            print("%-26s %s" % (name, verdict))
            failures += not ok
            continue
        exe = os.path.join(tmp, name)
        sources = [os.path.join(HERE, "cpp", s) for s in srcs.split()]
        r = subprocess.run([gxx] + FLAGS + (SAN if sanitize else []) + sources
                           + ["-o", exe], capture_output=True, text=True)
        if r.returncode != 0:
            print("%-26s COMPILE ERROR\n%s" % (name, r.stderr))
            failures += 1
            continue
        out_cpp = run([exe] + args, stdin)
        out_py = run([py, "-B", os.path.join("python", script)] + args, stdin)
        exp_path = os.path.join(HERE, "expected", name + ".txt")
        if update:
            with open(exp_path, "w") as f:
                f.write(out_cpp)
        expected = open(exp_path).read() if os.path.exists(exp_path) else None
        ok_cpp, ok_py = out_cpp == expected, out_py == expected
        print("%-26s C++ %-4s Python %s" % (name, "ok" if ok_cpp else "FAIL",
                                            "ok" if ok_py else "FAIL"))
        failures += (not ok_cpp) + (not ok_py)
    shutil.rmtree(tmp)
    print("all programs agree" if failures == 0 else "%d FAILURE(S)" % failures)
    sys.exit(1 if failures else 0)


if __name__ == "__main__":
    main()
```

Dengan `--sanitize`, program C++ juga dikompilasi dengan *address sanitizer* dan *undefined-behavior sanitizer*, yang menambahkan pemeriksaan akses memori di luar rentang, pemakaian memori yang sudah dibebaskan, luapan bilangan bulat, dan kesalahan serupa, lalu menghentikan program pada kesalahan pertama. Hasil pengujian bab ini:

```
$ python3 tests/run_tests.py --sanitize
```

```
hello_laundry              C++ ok   Python ok
strings_demo               C++ ok   Python ok
vector_demo                C++ ok   Python ok
references                 C++ ok   Python ok
overload_default           C++ ok   Python ok
new_delete                 C++ ok   Python ok
cashier                    C++ ok   Python ok
cashier_bad                C++ ok   Python ok
all programs agree
```

Dua bahasa yang mencetak hal yang persis sama saling memeriksa dengan kuat: bila salah satunya punya perbedaan pembulatan atau urutan baris yang lain, perbandingannya gagal. Setiap kali mengubah program, jalankan lagi pengujiannya.

13.12 Membuatnya Lebih Baik: Menolak Baris Buruk dengan Sopan

Program kasir bertemu masukan nyata, dan masukan nyata itu berantakan. Berkas `input/orders_bad.txt` mengumpulkan kesalahan yang memang terjadi di meja kasir: layanan

yang tidak ada, berat yang ditulis dengan koma, baris yang kehilangan satu kolom, nama kosong, "yes" yang salah eja, dan berat yang kelebihan satu nol.

```
# customer;service;grams;express
Bu Rina Wulandari;wash;3450;no
Pak Budi Santoso;dryclean;5200;yes
Mbak Sari Dewi;iron;2,5;no
Mas Joko Prasetyo;wash;4100
;wash;1000;no
Pak Hendra Gunawan;washiron;2750;maybe
Bu Lestari;iron;60000;no
Bu Lestari;iron;1800;yes
```

```
$ ./cashier input/orders_bad.txt
```

```
PROFITINA LAUNDRY - cashier v1.0 (input/orders_bad.txt)

 1 Bu Rina Wulandari    wash       3.450 kg  Rp  24150
   line 3 refused: unknown service 'dryclean'
   line 4 refused: weight '2,5' is not 1..50000 grams
   line 5 refused: expected 4 fields, found 3
   line 6 refused: customer name is empty
   line 7 refused: express must be yes or no, not 'maybe'
   line 8 refused: weight '60000' is not 1..50000 grams
 2 Bu Lestari           iron       1.800 kg  Rp  13500  express

Orders accepted : 2
Orders refused  : 6
Express orders  : 1
Total weight    : 5.250 kg
Revenue         : Rp 37650

Per service
  wash      1 orders   3.450 kg  Rp  24150
  washiron  0 orders   0.000 kg  Rp      0
  iron      1 orders   1.800 kg  Rp  13500
```

Setiap baris buruk ditolak beserta nomor barisnya dan alasan dalam kata-kata sederhana, sementara baris-baris baik di sekitarnya tetap diterima. Inilah perilaku program yang dapat dipercaya: ia tidak pernah menebak maksud baris yang salah, tidak pernah mogok, dan memberi tahu orang apa yang harus diperbaiki. Bandingkan dengan kasir C di Bab 11, tempat koma pada berat membuat `scanf` berhenti membaca dan sisa berkas hilang tanpa suara.

Tiga perbaikan lanjutan layak dikerjakan dan dijadikan soal latihan: menerima berat juga dalam kilogram dengan koma desimal seperti kebiasaan orang Indonesia (`3,45`) dan mengubahnya ke gram dengan cermat (Soal 3); membaca daftar harga dari berkas alih-alih dari kode sumber, sehingga Bu Rina dapat mengubah harga tanpa pemrogram (Soal 4); dan mengelompokkan struk menurut pelanggan, sehingga Bu Rina Wulandari yang punya dua pesanan hari ini mendapat satu total (Soal 5).

13.13 Memakainya Sungguhan

Kasir ini kecil, tetapi bentuknya cocok untuk banyak usaha kecil: daftar harga, berkas transaksi, pemeriksaan ketat setiap baris, satu baris cetakan per transaksi, dan sebuah ringkasan. Untuk menyesuaikannya dengan usaha Anda sendiri, ubahlah tiga hal.

- **Daftar harga.** Penjahit menarik biaya per potong dan per jenis pekerjaan; toko fotokopi per halaman dengan harga lebih rendah di atas 100 halaman; tempat cuci mobil per golongan kendaraan. Gantilah `SERVICES` dan perhitungan tarifnya. Bila harga dihitung per potong, kolom berat menjadi hitungan, dan `parseGrams` menjadi `parseCount` dengan batasnya sendiri.

- **Pemeriksaan.** Tanyakan kepada orang di meja kasir kesalahan apa yang benar-benar terjadi, lalu tambahkan satu pemeriksaan dengan satu pesan jelas untuk masing-masing. Pemeriksaan yang tidak pernah diperlukan tidak merugikan apa pun; pemeriksaan yang terlewat bisa mengacaukan pembukuan sehari.
- **Ringkasan.** Tanyakan kepada pemilik angka apa yang ia lihat di malam hari. Bagi Bu Rina itu pendapatan per layanan; toko fotokopi mungkin ingin halaman per jam, untuk tahu kapan perlu membuka mesin kedua.

Simpanlah uang dalam rupiah utuh dan aritmetika bilangan bulat, seperti Persamaan (13.1). Bila harga melibatkan persentase, bulatkan sekali, di tempat yang terdokumentasi, dengan cara yang sama di setiap program yang menghitungnya. Bab 15 mengubah aturan ini menjadi kelas `Rupiah` yang membuatnya sulit dilanggar.

13.14 Kesalahan yang Sering Terjadi, dan Cara Menghindarinya

- **Lupa `-std=c++17`.** Tanpa flag ini, versi kompilator yang berbeda dapat memilih versi bahasa yang berbeda, dan kode yang kemarin terkompilasi hari ini memberi peringatan atau galat. Selalu tulis flag-nya.
- **Mencampur `>>` dan `getline`.** `getline` yang langsung menyusul `cin >> number` membaca sisa baris yang kosong. Bacalah baris lebih dulu, atau panggil `std::cin.ignore(1000, '\n')` di antaranya.
- **Membandingkan string C dengan `==`.** Untuk `std::string`, `a == b` membandingkan teksnya. Untuk dua `char*`, yang dibandingkan adalah *alamat*, dan hasilnya hampir selalu salah. Bila program C++ masih memakai string C, ubahlah: `std::string(argv[1]) == "yes"`.
- **Mengindeks vektor melewati ujungnya.** `v[i]` tidak memeriksa. Pakailah perulangan berbasis rentang bila bisa, `v.at(i)` selama pengembangan, dan kompilasi pengujian dengan `-fsanitize=address`.
- **Referensi ke sesuatu yang sudah tidak ada.** Fungsi tidak boleh mengembalikan referensi ke salah satu variabel lokalnya sendiri: variabel itu mati saat fungsi kembali, dan referensinya menunjuk ke kekosongan (*dangling reference*). Kembalikan nilai saja; C++ memindahkan nilai itu keluar dengan murah.
- **`new` dan `delete` yang tidak sepadan.** `new[]` harus dibebaskan dengan `delete[]`, `new` dengan `delete`. Lebih baik lagi, pakailah `vector` dan jangan menulis keduanya.
- **Uang dalam `double`.** Jumlah rupiah yang dihitung dengan titik mengambang bisa tercetak satu rupiah lebih kecil dari seharusnya. Simpanlah rupiah utuh dalam `long long` dan bulatkan secara eksplisit, seperti Persamaan (13.1).
- **Overload yang ambigu.** Dua overload yang sama cocoknya untuk satu pemanggilan menghentikan kompilasi. Jagalah agar overload jelas berbeda, dan jangan menambahkan argumen bawaan yang membuat satu overload tampak seperti yang lain.

13.15 Ringkasan Bab

- C++ tumbuh dari C dan mengompilasi hampir setiap program C; `g++ -std=c++17 -Wall -Wextra` mengompilasi program-program buku ini di Windows (MSYS2), macOS, dan Linux. Komite ISO menuntaskan C++26 pada Maret 2026, tetapi C++17 sudah mencukupi seluruh keperluan buku ini.

- `std::cout << x` dan `std::cin >> x` mencetak dan membaca nilai bertipe apa pun tanpa string format; `std::getline` membaca satu baris utuh. Semua yang ada di pustaka standar tinggal di namespace `std`.
- `std::string` menampung teks sepanjang apa pun, disalin dan dibandingkan dengan operator biasa, dan menyediakan `size`, `find`, `substr`, dan `+`. `std::vector<T>` adalah array yang dapat bertambah panjang dengan `push_back`, `size`, `at`, dan perulangan berbasis rentang, serta bekerja dengan `std::sort`.
- Referensi adalah nama lain bagi variabel yang sudah ada. Parameter referensi (`int&`) memungkinkan fungsi mengubah variabel pemanggil tanpa pointer; referensi `const` mengirim objek besar tanpa menyalin dan tanpa hak mengubahnya.
- Overloading memberi beberapa fungsi satu nama, dibedakan oleh daftar parameternya; argumen bawaan memungkinkan pemanggil melewatkan argumen di bagian akhir. Keduanya diselesaikan pada waktu kompilasi.
- `new`/`delete` dan `new[]`/`delete[]` menggantikan `malloc`/`free` dan harus dipasangkan dengan benar; dalam praktik, kontainer seperti `vector` membuat keduanya jarang diperlukan.
- Python menawarkan kemudahan yang sama dengan caranya sendiri: `str`, `list`, nama yang terikat pada objek, argumen bawaan, memori otomatis, dan satu fungsi per nama. Kasir v1.0 ditulis dalam kedua bahasa dengan keluaran identik dan diuji dengan membandingkan keduanya terhadap teks yang diharapkan.

13.16 Soal Latihan

1. Tanpa menjalankannya, sebutkan apa yang dicetak `std::cout << 7 / 2 << ' ' << 7.0 / 2 << ' ' << (7 > 2) << '\n';`, dan mengapa nilai terakhir bukan `true`. Lalu ubah baris itu agar mencetak `true`.
2. Tulislah fungsi C++ `swapOrders` yang menukar dua nilai `std::string` melalui parameter referensi, beserta padanannya dalam Python. Mengapa versi Python harus mengembalikan sesuatu?
3. Perluas `parseGrams` pada kasir agar juga menerima kilogram dengan koma desimal, seperti `3,45` (berarti 3.450 g) atau `0,8` (800 g), dengan paling banyak tiga digit sesudah koma. Tetap tolak `2,5,1`, `,5`, dan `3,4567`. Kerjakan dalam C++ dan Python dan jaga agar keluarannya tetap sama.
4. Pindahkan daftar harga kasir ke berkas `input/services.txt` dengan baris seperti `wash;7000`, bacalah di awal `main` ke dalam `std::vector<Service>`, dan tolak untuk mulai bila berkasnya tidak ada atau ada tarif yang bukan bilangan positif.
5. Tambahkan ringkasan ketiga, “Per customer”, pada kasir: banyaknya pesanan dan total uang setiap pelanggan, dalam urutan kemunculan pertama setiap pelanggan.
6. Fungsi `kg(int)` dalam **Kode 13.13** hanya memanggil `kg(long long)`. Apa yang terjadi bila fungsi itu kita hapus? Apa yang terjadi bila justru `kg(long long)` yang dihapus dan `kg(int)` dipertahankan? Cobalah keduanya dan jelaskan perilaku kompilator.

13.17 Penyelesaian Soal Latihan

1. `7 / 2` adalah pembagian bulat, 3; `7.0 / 2` pembagian titik mengambang, 3.5; dan `(7 > 2)` adalah nilai `bool` `true`, yang secara bawaan dicetak stream sebagai `1`. Keluarannya `3 3.5 1`. Mengirim manipulator `std::boolalpha` ke stream lebih dulu mengubah pencetakan `bool` menjadi

kata: `std::cout << std::boolalpha << (7 > 2)` mencetak `true`. (`rupiah_demo.cpp` di Bab 15 memakai manipulator ini.)

2. Dalam C++:

```
void swapOrders(std::string& a, std::string& b) {
    std::string t = a;
    a = b;
    b = t;
}
```

Pemanggilannya `swapOrders(first, second);` dan kedua variabel berubah. (Pustaka standar sudah punya `std::swap` yang melakukan hal yang sama untuk tipe apa pun.) Dalam Python, fungsi tidak dapat mengikat ulang nama milik pemanggil, sehingga ia mengembalikan pasangan dalam urutan baru dan pemanggil membongkarnya: `def swap_orders(a, b): return b, a` dan `first, second = swap_orders(first, second)`; pemrogram Python biasanya langsung menulis `first, second = second, first`.

3. Pisahkan teks pada koma. Tanpa koma, pakai aturan lama. Dengan satu koma, kedua bagian harus berupa digit, bagian kiri paling banyak dua digit dan bagian kanan satu sampai tiga digit; isi bagian kanan dengan nol sampai tiga digit lalu hitung $1000 \cdot \text{left} + \text{right}$. Jadi `3,45` menjadi $3000 + 450 = 3450$ dan `0,8` menjadi $0 + 800 = 800$. Lalu terapkan batas 1..50000 yang sama. `2,5,1` punya dua koma, `,5` bagian kirinya kosong, dan `3,4567` punya empat digit di kanan; ketiganya ditolak. Dalam Python, `text.split(",")` memberi bagian-bagiannya dan pemeriksaan yang sama berlaku; jangan pernah memakai `float("3.45") * 1000`, yang menghasilkan `3450.0000000000005` dan menunjukkan mengapa uang dan ukuran lebih aman dalam bilangan bulat.

4. Ubah `SERVICES` dari konstanta menjadi `std::vector<Service>` yang diisi oleh `main`: buka `input/services.txt` dengan `std::ifstream`, lompati baris komentar, pisahkan setiap baris pada `;`, periksa bahwa ada dua kolom dan bahwa tarifnya hanya terdiri atas digit dan paling kecil 1, lalu dorong sebuah `Service`. Bila berkas tidak dapat dibuka atau ada baris yang salah, cetak masalahnya dan kembalikan 1 sebelum membaca pesanan apa pun. `findRate` dan perulangan ringkasan masing-masing perlu satu perubahan: keduanya menerima vektor itu sebagai parameter `const std::vector<Service>&` alih-alih memakai konstanta global. Versi Python membaca berkas itu ke dalam list berisi pasangan dengan cara yang sama.

5. Simpan `std::vector<std::string>` berisi nama pelanggan menurut urutan kemunculan pertama: untuk setiap pesanan yang diterima, cari di vektor itu (pencarian linear, Bab 6) dan dorong namanya bila belum ada. Lalu, untuk setiap nama, telusuri `orders` dan jumlahkan banyak pesanan dan uangnya, persis seperti perulangan per layanan. Untuk `input/orders.txt`, Bu Rina Wulandari muncul pertama dengan 2 pesanan dan $24{,}150 + 6{,}250 = 30{,}400$ rupiah. Dalam Python, `dict` menjaga urutan penyisipan sejak Python 3.7, sehingga `totals.setdefault(name, [0, 0])` mengurus pembukuannya dalam satu baris.

6. Tanpa `kg(int)`, pemanggilan `kg(o.grams)` dengan argumen `int` tetap terkompilasi: satu-satunya kandidat adalah `kg(long long)`, dan `int` diubah ke `long long` secara otomatis (sebuah *promosi* yang tidak kehilangan apa pun). Tanpa `kg(long long)`, pemanggilan `kg(totalGrams)` dengan argumen `long long` juga terkompilasi, karena `long long` dapat diubah ke `int`; dengan `-Wall -Wextra` tidak ada peringatan secara bawaan, tetapi konversi itu dapat kehilangan informasi untuk total di atas sekitar 2,1 miliar gram. Karena itulah versi `long long` yang kita jadikan versi sesungguhnya: konversi ke tipe yang lebih lebar selalu aman, konversi ke tipe yang lebih sempit tidak.

Daftar Pustaka

[1] B. Stroustrup, *The C++ Programming Language*, 4th ed. Upper Saddle River, NJ, USA: Addison-Wesley, 2013.

[2] H. Sutter, "C++26 is done! Trip report: March 2026 ISO C++ standards meeting (London Croydon, UK)," Sutter's Mill, Mar. 29, 2026. [Online]. Available: https://herbsutter.com/2026/03/29/c26-is-done-trip-report-march-2026-iso-c-standards-meeting-london-croydon-uk/

[3] GNU Project, "GCC 16 Release Series: Changes, New Features, and Fixes," Apr. 2026. [Online]. Available: https://gcc.gnu.org/gcc-16/

[4] B. Stroustrup, *A Tour of C++*, 3rd ed. Boston, MA, USA: Addison-Wesley, 2022.

[5] P. Deitel and H. Deitel, *C++ How to Program*, 10th ed. Hoboken, NJ, USA: Pearson, 2017.

[6] cppreference.com, "C++ reference: Strings library, Containers library (std::vector), Input/output library," accessed Sep. 2026. [Online]. Available: https://en.cppreference.com/w/cpp

[7] B. Stroustrup and H. Sutter, Eds., "C++ Core Guidelines," accessed Sep. 2026. [Online]. Available: https://isocpp.github.io/CppCoreGuidelines/CppCoreGuidelines

[8] Python Software Foundation, "The Python Tutorial, Section 4.9: More on Defining Functions," Python 3.14 documentation, accessed Sep. 2026. [Online]. Available: https://docs.python.org/3.14/tutorial/controlflow.html

Bab 14

Kelas dan Abstraksi Data

Sebuah struktur mengelompokkan nilai-nilai yang saling terkait, tetapi tidak dapat melindunginya: baris mana pun di mana pun dalam program boleh membuat berat sebuah pesanan menjadi nol atau nama pelanggannya menjadi kosong. Bab ini memperkenalkan *kelas*, gagasan inti C++, yang menyatukan data dengan fungsi-fungsi yang boleh menyentuhnya dan menyembunyikan selebihnya. Di akhir bab ini Anda akan mampu menjelaskan apa itu abstraksi data dan invarian; menulis kelas dengan data *private*, fungsi anggota *public*, konstruktor yang menolak nilai tidak sah, dan fungsi `const` yang berjanji tidak mengubah apa pun; memecah sebuah kelas menjadi berkas header dan berkas implementasi lalu mengompilasinya bersama; memakai anggota statis yang menjadi milik kelas secara keseluruhan; mengandalkan destruktor yang berjalan dengan sendirinya di akhir sebuah blok; membaca dan menulis hierarki kelas pertama yang kecil dengan pewarisan dan fungsi virtual; serta membangun buku pesanan Profitina Laundry sebagai dua kelas yang bekerja sama, `Order` dan `Shop`, dalam C++ dan Python, dengan keluaran yang identik.

14.1 Cucian Gratis, Tanpa Sengaja

Di penghujung suatu Sabtu yang sibuk, pegawai Bu Rina menghitung uang di laci dan mendapati kekurangan Rp24.150 dibandingkan angka dari program kasir. Keesokan harinya sebabnya terungkap. Satu pesanan dimasukkan dengan berat yang salah, dan untuk “membetulkannya” si pegawai memakai sebuah pilihan menu kecil yang dulu ditambahkan seorang mahasiswa ke program: pilihan itu mengubah berat pesanan terakhir secara langsung. Ia keliru mengetik 0, harganya menjadi nol, dan Bu Rina Wulandari membawa pulang cuciannya secara gratis. Tak seorang pun berbuat curang. Sebuah bilangan ditulis di tempat yang tidak semestinya, oleh kode yang jauh dari kode penghitung harga, dan tidak ada satu pun bagian program yang berkeberatan.

Dalam kasir versi C, pesanan adalah sebuah `struct`. Setiap fungsi yang dapat melihat pesanan dapat mengubah kolom mana pun menjadi nilai apa pun: berat 0 atau −500 gram, tarif 700 alih-alih 7.000, nama kosong. Aturan yang membuat pesanan *masuk akal* (pelanggan sungguhan, layanan yang dikenal, berat antara 1 gram dan 50 kilogram) hanya hidup di kepala pemrogram dan di beberapa pernyataan `if` pada fungsi masukan. Setiap fungsi baru adalah kesempatan baru untuk melupakannya, dan setiap pemrogram baru membawa kesempatan itu berlipat ganda.

Gagasan bab ini adalah menaruh aturan-aturan itu *di dalam* pesanan itu sendiri. Sebuah pesanan akan memeriksa nilai-nilainya saat dibuat dan menolak untuk ada bila nilainya salah. Datanya akan disembunyikan, sehingga satu-satunya cara mengubahnya adalah melalui sekumpulan kecil fungsi yang menjaga aturan. Pesanan yang sudah diambil tidak dapat dikembalikan ke mesin cuci; pesanan yang masih dicuci tidak dapat diambil. Setelah aturan berada di dalam, tidak ada pegawai, mahasiswa, atau pemrogram masa depan yang dapat melanggarnya secara tidak sengaja, karena program tidak akan terkompilasi atau akan menolak operasi itu dengan pesan yang jelas.

Ini membantu Bu Rina, yang pembukuannya akan cocok dan yang tidak perlu lagi menghitung ulang laci setiap malam dengan rasa curiga. Ini membantu setiap pemrogram yang kelak

menyentuh kasir, karena mereka dapat memakai `Order` tanpa perlu tahu isinya. Dan ini menyiapkan Anda untuk sisa perjalanan Anda sebagai pemrogram: kelas adalah batu bata dasar C++, Java, C#, Python, dan hampir setiap bahasa yang akan Anda pakai setelah buku ini.

14.2 Gagasannya dalam Kata-Kata Sederhana

14.2.1 Abstraksi data

Pikirkan cara Anda memakai *rice cooker*. Anda memasukkan beras dan air, menekan satu tombol, lalu menunggu lampunya berpindah dari "cook" ke "warm". Anda tidak membukanya untuk mengatur elemen pemanas, dan Anda tidak akan bisa melakukannya dengan aman seandainya mencoba: elemennya tersegel di dalam, dan termostat yang memutuskan kapan nasi matang bukan sesuatu yang dapat Anda sentuh. Yang Anda dapatkan adalah *antarmuka* yang sederhana (sebuah tombol dan sebuah lampu) dan sebuah jaminan (nasinya tidak gosong), sedangkan *implementasinya* (elemen, termostat, pengaturan waktu) tersembunyi. Pembuatnya bahkan dapat mengganti implementasi pada model tahun depan tanpa mengubah cara Anda memakainya.

Sebuah kelas melakukan hal yang sama untuk data. *Antarmukanya* adalah sekumpulan fungsi yang boleh dipanggil kode lain: membuat pesanan, menanyakan harganya, memindahkannya ke tahap berikutnya, menyerahkannya kepada pelanggan. *Implementasinya* adalah data di dalamnya dan kode fungsi-fungsi itu, dan semuanya tersembunyi. Pemisahan ini disebut *abstraksi data*, dan hasilnya, sebuah tipe yang didefinisikan oleh apa yang dapat Anda lakukan dengannya alih-alih oleh cara penyimpanannya, disebut *tipe data abstrak* (*abstract data type*) [1].

Contoh kedua yang akrab bagi kita adalah loket teller bank. Nasabah tidak masuk ke ruang brankas lalu mengambil uangnya sendiri; ia mengisi slip, menyerahkannya di loket, dan teller memeriksa slip itu: apakah tanda tangannya cocok, apakah saldonya cukup, apakah jumlahnya tidak melebihi batas penarikan harian. Hanya bila semuanya beres, teller mengambil uang dari laci dan mencatat transaksinya. Loket adalah antarmukanya, brankas dan buku besar adalah datanya yang tersembunyi, dan pemeriksaan teller adalah penjaga aturannya. Tidak ada nasabah yang dapat membuat saldonya negatif dengan sengaja atau tidak, karena satu-satunya jalan menuju uang melewati loket. Bank pun bebas mengganti sistem di balik loket, dari buku besar kertas ke komputer, tanpa mengubah cara nasabah menarik uang.

Kelas yang akan kita bangun untuk Profitina Laundry bekerja persis seperti loket itu. Kode di luar kelas adalah nasabah: ia boleh meminta pesanan dibuat, menanyakan harga, atau meminta pesanan dimajukan, tetapi semua permintaan melewati fungsi-fungsi *public* yang memeriksa aturan lebih dulu. Data pesanan sendiri, seperti uang di brankas, tidak dapat disentuh langsung.

14.2.2 Invarian

Apa yang kita peroleh dari penyembunyian? Kita memperoleh sebuah *invarian*.

> **DEFINISI.** *Invarian* sebuah kelas adalah syarat pada datanya yang bernilai benar untuk setiap objek kelas itu pada setiap saat kode di luar kelas dapat melihatnya. Konstruktor menegakkan invarian; setiap fungsi anggota *public* boleh mengandalkannya di awal dan wajib memulihkannya sebelum kembali.

Untuk pesanan Profitina Laundry, invariannya adalah: nama pelanggan tidak kosong, layanannya salah satu dari `wash`, `washiron`, dan `iron`, dan berat g dalam gram memenuhi

$$1 \le g \le 50\,000 \tag{14.1}$$

serta statusnya salah satu dari empat nilai yang hanya berubah dengan urutan Diterima, Dicuci, Siap, Diambil. Bila datanya *private*, hanya fungsi milik kelas itu sendiri yang dapat mengubahnya, sehingga bila masing-masing menjaga invarian, invarian itu berlaku selamanya. Itu jaminan luar biasa dari kode yang sedikit: kita memeriksa aturan di *satu* tempat, dan aturan itu lalu berlaku di *setiap* tempat, termasuk kode yang baru akan ditulis bertahun-tahun lagi [2].

14.2.3 Dari `struct` ke `class`

Dalam C++ sebuah kelas ditulis seperti struktur, dengan dua tambahan. Anggota-anggotanya dibagi ke dalam bagian-bagian oleh *access specifier* (penentu akses), dan fungsi yang bekerja pada data dideklarasikan *di dalam* kelas sebagai *fungsi anggota* (*member function*). **Tabel 14.1** menunjukkan siapa yang boleh memakai anggota di setiap bagian.

Tabel 14.1 Tiga penentu akses dan siapa yang boleh memakai anggota di masing-masing bagian.

Akses	Fungsi anggota kelas itu sendiri	Kelas turunan (Subbab 14.6)	Selain itu
`public`	ya	ya	ya
`protected`	ya	ya	tidak
`private`	ya	tidak	tidak

Dalam C++, `struct` dan `class` hampir sama; satu-satunya perbedaan ada pada bawaannya: anggota `struct` bersifat *public* kecuali Anda menyatakan lain, anggota `class` bersifat *private*. Menurut kebiasaan, kita memakai `struct` untuk kumpulan data biasa tanpa aturan yang perlu dijaga (seperti kasir di Bab 13) dan `class` untuk tipe yang memiliki invarian.

14.3 Masalahnya, Diukur

Sebelum memperbaiki masalah, mari kita lihat wujudnya dengan mata kepala sendiri, dalam program sekecil mungkin. **Kode 14.1** adalah pesanan dari bab-bab C, ditulis sebagai struktur C++, dan sebuah `main` yang melakukan terhadapnya apa yang dilakukan kecelakaan tadi.

Kode 14.1 `struct_problem.cpp`: struktur yang terbuka menerima nilai apa pun di kolom mana pun.

```cpp
// struct_problem.cpp - Chapter 14: why an open struct is not enough.
#include <iostream>
#include <string>

struct Order {                                  // every field is public
    std::string customer;
    int grams;
    long long ratePerKg;
};

long long price(const Order& o) { return (o.grams * o.ratePerKg + 500) / 1000; }

void show(const Order& o) {
    std::cout << o.customer << ", " << o.grams << " g at Rp " << o.ratePerKg
              << "/kg: Rp " << price(o) << '\n';
}

int main() {
    Order o{"Bu Rina", 3450, 7000};
    show(o);
    o.grams = 0;                                // a slip somewhere far away...
    show(o);                                    // ...and the laundry is free
    o.grams = 3450;
    o.ratePerKg = 700;                          // one zero missing
    show(o);
    o.customer = "";                            // an order for nobody
    show(o);
    return 0;
}
```

```
Bu Rina, 3450 g at Rp 7000/kg: Rp 24150
Bu Rina, 0 g at Rp 7000/kg: Rp 0
Bu Rina, 3450 g at Rp 700/kg: Rp 2415
, 3450 g at Rp 700/kg: Rp 2415
```

Tiga penugasan, tiga pesanan yang tidak masuk akal: cucian gratis, harga sepuluh kali terlalu murah, dan pesanan untuk tak seorang pun. Kompilator tidak dapat menolong, karena tidak ada apa pun dalam program yang mengatakan bahwa nilai-nilai ini salah. **Kode 14.2** menunjukkan bahwa Python berperilaku sama dengan rekaman biasa (sebuah *dataclass*, cara cepat Python mendeklarasikan struktur).

Kode 14.2 `struct_problem.py`: dataclass Python sama terbukanya.

```python
# struct_problem.py - Chapter 14: a plain record lets anyone write anything.
from dataclasses import dataclass


@dataclass
class Order:                                    # every field is open
    customer: str
    grams: int
    rate_per_kg: int

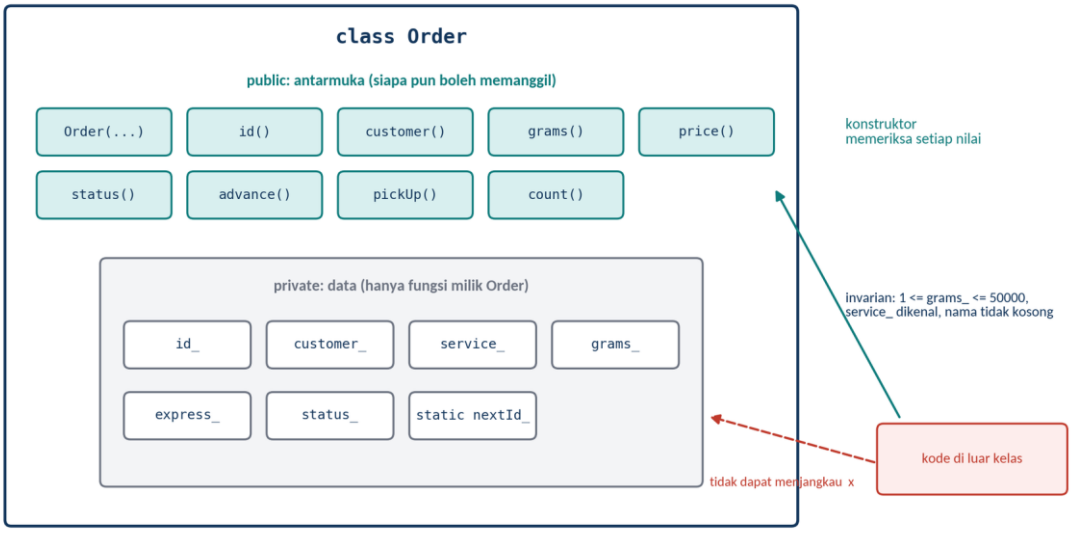

def price(o):
    return (o.grams * o.rate_per_kg + 500) // 1000


def show(o):
    print(f"{o.customer}, {o.grams} g at Rp {o.rate_per_kg}/kg: Rp {price(o)}")


o = Order("Bu Rina", 3450, 7000)
show(o)
o.grams = 0                                     # a slip somewhere far away...
show(o)                                         # ...and the laundry is free
o.grams = 3450
o.rate_per_kg = 700                             # one zero missing
show(o)
o.customer = ""                                 # an order for nobody
show(o)
```

14.4 Membangun Kelas `Order`

Gambar 14.1 adalah rancangannya. Bagian *public* adalah antarmuka: sebuah konstruktor, fungsi-fungsi yang *membaca* pesanan (`id`, `customer`, `grams`, `price`, `status`, dan seterusnya), dua fungsi yang *mengubahnya* dengan cara yang diizinkan (`advance` dan `pickUp`), dan penghitung semua pesanan yang sah. Bagian *private* menyimpan data, dan kode di luar kelas tidak dapat menjangkaunya.



Gambar 14.1 Kelas `Order`: antarmuka *public* mengelilingi data *private*; kode dari luar hanya dapat memanggil fungsi *public*, dan konstruktor memeriksa setiap nilai.

14.4.1 Header: antarmukanya

Dalam C++ sebuah kelas biasanya dipecah menjadi dua berkas. *Header* `order.h` menyatakan *apa* yang ditawarkan kelas; *implementasi* `order.cpp` menyatakan *bagaimana*. Berkas lain hanya menyertakan header-nya. **Kode 14.3** adalah header-nya.

Kode 14.3 `order.h`: antarmuka kelas `Order`.

```
// order.h - Chapter 14: the Order class of Profitina Laundry (the interface).
#ifndef ORDER_H
#define ORDER_H

#include <string>

enum class Status { Received, Washing, Ready, PickedUp };

std::string toString(Status s);

class Order {
public:
    Order(const std::string& customer, const std::string& service, int grams,
          bool express = false);

    int id() const { return id_; }
    const std::string& customer() const { return customer_; }
    const std::string& service() const { return service_; }
    int grams() const { return grams_; }
    bool express() const { return express_; }
    Status status() const { return status_; }
    long long price() const;                    // computed, never stored

    bool advance();                             // Received -> Washing -> Ready
    bool pickUp();                              // only from Ready
    static int count() { return nextId_ - 1; } // valid orders made so far

private:
    int id_;
    std::string customer_;
    std::string service_;
    int grams_;
    bool express_;
    Status status_;
    static int nextId_;                         // one counter shared by all orders
    static long long rateFor(const std::string& service);
};

#endif
```

Bacalah dari atas.

- `#ifndef ORDER_H`, `#define ORDER_H`, dan `#endif` adalah *include guard*, trik praprosesor dari Bab 11: bila sebuah berkas menyertakan `order.h` dua kali (langsung dan lewat header lain), salinan kedua dilewati.
- `enum class Status { Received, Washing, Ready, PickedUp };` adalah *enumerasi bercakupan* (*scoped enumeration*). Seperti `enum` C di Bab 10, ia memberi nama pada beberapa nilai, tetapi nama-nama itu harus ditulis bersama tipenya, `Status::Ready`, sehingga tidak bisa bertabrakan dengan nama lain, dan sebuah `Status` tidak diam-diam berubah menjadi `int`. `toString` mengubah status menjadi teks untuk dicetak.
- Konstruktor `Order(customer, service, grams, express = false)` bernama sama dengan kelasnya dan tidak punya tipe kembalian. Ia berjalan otomatis setiap kali sebuah `Order` dibuat, dan tugasnya menegakkan invarian.

- Fungsi-fungsi kecil seperti `int id() const { return id_; }` adalah *getter*: memberi akses baca ke satu kolom. Tubuhnya ditulis langsung di dalam kelas, yang membuatnya *inline*: kompilator mengganti setiap pemanggilan dengan tubuhnya, sehingga getter tidak memakan biaya apa pun saat program berjalan.
- Kata `const` sesudah daftar parameter adalah janji: *fungsi ini tidak mengubah objeknya*. Kompilator memeriksa janji itu (memberi nilai pada anggota di dalam fungsi `const` tidak akan terkompilasi), dan hanya fungsi `const` yang boleh dipanggil pada `const Order`, seperti parameter `const Order& o` milik `show` di bawah. Setiap fungsi yang hanya membaca sebaiknya `const`.
- `price()` dideklarasikan tetapi tidak disimpan. Harga selalu *dihitung* dari berat, layanan, dan tanda ekspres, sehingga tidak pernah bisa bertentangan dengannya.
- `static int count()` dan `static int nextId_` adalah *anggota statis*: milik kelas secara keseluruhan, bukan milik satu pesanan. Hanya ada tepat satu `nextId_`, dipakai bersama semua pesanan, dan `Order::count()` dapat dipanggil tanpa satu pesanan pun.
- Anggota data *private* diakhiri garis bawah (`id_`, `grams_`), kebiasaan umum yang membuat jelas, di dalam fungsi anggota, mana nama anggota dan mana parameter atau variabel lokal.

14.4.2 Implementasinya

Kode 14.4 mengisi tubuh-tubuh fungsinya.

Kode 14.4 `order.cpp`: implementasi kelas `Order`.

```cpp
// order.cpp - Chapter 14: the Order class of Profitina Laundry (implementation).
#include "order.h"

#include <stdexcept>

int Order::nextId_ = 1;

std::string toString(Status s) {
    switch (s) {
    case Status::Received: return "received";
    case Status::Washing:  return "washing";
    case Status::Ready:    return "ready";
    case Status::PickedUp: return "picked up";
    }
    return "?";
}

long long Order::rateFor(const std::string& service) {
    if (service == "wash") return 7000;
    if (service == "washiron") return 10000;
    if (service == "iron") return 5000;
    return -1;
}

Order::Order(const std::string& customer, const std::string& service, int grams,
             bool express)
    : id_(0), customer_(customer), service_(service), grams_(grams),
      express_(express), status_(Status::Received) {
    if (customer_.empty())
        throw std::invalid_argument("customer name is empty");
    if (rateFor(service_) < 0)
        throw std::invalid_argument("unknown service '" + service_ + "'");
    if (grams_ < 1 || grams_ > 50000)
        throw std::invalid_argument("weight must be 1..50000 g, not "
                                    + std::to_string(grams_));
    id_ = nextId_++;                            // only a valid order gets a number
}

long long Order::price() const {
    long long p = (grams_ * rateFor(service_) + 500) / 1000;
    if (express_) p += p / 2;
    return p;
}
```

Kode 14.4 (Lanjutan) *order.cpp: implementasi kelas Order.*

```cpp
bool Order::advance() {
    if (status_ == Status::Received) { status_ = Status::Washing; return true; }
    if (status_ == Status::Washing)  { status_ = Status::Ready;   return true; }
    return false;                               // Ready and PickedUp stay put
}

bool Order::pickUp() {
    if (status_ != Status::Ready) return false;
    status_ = Status::PickedUp;
    return true;
}
```

`int Order::nextId_ = 1;` *mendefinisikan* anggota statis: header hanya mendeklarasikannya, dan tepat satu berkas sumber harus menciptakannya. Awalan `Order::` berarti "`nextId_` milik `Order`". Awalan yang sama muncul di depan setiap fungsi anggota yang didefinisikan di luar kelas: `long long Order::price() const` adalah fungsi `price` milik `Order`.

Konstruktor diawali *member initializer list* (daftar penginisialisasi anggota), bagian sesudah titik dua: `: id_(0), customer_(customer), service_(service), ...`. Daftar ini menginisialisasi setiap anggota data langsung dengan sebuah nilai, sebelum tubuh konstruktor berjalan. Inilah cara yang dianjurkan untuk mengisi anggota dalam C++: bekerja untuk anggota yang tidak dapat diberi nilai belakangan (konstanta dan referensi), dan menghindari membuat string kosong dulu lalu menimpanya.

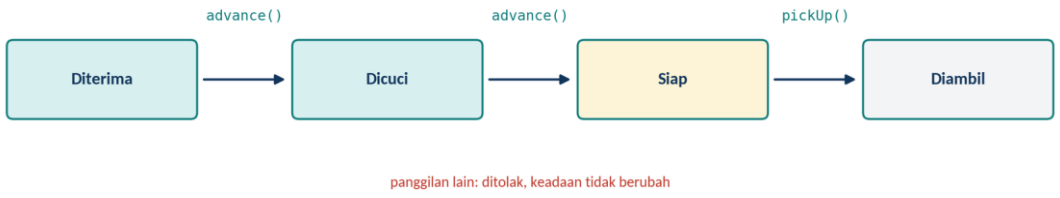
Lalu tubuh konstruktor memeriksa invarian Persamaan (14.1). Bila ada nilai yang salah, konstruktor *melempar eksepsi*: `throw std::invalid_argument("unknown service '" + service_ + "'");`. Subbab 13.5 menunjukkan eksepsi yang dilempar pustaka; di sini kita melemparnya sendiri. Melempar dari konstruktor adalah satu-satunya hal jujur yang dapat dilakukan konstruktor bila tidak mampu membangun objek yang sah: objeknya tidak pernah tercipta, dan kode yang mencoba menciptakannya melompat ke blok `catch`. Tidak ada pesanan setengah jadi dengan berat yang salah tergeletak di mana pun.

Hanya bila semua pemeriksaan lolos, pesanan mendapat nomornya: `id_ = nextId_++;` menyalin penghitung lalu menaikkannya. Karena baris ini berada *sesudah* pemeriksaan, pesanan yang ditolak tidak menghabiskan nomor, dan nomor-nomor pesanan sah adalah 1, 2, 3, … tanpa celah. Jadi banyaknya pesanan sah selalu

$$\text{count} = \text{nextId} - 1 \tag{14.2}$$

yang persis dikembalikan oleh `count()`.

`advance` dan `pickUp` adalah satu-satunya fungsi yang mengubah pesanan setelah dibuat, dan keduanya melaksanakan siklus hidup pada **Gambar 14.2**. `advance` memindahkan Diterima ke Dicuci dan Dicuci ke Siap, dan menolak yang lain dengan mengembalikan `false`; `pickUp` hanya berlaku dari Siap. Dengan mengembalikan `bool`, pemanggil dapat memutuskan apa yang akan disampaikan kepada orang di meja kasir.



Gambar 14.2 Siklus hidup sebuah pesanan: `advance()` memajukannya dua kali, `pickUp()` menyerahkannya, dan setiap panggilan lain ditolak tanpa mengubah keadaannya.

14.4.3 Memakai kelasnya

Kode 14.5 membuat pesanan, mencoba membuat dua pesanan yang tidak sah, lalu mengantar pesanan pertama melewati siklus hidupnya.

Kode 14.5 `order_main.cpp`: membuat, menolak, dan memajukan pesanan.

```cpp
// order_main.cpp - Chapter 14: using the Order class.
// Build: g++ -std=c++17 -Wall -Wextra order.cpp order_main.cpp -o order_main
#include <iostream>
#include <stdexcept>

#include "order.h"

void show(const Order& o) {
    std::cout << "#" << o.id() << " " << o.customer() << ", " << o.service() << ", "
              << o.grams() << " g" << (o.express() ? ", express" : "") << ": Rp "
              << o.price() << " [" << toString(o.status()) << "]\n";
}

int main() {
    Order a("Bu Rina", "wash", 3450);
    Order b("Pak Budi", "iron", 2000, true);
    show(a);
    show(b);

    try {
        Order bad("Mbak Sari", "wash", -500);
        show(bad);                              // never reached
    } catch (const std::invalid_argument& e) {
        std::cout << "refused: " << e.what() << '\n';
    }
    try {
        Order bad("Mas Joko", "dryclean", 1000);
        show(bad);
    } catch (const std::invalid_argument& e) {
        std::cout << "refused: " << e.what() << '\n';
    }

    std::cout << "pick up #1 now? " << (a.pickUp() ? "yes" : "no") << '\n';
    while (a.advance())
        std::cout << "#1 is now " << toString(a.status()) << '\n';
    std::cout << "pick up #1 now? " << (a.pickUp() ? "yes" : "no") << '\n';
    show(a);
    // a.grams_ = 0;   would not compile: 'int Order::grams_' is private
    std::cout << "valid orders created: " << Order::count() << '\n';
    return 0;
}
```

Program yang terdiri atas beberapa berkas sumber dikompilasi dengan menyebutkan semuanya; setiap berkas `.cpp` dikompilasi terpisah dan *linker* menggabungkan hasilnya:

```
$ g++ -std=c++17 -O2 -Wall -Wextra cpp/order.cpp cpp/order_main.cpp -o order_main
$ ./order_main
```

```
#1 Bu Rina, wash, 3450 g: Rp 24150 [received]
#2 Pak Budi, iron, 2000 g, express: Rp 15000 [received]
refused: weight must be 1..50000 g, not -500
refused: unknown service 'dryclean'
pick up #1 now? no
#1 is now washing
#1 is now ready
pick up #1 now? yes
#1 Bu Rina, wash, 3450 g: Rp 24150 [picked up]
valid orders created: 2
```

Kedua pesanan tidak sah tidak pernah tercipta: setiap konstruktornya melempar eksepsi, baris `show(bad)` dilewati, dan `catch` mencetak alasannya. Pesanan #1 tidak dapat diambil selagi masih “received”, lalu maju dua kali, berhenti maju pada “ready” (perulangan `while` selesai ketika `advance` mengembalikan `false`), dan diambil. Penghitung menunjukkan 2: pesanan yang ditolak tidak memakai nomor.

Perhatikan komentar di dekat akhir `main`. Bila Anda menghapus `//` di depan `a.grams_ = 0;`, kompilator berhenti dengan pesan bahwa `grams_` bersifat *private*. Kecelakaan di Subbab 14.1 kini menjadi mustahil: bukan sekadar kecil kemungkinannya, tetapi *mustahil*, diperiksa kompilator sebelum program sempat berjalan.

14.4.4 Kelas yang sama dalam Python

Python juga punya kelas, dan **Kode 14.6** adalah `Order` yang sama. Konstruktornya adalah metode `__init__`; `self` adalah objek yang sedang dibangun, ditulis secara eksplisit di tempat C++ memakai pointer tersembunyi `this`. Python tidak punya kata kunci `private`. Sebagai gantinya, nama yang diawali garis bawah (`_grams`) bersifat *private berdasarkan kesepakatan*: bahasanya tidak mencegah Anda menyentuhnya, tetapi setiap pemrogram Python membaca garis bawah itu sebagai "jangan dipakai dari luar" [3]. Objek `property` yang hanya-baca memberi antarmuka yang sama dengan getter C++ (`o.grams` alih-alih `o.grams()`), dan `raise ValueError(...)` berperan seperti `throw std::invalid_argument(...)`. `Enum` memberi enumerasi bercakupan, dan atribut kelas `_next_id` adalah anggota statisnya.

Kode 14.6 `order.py`: kelas `Order` dalam Python.

```
# order.py - Chapter 14: the Order class of Profitina Laundry in Python.
# A leading underscore marks a name as private by agreement; the read-only
# properties give the same public interface as order.h.
from enum import Enum


class Status(Enum):
    RECEIVED = "received"
    WASHING = "washing"
    READY = "ready"
    PICKED_UP = "picked up"


RATES = {"wash": 7000, "washiron": 10000, "iron": 5000}


class Order:
    _next_id = 1                                # one counter shared by all orders

    def __init__(self, customer, service, grams, express=False):
        if customer == "":
            raise ValueError("customer name is empty")
        if service not in RATES:
            raise ValueError(f"unknown service '{service}'")
        if not 1 <= grams <= 50000:
            raise ValueError(f"weight must be 1..50000 g, not {grams}")
        self._customer = customer
        self._service = service
        self._grams = grams
        self._express = express
        self._status = Status.RECEIVED
        self._id = Order._next_id               # only a valid order gets a number
        Order._next_id += 1

    id = property(lambda self: self._id)
    customer = property(lambda self: self._customer)
    service = property(lambda self: self._service)
    grams = property(lambda self: self._grams)
    express = property(lambda self: self._express)
    status = property(lambda self: self._status)
```

Kode 14.6 (Lanjutan) *order.py: kelas Order dalam Python.*

```
    def price(self):
        p = (self._grams * RATES[self._service] + 500) // 1000
        if self._express:
            p += p // 2
        return p

    def advance(self):
        if self._status == Status.RECEIVED:
            self._status = Status.WASHING
            return True
        if self._status == Status.WASHING:
            self._status = Status.READY
            return True
        return False                            # READY and PICKED_UP stay put

    def pick_up(self):
        if self._status != Status.READY:
            return False
        self._status = Status.PICKED_UP
        return True

    @staticmethod
    def count():
        return Order._next_id - 1
```

Kode 14.7 `order_main.py`: memakai `Order` versi Python; keluarannya persis sama dengan `order_main.cpp`.

```
# order_main.py - Chapter 14: using the Order class.
from order import Order


def show(o):
    print(f"#{o.id} {o.customer}, {o.service}, {o.grams} g"
          f"{', express' if o.express else ''}: Rp {o.price()} [{o.status.value}]")


a = Order("Bu Rina", "wash", 3450)
b = Order("Pak Budi", "iron", 2000, True)
show(a)
show(b)

for args in [("Mbak Sari", "wash", -500), ("Mas Joko", "dryclean", 1000)]:
    try:
        bad = Order(*args)
        show(bad)                               # never reached
    except ValueError as e:
        print(f"refused: {e}")

print(f"pick up #1 now? {'yes' if a.pick_up() else 'no'}")
while a.advance():
    print(f"#1 is now {a.status.value}")
print(f"pick up #1 now? {'yes' if a.pick_up() else 'no'}")
show(a)
print(f"valid orders created: {Order.count()}")
```

Kode 14.7 berbeda dari kembaran C++-nya dalam satu hal yang tampak: kedua pesanan tidak sah dicoba dalam perulangan atas list berisi tuple argumen, dan `Order(*args)` *membongkar* setiap tuple menjadi tiga argumen konstruktor. Keluarannya sama karakter demi karakter, dan hal itu diperiksa oleh program pengujian di Subbab 14.9.

14.4.5 Dua pertanyaan yang sering diajukan pemula

Mengapa menulis getter segala? Mengapa tidak membuat datanya public dan `const`? Karena antarmuka seharusnya menjanjikan *perilaku*, bukan *cara penyimpanan*. Hari ini `price()` menghitung harga dari berat; besok ia mungkin mencari di daftar harga yang berubah setiap Minggu. Pemanggil yang menulis `o.price()` tidak merasakan perubahan itu; pemanggil yang membaca anggota *public* semuanya harus ditulis ulang. Menjaga data tetap *private* juga membebaskan kelas untuk mengubah cara penyimpanannya, misalnya menyimpan berat dalam persepuluhan kilogram, tanpa merusak satu baris pun di tempat lain. Pemrogram C++ yang berpengalaman merumuskannya sebagai aturan: jadikan anggota data *private*, dan biarkan antarmuka terdiri atas fungsi [4].

Di dalam fungsi anggota, bagaimana `grams_` tahu berat pesanan *yang mana* yang dimaksud? Setiap fungsi anggota non-statis menerima satu parameter tambahan yang tersembunyi, sebuah pointer bernama `this`, yang menunjuk objek tempat fungsi itu dipanggil. Ketika `main` memanggil `a.price()`, `this` menunjuk `a`, dan `grams_` di dalam `price` berarti `this->grams_`, berat milik `a`. Anda boleh menulis `this->grams_` sendiri, dan kadang harus, misalnya bila sebuah parameter bernama sama dengan sebuah anggota. Fungsi anggota `const` menerima `this` sebagai pointer ke `const Order`, dan dari situlah kompilator tahu bahwa fungsi itu tidak boleh mengubah objeknya. Fungsi anggota statis tidak punya `this`, karena tidak dipanggil pada objek tertentu; itulah sebabnya `count()` hanya dapat memakai anggota statis seperti `nextId_`. Parameter `self` di Python adalah gagasan yang sama yang dibuat terlihat: Python mengirim objeknya secara eksplisit sebagai argumen pertama, dan Anda harus menulis `self._grams` setiap kali. Buku teks C++ menjelaskan mekanisme ini secara rinci dengan gambar-gambar memori [5]; untuk pemrograman sehari-hari cukup diketahui bahwa setiap fungsi anggota bekerja pada objek "miliknya sendiri".

14.4.6 Menelusuri pesanan yang ditolak langkah demi langkah

Ada gunanya mengikuti satu pesanan yang ditolak melewati mesin sekali saja, pelan-pelan. `main` menjalankan `Order bad("Mbak Sari", "wash", -500);` di dalam blok `try`.

1. Memori untuk `bad` dipesan di *stack*, dan konstruktor dipanggil dengan `customer = "Mbak Sari"`, `service = "wash"`, `grams = -500`, serta, dari argumen bawaan, `express = false`.
2. Daftar penginisialisasi anggota berjalan: `id_` menjadi 0, `customer_` salinan nama, `service_` salinan `"wash"`, `grams_` menjadi −500, `express_` bernilai false, `status_` bernilai `Status::Received`. Pada saat ini objek itu *melanggar* invariannya, tetapi belum ada pihak luar yang dapat melihatnya: konstruktornya belum selesai.
3. Pemeriksaan pertama, `customer_.empty()`, bernilai salah. Yang kedua, `rateFor(service_) < 0`, juga salah, karena `wash` bertarif 7.000. Yang ketiga, `grams_ < 1`, bernilai benar.
4. `throw` menciptakan objek `std::invalid_argument` yang memuat teks `weight must be 1..50000 g, not -500` dan langsung meninggalkan konstruktor. Baris `id_ = nextId_++;` tidak pernah berjalan, sehingga `nextId_` tetap 3.
5. Karena konstruktornya tidak selesai, objek itu dianggap tidak pernah ada. Anggota yang *sudah* diinisialisasi (kedua string) dihancurkan dengan benar, sehingga tidak ada memori yang bocor; tidak ada pemanggilan destruktor untuk `bad` sendiri.
6. Eksepsi itu merambat keluar ke `catch` terdekat yang cocok, yaitu yang ada di `main` untuk `const std::invalid_argument&`. Baris `show(bad);` dilewati. `e.what()` mengembalikan teks yang tersimpan, dan `main` mencetak `refused: weight must be 1..50000 g, not -500`.

Hal terpenting ada di langkah 5: dalam C++ tidak ada objek setengah jadi yang dapat disentuh bagian program lain. Konstruktor selesai dan invarian berlaku, atau objeknya tidak ada. Python memberi jaminan yang sama dalam praktik: bila `__init__` memunculkan eksepsi, ekspresi `Order(...)` tidak menghasilkan nilai apa pun, dan nama `bad` tidak pernah terikat.

14.5 Konstruktor dan Destruktor: Lahir Sah, Dibereskan Otomatis

Konstruktor berjalan ketika sebuah objek lahir. Pasangannya, *destruktor*, berjalan ketika objek itu mati. Destruktor bernama sama dengan kelas diawali tilde, `~ShiftLog()`, tanpa parameter dan tanpa tipe kembalian. Untuk variabel lokal biasa, "mati" berarti "mencapai kurung kurawal penutup blok tempat ia dideklarasikan", dan ini terjadi bagaimanapun blok itu ditinggalkan: di kurung kurawal, lewat `return`, atau karena sebuah eksepsi sedang melintas. **Kode 14.8** membuat saat-saat itu terlihat dengan catatan *shift* yang mencetak satu baris ketika *shift* seorang kasir dibuka dan ditutup.

Kode 14.8 `raii_demo.cpp`: konstruktor dan destruktor mengumumkan dirinya.

```cpp
// raii_demo.cpp - Chapter 14: constructors and destructors run by themselves.
#include <iostream>
#include <string>

class ShiftLog {
public:
    explicit ShiftLog(const std::string& cashier) : cashier_(cashier) {
        std::cout << "[open ] shift of " << cashier_ << '\n';
    }
    ~ShiftLog() {                               // runs when the object dies
        std::cout << "[close] shift of " << cashier_ << ", " << entries_
                  << " entries\n";
    }
    void record(const std::string& what) {
        ++entries_;
        std::cout << "        " << cashier_ << " #" << entries_ << ": " << what
                  << '\n';
    }

private:
    std::string cashier_;
    int entries_ = 0;
};

int main() {
    std::cout << "shop opens\n";
    {
        ShiftLog morning("Dewi");
        morning.record("order #1 received");
        morning.record("order #2 received");
    }                                           // morning's destructor runs here
    ShiftLog evening("Eko");
    evening.record("order #1 picked up");
    {
        ShiftLog helper("Fajar");               // two logs alive at once
        helper.record("ironing for order #2");
    }
    std::cout << "shop closes\n";
    return 0;                                   // evening's destructor runs now
}
```

```
shop opens
[open ] shift of Dewi
        Dewi #1: order #1 received
        Dewi #2: order #2 received
[close] shift of Dewi, 2 entries
[open ] shift of Eko
        Eko #1: order #1 picked up
[open ] shift of Fajar
        Fajar #1: ironing for order #2
[close] shift of Fajar, 1 entries
shop closes
[close] shift of Eko, 1 entries
```

Catatan Dewi ditutup pada kurung kurawal yang mengakhiri bloknya, sebelum catatan Eko dibuka. Catatan Fajar dibuka dan ditutup di dalam *shift* Eko. Catatan Eko ditutup paling akhir, setelah "shop closes", ketika `main` kembali. Objek-objek dalam cakupan yang sama dihancurkan dengan urutan *terbalik* dari penciptaannya, seperti piring yang diambil dari tumpukan.

Perilaku ini punya nama yang pada awalnya terdengar aneh: *RAII*, "*resource acquisition is initialization*" [6]. Gagasannya, setiap sumber daya (memori, berkas yang terbuka, kunci, *shift* seorang kasir) diambil di konstruktor dan dikembalikan di destruktor. Dengan begitu mustahil lupa mengembalikannya, karena bahasa yang memanggil destruktor untuk Anda. Anda sebenarnya sudah memakai RAII tanpa menyadarinya: `std::string` dan `std::vector` membebaskan memorinya di destruktor, dan `std::ifstream` menutup berkasnya di destruktor. Itulah sebabnya kasir di Bab 13 tidak punya `free`, `delete`, maupun `fclose`. Ketika menulis kelas sendiri, ikutilah aturan yang sama: bila sebuah kelas mengambil sesuatu, destruktornya mengembalikannya.

Python membebaskan memori dengan sendirinya, tetapi tidak menjanjikan *kapan* sebuah objek dihancurkan, sehingga "destruktor" Python (`__del__`) bukan tempat yang andal untuk menutup *shift*. Alat Python untuk pembersihan yang terjamin adalah pernyataan `with`: objek yang memiliki metode `__enter__` dan `__exit__` adalah *context manager*, dan `__exit__` berjalan ketika blok `with` berakhir, bagaimanapun cara berakhirnya. **Kode 14.9** mencetak baris-baris yang persis sama.

Kode 14.9 `raii_demo.py`: jaminan yang sama dengan pernyataan `with` milik Python.

```python
# raii_demo.py - Chapter 14: Python's 'with' gives the same guaranteed clean-up
# that a C++ destructor gives at the end of a block.


class ShiftLog:
    def __init__(self, cashier):
        self._cashier = cashier
        self._entries = 0
        print(f"[open ] shift of {cashier}")

    def __enter__(self):
        return self

    def __exit__(self, *exc):                  # runs when the 'with' block ends
        print(f"[close] shift of {self._cashier}, {self._entries} entries")
        return False

    def record(self, what):
        self._entries += 1
        print(f"        {self._cashier} #{self._entries}: {what}")


print("shop opens")
with ShiftLog("Dewi") as morning:
    morning.record("order #1 received")
    morning.record("order #2 received")
with ShiftLog("Eko") as evening:
    evening.record("order #1 picked up")
    with ShiftLog("Fajar") as helper:          # two logs alive at once
        helper.record("ironing for order #2")
    print("shop closes")
```

14.6 Sekilas tentang Pewarisan

Profitina Laundry tidak hanya menjual jasa per kilogram. Bed cover dan sepatu dihargai per potong, dan layanan apa pun dapat dipesan ekspres dengan tambahan 50%. Semuanya adalah *layanan*: masing-masing punya nama dan cara menghitung harga untuk suatu jumlah. Yang berbeda adalah *bagaimana* harganya dihitung. C++ mengungkapkan "beberapa jenis dari satu hal" dengan *pewarisan* (*inheritance*) [7].

> **DEFINISI.** Kelas `D` yang *diturunkan* (*derived*) dari *kelas dasar* (*base class*) `B`, ditulis `class D : public B`, memuat semua yang dimuat `B` dan boleh menambahkan anggotanya sendiri; objek `D` *adalah* objek `B` dan dapat dipakai di mana pun `B` diharapkan. Fungsi anggota yang dideklarasikan `virtual` di kelas dasar boleh *ditimpa* (*overridden*) di kelas turunan; pemanggilan lewat pointer atau referensi ke kelas dasar lalu menjalankan versi milik kelas sebenarnya dari objek itu. Fungsi virtual yang ditandai `= 0` bersifat *murni* (*pure*): kelas dasar tidak mendefinisikannya, dan setiap kelas turunan konkret wajib mendefinisikannya.

Kode 14.10 mendefinisikan kelas dasar `Service` dan tiga kelas turunan.

Kode 14.10 `services.cpp`: kelas dasar `Service` dan tiga jenis layanan.

```cpp
// services.cpp - Chapter 14: a first look at inheritance and virtual functions.
#include <iomanip>
#include <iostream>
#include <string>
#include <vector>

class Service {                                 // the base class
public:
    explicit Service(const std::string& name) : name_(name) {}
    virtual ~Service() = default;
    const std::string& name() const { return name_; }
    virtual long long price(int amount) const = 0;   // each kind decides
    virtual std::string unit() const = 0;

protected:
    std::string name_;                          // visible to derived classes
};

class PerKilo : public Service {                // a PerKilo IS-A Service
public:
    PerKilo(const std::string& name, long long ratePerKg)
        : Service(name), rate_(ratePerKg) {}
    long long price(int grams) const override {
        return (grams * rate_ + 500) / 1000;
    }
    std::string unit() const override { return "g"; }

private:
    long long rate_;
};

class PerPiece : public Service {
public:
    PerPiece(const std::string& name, long long ratePerPiece)
        : Service(name), rate_(ratePerPiece) {}
    long long price(int pieces) const override { return pieces * rate_; }
    std::string unit() const override { return "pcs"; }

private:
    long long rate_;
};
```

Kode 14.10 (Lanjutan) *services.cpp: kelas dasar Service dan tiga jenis layanan.*

```
class Express : public Service {               // wraps any service, adds 50%
public:
    explicit Express(const Service& base)
        : Service(base.name() + " express"), base_(base) {}
    long long price(int amount) const override {
        long long p = base_.price(amount);
        return p + p / 2;
    }
    std::string unit() const override { return base_.unit(); }

private:
    const Service& base_;
};

int main() {
    PerKilo wash("wash", 7000);
    PerKilo washIron("washiron", 10000);
    PerPiece bedCover("bed cover", 25000);
    Express fastWash(wash);
    Express fastCover(bedCover);

    std::vector<const Service*> menu = {&wash, &washIron, &bedCover, &fastWash,
                                        &fastCover};
    std::vector<int> amounts = {3450, 3450, 2, 3450, 2};
    for (std::size_t i = 0; i < menu.size(); ++i)   // one loop, five behaviors
        std::cout << std::left << std::setw(18) << menu[i]->name() << std::right
                  << std::setw(5) << amounts[i] << ' ' << std::left << std::setw(4)
                  << menu[i]->unit() << "Rp " << menu[i]->price(amounts[i]) << '\n';
    return 0;
}
```

```
wash               3450 g   Rp 24150
washiron           3450 g   Rp 34500
bed cover             2 pcs Rp 50000
wash express       3450 g   Rp 36225
bed cover express     2 pcs Rp 75000
```

Bagian-bagiannya, satu per satu:

- `Service` menyimpan nama dalam anggota `protected`. Menurut **Tabel 14.1**, kelas turunan boleh memakainya, tetapi bagian program lainnya tidak. `price` dan `unit` adalah fungsi virtual murni, sehingga `Service` merupakan kelas *abstrak*: Anda tidak dapat membuat `Service` polos, hanya salah satu jenis konkretnya.
- `virtual ~Service() = default;` memberi kelas dasar sebuah destruktor virtual. Aturannya sederhana: kelas yang dimaksudkan sebagai kelas dasar mendapat destruktor virtual, agar menghapus objek turunan lewat pointer dasar menjalankan destruktor yang benar.
- `PerKilo` dan `PerPiece` diturunkan dengan `: public Service`. **Konstruktor tidak diwariskan**: setiap kelas turunan menulis konstruktornya sendiri, dan membangun bagian dasarnya lebih dulu dengan memanggil konstruktor dasar di daftar penginisialisasinya, `: Service(name), rate_(ratePerKg)`. Masing-masing menimpa `price` dan `unit`; kata kunci `override` meminta kompilator memeriksa bahwa fungsi virtual yang cocok memang ada di kelas dasar, sehingga salah eja nama dan tipe parameter yang keliru tertangkap.
- `Express` adalah layanan yang *membungkus* layanan lain: harganya adalah harga layanan lain itu ditambah setengahnya, aturan ekspres dari Subbab 13.7 sekali lagi. Ia memegang `const Service&`, referensi ke layanan apa saja, bahkan ke `Express` lain.
- Di `main`, `menu` adalah `std::vector<const Service*>`: pointer ke kelas dasar yang menunjuk objek dari empat kelas berbeda. Perulangannya memanggil `menu[i]->price(amounts[i])` tanpa tahu sedang berbicara dengan kelas yang mana, dan setiap objek menjawab dengan caranya

sendiri. Inilah *polimorfisme* ("banyak bentuk"): satu pemanggilan, beberapa perilaku, dipilih pada waktu jalan oleh kelas sebenarnya dari objek.

Bagaimana program tahu, pada waktu jalan, `price` yang mana yang harus dipanggil? Mekanisme yang lazim cukup sederhana untuk dibayangkan. Untuk setiap kelas yang memiliki fungsi virtual, kompilator membuat satu tabel kecil, *tabel virtual* (*virtual table*), yang memuat alamat versi fungsi-fungsi virtual milik kelas itu: tabel `PerKilo` menunjuk `PerKilo::price` dan `PerKilo::unit`, tabel `Express` menunjuk `Express::price` dan `Express::unit`. Setiap objek dari kelas seperti itu membawa satu pointer tersembunyi ke tabel kelasnya, yang dipasang oleh konstruktor. Pemanggilan `menu[i]->price(amounts[i])` lalu menjalankan tiga langkah kecil: ikuti pointer tersembunyi objek ke tabelnya, baca alamat yang tersimpan di slot `price`, dan panggil fungsi di alamat itu. Telusuri baris keempat keluaran: `menu[3]` menunjuk `fastWash`, sebuah `Express`; tabelnya mengarah ke `Express::price`, yang memanggil `base_.price(3450)`; `base_` merujuk `wash`, sebuah `PerKilo`, yang tabelnya mengarah ke `PerKilo::price`, yang mengembalikan $(3450 \times 7000 + 500)/1000 = 24{,}150$; kembali di `Express::price`, $24{,}150 + 12{,}075 = 36{,}225$ dikembalikan dan dicetak. Biaya pemanggilan virtual adalah satu kali pencarian memori tambahan, dan karena itulah C++ membiarkan Anda memilih, fungsi demi fungsi, apakah mau membayarnya: fungsi yang tidak `virtual` dipanggil langsung, seperti di C.

Ini juga alasan kelas dasar memerlukan `virtual` pada destruktornya. Tanpanya, `delete` lewat `Service*` akan memanggil destruktor `Service` secara langsung, tanpa pernah melihat tabel, dan bagian turunan objek itu tidak akan dibereskan.

Pewarisan adalah pokok bahasan yang luas, dan pemrograman berorientasi objek biasanya menjadi langkah berikutnya setelah buku ini. Untuk buku ini Anda memerlukan persis apa yang ditunjukkan subbab ini: membaca `class D : public B`, menulis konstruktor yang memanggil konstruktor dasar, memakai anggota `protected`, dan menimpa fungsi virtual. Bab 16 (Soal 16.1) menerapkan keempat keterampilan itu pada bidak catur.

Kode 14.11 adalah versi Python-nya. `class PerKilo(Service):` menurunkan sebuah kelas; `super().__init__(name)` memanggil konstruktor dasar; modul `abc` (*abstract base classes*) dengan `@abstractmethod` menjadikan `Service` abstrak. Dalam Python *setiap* metode berperilaku seperti fungsi virtual C++, jadi tidak diperlukan kata kunci.

Kode 14.11 `services.py`: hierarki yang sama dalam Python.

```
# services.py - Chapter 14: inheritance in Python; every method is "virtual".
from abc import ABC, abstractmethod


class Service(ABC):                             # the base class
    def __init__(self, name):
        self._name = name

    def name(self):
        return self._name

    @abstractmethod
    def price(self, amount): ...

    @abstractmethod
    def unit(self): ...


class PerKilo(Service):                         # a PerKilo IS-A Service
    def __init__(self, name, rate_per_kg):
        super().__init__(name)
        self._rate = rate_per_kg

    def price(self, grams):
        return (grams * self._rate + 500) // 1000

    def unit(self):
        return "g"


class PerPiece(Service):
    def __init__(self, name, rate_per_piece):
        super().__init__(name)
        self._rate = rate_per_piece

    def price(self, pieces):
        return pieces * self._rate

    def unit(self):
        return "pcs"
```

Kode 14.11 (Lanjutan) *services.py: hierarki yang sama dalam Python.*

```
class Express(Service):                         # wraps any service, adds 50%
    def __init__(self, base):
        super().__init__(base.name() + " express")
        self._base = base

    def price(self, amount):
        p = self._base.price(amount)
        return p + p // 2

    def unit(self):
        return self._base.unit()


wash = PerKilo("wash", 7000)
wash_iron = PerKilo("washiron", 10000)
bed_cover = PerPiece("bed cover", 25000)
menu = [wash, wash_iron, bed_cover, Express(wash), Express(bed_cover)]
amounts = [3450, 3450, 2, 3450, 2]
for service, amount in zip(menu, amounts):      # one loop, five behaviors
    print(f"{service.name():<18}{amount:>5} {service.unit():<4}"
          f"Rp {service.price(amount)}")
```

14.7 Membangunnya: Toko

Sebuah pesanan saja baru separuh kasir. Separuh lainnya adalah buku semua pesanan hari itu: menerima pesanan baru, mencari pesanan menurut nomornya, memajukannya, menyerahkannya, menampilkan semuanya, dan melaporkan uangnya. Kita jadikan itu kelas kedua, `Shop`, yang *memiliki* sebuah vektor pesanan. Bila pewarisan mengungkapkan hubungan "adalah" (`PerKilo` adalah `Service`), anggota data mengungkapkan hubungan "memiliki" (`Shop` memiliki pesanan-pesanan), dan sebagian besar kelas dalam program nyata dibangun dengan cara kedua ini, dari kelas-kelas lain.

14.7.1 Kelas `Shop`

Kode 14.12 adalah antarmukanya. Setiap operasi mengembalikan satu baris teks yang ditampilkan program kepada orang di meja kasir; dengan begitu `Shop` bebas dari keputusan apa pun tentang *di mana* teks dicetak.

Kode 14.12 `shop.h`: antarmuka kelas `Shop`.

```
// shop.h - Chapter 14: the Shop class keeps all orders of one day.
#ifndef SHOP_H
#define SHOP_H

#include <string>
#include <vector>

#include "order.h"

class Shop {
public:
    explicit Shop(const std::string& name) : name_(name) {}

    // Each operation returns one line of text for the cashier's screen.
    std::string receive(const std::string& customer, const std::string& service,
                        int grams, bool express);
    std::string advance(int id);
    std::string pickUp(int id);
    std::string list() const;
    std::string report() const;

private:
    Order* find(int id);                        // nullptr when there is no such id
    std::string name_;
    std::vector<Order> orders_;
};

#endif
```

Fungsi bantu `find` bersifat *private*: kode lain tidak berkepentingan memegang pointer ke dalam vektor milik toko. Ia mengembalikan `nullptr` bila tidak ada pesanan bernomor itu, persis keadaan ketika pointer lebih cocok daripada referensi (Subbab 13.6): "tidak ada pesanan seperti itu" adalah jawaban yang wajar. `list` dan `report` bersifat `const`: melihat isi toko tidak mengubahnya.

Kode 14.13 adalah implementasinya.

Kode 14.13 `shop.cpp`: implementasi kelas `Shop`.

```cpp
// shop.cpp - Chapter 14: the Shop class (implementation).
#include "shop.h"

#include <stdexcept>

Order* Shop::find(int id) {
    for (Order& o : orders_)
        if (o.id() == id) return &o;
    return nullptr;
}

std::string Shop::receive(const std::string& customer, const std::string& service,
                          int grams, bool express) {
    try {
        orders_.push_back(Order(customer, service, grams, express));
    } catch (const std::invalid_argument& e) {
        return std::string("refused: ") + e.what();
    }
    const Order& o = orders_.back();
    return "order #" + std::to_string(o.id()) + " for " + o.customer() + ", Rp "
           + std::to_string(o.price());
}

std::string Shop::advance(int id) {
    Order* o = find(id);
    if (o == nullptr) return "order #" + std::to_string(id) + " does not exist";
    std::string before = toString(o->status());
    if (!o->advance())
        return "order #" + std::to_string(id) + " cannot advance (" + before + ")";
    return "order #" + std::to_string(id) + ": " + before + " -> "
           + toString(o->status());
}

std::string Shop::pickUp(int id) {
    Order* o = find(id);
    if (o == nullptr) return "order #" + std::to_string(id) + " does not exist";
    if (!o->pickUp())
        return "order #" + std::to_string(id) + " is not ready ("
               + toString(o->status()) + ")";
    return "order #" + std::to_string(id) + " picked up by " + o->customer()
           + ", pay Rp " + std::to_string(o->price());
}
```

Kode 14.13 (Lanjutan) *shop.cpp: implementasi kelas Shop.*

```
std::string Shop::list() const {
    std::string text = name_ + ", " + std::to_string(orders_.size()) + " orders";
    for (const Order& o : orders_)
        text += "\n  #" + std::to_string(o.id()) + " " + o.customer() + " ("
                + o.service() + ", " + std::to_string(o.grams()) + " g): "
                + toString(o.status());
    return text;
}

std::string Shop::report() const {
    long long paid = 0, open = 0;
    int done = 0;
    for (const Order& o : orders_) {
        if (o.status() == Status::PickedUp) {
            paid += o.price();
            ++done;
        } else {
            open += o.price();
        }
    }
    return "picked up " + std::to_string(done) + " of "
           + std::to_string(orders_.size()) + ", paid Rp " + std::to_string(paid)
           + ", still in the shop Rp " + std::to_string(open);
}
```

Perhatikan betapa sedikit yang harus diperiksa `Shop`, dan betapa hal itu justru membuatnya lebih dapat dipercaya. `receive` sama sekali tidak memeriksa berat atau layanan: ia cukup mencoba membangun sebuah `Order`, dan bila konstruktor pesanan melempar eksepsi, `receive` menangkapnya dan mengubah pesannya menjadi baris “refused”. Aturannya tinggal di satu tempat, `Order`, dan `Shop` memercayainya. `advance` dan `pickUp` mencari pesanannya, mengingat statusnya sebagai teks untuk pesan, lalu memanggil fungsi milik pesanan itu sendiri; bila hasilnya `false`, pesanan itu menolak, dan toko melaporkan alasannya. `report` menelusuri pesanan dan memisahkan uang yang sudah dibayar (pesanan yang sudah diambil) dari uang yang masih tertahan di toko.

14.7.2 Program di sekelilingnya

Program pada **Kode 14.14** membaca perintah, satu per baris, dan mencetak setiap perintah beserta jawaban toko. Perintahnya adalah `new customer;service;grams;express`, `advance N`, `pickup N`, `list`, dan `report`.

Kode 14.14 `shop_main.cpp`: membaca perintah dan meneruskannya ke toko.

```cpp
// shop_main.cpp - Chapter 14: runs a day of commands against a Shop.
// Build: g++ -std=c++17 -Wall -Wextra order.cpp shop.cpp shop_main.cpp -o shop
#include <iostream>
#include <sstream>
#include <stdexcept>
#include <string>
#include <vector>

#include "shop.h"

std::vector<std::string> split(const std::string& line, char sep) {
    std::vector<std::string> parts;
    std::string part;
    std::istringstream in(line);
    while (std::getline(in, part, sep)) parts.push_back(part);
    return parts;
}

std::string execute(Shop& shop, const std::string& line) {
    std::size_t space = line.find(' ');
    std::string command = line.substr(0, space);
    std::string rest = space == std::string::npos ? "" : line.substr(space + 1);
    try {
        if (command == "new") {
            std::vector<std::string> f = split(rest, ';');
            if (f.size() != 4) return "new needs customer;service;grams;express";
            return shop.receive(f[0], f[1], std::stoi(f[2]), f[3] == "yes");
        }
        if (command == "advance") return shop.advance(std::stoi(rest));
        if (command == "pickup") return shop.pickUp(std::stoi(rest));
    } catch (const std::logic_error&) {        // stoi failed: not a number
        return "not a number in '" + line + "'";
    }
    if (command == "list") return shop.list();
    if (command == "report") return shop.report();
    return "unknown command '" + command + "'";
}
```

Kode 14.14 (Lanjutan) *shop_main.cpp: membaca perintah dan meneruskannya ke toko.*

```cpp
int main() {
    Shop shop("Profitina Laundry");
    std::string line;
    while (std::getline(std::cin, line)) {
        if (line.empty() || line[0] == '#') continue;
        std::cout << "> " << line << '\n' << execute(shop, line) << '\n';
    }
    return 0;
}
```

Dua rincian di sini baru. `split` memakai `std::istringstream`, stream yang membaca dari string alih-alih dari papan ketik, bersama `std::getline(in, part, sep)` tiga argumen, yang membaca sampai pemisah pilihan Anda. Dan `std::stoi` (“*string to int*”) mengubah teks menjadi bilangan dan melempar `std::invalid_argument` bila teksnya tidak diawali bilangan; karena kelas eksepsi itu diturunkan dari `std::logic_error`, satu `catch (const std::logic_error&)` juga menangkap galat di luar rentang yang dilempar `stoi` untuk bilangan raksasa. Di sini pewarisan membantu kita menangkap satu keluarga galat sekaligus.

Versi Python-nya (**Kode 14.15** dan **Kode 14.16**) berstruktur sama, metode demi metode.

Kode 14.15 `shop.py`: kelas `Shop` dalam Python.

```python
# shop.py - Chapter 14: the Shop class keeps all orders of one day.
from order import Order, Status


class Shop:
    def __init__(self, name):
        self._name = name
        self._orders = []

    def _find(self, order_id):
        for o in self._orders:
            if o.id == order_id:
                return o
        return None

    def receive(self, customer, service, grams, express):
        try:
            o = Order(customer, service, grams, express)
        except ValueError as e:
            return f"refused: {e}"
        self._orders.append(o)
        return f"order #{o.id} for {o.customer}, Rp {o.price()}"

    def advance(self, order_id):
        o = self._find(order_id)
        if o is None:
            return f"order #{order_id} does not exist"
        before = o.status.value
        if not o.advance():
            return f"order #{order_id} cannot advance ({before})"
        return f"order #{order_id}: {before} -> {o.status.value}"

    def pick_up(self, order_id):
        o = self._find(order_id)
        if o is None:
            return f"order #{order_id} does not exist"
        if not o.pick_up():
            return f"order #{order_id} is not ready ({o.status.value})"
        return f"order #{order_id} picked up by {o.customer}, pay Rp {o.price()}"
```

Kode 14.15 (Lanjutan) *shop.py: kelas Shop dalam Python.*

```python
    def list(self):
        text = f"{self._name}, {len(self._orders)} orders"
        for o in self._orders:
            text += (f"\n  #{o.id} {o.customer} ({o.service}, {o.grams} g): "
                     f"{o.status.value}")
        return text

    def report(self):
        done = [o for o in self._orders if o.status == Status.PICKED_UP]
        paid = sum(o.price() for o in done)
        still = sum(o.price() for o in self._orders) - paid
        return (f"picked up {len(done)} of {len(self._orders)}, paid Rp {paid}, "
                f"still in the shop Rp {still}")
```

Kode 14.16 `shop_main.py`: perulangan perintah dalam Python.

```python
# shop_main.py - Chapter 14: runs a day of commands against a Shop.
import sys

from shop import Shop


def execute(shop, line):
    command, _, rest = line.partition(" ")
    try:
        if command == "new":
            f = rest.split(";")
            if len(f) != 4:
                return "new needs customer;service;grams;express"
            return shop.receive(f[0], f[1], int(f[2]), f[3] == "yes")
        if command == "advance":
            return shop.advance(int(rest))
        if command == "pickup":
            return shop.pick_up(int(rest))
    except ValueError:                          # int() failed: not a number
        return f"not a number in '{line}'"
    if command == "list":
        return shop.list()
    if command == "report":
        return shop.report()
    return f"unknown command '{command}'"


shop = Shop("Profitina Laundry")
for line in sys.stdin.read().splitlines():
    if line == "" or line.startswith("#"):
        continue
    print(f"> {line}\n{execute(shop, line)}")
```

14.8 Menjalankannya

Berkas `input/day.txt` adalah satu hari di meja kasir, lengkap dengan kesalahannya: pesanan dengan berat negatif, pengambilan sebelum cucian siap, nomor pesanan yang tidak ada, kata di tempat yang seharusnya bilangan, dan perintah yang tidak dikenal.

```
$ g++ -std=c++17 -O2 -Wall -Wextra cpp/order.cpp cpp/shop.cpp cpp/shop_main.cpp -o shop
$ ./shop < input/day.txt
$ python3 python/shop_main.py < input/day.txt
```

```
> new Bu Rina;wash;3450;no
order #1 for Bu Rina, Rp 24150
> new Pak Budi;iron;2000;yes
order #2 for Pak Budi, Rp 15000
> new Mbak Sari;washiron;-500;no
refused: weight must be 1..50000 g, not -500
> new Mas Joko;washiron;5200;no
order #3 for Mas Joko, Rp 52000
> advance 1
order #1: received -> washing
> advance 1
order #1: washing -> ready
> pickup 2
order #2 is not ready (received)
> advance 2
order #2: received -> washing
> advance 2
order #2: washing -> ready
> pickup 2
order #2 picked up by Pak Budi, pay Rp 15000
> advance 2
order #2 cannot advance (picked up)
> pickup 1
order #1 picked up by Bu Rina, pay Rp 24150
> advance 9
order #9 does not exist
> pickup two
not a number in 'pickup two'
> refund 1
unknown command 'refund'
> list
Profitina Laundry, 3 orders
  #1 Bu Rina (wash, 3450 g): picked up
  #2 Pak Budi (iron, 2000 g): picked up
  #3 Mas Joko (washiron, 5200 g): received
> report
picked up 2 of 3, paid Rp 39150, still in the shop Rp 52000
```

Ikutilah riwayat pesanan #2 pada keluaran. Pesanan Pak Budi diterima; pengambilan terlalu dini ditolak karena statusnya masih “received”; dua perintah `advance` membawanya ke “ready”; pengambilan berhasil dan meminta Rp15.000; satu `advance` lagi ditolak karena pesanan yang sudah diambil tidak ke mana-mana. Setiap langkah persis diagram **Gambar 14.2**. Pesanan Mbak Sari dengan −500 gram tidak pernah mendapat nomor, sehingga pesanan Mas Joko menjadi #3, seperti dijanjikan Persamaan (14.2). Laporan membagi Rp91.150 dari ketiga pesanan menjadi Rp39.150 yang sudah dibayar dan Rp52.000 untuk cucian yang masih di rak.

14.9 Mengujinya

Program pengujian dari Subbab 13.11 juga menjalankan bab ini; daftar kasusnya menyebutkan berkas-berkas sumber program yang terdiri atas beberapa berkas:

```
# name | C++ sources | Python script | stdin | arguments
struct_problem | struct_problem.cpp | struct_problem.py | - | -
order_main | order.cpp order_main.cpp | order_main.py | - | -
raii_demo | raii_demo.cpp | raii_demo.py | - | -
services | services.cpp | services.py | - | -
shop | order.cpp shop.cpp shop_main.cpp | shop_main.py | input/day.txt | -
$ python3 tests/run_tests.py --sanitize
```

```
struct_problem             C++ ok   Python ok
order_main                 C++ ok   Python ok
raii_demo                  C++ ok   Python ok
services                   C++ ok   Python ok
shop                       C++ ok   Python ok
all programs agree
```

Sebuah kelas mengundang jenis uji kedua yang lebih halus: *menguji invariannya secara langsung*. Untuk `Order`, tuliskan nilai-nilai batas Persamaan (14.1) dan cobalah satu per satu: 0 g dan 50.001 g harus ditolak, 1 g dan 50.000 g harus diterima. Cobalah setiap transisi pada **Gambar 14.2**, yang diizinkan maupun yang terlarang: ada 4 keadaan dan 2 operasi, jadi 8 kasus, dan tepat 3 di antaranya berhasil. Soal 2 meminta Anda menulis uji-uji ini sebagai program.

14.10 Membuatnya Lebih Baik

Kedua kelas dapat berkembang ke beberapa arah yang berguna tanpa mengubah cara memakainya.

- **Uang sebagai tipe tersendiri.** Harga masih berupa `long long` yang bisa tidak sengaja dijumlahkan dengan berat. Bab 15 memperkenalkan kelas `Rupiah` dengan operatornya sendiri, sehingga `price + grams` tidak lagi terkompilasi.
- **Mencetak dengan `<<`.** `show` di `order_main.cpp` adalah fungsi bebas dengan nama buatan. Bab 15 menunjukkan cara mengajari `std::cout << order` untuk mencetak pesanan, sehingga pesanan tercetak seperti bilangan.
- **Menyimpan hari itu.** Saat tutup, pesanan-pesanan lenyap. Dengan pemrosesan berkas dari Bab 11, `Shop` dapat menyimpan pesanannya ke berkas di destruktornya dan memuatnya di konstruktornya, pemakaian RAII yang sempurna. Soal 5 meminta separuh bagian penyimpanannya.
- **Mencari lebih cepat.** `find` adalah pencarian linear, cukup untuk seratus pesanan sehari. Untuk toko besar, `std::map<int, Order>` menemukan pesanan menurut nomornya dalam $O(\log n)$ langkah; Bab 15 memakai `std::map` untuk pelanggan dan layanan.

14.11 Memakainya Sungguhan

Dalam dunia usaha kecil di sekitar kita, pola ini bahkan lebih sering muncul daripada yang disadari para pemiliknya sendiri. Pola bab ini, kelas kecil yang menjaga aturannya sendiri ditambah kelas yang lebih besar yang mengelola banyak di antaranya, muncul di setiap sistem bisnis yang akan Anda temui. Antrean klinik punya pasien yang statusnya Terdaftar, Menunggu, Bersama dokter, Selesai; layanan antar warung punya pesanan yang Dipesan, Dimasak, Dalam perjalanan, Diterima; perpustakaan punya buku yang Di rak, Dipinjam, Terlambat, Dikembalikan. Untuk masing-masing, ajukan tiga pertanyaan, dan kelasnya nyaris menulis dirinya sendiri:

1. *Apa yang membuat satu butir sah?* Syarat-syarat itu masuk ke konstruktor, seperti Persamaan (14.1).
2. *Dengan urutan apa keadaannya boleh berubah?* Gambarlah diagramnya, seperti **Gambar 14.2**, dan tulis satu fungsi anggota untuk setiap anak panah; setiap fungsi menolak apa yang tidak diizinkan diagram.
3. *Apa yang ingin diketahui pemilik di akhir hari?* Itulah `report` milik kelas pengelola.

Ambil warung katering Mbak Wulan sebagai contoh yang dikerjakan. Sebuah pesanan sah bila punya pelanggan, paling sedikit satu porsi dan paling banyak 200 (dapurnya tidak sanggup

memasak lebih dari itu dalam satu pagi), dan waktu antar antara 06.00 dan 20.00: tiga syarat untuk konstruktor. Keadaannya Dipesan, Dimasak, Dalam perjalanan, Diterima, dan satu anak panah tambahan, Dipesan ke Dibatalkan, karena pelanggan hanya membatalkan sebelum mulai dimasak: paling banyak lima fungsi anggota, masing-masing menolak apa yang tidak tampak di diagram. Di akhir hari ia ingin tahu porsi yang dimasak, uang yang diterima, dan uang yang masih terutang oleh pelanggan kantor yang membayar di akhir bulan: itulah `report`. Ditulis dengan cara ini, programnya tidak akan mengantar pesanan yang belum pernah dimasak atau memasak pesanan yang sudah dibatalkan, seterburu-buru apa pun orang yang menerima telepon.

Untuk klinik, program seperti ini hanya menjaga antrean tetap tertib. Menentukan siapa yang mendesak adalah urusan perawat dan dokter; program antrean seharusnya memudahkan mereka memajukan seorang pasien, bukan memutuskannya untuk mereka.

14.12 Kesalahan yang Sering Terjadi, dan Cara Menghindarinya

- **Data public "untuk sementara saja".** Menjadikan anggota *public* demi memperbaiki masalah dengan cepat membuat setiap baris program dapat melanggar invarian lagi. Tambahkan fungsi anggota yang melakukan perubahan yang diperlukan *dan menjaga aturannya*.
- **Lupa `const` pada fungsi pembaca.** Getter tanpa `const` tidak dapat dipanggil lewat `const Order&`, dan pesan galat kompilator muncul jauh dari penyebab sebenarnya. Tandai setiap fungsi yang tidak mengubah objek dengan `const` sejak awal.
- **Memeriksa invarian di luar kelas.** Bila `Shop::receive` juga memeriksa berat, kedua pemeriksaan itu akan bergeser saat pertama kali seseorang mengubah salah satunya. Periksa sekali, di konstruktor, dan biarkan kode lain mengandalkannya.
- **Memberi nilai di tubuh konstruktor alih-alih menginisialisasi.** Anggota referensi atau anggota `const` *wajib* diisi di daftar penginisialisasi; untuk yang lain, daftar itu tetap cara yang bersih dan efisien. Inisialisasikan anggota dengan urutan yang sama seperti urutan deklarasinya di kelas, karena itulah urutan C++ menginisialisasinya, apa pun urutan tulisan daftarnya (`-Wall` memperingatkan bila urutannya berbeda).
- **Mendefinisikan anggota statis dua kali atau tidak sama sekali.** `static int nextId_;` di dalam kelas hanya mendeklarasikannya; tepat satu berkas `.cpp` harus memuat `int Order::nextId_ = 1;`. Tanpanya *linker* melaporkan "undefined reference"; dengan dua definisi ia melaporkan "multiple definition".
- **Hanya mengompilasi satu berkas dari program multiberkas.** `g++ order_main.cpp` saja berakhir dengan "undefined reference to `Order::Order(...)`". Sebutkan setiap berkas `.cpp` pada baris perintah (atau biarkan alat *build* yang melakukannya).
- **Kelas dasar tanpa destruktor virtual.** Menghapus objek turunan lewat pointer dasar lalu melewatkan destruktor bagian turunannya. Beri setiap kelas dasar `virtual ~Base() = default;`.
- **Mengandalkan `__del__` Python.** Metode ini tidak dipanggil pada saat yang terjamin. Pakailah `with` untuk apa pun yang harus ditutup atau dilepaskan.

14.13 Ringkasan Bab

- Abstraksi data memisahkan antarmuka sebuah tipe (apa yang boleh Anda lakukan dengannya) dari implementasinya (bagaimana ia disimpan dan dihitung); tipe yang didefinisikan oleh operasinya adalah tipe data abstrak.

- Kelas menyatukan data dan fungsi anggota. Data `private` hanya dapat diubah oleh fungsi milik kelas itu, sehingga aturan yang dijaga fungsi-fungsi itu, yaitu invarian, berlaku untuk setiap objek setiap saat; anggota `public` membentuk antarmuka; anggota `protected` dibagi dengan kelas turunan.
- Konstruktor menegakkan invarian dan boleh melempar eksepsi untuk menolak nilai tidak sah, sehingga tidak pernah ada objek yang tidak sah; daftar penginisialisasi anggota mengisi anggota secara langsung. Destruktor berjalan otomatis di akhir hidup sebuah objek, yang memungkinkan RAII: sumber daya yang diambil di konstruktor selalu dilepaskan di destruktor.
- Fungsi anggota `const` berjanji tidak mengubah objek dan hanya fungsi seperti itulah yang dapat dipanggil pada objek `const`. Anggota statis adalah milik kelas secara keseluruhan.
- Sebuah kelas biasanya dipecah menjadi header (antarmuka) dan berkas sumber (implementasi); program yang terdiri atas beberapa berkas sumber dikompilasi dengan menyebutkan semuanya.
- Pewarisan (`class D : public B`) mengungkapkan hubungan “adalah”; fungsi virtual membuat satu pemanggilan berperilaku berbeda untuk setiap kelas turunan; fungsi virtual murni menjadikan kelas dasar abstrak. Konstruktor tidak diwariskan; setiap konstruktor turunan memanggil konstruktor dasarnya.
- Dalam Python, `__init__` adalah konstruktor, garis bawah di depan menandai data *private* berdasarkan kesepakatan, *property* memberi akses hanya-baca, `with` memberi pembersihan yang terjamin, dan setiap metode bersifat virtual. Kelas `Order` dan `Shop` berperilaku identik dalam kedua bahasa.

14.14 Soal Latihan

1. Tambahkan fungsi anggota `bool changeWeight(int grams)` ke `Order` yang membetulkan berat sebuah pesanan, tetapi hanya selama statusnya masih `Received`, dan hanya ke nilai yang menjaga invarian. Mengapa ini aman padahal pilihan menu “ubah berat” yang lama di Subbab 14.1 tidak?
2. Tulislah program uji `order_test.cpp` (dan `order_test.py`) yang memeriksa empat berat batas 0, 1, 50.000, dan 50.001 serta kedelapan kombinasi keadaan dan operasi pada **Gambar 14.2**, mencetak satu baris per pemeriksaan dan jumlah akhir pemeriksaan yang lolos.
3. Tambahkan status kelima, `Cancelled`, ke `Order`: pesanan hanya boleh dibatalkan selama masih `Received`, dan pesanan yang dibatalkan tidak dapat maju atau diambil. Perbarui diagramnya, `toString`, dan `Shop::report` (pesanan yang dibatalkan tidak dihitung sebagai sudah dibayar maupun sebagai masih di toko).
4. Tambahkan kelas turunan `Minimum` ke `services.cpp` yang membungkus layanan per kilo dan menarik biaya paling sedikit seharga 3 kg: pelanggan dengan cucian 1,2 kg membayar untuk 3 kg. Berapa tagihan `Minimum(wash)` untuk 1.200 g dan untuk 3.450 g?
5. Beri `Shop` fungsi anggota `save(const std::string& path) const` yang menulis satu baris per pesanan (`id;customer;service;grams;express;status`) ke berkas dengan `std::ofstream`. Mengapa `save` sebaiknya `const`?
6. Di `order.cpp`, apa yang akan salah bila baris `id_ = nextId_++;` dipindahkan ke awal tubuh konstruktor, sebelum pemeriksaan? Jelaskan keluaran `shop_main` untuk `input/day.txt` dalam kasus itu.

14.15 Penyelesaian Soal Latihan

1. Deklarasikan `bool changeWeight(int grams);` di bagian *public* dan definisikan: bila `status_ != Status::Received` atau `grams < 1` atau `grams > 50000`, kembalikan `false`; selain itu isi `grams_ = grams` dan kembalikan `true`. Ini aman karena merupakan fungsi anggota: ia dapat melihat berat yang *private*, dan menjaga kedua bagian invarian, yaitu rentang Persamaan (14.1) dan aturan bahwa berat sudah tetap begitu pencucian dimulai. Pilihan menu lama menulis bilangan apa saja langsung ke struktur, tanpa aturan sama sekali. Dalam Python, `def change_weight(self, grams)` melakukan hal yang sama dengan `self._grams`.

2. Untuk berat, fungsi bantu kecil `bool accepted(int g)` mencoba `Order("Test", "wash", g)` di dalam `try` dan mengembalikan `false` di `catch`; jawaban yang diharapkan adalah `false, true, true, false`. Untuk transisi, tulislah fungsi yang membuat pesanan baru dan membawanya ke masing-masing dari empat keadaan (0, 1, atau 2 kali `advance`, ditambah `pickUp` untuk yang keempat), lalu mencoba `advance` atau `pickUp` dan membandingkan hasilnya dengan tabel: `advance` berhasil dari Received dan Washing, `pickUp` hanya dari Ready, sehingga tepat 3 dari 8 kasus mengembalikan `true`. Programnya mencetak, misalnya, `advance from ready: false (expected false) ok` dan di akhir `12 of 12 checks passed`.

3. Tambahkan `Cancelled` ke `enum class` dan ke `toString`. Tambahkan `bool cancel()` yang hanya berhasil dari `Received`. `advance` dan `pickUp` sudah menolak setiap keadaan yang tidak mereka sebutkan, sehingga tidak perlu diubah: inilah manfaat menulis “izinkan yang ini” alih-alih “larang yang itu”. Di `Shop`, tambahkan perintah `cancel N` dan, di `report`, lewati pesanan yang dibatalkan saat menjumlahkan uang; hitung pesanan itu tersendiri di baris laporan.

4. `class Minimum : public Service` memegang `const Service& base_` seperti `Express`, dan `price(int grams)`-nya mengembalikan `base_.price(grams < 3000 ? 3000 : grams)`. Untuk 1.200 g `wash`, tagihannya seharga 3.000 g: $(3000 \times 7000 + 500)/1000 = 21{,}000$. Untuk 3.450 g batas minimum tidak berlaku, dan harganya tetap 24,150.

5. `std::ofstream out(path);`, lalu satu baris `out << o.id() << ';' << o.customer() << ...` per pesanan, dan berkasnya ditutup oleh destruktor `out` di akhir fungsi. `save` harus `const` karena menyimpan tidak mengubah toko; mendeklarasikannya `const` memungkinkannya dipanggil pada `const Shop&` dan mendokumentasikan janji itu. Perhatikan bahwa ia hanya dapat memanggil fungsi anggota `const` milik pesanan, dan semua getter memang `const`.

6. Setiap pemanggilan konstruktor, sah atau tidak, akan mengambil nomor, karena nomor diambil sebelum pemeriksaan berjalan. Pesanan Mbak Sari yang ditolak akan diam-diam menghabiskan nomor 3, dan pesanan Mas Joko menjadi #4. `list` akan menampilkan pesanan #1, #2, dan #4, dan `Order::count()` akan menyebut 4 padahal hanya ada 3 pesanan, melanggar Persamaan (14.2). Pelajarannya: ubahlah keadaan bersama hanya setelah semua pemeriksaan lolos.

Daftar Pustaka

[1] B. Liskov and S. Zilles, “Programming with abstract data types,” *ACM SIGPLAN Notices*, vol. 9, no. 4, pp. 50–59, 1974.

[2] B. Stroustrup and H. Sutter, Eds., “C++ Core Guidelines, Section C: Classes and class hierarchies (C.2: Use class if the class has an invariant),” accessed Sep. 2026. [Online]. Available: https://isocpp.github.io/CppCoreGuidelines/CppCoreGuidelines

[3] Python Software Foundation, “The Python Tutorial, Chapter 9: Classes,” Python 3.14 documentation, accessed Sep. 2026. [Online]. Available: https://docs.python.org/3.14/tutorial/classes.html

[4] S. Meyers, *Effective C++: 55 Specific Ways to Improve Your Programs and Designs*, 3rd ed. Upper Saddle River, NJ, USA: Addison-Wesley, 2005.

[5] P. Deitel and H. Deitel, *C++ How to Program*, 10th ed. Hoboken, NJ, USA: Pearson, 2017.

[6] B. Stroustrup, *The C++ Programming Language*, 4th ed. Upper Saddle River, NJ, USA: Addison-Wesley, 2013.

[7] cppreference.com, "Derived classes" and "virtual function specifier," accessed Sep. 2026. [Online]. Available: https://en.cppreference.com/w/cpp/language/derived_class

Bab 15

Overloading Operator dan Studi Kasus Utuh

Anda sudah menulis `a + b` sejak Bab 1, dan `std::cout << x` sejak Bab 13, tanpa bertanya apa sebenarnya `+` dan `<<` itu. Dalam C++ keduanya adalah fungsi dengan nama yang tidak biasa, dan Anda dapat menulis fungsi semacam itu untuk kelas Anda sendiri. Bab ini menunjukkan caranya dan kapan melakukannya, lalu memakai segala sesuatu dari Bab 13 sampai 15 dalam satu program utuh. Di akhir bab ini Anda akan mampu menjelaskan apa yang dilakukan kompilator terhadap ekspresi seperti `wash + iron` bila operannya objek dari kelas Anda; memutuskan apakah sebuah operator sebaiknya menjadi fungsi anggota atau fungsi non-anggota; menulis operator aritmetika, penugasan gabungan, perbandingan, dan keluaran untuk tipe uang dan tipe pecahan; menghitung faktor persekutuan terbesar dan kelipatan persekutuan terkecil dengan algoritma Euclid; serta membangun, menjalankan, dan menguji program Profitina Laundry yang lengkap, yang membaca daftar harga, pelanggan, dan pesanan dari berkas, mencetak struk dengan biaya ekspres dan diskon anggota, dan menulis laporan harian yang terurut, dalam C++ dan Python dengan keluaran identik.

15.1 Struk yang Jumlahnya Cocok

Setiap akhir bulan Bu Rina menjumlahkan struk-struknya, dan setiap bulan totalnya selisih beberapa ratus rupiah dari angka program kasir. Selisihnya tidak pernah cukup besar untuk dikhawatirkan, tetapi juga tidak pernah cukup kecil untuk diabaikan. Ketika akhirnya ia meminta kedua pegawainya menghitung struk yang sama dengan tangan, sebabnya langsung tampak: yang satu membulatkan diskon anggota 10% *ke bawah*, yang lain *ke rupiah terdekat*, dan program melakukan hal yang lain lagi. Tampilan struknya pun berbeda: pegawai yang satu menulis “Rp24150”, yang lain “Rp24.150”, dan program lama mencetak “Rp24150”.

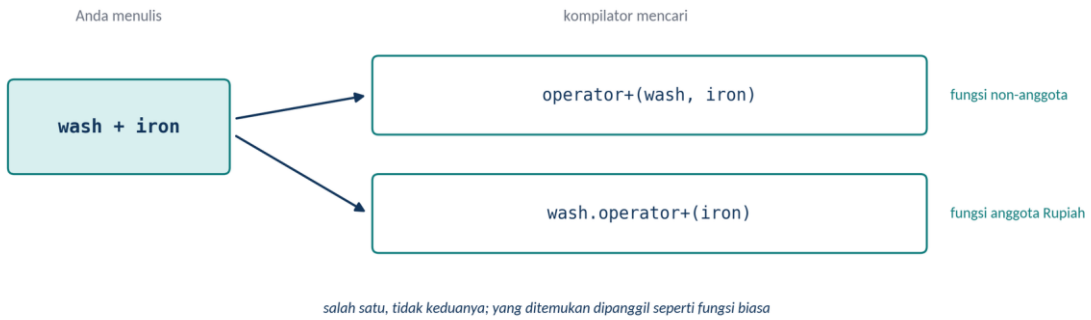
Masalahnya bukan aritmetika; semua orang dapat menghitung 10% dari sebuah bilangan. Masalahnya, *uang* diperlakukan sebagai bilangan biasa, dan setiap potong kode yang menanganinya membuat keputusan kecilnya sendiri: bagaimana membulatkan, bagaimana mencetak, apakah boleh menjumlahkan berat dengan harga. Dalam program beberapa ratus baris ada puluhan tempat seperti itu, dan semuanya lama-lama bergeser.

Selisih kecil semacam ini tampak sepele, tetapi pembukuan yang tidak pernah cocok perlahan-lahan mengikis kepercayaan pemilik kepada programnya sendiri, dan program yang tidak dipercaya akhirnya tidak dipakai. Obatnya adalah gagasan Bab 14 yang diterapkan pada sebuah bilangan: jadikan uang sebuah *kelas*. Objek `Rupiah` menyimpan jumlah rupiah utuh; ia tahu cara menjumlahkan dan mengurangkan dirinya, cara mengambil persentase dengan satu aturan pembulatan yang tetap, cara membandingkan dirinya, dan cara mencetak dirinya dengan gaya Indonesia, dengan titik di antara setiap kelompok tiga digit. Dan berkat *operator overloading*, kode yang memakainya tetap tampak seperti aritmetika biasa: `base + express - discount`.

15.2 Gagasannya dalam Kata-Kata Sederhana

15.2.1 Operator adalah fungsi dengan nama khusus

Ketika kompilator melihat `wash + iron` dan kedua operannya `int`, ia menghasilkan instruksi penjumlahan mesin. Bila operannya objek dari suatu kelas, tidak ada instruksi mesin untuk itu, sehingga kompilator mencari sebuah *fungsi* bernama `operator+` yang menerima keduanya. **Gambar 15.1** menunjukkan dua tempat yang dicarinya.



Gambar 15.1 Apa yang dilakukan kompilator terhadap `wash + iron` untuk operan berupa kelas: ia mencari `operator+(wash, iron)` non-anggota atau `wash.operator+(iron)` anggota, lalu memanggil yang ditemukannya.

Sesungguhnya hanya itu saja [1]. `operator+` adalah fungsi biasa yang kebetulan bernama `operator+`. Anda bahkan boleh memanggilnya dengan nama itu, `operator+(wash, iron)`, dan mendapat hasil yang sama. Sintaksis khususnya hanya membuat Anda dapat *memakainya* sebagai `wash + iron`. Demikian pula, `std::cout << x` memanggil fungsi `operator<<(std::cout, x)`, yang didefinisikan pustaka standar untuk setiap tipe bawaan dan boleh Anda definisikan untuk tipe Anda sendiri. Anda sudah memakai operator yang di-*overload* sejak Bab 13: `+` pada dua `std::string` adalah salah satunya, begitu pula `==` dan `<` pada string, dan `[]` pada `std::vector`.

> **DEFINISI.** *Operator overloading* adalah mendefinisikan makna sebuah operator C++ untuk operan bertipe buatan pengguna dengan menulis fungsi bernama `operator` diikuti lambang operatornya. Operator biner berupa fungsi non-anggota dengan dua parameter atau fungsi anggota dengan satu parameter (operan kirinya adalah objek itu sendiri); operator uner berupa non-anggota dengan satu parameter atau anggota tanpa parameter.

15.2.2 Aturannya

C++ memberi batas yang masuk akal pada overloading [2]. Anda tidak dapat menciptakan operator baru (tidak ada `**` untuk pangkat); tidak dapat mengubah banyaknya operan sebuah operator, presedensinya, atau asosiativitasnya (`a + b * c` selalu mengalikan lebih dulu); tidak dapat mendefinisikan ulang operator pada tipe bawaan (`1 + 1` selalu 2); dan beberapa operator sama sekali tidak dapat di-*overload*: `.` (akses anggota), `::` (cakupan), `?:`, dan `sizeof`. **Tabel 15.1** mendaftar operator-operator yang di-*overload* dalam bab ini, di mana biasanya ditulis, dan metode Python yang memainkan peran yang sama.

Tabel 15.1 Operator-operator yang di-*overload* dalam bab ini, bentuk lazimnya di C++, dan padanannya di Python.

Operator	Bentuk lazim di C++	Mengembalikan	Metode Python
`a += b`, `a -= b`	anggota	`*this`, lewat referensi	`__iadd__`, `__isub__` (opsional)
`a + b`, `a - b`, `a * k`	non-anggota	objek baru	`__add__`, `__sub__`, `__mul__`, `__rmul__`
`-a`	anggota	objek baru	`__neg__`
`a == b`, `a != b`	non-anggota	`bool`	`__eq__`, `__ne__`
`a < b` dan kawan-kawannya	non-anggota	`bool`	`__lt__` (+ `functools.total_ordering`)
`os << a`	non-anggota	`os`, lewat referensi	`__str__`, `__format__`

Tiga pedoman menentukan kolom "bentuk lazim" [3], [4].

- **Operator yang mengubah operan kirinya adalah anggota.** `a += b` mengubah `a`, sehingga memerlukan akses ke data *private* milik `a`; ia mengembalikan `*this` (objek itu sendiri) lewat referensi agar dapat dirangkai seperti versi bawaannya.
- **Operator simetris adalah non-anggota yang dibangun di atas anggota.** `a + b` tidak mengubah operan mana pun, sehingga ditulis di luar kelas sebagai "salin `a`, lalu `+= b`". Ini menjaga kelas tetap ramping dan memperlakukan kedua operan setara, hal yang penting bila ada konversi (Subbab 15.4 menunjukkan `2 + half`).
- `<<` **harus non-anggota**, karena operan kirinya adalah stream, bukan objek Anda, dan Anda tidak dapat menambahkan anggota ke `std::ostream`.

Aturan terpenting justru tidak tercantum dalam bahasa sama sekali: *operator yang di-overload harus melakukan apa yang diharapkan pembaca*. `+` pada uang menjumlahkan uang. Memakai `+` untuk "menambahkan pesanan ke toko" atau `<<` untuk "mengirim surel" membuat kode lebih pendek tetapi jauh lebih sulit dibaca. Bila ragu, tulislah fungsi bernama.

Dua operator lagi sering muncul di pustaka standar, dan Anda sudah memakai keduanya. `operator[]` memberi pengindeksan: `grams[2]` pada `std::vector` dan `services_[code]` pada `std::map` (Subbab 15.5) adalah pemanggilan fungsi anggota bernama `operator[]`, sehingga sebuah *map* dapat memutuskan untuk *menciptakan* entri yang belum ada sementara vektor sama sekali tidak memeriksa indeksnya; perilakunya adalah apa pun yang ditulis dalam fungsinya. `operator()` membuat objek dapat dipanggil seperti fungsi; objek semacam itu disebut *function object*, dan fungsi perbandingan kecil yang diberikan Bab 16 kepada `std::stable_sort` dikompilasi menjadi objek jenis ini. Keduanya tidak diperlukan oleh kelas-kelas dalam bab ini, tetapi mengenalinya menghapus sisa-sisa keajaiban dari pustaka standar: setiap sintaksis khusus yang Anda temui pada sebuah kelas adalah fungsi yang dapat Anda baca.

15.3 Kelas Uang: `Rupiah`

15.3.1 Kelasnya

Kode 15.1 adalah kelasnya secara lengkap. Seluruhnya ditulis dalam header, dengan setiap fungsi `inline`, sehingga program mana pun dapat memakainya cukup dengan menyertakan satu berkas.

Kode 15.1 `rupiah.h`: jumlah uang dalam rupiah utuh, beserta operator-operatornya.

```cpp
// rupiah.h - Chapter 15: an amount of money in whole rupiah, with operators.
#ifndef RUPIAH_H
#define RUPIAH_H

#include <ostream>
#include <string>

class Rupiah {
public:
    Rupiah() = default;                          // Rp 0
    explicit Rupiah(long long amount) : amount_(amount) {}
    long long value() const { return amount_; }

    Rupiah& operator+=(const Rupiah& other) {  // members change *this
        amount_ += other.amount_;
        return *this;
    }
    Rupiah& operator-=(const Rupiah& other) {
        amount_ -= other.amount_;
        return *this;
    }
    Rupiah operator-() const { return Rupiah(-amount_); }

    // p percent of this amount, rounded half away from zero.
    Rupiah percent(int p) const {
        long long a = amount_ < 0 ? -amount_ : amount_;
        long long r = (a * p + 50) / 100;
        return Rupiah(amount_ < 0 ? -r : r);
    }

private:
    long long amount_ = 0;
};

// Non-members, built on the members above.
inline Rupiah operator+(Rupiah a, const Rupiah& b) { return a += b; }
inline Rupiah operator-(Rupiah a, const Rupiah& b) { return a -= b; }
inline Rupiah operator*(const Rupiah& a, long long k) {
    return Rupiah(a.value() * k);
}
inline Rupiah operator*(long long k, const Rupiah& a) { return a * k; }
```

Kode 15.1 (Lanjutan) *rupiah.h: jumlah uang dalam rupiah utuh, beserta operator-operatornya.*

```
inline bool operator==(const Rupiah& a, const Rupiah& b) {
    return a.value() == b.value();
}
inline bool operator!=(const Rupiah& a, const Rupiah& b) { return !(a == b); }
inline bool operator<(const Rupiah& a, const Rupiah& b) {
    return a.value() < b.value();
}
inline bool operator>(const Rupiah& a, const Rupiah& b) { return b < a; }
inline bool operator<=(const Rupiah& a, const Rupiah& b) { return !(b < a); }
inline bool operator>=(const Rupiah& a, const Rupiah& b) { return !(a < b); }

// "Rp 1.234.500": Indonesian style, a dot between groups of three digits.
inline std::string toText(const Rupiah& r) {
    long long v = r.value();
    std::string digits = std::to_string(v < 0 ? -v : v);
    std::string out;
    int count = 0;
    for (int i = static_cast<int>(digits.size()) - 1; i >= 0; --i) {
        out.insert(out.begin(), digits[i]);
        if (++count % 3 == 0 && i > 0) out.insert(out.begin(), '.');
    }
    return (v < 0 ? "-Rp " : "Rp ") + out;
}

inline std::ostream& operator<<(std::ostream& os, const Rupiah& r) {
    return os << toText(r);                    // honors setw, left and right
}

#endif
```

Mari kita telusuri.

Data dan konstruktornya. Satu-satunya anggota data adalah `long long amount_`, rupiah utuh. `long long` menampung jumlah sampai sekitar 9.2×10^{18}, jauh melebihi keperluan laundry mana pun, dan bilangan bulat membuat setiap keputusan pembulatan menjadi eksplisit. `Rupiah() = default;` meminta konstruktor bawaan biasa, yang memakai penginisialisasi anggota `= 0`, sehingga `Rupiah` baru bernilai Rp0. Konstruktor kedua ditandai **`explicit`**: `Rupiah r(5000);` diizinkan, tetapi `Rupiah r = 5000;` dan mengirim `5000` polos di tempat yang mengharapkan `Rupiah` tidak diizinkan. Itu disengaja. Dengan `explicit`, kompilator tidak akan pernah diam-diam mengubah berat atau hitungan menjadi uang; Anda harus menulis `Rupiah(5000)`.

Penugasan gabungan sebagai anggota. `operator+=` menambahkan jumlah lain ke jumlah ini dan mengembalikan `*this` lewat referensi. `this` adalah pointer ke objek saat ini (Subbab 14.4.5), jadi `*this` adalah objek itu sendiri. Mengembalikan referensi itulah yang membuat `till += wash;` berperilaku seperti `+=` bawaan.

Minus uner. `Rupiah operator-() const` mengembalikan jumlah yang dinegasikan tanpa mengubah objeknya. Struk memakainya untuk mencetak diskon sebagai jumlah negatif.

Persentase, satu aturan untuk semua. `percent(p)` menghitung $p\%$ dari jumlahnya, dibulatkan *setengah menjauhi nol*. Untuk jumlah tak negatif a rumusnya adalah

$$\text{percent}(a, p) = \left\lfloor \frac{a \cdot p + 50}{100} \right\rfloor \tag{15.1}$$

dan untuk jumlah negatif, hal yang sama dihitung untuk $|a|$ lalu tandanya dikembalikan, sehingga -10% dari pengembalian dana dibulatkan dengan cara yang sama seperti 10% dari penjualan. Menambahkan 50 sebelum membagi dengan 100 adalah trik yang sama dengan menambahkan

500 sebelum membagi dengan 1000 di Bab 13: mengubah "potong pecahannya" menjadi "bulatkan ke terdekat", dengan setengah dibulatkan ke atas. Contohnya, 10% dari Rp24.150 adalah $\lfloor(241{,}500 + 50)/100\rfloor = \lfloor 2415.5\rfloor = 2415$: nilai tepatnya, 2.415,0, tetap 2.415. Dan 50% dari Rp35.001 adalah $\lfloor(1{,}750{,}050 + 50)/100\rfloor = 17{,}501$, setengah rupiahnya dibulatkan ke atas. Kini setiap struk di toko dibulatkan dengan tepat satu cara, karena hanya ada tepat satu fungsi yang melakukannya.

Aritmetika sebagai non-anggota. `operator+(Rupiah a, const Rupiah& b)` menerima operan kirinya *lewat nilai*, sehingga membuat salinan; ia menambahkan `b` ke salinan itu dengan `+=` lalu mengembalikannya. Satu baris, dan seluruh pekerjaan sesungguhnya tetap di `+=`. `operator*` hadir dalam dua versi, `Rupiah * long long` dan `long long * Rupiah`, agar `rate * amount` maupun `3 * iron` sama-sama berjalan. Tidak ada `Rupiah * Rupiah`, karena uang dikali uang tidak bermakna, dan kesalahan seperti `price * price` tidak akan terkompilasi.

Perbandingan. `==` dan `<` membandingkan jumlahnya; keempat operator lainnya ditulis dengan memakai keduanya (`a > b` adalah `b < a`, `a <= b` adalah `!(b < a)`), sehingga tidak mungkin saling bertentangan. (C++20 dapat membangkitkan keenamnya dari satu `operator<=>`; di C++17 kita menuliskannya.)

Pencetakan. `toText` membangun teks dari kanan, menyisipkan titik setiap tiga digit, dan menaruh `Rp` atau `-Rp` di depan: 1234567890 menjadi `Rp 1.234.567.890`, seperti cara menulis jumlah uang di Indonesia. `operator<<` mengirim teks itu ke stream. Karena yang disisipkan adalah satu string, manipulator `std::setw`, `std::left`, dan `std::right` bekerja pada `Rupiah` persis seperti pada bilangan, dan itulah yang membuat kolom-kolom struk di bawah rata.

15.3.2 Kelasnya beraksi

Kode 15.2 mencoba setiap operatornya.

Kode 15.2 `rupiah_demo.cpp`: setiap operator `Rupiah` dipakai.

```
// rupiah_demo.cpp - Chapter 15: the Rupiah class at work.
#include <iomanip>
#include <iostream>

#include "rupiah.h"

int main() {
    Rupiah wash(24150), iron(10000);
    Rupiah sum = wash + iron;                  // operator+(wash, iron)
    std::cout << "wash + iron      = " << sum << '\n';
    std::cout << "wash - iron      = " << wash - iron << '\n';
    std::cout << "iron - wash      = " << iron - wash << '\n';
    std::cout << "3 * iron         = " << 3 * iron << '\n';
    std::cout << "10% of the sum   = " << sum.percent(10) << '\n';
    std::cout << "big amount       = " << Rupiah(1234567890) << '\n';

    Rupiah till;                               // Rp 0
    till += wash;
    till += iron;
    till -= Rupiah(5000);
    std::cout << "till             = " << till << '\n';

    std::cout << std::boolalpha;
    std::cout << "wash == iron     : " << (wash == iron) << '\n';
    std::cout << "iron <  wash     : " << (iron < wash) << '\n';
    std::cout << "till >= sum      : " << (till >= sum) << '\n';

    std::cout << '[' << std::setw(14) << wash << "]\n";   // right aligned
    std::cout << '[' << std::left << std::setw(14) << wash << "]\n";
    // Rupiah x = 5000;   would not compile: the constructor is explicit
    // wash * wash;       would not compile: no operator for Rupiah * Rupiah
    return 0;
}
```

```
wash + iron      = Rp 34.150
wash - iron      = Rp 14.150
iron - wash      = -Rp 14.150
3 * iron         = Rp 30.000
10% of the sum   = Rp 3.415
big amount       = Rp 1.234.567.890
till             = Rp 29.150
wash == iron     : false
iron <  wash     : true
till >= sum      : false
[     Rp 24.150]
[Rp 24.150     ]
```

Setiap baris terbaca sebagai aritmetika biasa, dan setiap baris adalah pemanggilan fungsi di baliknya. `3 * iron` menemukan `operator*(long long, const Rupiah&)` non-anggota. `iron - wash` menghasilkan jumlah negatif, dicetak dengan tandanya di depan `Rp`. Dua baris terakhir menunjukkan `setw(14)` meratakan jumlah ke kanan dan ke kiri di dalam kurung siku. Dua baris yang dijadikan komentar menunjukkan apa yang ditolak kompilator: `Rupiah x = 5000;` karena konstruktornya `explicit`, dan `wash * wash` karena operator itu tidak ada. Tipe yang menolak hal tak masuk akal pada waktu kompilasi lebih berharga daripada sebanyak apa pun komentar yang cermat.

15.3.3 Menelusuri satu ekspresi dengan tangan

Ada gunanya mengikuti satu ekspresi sampai ke dasarnya sekali saja, karena sesudahnya operator tidak lagi menyimpan misteri. Ambil total sebuah struk, `base + express - discount`, dengan `base` = Rp27.500, `express` = Rp13.750, dan `discount` = Rp4.125. Seperti operator bawaan,

`+` dan `-` yang presedensinya sama dievaluasi dari kiri ke kanan, jadi ekspresi itu berarti `(base + express) - discount`.

1. Kompilator memerlukan `operator+` untuk dua `Rupiah`. Ia menemukan `operator+(Rupiah a, const Rupiah& b)` non-anggota. Parameter `a` adalah *salinan* `base` (27.500), dan `b` adalah nama lain bagi `express`.
2. Di dalamnya, `a += b` memanggil `operator+=` anggota pada salinan itu: `amount_` menjadi $27{,}500 + 13{,}750 = 41{,}250$, dan fungsinya mengembalikan referensi ke salinan tersebut.
3. `operator+` mengembalikan salinan itu sebagai hasilnya, sebuah `Rupiah` sementara berisi 41.250. `base` sendiri tidak pernah tersentuh.
4. Kompilator kini memerlukan `operator-` untuk objek sementara itu dan `discount`. Ia menemukan `operator-` non-anggota, yang menyalin objek sementara, memanggil `-=` dengan 4.125, dan mengembalikan objek sementara baru berisi $41{,}250 - 4{,}125 = 37{,}125$.
5. Objek-objek sementara mati di akhir pernyataan penuh; untuk kelas yang satu-satunya anggotanya `long long`, kompilator mengoptimalkan semua ini menjadi beberapa instruksi mesin yang sama dengan versi bilangan bulat biasa.

Jadi ekspresi itu tidak lebih mahal daripada aritmetika bilangan bulat, namun tidak mungkin tertukar dengan berat atau hitungan. Perpaduan itu, keamanan tanpa biaya waktu jalan, adalah alasan pemrogram C++ begitu leluasa memakai kelas-kelas kecil seperti `Rupiah`.

15.3.4 Kelas yang sama dalam Python

Python juga punya operator overloading; hanya saja ia memakai nama-nama metode yang tetap, yaitu metode *khusus* atau metode "*dunder*" (*double underscore*) dari model datanya [5]. `a + b` memanggil `a.__add__(b)`; `3 * a` mula-mula mencoba perkalian milik `int`, yang tidak mengenal `Rupiah`, lalu memanggil `a.__rmul__(3)` ("perkalian terpantul"); `a < b` memanggil `a.__lt__(b)`; `str(a)` memanggil `a.__str__()`; dan kolom f-string seperti `{wash:>14}` memanggil `wash.__format__(">14")`. Dekorator `@functools.total_ordering` menurunkan `<=`, `>`, dan `>=` dari `__eq__` dan `__lt__`, persis seperti yang dilakukan header C++ dengan tangan. **Kode 15.3** adalah kelasnya, dan **Kode 15.4** mencetak keluaran yang sama dengan demo C++.

Kode 15.3 `rupiah.py`: kelas `Rupiah` dalam Python.

```python
# rupiah.py - Chapter 15: an amount of money in whole rupiah, with operators.
from functools import total_ordering


@total_ordering                                 # derives <=, >, >= from == and <
class Rupiah:
    __slots__ = ("_amount",)

    def __init__(self, amount=0):
        self._amount = int(amount)

    def value(self):
        return self._amount

    def __add__(self, other):
        return Rupiah(self._amount + other._amount)

    def __sub__(self, other):
        return Rupiah(self._amount - other._amount)

    def __neg__(self):
        return Rupiah(-self._amount)

    def __mul__(self, k):                       # Rupiah * int
        return Rupiah(self._amount * k)

    __rmul__ = __mul__                          # int * Rupiah

    def __eq__(self, other):
        return isinstance(other, Rupiah) and self._amount == other._amount

    def __lt__(self, other):
        return self._amount < other._amount

    __hash__ = None                             # changeable, so not a dict key

    def percent(self, p):
        """p percent of this amount, rounded half away from zero."""
        a = abs(self._amount)
        r = (a * p + 50) // 100
        return Rupiah(-r if self._amount < 0 else r)

    def __str__(self):                          # "Rp 1.234.500"
        text = f"{abs(self._amount):,}".replace(",", ".")
        return ("-Rp " if self._amount < 0 else "Rp ") + text
```

Kode 15.3 (Lanjutan) *rupiah.py: kelas Rupiah dalam Python.*

```python
    def __format__(self, spec):                 # lets f"{r:>12}" align it
        return format(str(self), spec)
```

Kode 15.4 `rupiah_demo.py`: demo Python mencetak persis apa yang dicetak `rupiah_demo.cpp`.

```
# rupiah_demo.py - Chapter 15: the Rupiah class at work.
from rupiah import Rupiah

wash, iron = Rupiah(24150), Rupiah(10000)
total = wash + iron                          # wash.__add__(iron)
print(f"wash + iron      = {total}")
print(f"wash - iron      = {wash - iron}")
print(f"iron - wash      = {iron - wash}")
print(f"3 * iron         = {3 * iron}")
print(f"10% of the sum   = {total.percent(10)}")
print(f"big amount       = {Rupiah(1234567890)}")

till = Rupiah()                              # Rp 0
till += wash                                 # Python falls back on __add__
till += iron
till -= Rupiah(5000)
print(f"till             = {till}")

print(f"wash == iron     : {str(wash == iron).lower()}")
print(f"iron <  wash     : {str(iron < wash).lower()}")
print(f"till >= sum      : {str(till >= total).lower()}")

print(f"[{wash:>14}]")                       # right aligned
print(f"[{wash:<14}]")
```

Dua rincian Python patut disebut. `__hash__ = None` menyatakan bahwa `Rupiah` tidak dapat dipakai sebagai kunci *dictionary*; aturan di Python adalah bahwa objek yang mendefinisikan `__eq__` harus memutuskan soal *hashing*, dan uang yang dapat dibandingkan memang bukan sesuatu yang kita inginkan sebagai kunci. Dan pemisah ribuan berasal dari bahasa mini format milik Python sendiri: `f"{24150:,}"` menghasilkan `24,150`, dan mengganti koma dengan titik menghasilkan bentuk Indonesia.

15.4 Contoh Kedua: Pecahan dan Algoritma Euclid

Uang hanya memerlukan bilangan bulat. Sebagian besaran secara alami berupa pecahan: setengah sendok takar deterjen ditambah sepertiga sendok pelembut, tiga perempat botol pewangi. Kelas `Fraction` adalah latihan klasik operator overloading, dan ia menghadirkan kembali salah satu algoritma tertua di dunia.

Pecahan $\frac{a}{b}$ disimpan dalam *bentuk paling sederhana*: $\frac{6}{8}$ disimpan sebagai $\frac{3}{4}$. Untuk menyederhanakan pecahan, kita membagi pembilang dan penyebutnya dengan *faktor persekutuan terbesar* (FPB, *greatest common divisor*, gcd), bilangan terbesar yang membagi keduanya. Euclid menjelaskan cara menemukannya sekitar 300 SM [6]: FPB tidak berubah bila bilangan yang lebih besar diganti dengan sisa pembagiannya oleh bilangan yang lebih kecil,

$$\gcd(a, b) = \gcd(b,\ a \bmod b), \qquad \gcd(a, 0) = a \tag{15.2}$$

Untuk 270 dan 192: $\gcd(270,192) = \gcd(192,78) = \gcd(78,36) = \gcd(36,6) = \gcd(6,0) = 6$. Setiap langkah membuat bilangannya mengecil dengan cepat: banyaknya langkah hanya tumbuh seiring banyaknya *digit*, bukan besarnya bilangan. Mengapa ini benar? Setiap bilangan yang membagi a dan b juga membagi $a - qb$ untuk bilangan bulat q mana pun, khususnya sisa $a \bmod b$; dan setiap bilangan yang membagi b dan sisa itu juga membagi $a = qb + (a \bmod b)$. Jadi pasangan (a, b) dan $(b, a \bmod b)$ memiliki pembagi persekutuan yang persis sama, sehingga yang terbesar pun sama.

Kelipatan persekutuan terkecil (KPK, *least common multiple*, lcm), bilangan terkecil yang dapat dibagi keduanya, diperoleh dari FPB:

$$\operatorname{lcm}(a, b) = \frac{a}{\gcd(a, b)} \cdot b \tag{15.3}$$

Membagi *lebih dulu* penting di dalam program. $\frac{a \cdot b}{\gcd(a,b)}$ adalah bilangan yang sama secara matematis, tetapi hasil kali $a \cdot b$ dapat meluap walaupun KPK-nya sendiri muat; $a/\gcd(a, b)$ selalu bilangan bulat, dan mengalikannya dengan b memberi jawaban dengan nilai antara yang lebih kecil. Penjumlahan pecahan memakai KPK penyebut-penyebutnya sebagai penyebut bersama:

$$\frac{a}{b} + \frac{c}{d} = \frac{a \cdot \frac{m}{b} + c \cdot \frac{m}{d}}{m}, \qquad m = \operatorname{lcm}(b, d) \tag{15.4}$$

Untuk $\frac{1}{2} + \frac{1}{3}$: $m = 6$, dan jumlahnya $\frac{1 \cdot 3 + 1 \cdot 2}{6} = \frac{5}{6}$. **Kode 15.5** adalah kelasnya beserta operator-operatornya.

Kode 15.5 `fraction.cpp`: kelas `Fraction` yang dibangun di atas FPB Euclid, dengan operator aritmetika, perbandingan, dan keluaran.

```cpp
// fraction.cpp - Chapter 15: a Fraction class built on Euclid's gcd.
#include <iostream>
#include <stdexcept>

long long gcd(long long a, long long b) {     // Euclid, about 300 BC
    if (a < 0) a = -a;
    if (b < 0) b = -b;
    while (b != 0) {
        long long r = a % b;
        a = b;
        b = r;
    }
    return a;
}

long long lcm(long long a, long long b) {
    return a / gcd(a, b) * b;                  // divide first: no overflow
}

class Fraction {
public:
    Fraction(long long num = 0, long long den = 1) : num_(num), den_(den) {
        if (den_ == 0) throw std::invalid_argument("denominator is zero");
        if (den_ < 0) { num_ = -num_; den_ = -den_; }   // sign lives on top
        long long g = gcd(num_, den_);
        if (g > 1) { num_ /= g; den_ /= g; }            // always in lowest terms
    }
    long long num() const { return num_; }
    long long den() const { return den_; }

private:
    long long num_, den_;
};

Fraction operator+(const Fraction& a, const Fraction& b) {
    long long d = lcm(a.den(), b.den());       // the least common denominator
    return Fraction(a.num() * (d / a.den()) + b.num() * (d / b.den()), d);
}
Fraction operator-(const Fraction& a, const Fraction& b) {
    return a + Fraction(-b.num(), b.den());
}
Fraction operator*(const Fraction& a, const Fraction& b) {
    return Fraction(a.num() * b.num(), a.den() * b.den());
}
Fraction operator/(const Fraction& a, const Fraction& b) {
    return Fraction(a.num() * b.den(), a.den() * b.num());
```

Kode 15.5 (Lanjutan) *fraction.cpp: kelas Fraction yang dibangun di atas FPB Euclid, dengan operator aritmetika, perbandingan, dan keluaran.*

```
}
bool operator==(const Fraction& a, const Fraction& b) {
    return a.num() == b.num() && a.den() == b.den();   // lowest terms: unique
}
bool operator<(const Fraction& a, const Fraction& b) {
    return a.num() * b.den() < b.num() * a.den();      // both dens positive
}
std::ostream& operator<<(std::ostream& os, const Fraction& f) {
    if (f.den() == 1) return os << f.num();
    return os << f.num() << '/' << f.den();
}

int main() {
    long long pairs[][2] = {{12, 18}, {7, 13}, {100, 75}, {270, 192}};
    for (auto& p : pairs)
        std::cout << "gcd(" << p[0] << ", " << p[1] << ") = " << gcd(p[0], p[1])
                  << ", lcm = " << lcm(p[0], p[1]) << '\n';

    Fraction half(1, 2), third(1, 3), soap(3, 4);
    std::cout << "1/2 + 1/3   = " << half + third << '\n';
    std::cout << "1/2 - 1/3   = " << half - third << '\n';
    std::cout << "3/4 * 2/3   = " << soap * Fraction(2, 3) << '\n';
    std::cout << "3/4 / 1/2   = " << soap / half << '\n';
    std::cout << "6/8 == 3/4  : " << (Fraction(6, 8) == soap ? "yes" : "no")
              << '\n';
    std::cout << "1/3 < 1/2   : " << (third < half ? "yes" : "no") << '\n';
    std::cout << "2 + 1/2     = " << 2 + half << '\n';   // 2 becomes Fraction(2)
    std::cout << "4/-6        = " << Fraction(4, -6) << '\n';
    try {
        std::cout << half / Fraction(0) << '\n';
    } catch (const std::invalid_argument& e) {
        std::cout << "1/2 / 0     : refused, " << e.what() << '\n';
    }
    return 0;
}
```

```
gcd(12, 18) = 6, lcm = 36
gcd(7, 13) = 1, lcm = 91
gcd(100, 75) = 25, lcm = 300
gcd(270, 192) = 6, lcm = 8640
1/2 + 1/3   = 5/6
1/2 - 1/3   = 1/6
3/4 * 2/3   = 1/2
3/4 / 1/2   = 3/2
6/8 == 3/4  : yes
1/3 < 1/2   : yes
2 + 1/2     = 5/2
4/-6        = -2/3
1/2 / 0     : refused, denominator is zero
```

Tiga keputusan rancangan membuat kelas ini mudah dipercaya.

- **Invarian ditegakkan di konstruktor.** Penyebut nol melempar eksepsi; penyebut negatif memindahkan tandanya ke pembilang; dan setiap pecahan disederhanakan dengan FPB. Karena *setiap* pecahan melewati konstruktor, termasuk hasil `+`, `-`, `*`, dan `/`, setiap pecahan dalam program selalu berada dalam bentuk paling sederhana dengan penyebut positif.
- **Invarian itu membuat `==` menjadi sepele.** Dua pecahan dalam bentuk paling sederhana sama tepat bila pembilang dan penyebutnya sama, sehingga $\frac{6}{8} = \frac{3}{4}$ diputuskan dengan membandingkan $3 = 3$ dan $4 = 4$. Tanpa invarian, `==` memerlukan perkalian silang. Dan `<` dapat

memakai perkalian silang dengan aman, $\frac{a}{b} < \frac{c}{d} \Leftrightarrow ad < cb$, hanya karena kedua penyebutnya positif; dengan penyebut negatif pertidaksamaannya akan terbalik.

- **Konstruktor ini sengaja *tidak* `explicit`.** `Fraction(long long num = 0, long long den = 1)` dapat dipanggil dengan satu argumen, sehingga juga berfungsi sebagai konversi dari bilangan bulat: dalam `2 + half`, kompilator mengubah `2` menjadi `Fraction(2)` (yaitu $\frac{2}{1}$) lalu memanggil `operator+` non-anggota. Ini hanya berhasil karena `operator+` adalah non-anggota: `operator+` anggota akan memerlukan `Fraction` di sebelah kiri, dan `2` bukan `Fraction`. Untuk pecahan, "setiap bilangan bulat adalah pecahan" memang benar, jadi konversinya disambut baik. Untuk uang tidak demikian ("setiap bilangan bulat adalah sejumlah rupiah" adalah jalan berat berubah menjadi harga), dan karena itulah konstruktor `Rupiah` dibuat `explicit`. Memilih di antara keduanya adalah keputusan rancangan yang Anda ambil untuk setiap konstruktor satu argumen.

Kode 15.6 adalah versi Python-nya. Python memerlukan `__radd__` untuk `2 + half`, karena `int.__add__` tidak mengenal kelas kita, dan fungsi bantu kecil `_of` mengubah bilangan bulat di sebelah kanan menjadi `Fraction`. (Pustaka standar sudah punya `fractions.Fraction` yang lengkap [7]; membangun milik kita sendiri menunjukkan apa isinya.)

Kode 15.6 `fraction.py`: kelas `Fraction` dalam Python.

```python
# fraction.py - Chapter 15: a Fraction class built on Euclid's gcd.
# (Python's standard library has fractions.Fraction; we build our own to learn.)


def gcd(a, b):                                  # Euclid, about 300 BC
    a, b = abs(a), abs(b)
    while b != 0:
        a, b = b, a % b
    return a


def lcm(a, b):
    return a // gcd(a, b) * b


class Fraction:
    def __init__(self, num=0, den=1):
        if den == 0:
            raise ValueError("denominator is zero")
        if den < 0:                             # sign lives on top
            num, den = -num, -den
        g = gcd(num, den)
        if g > 1:                               # always in lowest terms
            num, den = num // g, den // g
        self.num, self.den = num, den

    @staticmethod
    def _of(x):
        return x if isinstance(x, Fraction) else Fraction(x)

    def __add__(self, other):
        other = Fraction._of(other)
        d = lcm(self.den, other.den)
        top = self.num * (d // self.den) + other.num * (d // other.den)
        return Fraction(top, d)

    __radd__ = __add__                          # 2 + half

    def __sub__(self, other):
        other = Fraction._of(other)
        return self + Fraction(-other.num, other.den)

    def __mul__(self, other):
        other = Fraction._of(other)
        return Fraction(self.num * other.num, self.den * other.den)
```

Kode 15.6 (Lanjutan) *fraction.py: kelas Fraction dalam Python.*

```
    def __truediv__(self, other):
        other = Fraction._of(other)
        return Fraction(self.num * other.den, self.den * other.num)

    def __eq__(self, other):
        other = Fraction._of(other)
        return self.num == other.num and self.den == other.den

    def __lt__(self, other):
        other = Fraction._of(other)
        return self.num * other.den < other.num * self.den

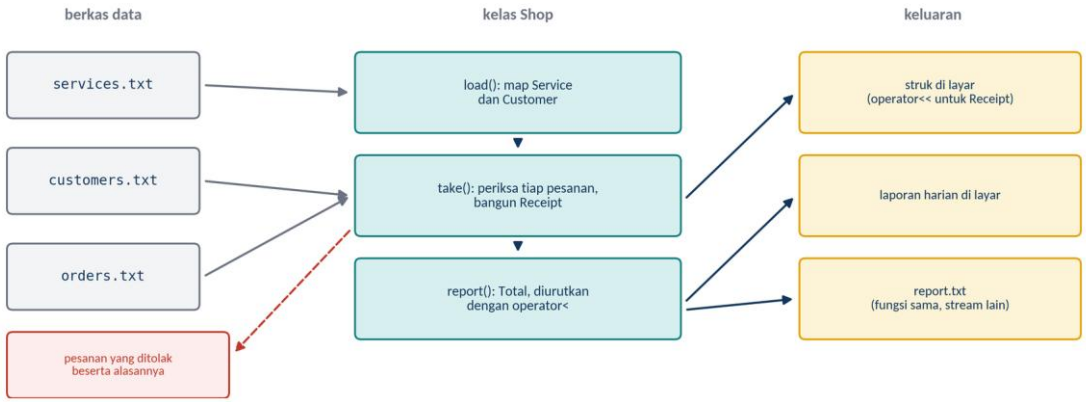
    def __str__(self):
        return str(self.num) if self.den == 1 else f"{self.num}/{self.den}"


for a, b in [(12, 18), (7, 13), (100, 75), (270, 192)]:
    print(f"gcd({a}, {b}) = {gcd(a, b)}, lcm = {lcm(a, b)}")

half, third, soap = Fraction(1, 2), Fraction(1, 3), Fraction(3, 4)
print(f"1/2 + 1/3   = {half + third}")
print(f"1/2 - 1/3   = {half - third}")
print(f"3/4 * 2/3   = {soap * Fraction(2, 3)}")
print(f"3/4 / 1/2   = {soap / half}")
print(f"6/8 == 3/4  : {'yes' if Fraction(6, 8) == soap else 'no'}")
print(f"1/3 < 1/2   : {'yes' if third < half else 'no'}")
print(f"2 + 1/2     = {2 + half}")
print(f"4/-6        = {Fraction(4, -6)}")
try:
    print(half / Fraction(0))
except ValueError as e:
    print(f"1/2 / 0     : refused, {e}")
```

15.5 Studi Kasus Utuh: Profitina Laundry

Sekarang kita satukan Bab 13, 14, dan 15 menjadi satu program yang dapat dipakai Bu Rina setiap akhir hari. **Gambar 15.2** adalah rancangannya.



Gambar 15.2 Studi kasus: tiga berkas data dimuat ke dalam `Shop`, setiap pesanan diperiksa dan diubah menjadi `Receipt`, lalu struk-struk dicetak dan diringkas dalam laporan yang ditulis ke layar sekaligus ke `report.txt`.

15.5.1 Datanya

Daftar harga tidak lagi berada di dalam program. `data/services.txt` memuat satu baris per layanan: kode pendek, nama, satuan (`kg` atau `pcs`), dan tarif dalam rupiah.

```
# code;name;unit;rate in rupiah per kg or per piece
W;Wash & fold;kg;7000
WI;Wash & iron;kg;10000
I;Iron only;kg;5000
BC;Bed cover;pcs;25000
SH;Shoes;pcs;35000
```

`data/customers.txt` mendaftar pelanggan beserta keanggotaannya; anggota mendapat diskon 10%:

```
# id;name;member (yes/no)
C01;Bu Rina Wulandari;yes
C02;Pak Budi Santoso;no
C03;Mbak Sari Dewi;yes
C04;Mas Joko Prasetyo;no
```

`data/orders.txt` adalah pesanan hari itu: nomor pesanan, id pelanggan, kode layanan, jumlah (gram untuk layanan `kg`, potong untuk layanan `pcs`), dan apakah ekspres. Dua baris sengaja dibuat salah: pesanan 1006 menyebut pelanggan yang tidak ada, dan pesanan 1009 layanan yang tidak ada.

```
# number;customer;service;amount (grams or pieces);express (yes/no)
1001;C01;W;3450;no
1002;C02;WI;5200;yes
1003;C03;BC;2;no
1004;C01;I;2000;no
1005;C04;SH;1;yes
1006;C05;W;3000;no
1007;C02;W;4100;no
1008;C03;WI;2750;yes
1009;C04;DC;1000;no
```

15.5.2 Aturannya

Harga satu pesanan dihitung dalam tiga langkah. *Harga dasar* adalah tarif kali jumlah, dengan pembulatan dari Bab 13 untuk layanan per kilogram. *Biaya ekspres* adalah 50% dari harga dasar, menurut Persamaan (15.1), untuk pesanan ekspres, dan nol untuk yang lain. *Diskon anggota* adalah 10% dari harga dasar ditambah biaya ekspres, sekali lagi menurut Persamaan (15.1), untuk anggota, dan nol untuk yang lain. Totalnya adalah

$$\text{total} = \text{base} + \text{express} - \text{discount} \tag{15.5}$$

dan karena keempat besaran itu objek `Rupiah`, baris ini muncul di program secara harfiah sebagai `base + express - discount`.

15.5.3 Programnya

Kode 15.7 adalah programnya secara lengkap. Ia menyertakan `rupiah.h` dari Subbab 15.3 tanpa perubahan.

Kode 15.7 `laundry.cpp`: program Profitina Laundry yang lengkap dalam C++.

```cpp
// laundry.cpp - Chapter 15: the complete Profitina Laundry case study.
// Reads services, customers and orders from data/, prints one receipt per
// order and the daily report, and saves the report to report.txt.
// Build: g++ -std=c++17 -O2 -Wall -Wextra laundry.cpp -o laundry
#include <algorithm>
#include <fstream>
#include <iomanip>
#include <iostream>
#include <map>
#include <sstream>
#include <string>
#include <vector>

#include "rupiah.h"

std::vector<std::string> split(const std::string& line, char sep) {
    std::vector<std::string> parts;
    std::string part;
    std::istringstream in(line);
    while (std::getline(in, part, sep)) parts.push_back(part);
    if (!line.empty() && line.back() == sep) parts.push_back("");
    return parts;
}

// Reads the non-comment lines of a data file, each split at ';'.
std::vector<std::vector<std::string>> readTable(const std::string& path) {
    std::vector<std::vector<std::string>> rows;
    std::ifstream in(path);
    std::string line;
    while (std::getline(in, line))
        if (!line.empty() && line[0] != '#') rows.push_back(split(line, ';'));
    return rows;
}

bool isNumber(const std::string& s) {
    if (s.empty() || s.size() > 6) return false;
    for (char c : s)
        if (c < '0' || c > '9') return false;
    return true;
}
```

Kode 15.7 (Lanjutan) *laundry.cpp: program Profitina Laundry yang lengkap dalam C++.*

```
struct Service {
    std::string code, name, unit;              // unit is "kg" or "pcs"
    Rupiah rate;
    Rupiah price(int amount) const {           // amount: grams or pieces
        if (unit == "kg") return Rupiah((amount * rate.value() + 500) / 1000);
        return rate * amount;
    }
    std::string describe(int amount) const {   // "3.450 kg" or "2 pcs"
        if (unit != "kg") return std::to_string(amount) + " pcs";
        std::string g = std::to_string(amount % 1000);
        return std::to_string(amount / 1000) + "." + std::string(3 - g.size(), '0')
               + g + " kg";
    }
};

struct Customer {
    std::string id, name;
    bool member;
};

struct Receipt {
    std::string number;
    const Customer* customer;
    const Service* service;
    int amount;
    Rupiah base, express, discount;
    Rupiah total() const { return base + express - discount; }
};

std::string line40(const std::string& label, const Rupiah& money) {
    std::ostringstream out;
    out << "  " << std::left << std::setw(24) << label << std::right
        << std::setw(14) << money;
    return out.str();
}
```

Kode 15.7 (Lanjutan) *laundry.cpp: program Profitina Laundry yang lengkap dalam C++.*

```
std::ostream& operator<<(std::ostream& os, const Receipt& r) {
    os << std::string(40, '=') << '\n'
       << std::left << std::setw(31) << "PROFITINA LAUNDRY" << "No. " << r.number
       << '\n'
       << r.customer->name << (r.customer->member ? " (member)" : "") << '\n'
       << r.service->name << ", " << r.service->describe(r.amount) << " x "
       << r.service->rate << '\n'
       << line40("Base price", r.base) << '\n';
    if (r.express.value() != 0) os << line40("Express +50%", r.express) << '\n';
    if (r.discount.value() != 0)
        os << line40("Member discount 10%", -r.discount) << '\n';
    return os << line40("TOTAL", r.total()) << '\n';
}

struct Total {                                  // one line of the daily report
    std::string key, name;
    int count = 0;
    Rupiah money;
};

bool operator<(const Total& a, const Total& b) {   // largest money first
    if (a.money != b.money) return a.money > b.money;
    return a.key < b.key;
}

class Shop {
public:
    void load(const std::string& dir) {
        for (auto& f : readTable(dir + "/services.txt"))
            if (f.size() == 4 && isNumber(f[3]))
                services_[f[0]] =
                    Service{f[0], f[1], f[2], Rupiah(std::stoll(f[3]))};
        for (auto& f : readTable(dir + "/customers.txt"))
            if (f.size() == 3)
                customers_[f[0]] = Customer{f[0], f[1], f[2] == "yes"};
        for (auto& f : readTable(dir + "/orders.txt")) take(f);
    }

    void printReceipts(std::ostream& os) const {
        for (const Receipt& r : receipts_) os << r;
        os << std::string(40, '=') << '\n';
        for (const std::string& why : refused_) os << "refused: " << why << '\n';
    }
```

Kode 15.7 (Lanjutan) *laundry.cpp: program Profitina Laundry yang lengkap dalam C++.*

```
    void report(std::ostream& os) const {
        std::map<std::string, Total> byService, byCustomer;
        Rupiah grand, discounts;
        for (const Receipt& r : receipts_) {
            add(byService[r.service->code], r.service->code, r.service->name, r);
            add(byCustomer[r.customer->id], r.customer->id, r.customer->name, r);
            grand += r.total();
            discounts += r.discount;
        }
        os << "DAILY REPORT - " << receipts_.size() << " receipts, "
           << refused_.size() << " orders refused\n";
        section(os, "Revenue by service (largest first)", byService);
        section(os, "Revenue by customer (largest first)", byCustomer);
        os << std::left << std::setw(26) << "Member discounts given" << std::right
           << std::setw(14) << discounts << '\n'
           << std::left << std::setw(26) << "GRAND TOTAL" << std::right
           << std::setw(14) << grand << '\n';
    }

private:
    void take(const std::vector<std::string>& f) {
        std::string no = f.empty() ? "?" : f[0];
        if (f.size() != 5) return refuse(no, "expected 5 fields");
        auto c = customers_.find(f[1]);
        if (c == customers_.end()) return refuse(no, "unknown customer " + f[1]);
        auto s = services_.find(f[2]);
        if (s == services_.end()) return refuse(no, "unknown service " + f[2]);
        if (!isNumber(f[3]) || std::stoi(f[3]) < 1) return refuse(no, "bad amount");
        if (f[4] != "yes" && f[4] != "no") return refuse(no, "express is yes/no");

        Receipt r{no, &c->second, &s->second, std::stoi(f[3]), {}, {}, {}};
        r.base = r.service->price(r.amount);
        if (f[4] == "yes") r.express = r.base.percent(50);
        if (r.customer->member) r.discount = (r.base + r.express).percent(10);
        receipts_.push_back(r);
    }

    void refuse(const std::string& no, const std::string& why) {
        refused_.push_back("order " + no + ", " + why);
    }
```

Kode 15.7 (Lanjutan) *laundry.cpp: program Profitina Laundry yang lengkap dalam C++.*

```
    static void add(Total& t, const std::string& key, const std::string& name,
                    const Receipt& r) {
        t.key = key;
        t.name = name;
        ++t.count;
        t.money += r.total();
    }

    static void section(std::ostream& os, const std::string& title,
                        const std::map<std::string, Total>& totals) {
        std::vector<Total> rows;
        for (const auto& kv : totals) rows.push_back(kv.second);
        std::sort(rows.begin(), rows.end());   // uses our operator<
        os << title << '\n';
        for (const Total& t : rows)
            os << "  " << std::left << std::setw(4) << t.key << std::setw(20)
               << t.name << std::right << std::setw(2) << t.count << std::setw(12)
               << t.money << '\n';
    }

    std::map<std::string, Service> services_;
    std::map<std::string, Customer> customers_;
    std::vector<Receipt> receipts_;
    std::vector<std::string> refused_;
};

int main(int argc, char* argv[]) {
    std::string dir = argc > 1 ? argv[1] : "data";
    Shop shop;
    shop.load(dir);
    shop.printReceipts(std::cout);
    std::cout << '\n';
    shop.report(std::cout);                    // the same function writes...
    std::ofstream file("report.txt");
    shop.report(file);                         // ...to the screen and to a file
    std::cout << "\nreport saved to report.txt\n";
    return 0;
}
```

Programnya sekitar 200 baris. Beginilah bagian-bagiannya bekerja sama.

Membaca berkas. `readTable` membuka berkas dengan `std::ifstream`, melompati baris kosong dan komentar, lalu memecah setiap baris lainnya pada `;` dengan `split` dari Bab 14. Hasilnya vektor baris, masing-masing berupa vektor string. Satu perbaikan kecil dibanding Bab 14: bila sebuah baris diakhiri `;`, `split` menambahkan kolom terakhir yang kosong, sehingga baris dengan kolom terakhir kosong tetap memiliki jumlah kolom yang benar dan diperiksa dengan semestinya. Bila sebuah berkas tidak ada, stream-nya tidak mengirim baris apa pun, dan toko dimulai dengan daftar harga kosong, sehingga setiap pesanan ditolak dengan alasan yang jelas.

`Service` dan `Customer`. Dua struktur kecil tanpa invarian yang perlu dijaga: setelah dimuat, tidak ada yang mengubahnya. `Service::price` menghitung harga dasar sebagai `Rupiah`, untuk gram atau potong, dan `describe` mencetak jumlah itu dalam satuannya.

`Receipt`. Satu pesanan yang diterima beserta seluruh uangnya. Struk memegang *pointer* ke pelanggan dan layanannya, bukan salinan, karena pelanggan dan layanan itu sudah ada di dalam *map* milik toko dan tidak boleh digandakan. `total()` adalah Persamaan (15.5).

Mencetak struk dengan `<<`. `operator<<(std::ostream& os, const Receipt& r)` mencetak seluruh blok struk: garis pembatas, nama toko dengan nomor pesanan, pelanggan (dengan "(member)" bila berlaku), baris layanan, lalu satu baris untuk setiap jumlah, dengan baris ekspres

dan diskon hanya bila nilainya tidak nol. Setiap baris jumlah diformat oleh `line40`, yang memakai `std::ostringstream`, stream yang menulis ke dalam string, untuk meratakan `Rupiah` ke kanan dengan `setw(14)`. Karena operatornya menerima `std::ostream&` apa saja, kode yang sama dapat mencetak struk ke berkas atau ke string.

`Total` dan `operator<`-nya. Satu baris laporan: kunci (kode layanan atau id pelanggan), nama, banyaknya, dan jumlah uang. Laporan harus mendaftar jumlah terbesar lebih dulu, dan untuk jumlah yang sama, menurut urutan kuncinya. Urutan itu kita ungkapkan *sekali*, sebagai `operator<` untuk `Total`: "`a` datang sebelum `b` bila `a` punya uang lebih banyak, atau uangnya sama dan kuncinya lebih kecil". Lalu `std::sort(rows.begin(), rows.end())`, yang memakai `<` setiap kali membandingkan dua elemen, mengurutkan laporan tanpa petunjuk tambahan apa pun. Inilah overloading pada saat paling bergunanya: kita mengajari algoritma standar apa arti "sebelum" untuk tipe kita.

`Shop`. Kelas yang memiliki segalanya. Data *private*-nya adalah dua `std::map` dan dua vektor. `std::map<std::string, Service>` adalah sebuah *kamus*: ia menyimpan pasangan kunci (kode layanan) dan nilai (`Service`), menjaganya terurut menurut kunci, dan menemukan sebuah kunci dalam $O(\log n)$ langkah. `services_[code] = service` menyisipkan atau mengganti; `services_.find(code)` mengembalikan iterator ke pasangannya, atau `services_.end()` bila kuncinya tidak ada; `it->second` adalah nilai pasangan itu. `load` mengisi kedua *map* lalu memanggil `take` untuk setiap baris pesanan. `take` memeriksa kolom satu per satu, kembali lebih awal lewat `refuse` pada masalah pertama, persis seperti `parseOrder` di Bab 13; bila semuanya beres, ia membangun `Receipt` dan menghitung ketiga jumlahnya dengan aturan Subbab 15.5.2. Perhatikan bagaimana pemeriksaannya terbaca: `if (c == customers_.end()) return refuse(no, "unknown customer " + f[1]);`, satu baris per aturan.

Laporannya. `report` mengelompokkan struk dua kali, per layanan dan per pelanggan, dengan `std::map<std::string, Total>`: `byService[r.service->code]` menemukan total untuk kode itu, *dan menciptakan total kosong pada kali pertama*, satu-satunya sifat mengejutkan dari `operator[]` pada *map*, dan di sini justru praktis. Fungsi bantu statis `section` menyalin total-total ke vektor, mengurutkannya dengan `operator<` kita, lalu mencetaknya. Di akhir menyusul diskon yang diberikan dan total keseluruhan.

Satu laporan, dua tujuan. `report` menerima `std::ostream&`. `main` memanggilnya dua kali, sekali dengan `std::cout` dan sekali dengan `std::ofstream` yang dibuka pada `report.txt`. `std::cout` maupun `std::ofstream` *adalah* `std::ostream` (pustaka standar memakai pewarisan dari Subbab 14.6 tepat untuk ini), sehingga fungsi yang sama menulis ke layar dan ke berkas, dan keduanya tidak mungkin berbeda. Berkasnya ditutup oleh destruktor `ofstream` ketika `main` berakhir.

15.5.4 Mengikuti dua pesanan melewati `take`

Ikuti pesanan 1006 lebih dulu. `readTable` sudah memecah barisnya menjadi lima string `1006`, `C05`, `W`, `3000`, dan `no`, jadi jumlah kolomnya benar. `customers_.find("C05")` menyusuri kunci-kunci *map* yang terurut, C01, C02, C03, C04, tidak menemukan C05, dan mengembalikan `customers_.end()`. Uji `c == customers_.end()` bernilai benar, sehingga `take` memanggil `refuse("1006", "unknown customer C05")`, yang menambahkan teks itu ke `refused_`, lalu kembali. Tidak ada hal lain yang terjadi: tidak ada struk, tidak ada total setengah jadi. Layanan, jumlah, dan tanda ekspres baris itu tidak pernah diperiksa, karena masalah pertama sudah cukup untuk menolak pesanan.

Sekarang pesanan 1002, `1002;C02;WI;5200;yes`. Pelanggan C02 ditemukan (Pak Budi Santoso, bukan anggota), layanan WI ditemukan (cuci setrika, 10.000 per kg), `5200` adalah bilangan paling banyak enam digit dan paling kecil 1, dan `yes` adalah tanda ekspres yang sah. `take` membangun struk dengan pointer ke pelanggan dan layanan, menghitung harga dasar $(5200 \times 10000 + 500)/1000 = 52{,}000$, biaya ekspres `r.base.percent(50)`, yaitu 26,000, dan tanpa diskon, karena `r.customer->member` bernilai salah. Struk itu masuk ke ujung `receipts_`, dan belakangan `operator<<` mencetaknya dengan baris "Base price", "Express +50%", dan "TOTAL", melewati baris diskon karena diskonnya nol.

Kode 15.8 adalah program yang sama dalam Python. `@dataclass` membangkitkan konstruktor kelas-kelas rekaman biasa; `Receipt.__str__` berperan sebagai `operator<<`; `Total.__lt__` adalah urutan pengurutan yang dipakai `sorted`; `dict` menggantikan `std::map` (dan `setdefault` menggantikan penciptaan-saat-dipakai milik `[]`); dan fungsi laporan menerima objek apa saja yang punya metode `write`, sehingga ia menulis ke `sys.stdout` dan ke berkas, persis seperti versi C++ menulis ke `std::cout` dan ke `ofstream`.

Kode 15.8 `laundry.py`: program Profitina Laundry yang lengkap dalam Python.

```python
# laundry.py - Chapter 15: the complete Profitina Laundry case study in Python.
# Reads services, customers and orders from data/, prints one receipt per
# order and the daily report, and saves the report to report.txt.
import sys
from dataclasses import dataclass, field

from rupiah import Rupiah


def read_table(path):
    """The non-comment lines of a data file, each split at ';'."""
    with open(path, encoding="utf-8") as f:
        return [line.split(";") for line in f.read().splitlines()
                if line and not line.startswith("#")]


def is_number(s):
    return 0 < len(s) <= 6 and all(c in "0123456789" for c in s)


@dataclass
class Service:
    code: str
    name: str
    unit: str                                   # "kg" or "pcs"
    rate: Rupiah

    def price(self, amount):                    # amount: grams or pieces
        if self.unit == "kg":
            return Rupiah((amount * self.rate.value() + 500) // 1000)
        return self.rate * amount

    def describe(self, amount):                 # "3.450 kg" or "2 pcs"
        if self.unit != "kg":
            return f"{amount} pcs"
        return f"{amount // 1000}.{amount % 1000:03d} kg"


@dataclass
class Customer:
    id: str
    name: str
    member: bool
```

Kode 15.8 (Lanjutan) *laundry.py: program Profitina Laundry yang lengkap dalam Python.*

```
@dataclass
class Receipt:
    number: str
    customer: Customer
    service: Service
    amount: int
    base: Rupiah = field(default_factory=Rupiah)
    express: Rupiah = field(default_factory=Rupiah)
    discount: Rupiah = field(default_factory=Rupiah)

    def total(self):
        return self.base + self.express - self.discount

    def __str__(self):                          # plays the role of operator<<
        lines = ["=" * 40, f"{'PROFITINA LAUNDRY':<31}No. {self.number}",
                 self.customer.name + (" (member)" if self.customer.member else ""),
                 f"{self.service.name}, {self.service.describe(self.amount)} x "
                 f"{self.service.rate}", line40("Base price", self.base)]
        if self.express.value() != 0:
            lines.append(line40("Express +50%", self.express))
        if self.discount.value() != 0:
            lines.append(line40("Member discount 10%", -self.discount))
        lines.append(line40("TOTAL", self.total()))
        return "\n".join(lines) + "\n"


def line40(label, money):
    return f"  {label:<24}{money:>14}"


@dataclass
class Total:                                    # one line of the daily report
    key: str = ""
    name: str = ""
    count: int = 0
    money: Rupiah = field(default_factory=Rupiah)

    def __lt__(self, other):                    # largest money first
        if self.money != other.money:
            return self.money > other.money
        return self.key < other.key
```

Kode 15.8 (Lanjutan) *laundry.py: program Profitina Laundry yang lengkap dalam Python.*

```
class Shop:
    def __init__(self):
        self._services, self._customers = {}, {}
        self._receipts, self._refused = [], []

    def load(self, folder):
        for f in read_table(folder + "/services.txt"):
            if len(f) == 4 and is_number(f[3]):
                self._services[f[0]] = Service(f[0], f[1], f[2], Rupiah(int(f[3])))
        for f in read_table(folder + "/customers.txt"):
            if len(f) == 3:
                self._customers[f[0]] = Customer(f[0], f[1], f[2] == "yes")
        for f in read_table(folder + "/orders.txt"):
            self._take(f)

    def _take(self, f):
        no = f[0] if f else "?"
        if len(f) != 5:
            return self._refuse(no, "expected 5 fields")
        c = self._customers.get(f[1])
        if c is None:
            return self._refuse(no, "unknown customer " + f[1])
        s = self._services.get(f[2])
        if s is None:
            return self._refuse(no, "unknown service " + f[2])
        if not is_number(f[3]) or int(f[3]) < 1:
            return self._refuse(no, "bad amount")
        if f[4] not in ("yes", "no"):
            return self._refuse(no, "express is yes/no")
        r = Receipt(no, c, s, int(f[3]))
        r.base = s.price(r.amount)
        if f[4] == "yes":
            r.express = r.base.percent(50)
        if c.member:
            r.discount = (r.base + r.express).percent(10)
        self._receipts.append(r)

    def _refuse(self, no, why):
        self._refused.append(f"order {no}, {why}")
```

Kode 15.8 (Lanjutan) *laundry.py: program Profitina Laundry yang lengkap dalam Python.*

```
    def print_receipts(self, out):
        for r in self._receipts:
            out.write(str(r))
        out.write("=" * 40 + "\n")
        for why in self._refused:
            out.write(f"refused: {why}\n")

    def report(self, out):
        by_service, by_customer = {}, {}
        grand, discounts = Rupiah(), Rupiah()
        for r in self._receipts:
            _add(by_service.setdefault(r.service.code, Total()),
                 r.service.code, r.service.name, r)
            _add(by_customer.setdefault(r.customer.id, Total()),
                 r.customer.id, r.customer.name, r)
            grand += r.total()
            discounts += r.discount
        out.write(f"DAILY REPORT - {len(self._receipts)} receipts, "
                  f"{len(self._refused)} orders refused\n")
        _section(out, "Revenue by service (largest first)", by_service)
        _section(out, "Revenue by customer (largest first)", by_customer)
        out.write(f"{'Member discounts given':<26}{discounts:>14}\n")
        out.write(f"{'GRAND TOTAL':<26}{grand:>14}\n")


def _add(t, key, name, r):
    t.key, t.name = key, name
    t.count += 1
    t.money += r.total()


def _section(out, title, totals):
    out.write(title + "\n")
    for t in sorted(totals.values()):          # uses our __lt__
        out.write(f"  {t.key:<4}{t.name:<20}{t.count:>2}{t.money:>12}\n")
```

Kode 15.8 (Lanjutan) *laundry.py: program Profitina Laundry yang lengkap dalam Python.*

```
def main():
    folder = sys.argv[1] if len(sys.argv) > 1 else "data"
    shop = Shop()
    shop.load(folder)
    shop.print_receipts(sys.stdout)
    print()
    shop.report(sys.stdout)                    # the same function writes...
    with open("report.txt", "w", encoding="utf-8") as f:
        shop.report(f)                         # ...to the screen and to a file
    print("\nreport saved to report.txt")


if __name__ == "__main__":
    main()
```

15.6 Menjalankannya

```
$ g++ -std=c++17 -O2 -Wall -Wextra cpp/laundry.cpp -o laundry
$ ./laundry
$ python3 python/laundry.py
```

Kedua program mencetak berikut ini dan menulis bagian laporannya ke `report.txt`:

```
======================================
PROFITINA LAUNDRY             No. 1001
Bu Rina Wulandari (member)
Wash & fold, 3.450 kg x Rp 7.000
  Base price                  Rp 24.150
  Member discount 10%         -Rp 2.415
  TOTAL                       Rp 21.735
======================================
PROFITINA LAUNDRY             No. 1002
Pak Budi Santoso
Wash & iron, 5.200 kg x Rp 10.000
  Base price                  Rp 52.000
  Express +50%                Rp 26.000
  TOTAL                       Rp 78.000
======================================
PROFITINA LAUNDRY             No. 1003
Mbak Sari Dewi (member)
Bed cover, 2 pcs x Rp 25.000
  Base price                  Rp 50.000
  Member discount 10%         -Rp 5.000
  TOTAL                       Rp 45.000
======================================
PROFITINA LAUNDRY             No. 1004
Bu Rina Wulandari (member)
Iron only, 2.000 kg x Rp 5.000
  Base price                  Rp 10.000
  Member discount 10%         -Rp 1.000
  TOTAL                        Rp 9.000
======================================
PROFITINA LAUNDRY             No. 1005
Mas Joko Prasetyo
Shoes, 1 pcs x Rp 35.000
  Base price                  Rp 35.000
  Express +50%                Rp 17.500
  TOTAL                       Rp 52.500
======================================
PROFITINA LAUNDRY             No. 1007
Pak Budi Santoso
Wash & fold, 4.100 kg x Rp 7.000
  Base price                  Rp 28.700
  TOTAL                       Rp 28.700
======================================
PROFITINA LAUNDRY             No. 1008
Mbak Sari Dewi (member)
Wash & iron, 2.750 kg x Rp 10.000
  Base price                  Rp 27.500
  Express +50%                Rp 13.750
  Member discount 10%         -Rp 4.125
  TOTAL                       Rp 37.125
======================================
refused: order 1006, unknown customer C05
refused: order 1009, unknown service DC
```

```
DAILY REPORT - 7 receipts, 2 orders refused
Revenue by service (largest first)
  WI  Wash & iron          2  Rp 115.125
  SH  Shoes                1   Rp 52.500
  W   Wash & fold          2   Rp 50.435
  BC  Bed cover            1   Rp 45.000
  I   Iron only            1    Rp 9.000
Revenue by customer (largest first)
  C02 Pak Budi Santoso     2  Rp 106.700
  C03 Mbak Sari Dewi       2   Rp 82.125
  C04 Mas Joko Prasetyo    1   Rp 52.500
  C01 Bu Rina Wulandari    2   Rp 30.735
Member discounts given         Rp 12.540
GRAND TOTAL                   Rp 272.060

report saved to report.txt
```

Periksa struk yang paling rumit dengan tangan, pesanan 1008 untuk Mbak Sari Dewi, seorang anggota, 2.750 g cuci setrika, ekspres. Harga dasarnya $(2750 \times 10000 + 500)/1000 = 27{,}500$. Biaya ekspresnya 50% dari itu menurut Persamaan (15.1): $(27{,}500 \times 50 + 50)/100 = 13{,}750$. Diskonnya 10% dari $27{,}500 + 13{,}750 = 41{,}250$, yaitu $(412{,}500 + 50)/100 = 4{,}125$. Menurut Persamaan (15.5), totalnya $27{,}500 + 13{,}750 - 4{,}125 = 37{,}125$. Struknya cocok.

Lalu periksa laporannya. Cuci setrika punya dua struk, $78{,}000 + 37{,}125 = 115{,}125$, jumlah terbesar, sehingga berada paling atas. Pelanggan C01 punya $21{,}735 + 9{,}000 = 30{,}735$ dan berada paling bawah di antara pelanggan. Keempat diskon anggota adalah $2{,}415 + 5{,}000 + 1{,}000 + 4{,}125 = 12{,}540$. Dan kedua pesanan yang ditolak didaftar beserta alasannya alih-alih lenyap.

15.7 Mengujinya

Daftar uji bab ini menyebutkan ketiga program; `laundry` membaca berkas datanya dari folder `data/`.

```
# name | C++ sources | Python script | stdin | arguments
rupiah_demo | rupiah_demo.cpp | rupiah_demo.py | - | -
fraction | fraction.cpp | fraction.py | - | -
laundry | laundry.cpp | laundry.py | - | -
$ python3 tests/run_tests.py --sanitize
```

```
rupiah_demo                C++ ok   Python ok
fraction                   C++ ok   Python ok
laundry                    C++ ok   Python ok
all programs agree
```

Kelas yang memiliki operator layak diuji terhadap *hukum-hukum* yang seharusnya dipatuhi operatornya, karena pembaca `a + b` mengandalkannya. Untuk `Rupiah`: $a + b = b + a$; $(a + b) - b = a$; $a + (-a) = 0$; $3 \times a = a + a + a$; dan tepat satu dari $a < b$, $a = b$, $b < a$ yang berlaku. Untuk `Fraction`: $x + 0 = x$, $x \times 1 = x$, $x/x = 1$ untuk $x \neq 0$, dan $\frac{a}{b} \times \frac{b}{a} = 1$. Soal 3 mengubah hukum-hukum ini menjadi program kecil yang mencobanya pada banyak nilai. Hukum yang gagal langsung menunjuk operator yang salah.

15.8 Membuatnya Lebih Baik

- **Kunci urutan yang lebih baik untuk pelanggan.** Laporan pelanggan diurutkan menurut uang; pemilik mungkin lebih suka menurut nama. Karena urutan itu tinggal di satu `operator<`, urutan

kedua adalah fungsi perbandingan kedua yang diberikan kepada `std::sort` sebagai argumen ketiganya, seperti Bab 16 mengurutkan waktu pengambilan.

- **Pesanan yang ditolak ke berkas tersendiri.** Pegawai di meja kasir memerlukan daftar pesanan yang ditolak untuk menelepon pelanggannya. `printReceipts` sudah menerima `std::ostream&`, jadi menulis baris-baris penolakan ke `refused.txt` hanya memerlukan stream kedua.
- **Berat sebagai tipe tersendiri.** Kelas `Grams`, seperti `Rupiah`, akan membuat `Rupiah + Grams` mustahil dan memberi `describe` rumah yang wajar sebagai `operator<<` untuk `Grams`.
- **Beberapa hari.** Kolom tanggal di `orders.txt` dan sebuah `std::map` dari tanggal ke `Shop` akan memberi laporan per hari dan ringkasan sebulan. Bab 16 (Soal 16.4) menunjukkan bagaimana perkalian matriks mengubah kilogram harian menjadi pendapatan harian dalam satu langkah.

15.9 Memakainya Sungguhan

Program ini cukup lengkap untuk disesuaikan dengan usaha kecil sungguhan, dan penyesuaiannya mengikuti berkas-berkasnya. Penjahit menaruh “hem trousers;pcs;15000” dan “take in a dress;pcs;45000” di `services.txt`; tempat cuci mobil menaruh golongan kendaraan; usaha katering menarik biaya per porsi. Berkas pelanggan mendapat kolom persentase diskon bila anggota tidak semuanya mendapat diskon yang sama. Selebihnya, yaitu pemeriksaan, struk, dan laporan, tetap sama.

Satu hal yang khas di Indonesia patut dipikirkan sejak awal: pecahan uang terkecil yang masih lazim dipakai di kasir adalah koin ratusan rupiah, dan banyak toko membulatkan total belanja ke ratusan terdekat karena koin yang lebih kecil hampir tidak pernah beredar. Bila Bu Rina ingin totalnya dibulatkan ke Rp100 terdekat, jangan menaburkan pembulatan itu di setiap baris struk. Tambahkan satu fungsi anggota ke `Rupiah`, misalnya `roundTo(100)`, yang memakai trik yang sama dengan rumus persentase di Subbab 15.3.1, yaitu menambahkan setengah pembagi sebelum membagi, lalu terapkan hanya pada total akhir, sekali, dan cetak selisih pembulatannya sebagai baris tersendiri di struk agar pelanggan melihatnya. Dengan begitu aturan pembulatan tetap tinggal di satu tempat, laporan harian tetap menjumlahkan angka yang sama dengan yang tercetak di struk, dan bila kelak aturan berubah (misalnya pembulatan ke Rp500), hanya satu argumen yang perlu diganti.

Contoh kecilnya: total Rp37.125 dibulatkan ke ratusan terdekat menjadi Rp37.100, karena 25 rupiah kurang dari setengah seratus; total Rp21.735 menjadi Rp21.700; dan total Rp45.050 menjadi Rp45.100, karena tepat setengah dibulatkan ke atas. Perhatikan bahwa selisih pembulatan sepanjang hari dapat positif maupun negatif, sehingga dalam jangka panjang toko tidak diuntungkan dan pelanggan tidak dirugikan; laporan harian sebaiknya tetap mencantumkan jumlah selisih itu agar pemilik dapat memeriksanya setiap malam dengan tenang. Tuliskan ketiga contoh ini sebagai uji sebelum menulis fungsinya, lalu pastikan versi C++ dan versi Python menghasilkan angka yang sama.

Pertahankan tiga kebiasaan dari bab ini ketika menyesuaikannya. Simpanlah uang dalam tipe tersendiri, agar selalu dibulatkan dan dicetak dengan satu cara. Letakkan aturan usaha, seperti biaya tambahan dan diskon, di satu tempat yang terlihat, seperti Subbab 15.5.2 dan beberapa baris `take` yang melaksanakannya, agar pemilik dapat membaca dan menyetujuinya. Dan periksalah program dengan tangan terhadap satu struk nyata dari toko sebelum memercayakan pembukuan sebulan kepadanya. Program membantu pemilik menghitung; keputusan tentang harga dan diskon tetap milik pemilik.

15.10 Kesalahan yang Sering Terjadi, dan Cara Menghindarinya

- **Mengembalikan salinan dari `+=`.** `Rupiah operator+=(...)` (tanpa `&`) terkompilasi, tetapi `(a += b) += c` lalu mengubah salinan sementara, bukan `a`. Penugasan gabungan mengembalikan `*this` lewat referensi.
- **Menjadikan `operator+` sebagai anggota.** Operator itu lalu memerlukan objek Anda di sebelah kiri, sehingga `3 * iron` atau `2 + half` gagal dikompilasi. Tulislah operator simetris sebagai non-anggota yang dibangun di atas `+=`.
- **Mendefinisikan `==` dan `!=` (atau `<` dan `>`) secara terpisah.** Cepat atau lambat keduanya akan bertentangan. Definisikan `==` dan `<`, lalu tulis yang lain dengan memakai keduanya.
- **`<` yang bukan urutan ketat.** `std::sort` memerlukan perbandingan yang membuat `a < a` bernilai salah dan yang "baik `a < b` maupun `b < a` salah" berarti "setara". Menulis `<=` alih-alih `<` dalam `operator<` milik `Total` melanggar aturan ini dan merupakan perilaku tak terdefinisi bagi `std::sort`.
- **Lupa `explicit` pada konstruktor pengubah.** Setiap konstruktor satu argumen adalah konversi diam-diam kecuali dibuat `explicit`. Putuskan untuk masing-masing: apakah setiap nilai bertipe parameter itu sungguh merupakan nilai tipe Anda?
- **Luapan pada `a * b / gcd`.** Bagi lebih dulu, seperti Persamaan (15.3). Penalaran yang sama berlaku untuk persentase dari jumlah yang sangat besar.
- **Makna yang mengejutkan.** Operator yang melakukan sesuatu yang tidak akan ditebak pembaca dari lambangnya membuat kode menjadi teka-teki. Pakailah fungsi bernama.
- **Python: mengembalikan `NotImplemented` versus memunculkan eksepsi.** Bila `__add__` tidak dapat menangani operan yang lain, kelas Python yang cermat mengembalikan nilai khusus `NotImplemented` agar Python dapat mencoba metode terpantul milik operan lain; kelas-kelas kecil kita mengandaikan operannya benar, dan itu wajar untuk program yang mengendalikan semua pemanggilannya sendiri.

15.11 Ringkasan Bab

- Operator yang di-*overload* adalah fungsi bernama `operator` diikuti lambangnya; `a + b` pada objek kelas menjadi `operator+(a, b)` atau `a.operator+(b)`. Presedensi, banyaknya operan, dan makna bawaan tidak dapat diubah, dan beberapa operator sama sekali tidak dapat di-*overload*.
- Operator yang mengubah operan kirinya (`+=`, `-=`) adalah anggota yang mengembalikan `*this` lewat referensi; operator simetris (`+`, `-`, `==`, `<`) adalah non-anggota yang dibangun di atasnya; `<<` untuk keluaran adalah non-anggota yang mengembalikan stream-nya.
- Kelas `Rupiah` menyimpan uang dalam rupiah utuh, membulatkan persentase dengan satu cara yang terdokumentasi (Persamaan (15.1)), mencetak jumlah dengan gaya Indonesia, dan menolak hal tak masuk akal seperti `Rupiah * Rupiah` serta konversi diam-diam dari bilangan (konstruktor `explicit`).
- Kelas `Fraction` menjaga nilainya dalam bentuk paling sederhana dengan algoritma Euclid (Persamaan (15.2)) dan menjumlahkan dengan KPK (Persamaan (15.3), Persamaan (15.4)); konstruktornya yang tidak `explicit` membuat `2 + half` berjalan.
- Program Profitina Laundry yang lengkap membaca daftar harga, pelanggan, dan pesanan dari berkas, memeriksa setiap pesanan, mencetak struk melalui `operator<<`, mengurutkan laporannya dengan `operator<`, memakai `std::map` untuk pencarian, dan menulis satu fungsi laporan ke layar dan ke berkas.

- Python menawarkan hal yang sama melalui metode khusus (`__add__`, `__rmul__`, `__lt__`, `__str__`, `__format__`) dan `functools.total_ordering`; kedua versi setiap program mencetak keluaran yang identik.

15.12 Soal Latihan

1. Tambahkan `operator/` ke `Rupiah` untuk membagi sejumlah uang menjadi k bagian yang sama, dibulatkan setengah ke atas, dan fungsi anggota `split(k)` yang mengembalikan `std::vector<Rupiah>` berisi k bagian yang *jumlahnya persis* sama dengan jumlah semula. Bagilah Rp100.000 menjadi 3 bagian.
2. Tulislah fungsi faktorial dalam dua versi, iteratif dan rekursif, keduanya mengembalikan `long long`, lalu cetak $n!$ untuk $n = 0,5,10,15,20,21$. Apa yang terjadi pada 21, dan bagaimana program dapat mendeteksinya sebelum terjadi?
3. Tulislah program yang memeriksa hukum-hukum pada Subbab 15.7 untuk `Rupiah` pada semua pasangan jumlah $-1000, -1, 0, 1, 999, 24150$ dan untuk `Fraction` pada semua pasangan $\frac{1}{2}, -\frac{2}{3}, \frac{5}{4}, 3, 0$ (lewati pembagian dengan nol), lalu cetak berapa pemeriksaan yang lolos.
4. Jumlah kuadrat $1^2 + 2^2 + \cdots + n^2$ dapat dihitung dengan perulangan atau dengan rumus $\frac{n(n+1)(2n+1)}{6}$. Tulislah keduanya, bandingkan untuk $n = 1, \ldots, 1000$, dan jelaskan mengapa rumusnya harus dihitung sebagai `n * (n + 1) / 2 * (2 * n + 1) / 3` atau dengan urutan cermat lain untuk menghindari luapan sekaligus pembulatan yang salah.
5. Ubahlah `laundry.cpp` agar diskon anggota dibaca dari berkas pelanggan (kolom keempat berisi persentase, misalnya 10 atau 15), dan bukan anggota mendapat 0. Apa yang berubah pada keluaran bila diskon Mbak Sari Dewi 15%?
6. Tambahkan `operator<<` untuk `Fraction` yang mencetak pecahan campuran bila nilai mutlak pecahannya lebih dari 1: $\frac{7}{2}$ sebagai `3 1/2` dan $-\frac{7}{2}$ sebagai `-3 1/2`. Mengapa ini perubahan yang meragukan pada operator keluaran standar, dan apa yang sebaiknya Anda lakukan?

15.13 Penyelesaian Soal Latihan

1. `Rupiah operator/(const Rupiah& a, long long k)` mengembalikan `Rupiah((a.value() * 2 + k) / (2 * k))` untuk jumlah tak negatif, yang membulatkan setengah ke atas (inilah Persamaan (15.1) dengan pembagi k alih-alih 100). Bagian-bagian yang dibulatkan tidak selalu berjumlah tepat: 100,000/3 menghasilkan 33,333 tiga kali, yaitu 99,999. `split(k)` memperbaikinya: hitung bagian bulat $q = \lfloor a/k \rfloor = 33{,}333$ dan sisa $r = a \bmod k = 1$, lalu berikan $q + 1$ kepada r bagian pertama dan q kepada sisanya: $33{,}334 + 33{,}333 + 33{,}333 = 100{,}000$. Beginilah tagihan dibagi adil di antara teman tanpa kehilangan satu rupiah pun.

2. Versi iteratif mengalikan `f *= i` untuk $i = 2, \ldots, n$; versi rekursif mengembalikan `n <= 1 ? 1 : n * factorial(n - 1)`. Keduanya sama sampai $20! = 2{,}432{,}902{,}008{,}176{,}640{,}000$, faktorial terbesar yang muat dalam `long long`. 21! kira-kira 5.1×10^{19}, melebihi batas maksimum sekitar 9.2×10^{18}: perkaliannya meluap, yang untuk bilangan bertanda merupakan perilaku tak terdefinisi, dan dalam praktik tercetak bilangan yang salah, sering negatif (*sanitizer* dari Subbab 13.11 justru menghentikan programnya). Untuk mendeteksinya lebih dulu, periksalah sebelum mengalikan: bila `f > LLONG_MAX / i`, hasil kalinya tidak akan muat. Dalam Python, bilangan bulat tumbuh tanpa batas, sehingga `math.factorial(21)` benar begitu saja; ini perbedaan nyata antara kedua bahasa yang patut diingat.

3. Telusuri semua pasangan dengan dua perulangan berbasis rentang bersarang, dan untuk setiap pasangan uji setiap hukum dengan `==`, hitung yang berhasil dan cetak setiap kegagalan beserta nilainya. Dengan enam jumlah ada 36 pasangan; dengan keempat hukum `Rupiah` yang berlaku pada pasangan $(a + b = b + a$, $(a + b) - b = a$, trikotomi, dan $3 \times a = a + a + a$ yang diterapkan pada elemen pertama), setiap pemeriksaan lolos. Untuk pecahan, $x/x = 1$ dilewati untuk $x = 0$, dan semua pemeriksaan lainnya lolos. Nilai program ini tampak ketika kelak Anda mengubah sebuah operator: perubahan yang salah langsung muncul sebagai hukum yang gagal.

4. Perulangannya menjumlahkan `1LL * i * i`; rumusnya memberi nilai yang sama, misalnya 385 untuk $n = 10$ dan 333,833,500 untuk $n = 1000$. Hasil kali $n(n + 1)(2n + 1)$ selalu habis dibagi 6, tetapi program yang membagi *sebagian* darinya terlalu awal bisa membulatkan: $n/6$ saja salah setiap kali 6 tidak membagi n. Cara yang aman adalah membagi dengan faktor yang pasti membagi apa yang sedang Anda pegang: $n(n + 1)$ selalu genap, sehingga `n * (n + 1) / 2` tepat; lalu mengalikannya dengan $2n + 1$ memberi bilangan yang habis dibagi 3, karena $n(n + 1)(2n + 1)$ habis dibagi 3; sehingga `/ 3` pun tepat. Membagi lebih awal juga menjaga nilai antara tetap kecil, yang menunda luapan.

5. Tambahkan kolom keempat di setiap baris `customers.txt`, simpan sebagai `int discount` di `Customer`, periksa di `load` dengan `isNumber` dan batas 0 sampai 100, lalu di `take` tulis `r.discount = (r.base + r.express).percent(r.customer->discount);` dan hapus uji `member` (diskon 0 menghasilkan Rp0). Dengan 15% untuk Mbak Sari Dewi, pesanan 1003 mendapat diskon $(50{,}000 \times 15 + 50)/100 = 7{,}500$ dan total 42,500; pesanan 1008 mendapat $(41{,}250 \times 15 + 50)/100 = 6{,}188$ (nilai tepatnya 6.187,5 dibulatkan ke atas) dan total 35,062. Baris diskonnya lalu seharusnya berbunyi “Member discount 15%”, sehingga `line40` menerima label yang disusun dari persentasenya.

6. Hitung bagian bulat `w = num / den` dan sisa `r = num % den` (pembagian C++ memotong ke arah nol, sehingga untuk $-\frac{7}{2}$, $w = -3$ dan $r = -1$); cetak `w`, spasi, `std::abs(r)`, `/`, dan `den` bila keduanya bukan nol. Ini meragukan karena `<<` diharapkan mencetak nilai dalam bentuk yang dapat dibaca kembali dan dipahami semua orang; pecahan campuran adalah pilihan *penyajian*. Lebih baik: biarkan `<<` seperti adanya dan tambahkan fungsi bernama `toMixed(const Fraction&)` yang mengembalikan bentuk campuran sebagai string, dipakai hanya di tempat yang membantu pembaca.

Daftar Pustaka

[1] B. Stroustrup, *The C++ Programming Language*, 4th ed. Upper Saddle River, NJ, USA: Addison-Wesley, 2013.

[2] cppreference.com, “operator overloading,” accessed Sep. 2026. [Online]. Available: https://en.cppreference.com/w/cpp/language/operators

[3] B. Stroustrup and H. Sutter, Eds., “C++ Core Guidelines, Section C.over: Overloading and overloaded operators,” accessed Sep. 2026. [Online]. Available: https://isocpp.github.io/CppCoreGuidelines/CppCoreGuidelines

[4] S. Meyers, *Effective C++: 55 Specific Ways to Improve Your Programs and Designs*, 3rd ed. Upper Saddle River, NJ, USA: Addison-Wesley, 2005.

[5] Python Software Foundation, “The Python Language Reference, Section 3.3: Special method names,” Python 3.14 documentation, accessed Sep. 2026. [Online]. Available: https://docs.python.org/3.14/reference/datamodel.html

[6] D. E. Knuth, *The Art of Computer Programming, Volume 2: Seminumerical Algorithms*, 3rd ed. Reading, MA, USA: Addison-Wesley, 1997.

[7] Python Software Foundation, “fractions — Rational numbers,” Python 3.14 documentation, accessed Sep. 2026. [Online]. Available: https://docs.python.org/3.14/library/fractions.html

Bab 16

Evaluasi dan Pembahasan IV: Kelas, Overloading, dan Seluruh Buku, Diselesaikan Langkah demi Langkah

Bab terakhir ini tidak menambahkan fitur bahasa baru. Ia mengambil C++ dari Bab 13 sampai 15, dan bersamanya semua yang dibangun buku ini sejak Bab 1, lalu memakainya pada lima soal utuh. Empat di antaranya adalah soal penutup klasik mata pelajaran pemrograman pertama: keluarga bidak catur yang harus dilengkapi dengan meniru satu kelas yang diberikan; fungsi `add` yang di-*overload* untuk dua sampai lima argumen; menemukan setiap anagram sebuah kata di dalam teks yang lebih panjang; dan mengalikan dua matriks setelah memeriksa bahwa ukurannya cocok. Soal kelima baru dan milik Profitina Laundry: kelas `Time` dengan operator-operatornya yang memberi tahu setiap pelanggan kapan cuciannya dapat diambil. Setiap soal diselesaikan dengan urutan yang sama seperti di Bab 4, 8, dan 12: soalnya dinyatakan ulang dengan kata-kata kita sendiri; contoh-contoh dikerjakan dengan tangan; rancangan di atas kertas; program C++ yang lengkap beserta kembaran Python-nya, dengan keluaran nyata keduanya; sebuah *penyelesaian keliru yang menggoda* beserta cara persis ia gagal; dan varian *periksa diri Anda* beserta jawabannya. Di akhir bab ini Anda akan memiliki metode yang teruji untuk soal kecil apa pun yang melibatkan kelas, overloading, string, dan array, lima contoh yang dikerjakan tuntas, dan gambaran jelas tentang seberapa jauh Anda telah melangkah sejak program pertama Anda mencetak satu baris.

16.1 Di Mana Kita Berhenti

Inilah perkakas yang dipakai soal-soal ini, bab demi bab.

Dari **Bab 1 sampai 3** Anda memiliki dasar-dasar yang tidak pernah hilang: variabel dan tipe, pembagian bulat dan pembulatannya, `if`, `switch`, ketiga jenis perulangan, dan kebiasaan menghitung hasil dengan tangan sebelum memercayai program. Dari **Bab 5 sampai 7** Anda memiliki fungsi dengan parameter dan nilai kembalian, rekursi, array dan array dua dimensi, pengurutan dan pencarian, serta pointer, alamat di balik setiap array. Dari **Bab 9 sampai 11** Anda memiliki string, struktur, dan pemrosesan berkas.

Dari **Bab 13** Anda memiliki C++ itu sendiri [1]: `cin` dan `cout`, `std::string`, `std::vector`, referensi, overloading, dan argumen bawaan. Dari **Bab 14** Anda memiliki kelas: data *private* yang dijaga invarian, konstruktor yang menolak nilai tidak sah, fungsi anggota `const`, dan hierarki kelas pertama dengan anggota `protected`, kelas turunan, dan fungsi virtual. Dari **Bab 15** Anda memiliki operator overloading: operator aritmetika dan perbandingan, `operator<<` untuk mencetak, dan aturan bahwa setiap operator harus bermakna seperti yang diharapkan pembaca.

Soal 16.1 memerlukan pewarisan (Bab 14); Soal 16.2 memerlukan overloading (Bab 13); Soal 16.3 memerlukan string (Bab 9 dan 13) dan perulangan yang batasnya harus tepat (Bab 3); Soal 16.4 memerlukan array dua dimensi (Bab 6), kini sebagai vektor berisi vektor; dan Soal 16.5

memerlukan kelas dengan invarian dan operator (Bab 14 dan 15). Menyebut nama alatnya saja sudah separuh penyelesaian.

16.2 Cara Menghadapi Sebuah Soal

16.2.1 Tujuh langkah

Setiap soal dalam bab ini melewati tujuh langkah yang sama. Langkah-langkah ini adalah gagasan lama, metode pemecahan masalah Pólya [2] yang disesuaikan untuk program, dan layak dipelajari sebagai rutinitas, karena rutinitaslah yang membawa Anda melewati sebuah soal ketika Anda lelah atau ragu.

1. **Nyatakan ulang soalnya dengan kata-kata Anda sendiri.** Tuliskan apa yang masuk, apa yang keluar, dan setiap keputusan yang dibiarkan terbuka oleh soal. Keputusan yang Anda ambil diam-diam adalah keputusan yang tidak dapat diperiksa pembaca.
2. **Kerjakan contoh dengan tangan.** Ambil contoh dari soal, bila ada, dan setidaknya satu contoh Anda sendiri, termasuk kasus-kasus tepi: masukan kosong, nilai terkecil dan terbesar, masukan yang harus ditolak.
3. **Rancang di atas kertas.** Pilih datanya (kelas, vektor, string) dan tuliskan algoritmanya sebagai pseudokode atau daftar langkah singkat. Perkirakan biayanya.
4. **Tulis programnya.** Lengkap, terkompilasi tanpa peringatan, dengan rancangan yang tampak dalam kode.
5. **Jalankan pada contoh-contoh tangan.** Keluarannya harus sesuai dengan langkah 2, karakter demi karakter bila formatnya ditentukan.
6. **Cobalah merusaknya.** Carilah masukan yang akan salah ditangani pemrogram yang terburu-buru. “Penyelesaian keliru yang menggoda” pada setiap soal adalah program seperti itu.
7. **Ubah satu hal dan ramalkan hasilnya.** Kotak “periksa diri Anda” meminta Anda mengubah soal sedikit dan meramalkan keluaran baru *sebelum* menjalankan apa pun. Bila ramalan Anda benar, Anda memahami penyelesaiannya; bila tidak, Anda telah menemukan apa yang perlu dipelajari.

16.2.2 Susunan setiap soal, dan letak kodenya

Setiap soal terdiri atas enam bagian: soalnya, memahaminya dengan tangan, merancangnya di atas kertas, penyelesaian C++, penyelesaian Python, dan penyelesaian keliru yang menggoda beserta kotak “periksa diri Anda”. Kodenya berada di `FP/Code_new/Chapter16`: penyelesaian di `cpp/` dan `python/`, penyelesaian keliru yang menggoda di `cpp/tempting/` dan `python/tempting/`, berkas masukan di `input/`, dan keluaran yang diharapkan dari setiap program di `expected/`. Program pengujian dari Subbab 13.11 menjalankan setiap penyelesaian dalam kedua bahasa dan membandingkan keluarannya; daftar kasusnya juga memeriksa bahwa penyelesaian keliru Soal 16.2 memang *ditolak* kompilator.

16.3 Soal 16.1: Bidak Catur

16.3.1 Soalnya

> **SOAL 16.1.** Sebuah program menyatakan bidak catur sebagai kelas. Kelas dasar `Piece` menyimpan nama bidak, pihaknya (putih atau hitam), dan posisinya di papan, serta menyediakan fungsi yang mencetak posisi itu. Kelas `Pawn` (pion) yang lengkap, diturunkan dari

`Piece`, sudah diberikan: konstruktornya mengisi nama dan pihak, dan fungsi geraknya memeriksa aturan pion, memindahkan bidak dan mencetak posisi barunya, atau mencetak `Invalid movement!!`. Tambahkan kelas `Bishop` (gajah), `Knight` (kuda), `Rook` (benteng), dan `Queen` (ratu) dengan cara yang sama, masing-masing mulai di (0, 0), dan uji masing-masing dengan satu gerakan sah, satu gerakan tidak sah, dan satu gerakan sah dari posisi yang baru.

Kita mengambil empat keputusan. *Papannya:* koordinat (x, y) dengan $0 \le x, y \le 7$, x ke kanan dan y ke atas, seperti pada kertas berpetak; gerakan ke petak di luar papan tidak sah. *Pihaknya:* anggota `faction_` bernilai $+1$ untuk putih dan -1 untuk hitam; pion putih bergerak ke atas (y-nya bertambah 1) dan pion hitam ke bawah. *Sendirian di papan:* setiap bidak diuji sendirian, sehingga tidak ada bidak lain yang menghalangi jalan atau dimakan; program catur sungguhan memerlukan objek papan yang mengenal setiap bidak. *Diam di tempat bukan gerakan:* bergerak ke petak tempat bidak itu berada tidak sah untuk bidak mana pun.

16.3.2 Memahaminya dengan tangan

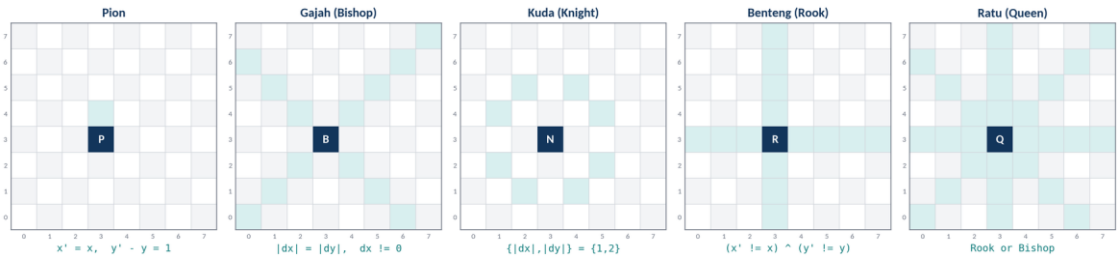
Tulislah $d_x = x' - x$ dan $d_y = y' - y$ untuk perubahan dari petak saat ini (x, y) ke petak tujuan (x', y'), dan $|d_x|$, $|d_y|$ untuk besarnya. Kelima aturan gerak catur, untuk bidak yang sendirian di papan, menjadi lima syarat:

- **Pion** (putih): kolom yang sama dan satu langkah ke atas: $d_x = 0$ dan $d_y = 1$; untuk hitam, $d_y = -1$. Keduanya tercakup oleh $d_y = 1 \cdot \text{faction}$.
- **Gajah:** diagonal, sejauh apa pun: $|d_x| = |d_y|$ dan $d_x \neq 0$.
- **Kuda:** berbentuk "L", dua petak ke satu arah dan satu petak ke arah lain: $\{|d_x|, |d_y|\} = \{1,2\}$.
- **Benteng:** sepanjang baris atau kolom, sejauh apa pun: *tepat satu* dari x dan y yang berubah.
- **Ratu:** seperti benteng atau seperti gajah.

"Tepat satu dari dua hal" milik benteng adalah operasi logika *exclusive or* (XOR, atau-eksklusif), ditulis $\oplus$: bernilai benar bila tepat satu dari kedua operannya benar. Aturan benteng karenanya adalah

$$\text{rook}(x, y \to x', y') = (x' \neq x) \oplus (y' \neq y) \tag{16.1}$$

Dalam C++ dan Python, `^` adalah operator XOR bitwise; bila diterapkan pada dua nilai `bool`, ia persis $\oplus$. **Gambar 16.1** menunjukkan, untuk setiap bidak yang berdiri di (3,3), petak-petak yang boleh dituju menurut syarat-syarat ini; periksalah beberapa petak terhadap rumusnya dengan pensil.



Gambar 16.1 Petak-petak yang dapat dicapai setiap bidak dari (3, 3) di papan kosong, beserta syarat yang menentukannya.

Sekarang uji-ujinya, dikerjakan dengan tangan. Gajah mulai di (0,0). Ke (3,3): $|d_x| = |d_y| = 3$, $d_x \neq 0$, sah. Dari (3,3) ke (3,5): $|d_x| = 0 \neq 2 = |d_y|$, tidak sah, dan gajah tetap di (3,3). Ke (5,1): $|d_x| = |d_y| = 2$, sah. Benteng dari (0,0) ke (0,6): hanya y yang berubah, sah; dari (0,6) ke (3,3):

keduanya berubah, tidak sah; ke (7,6): hanya *x* yang berubah, sah; ke (8,6): di luar papan, tidak sah.

16.3.3 Merancangnya di atas kertas

`Pawn` yang diberikan menunjukkan polanya: konstruktor yang meneruskan nama, pihak, dan petak awal ke konstruktor `Piece`, dan sebuah fungsi dengan syarat pion. Setiap kelas baru mengulang pola itu dengan syaratnya sendiri, sehingga hampir tidak ada yang perlu diciptakan. Satu perbaikan membuat peniruannya lebih sederhana lagi: langkah-langkah yang sama untuk setiap bidak (periksa papan, periksa syarat, pindahkan dan cetak, atau mengeluh) kita taruh di *satu* fungsi `move` milik `Piece`, dan biarkan ia memanggil fungsi virtual murni `moveCondition` yang ditimpa oleh setiap bidak. Inilah polimorfisme dari Subbab 14.6: kelas dasar menentukan *kapan* bertanya, setiap kelas turunan menentukan *apa* jawabannya.

Pseudokode 16.1 Gerakan yang dipakai bersama semua bidak; MOVE-CONDITION berbeda untuk setiap jenis bidak.

```
PIECE-MOVE(piece, x', y')
1  if 0 <= x' <= 7 and 0 <= y' <= 7 and MOVE-CONDITION(piece, x', y')
2      piece.x = x'
3      piece.y = y'
4      print "moved, " and the new location
5  else
6      print "Invalid movement!!"
```

Pseudokode 16.1 memakan kerja yang tetap untuk setiap gerakan. Seluruh kesulitan soal ini terletak pada menulis kelima syarat itu dengan *tepat*, termasuk tanda kurungnya.

16.3.4 Penyelesaian C++

Kode 16.1 adalah programnya secara lengkap. Bagian bertanda “given” adalah modelnya; keempat kelas sesudahnya adalah jawaban kita.

Kode 16.1 `chess.cpp`: `Piece` dan `Pawn` yang diberikan, serta keempat bidak tambahan.

```cpp
// chess.cpp - Problem 16.1: chess pieces as a small class hierarchy.
// Piece and Pawn are the given part; Bishop, Knight, Rook and Queen are added
// "like the Pawn class". One piece alone on an 8 x 8 board, no captures.
#include <cstdlib>
#include <iostream>
#include <string>
#include <utility>
#include <vector>

class Piece {
public:
    Piece(const std::string& name, int faction, int x, int y)
        : name_(name), faction_(faction), x_(x), y_(y) {}
    virtual ~Piece() = default;

    std::string getLocation() const {
        return name_ + (faction_ == 1 ? " (white)" : " (black)") + " at ("
               + std::to_string(x_) + ", " + std::to_string(y_) + ")";
    }

    void move(int xDest, int yDest) {          // the same steps for every piece
        std::cout << "  to (" << xDest << ", " << yDest << "): ";
        if (onBoard(xDest, yDest) && moveCondition(xDest, yDest)) {
            x_ = xDest;
            y_ = yDest;
            std::cout << "moved, " << getLocation() << '\n';
        } else {
            std::cout << "Invalid movement!!\n";
        }
    }

protected:
    virtual bool moveCondition(int xDest, int yDest) const = 0;
    std::string name_;
    int faction_;                              // +1 white (moves up), -1 black
    int x_, y_;

private:
    static bool onBoard(int x, int y) { return x >= 0 && x < 8 && y >= 0 && y < 8; }
};

// ---- given: the model to imitate ----------------------------------------
class Pawn : public Piece {
public:
    Pawn(int faction, int x = 0, int y = 0) : Piece("Pawn", faction, x, y) {}
```

Kode 16.1 (Lanjutan) *chess.cpp: Piece dan Pawn yang diberikan, serta keempat bidak tambahan.*

```
protected:
    bool moveCondition(int xDest, int yDest) const override {
        return (xDest == x_) && ((yDest - y_) == 1 * faction_);
    }
};

// ---- added: four more pieces, built exactly like Pawn ----------------------
class Bishop : public Piece {
public:
    Bishop(int faction, int x = 0, int y = 0) : Piece("Bishop", faction, x, y) {}

protected:
    bool moveCondition(int xDest, int yDest) const override {
        return (xDest != x_) && (std::abs(xDest - x_) == std::abs(yDest - y_));
    }
};

class Knight : public Piece {
public:
    Knight(int faction, int x = 0, int y = 0) : Piece("Knight", faction, x, y) {}

protected:
    bool moveCondition(int xDest, int yDest) const override {
        int dx = std::abs(xDest - x_), dy = std::abs(yDest - y_);
        return (dx == 1 && dy == 2) || (dx == 2 && dy == 1);
    }
};

class Rook : public Piece {
public:
    Rook(int faction, int x = 0, int y = 0) : Piece("Rook", faction, x, y) {}

protected:
    bool moveCondition(int xDest, int yDest) const override {
        return (xDest != x_) ^ (yDest != y_);  // exactly one coordinate changes
    }
};

class Queen : public Piece {
public:
    Queen(int faction, int x = 0, int y = 0) : Piece("Queen", faction, x, y) {}
```

Kode 16.1 (Lanjutan) *chess.cpp: Piece dan Pawn yang diberikan, serta keempat bidak tambahan.*

```
protected:
    bool moveCondition(int xDest, int yDest) const override {
        bool straight = (xDest != x_) ^ (yDest != y_);
        bool diagonal =
            (xDest != x_) && (std::abs(xDest - x_) == std::abs(yDest - y_));
        return straight || diagonal;
    }
};

int main() {
    Pawn whitePawn(1), blackPawn(-1, 3, 6);
    Bishop bishop(1);
    Knight knight(1);
    Rook rook(1);
    Queen queen(1);

    using Moves = std::vector<std::pair<int, int>>;
    std::vector<std::pair<Piece*, Moves>> tests = {
        {&whitePawn, {{0, 1}, {1, 2}, {0, 2}}},
        {&blackPawn, {{3, 5}, {3, 6}, {3, 4}}},
        {&bishop, {{3, 3}, {3, 5}, {5, 1}}},
        {&knight, {{1, 2}, {2, 2}, {3, 1}}},
        {&rook, {{0, 6}, {3, 3}, {7, 6}, {8, 6}}},
        {&queen, {{4, 4}, {5, 6}, {4, 0}, {4, 0}}},
    };
    for (auto& [piece, moves] : tests) {       // C++17 structured binding
        std::cout << piece->getLocation() << '\n';
        for (auto& [x, y] : moves) piece->move(x, y);
    }
    return 0;
}
```

```
Pawn (white) at (0, 0)
  to (0, 1): moved, Pawn (white) at (0, 1)
  to (1, 2): Invalid movement!!
  to (0, 2): moved, Pawn (white) at (0, 2)
Pawn (black) at (3, 6)
  to (3, 5): moved, Pawn (black) at (3, 5)
  to (3, 6): Invalid movement!!
  to (3, 4): moved, Pawn (black) at (3, 4)
Bishop (white) at (0, 0)
  to (3, 3): moved, Bishop (white) at (3, 3)
  to (3, 5): Invalid movement!!
  to (5, 1): moved, Bishop (white) at (5, 1)
Knight (white) at (0, 0)
  to (1, 2): moved, Knight (white) at (1, 2)
  to (2, 2): Invalid movement!!
  to (3, 1): moved, Knight (white) at (3, 1)
Rook (white) at (0, 0)
  to (0, 6): moved, Rook (white) at (0, 6)
  to (3, 3): Invalid movement!!
  to (7, 6): moved, Rook (white) at (7, 6)
  to (8, 6): Invalid movement!!
Queen (white) at (0, 0)
  to (4, 4): moved, Queen (white) at (4, 4)
  to (5, 6): Invalid movement!!
  to (4, 0): moved, Queen (white) at (4, 0)
  to (4, 0): Invalid movement!!
```

Setiap baris sesuai dengan pekerjaan tangan di Subbab 16.3.2. Perhatikan rincian yang membuat peniruannya benar. Setiap kelas punya konstruktornya sendiri, karena konstruktor tidak diwariskan, dan masing-masing meneruskan namanya sendiri. Setiap syarat ditulis dengan `std::abs` dari `<cstdlib>`, sebagaimana syarat-syarat pada soal itu sendiri. `Queen` memakai ulang

kedua rumus benteng dan gajah dengan nama, `straight` dan `diagonal`, sehingga kodenya terbaca seperti aturannya. Tabel uji di `main` adalah vektor berisi pasangan, masing-masing memegang pointer ke sebuah bidak dan daftar gerakannya; perulangannya memakai *structured binding* C++17, `auto& [piece, moves]`, untuk memberi kedua bagian pasangan itu namanya sendiri.

16.3.5 Penyelesaian Python

Kode 16.2 adalah hierarki yang sama dalam Python, dengan `_move_condition` yang abstrak.

Kode 16.2 `chess.py`: bidak-bidak catur dalam Python; keluarannya identik.

```python
# chess.py - Problem 16.1: chess pieces as a small class hierarchy.
# Piece and Pawn are the given part; Bishop, Knight, Rook and Queen are added
# "like the Pawn class". One piece alone on an 8 x 8 board, no captures.
from abc import ABC, abstractmethod


class Piece(ABC):
    def __init__(self, name, faction, x, y):
        self._name, self._faction = name, faction   # +1 white, -1 black
        self._x, self._y = x, y

    def get_location(self):
        side = " (white)" if self._faction == 1 else " (black)"
        return f"{self._name}{side} at ({self._x}, {self._y})"

    def move(self, x_dest, y_dest):            # the same steps for every piece
        print(f"  to ({x_dest}, {y_dest}): ", end="")
        on_board = 0 <= x_dest < 8 and 0 <= y_dest < 8
        if on_board and self._move_condition(x_dest, y_dest):
            self._x, self._y = x_dest, y_dest
            print(f"moved, {self.get_location()}")
        else:
            print("Invalid movement!!")

    @abstractmethod
    def _move_condition(self, x_dest, y_dest): ...


class Pawn(Piece):                             # given: the model to imitate
    def __init__(self, faction, x=0, y=0):
        super().__init__("Pawn", faction, x, y)

    def _move_condition(self, x_dest, y_dest):
        return x_dest == self._x and (y_dest - self._y) == 1 * self._faction


class Bishop(Piece):
    def __init__(self, faction, x=0, y=0):
        super().__init__("Bishop", faction, x, y)

    def _move_condition(self, x_dest, y_dest):
        return x_dest != self._x and abs(x_dest - self._x) == abs(y_dest - self._y)
```

Kode 16.2 (Lanjutan) *chess.py: bidak-bidak catur dalam Python; keluarannya identik.*

```
class Knight(Piece):
    def __init__(self, faction, x=0, y=0):
        super().__init__("Knight", faction, x, y)

    def _move_condition(self, x_dest, y_dest):
        dx, dy = abs(x_dest - self._x), abs(y_dest - self._y)
        return (dx == 1 and dy == 2) or (dx == 2 and dy == 1)


class Rook(Piece):
    def __init__(self, faction, x=0, y=0):
        super().__init__("Rook", faction, x, y)

    def _move_condition(self, x_dest, y_dest):
        return (x_dest != self._x) ^ (y_dest != self._y)   # exactly one changes


class Queen(Piece):
    def __init__(self, faction, x=0, y=0):
        super().__init__("Queen", faction, x, y)

    def _move_condition(self, x_dest, y_dest):
        straight = (x_dest != self._x) ^ (y_dest != self._y)
        diagonal = (x_dest != self._x
                    and abs(x_dest - self._x) == abs(y_dest - self._y))
        return straight or diagonal


tests = [
    (Pawn(1), [(0, 1), (1, 2), (0, 2)]),
    (Pawn(-1, 3, 6), [(3, 5), (3, 6), (3, 4)]),
    (Bishop(1), [(3, 3), (3, 5), (5, 1)]),
    (Knight(1), [(1, 2), (2, 2), (3, 1)]),
    (Rook(1), [(0, 6), (3, 3), (7, 6), (8, 6)]),
    (Queen(1), [(4, 4), (5, 6), (4, 0), (4, 0)]),
]
for piece, moves in tests:
    print(piece.get_location())
    for x, y in moves:
        piece.move(x, y)
```

`^` Python pada dua `bool` juga merupakan atau-eksklusif, dan perbandingan berantainya, `0 <= x_dest < 8`, terbaca persis seperti matematika.

16.3.6 Penyelesaian keliru yang menggoda

Aturan benteng terdengar seperti “benteng bergerak bila ia bergerak sepanjang baris atau kolom”, dan terjemahan yang menggoda untuk “atau” adalah `||`. **Kode 16.3** menuliskannya begitu, bersama gajah yang lupa $d_x \neq 0$.

Kode 16.3 `wrong_rook.cpp`: syarat benteng dengan `||` dan syarat gajah tanpa $d_x \neq 0$.

```cpp
// wrong_rook.cpp - a tempting wrong solution to Problem 16.1: the Rook's
// condition written with || instead of ^ (and the bishop's without x != x0).
#include <cstdlib>
#include <iostream>

bool rookWrong(int x, int y, int xDest, int yDest) {
    return (xDest != x) || (yDest != y);       // "something changed"
}

bool bishopWrong(int x, int y, int xDest, int yDest) {
    return std::abs(xDest - x) == std::abs(yDest - y);   // allows standing still
}

int main() {
    std::cout << "rook   (0,6) -> (3,3): " << rookWrong(0, 6, 3, 3) << '\n';
    std::cout << "rook   (0,6) -> (5,1): " << rookWrong(0, 6, 5, 1) << '\n';
    std::cout << "bishop (3,3) -> (3,3): " << bishopWrong(3, 3, 3, 3) << '\n';
    return 0;
}
```

```
rook   (0,6) -> (3,3): 1
rook   (0,6) -> (5,1): 1
bishop (3,3) -> (3,3): 1
```

Stream mencetak `true` sebagai `1`. Benteng yang keliru menerima (0,6) → (3,3) dan (0,6) → (5,1), dua gerakan diagonal, karena `||` bernilai benar bila *setidaknya satu* koordinat berubah, dan pada diagonal keduanya berubah. `||` hanya akan benar bila artinya "satu atau yang lain tetapi tidak keduanya", dan itulah yang dikatakan $\oplus$ dalam Persamaan (16.1). Gajah yang keliru menerima "bergerak" dari (3,3) ke (3,3), karena $0 = 0$. Kedua kesalahan berakar sama: aturannya diterjemahkan dari kalimat yang longgar, bukan dari syarat yang tepat yang diperiksa pada contoh-contoh, termasuk contoh "tetap di tempat".

> **PERIKSA DIRI ANDA.** *Raja* bergerak satu petak ke arah mana pun. Tulislah `moveCondition`-nya dengan $|d_x|$ dan $|d_y|$, lalu ramalkan keluarannya untuk raja putih di (0,0) yang digerakkan ke (1,1), lalu (1,3), lalu (1,1).
>
> *Jawaban.* `return std::max(std::abs(xDest - x_), std::abs(yDest - y_)) == 1;`: jarak yang lebih besar dari keduanya harus tepat 1, yang sekaligus menolak diam di tempat. (0,0) → (1,1): kedua jarak 1, berpindah. (1,1) → (1,3): $|d_y| = 2$, `Invalid movement!!`, dan raja tetap di (1,1). (1,1) → (1,1): jarak terbesarnya 0, `Invalid movement!!`.

16.4 Soal 16.2: Overloading `add`

16.4.1 Soalnya

> **SOAL 16.2.** Fungsi `int add(int a, int b)` yang mengembalikan $a + b$ sudah diberikan. Tulislah overload `add` untuk tiga, empat, dan lima argumen bilangan bulat, dan cetak hasil satu pemanggilan masing-masing.

Keputusan kita: argumennya adalah berat kantong cucian dalam gram dari timbangan Profitina Laundry; program mencetak setiap pemanggilan dan hasilnya dalam kolom yang rata; dan sebagai tambahan kita menyertakan overload untuk dua `double` serta *template* C++17 yang menerima berapa pun argumen, untuk menunjukkan di mana batas overloading.

16.4.2 Memahaminya dengan tangan

Kantong-kantongnya seberat 3.450, 1.200, 5.000, 2.750, dan 4.100 gram. Jumlah dua, tiga, empat, dan lima kantong pertama adalah 4.650, 9.650, 12.400, dan 16.500 gram. Untuk overload `double`, $2.5 + 1.25 = 3.75$.

16.4.3 Merancangnya di atas kertas

Resolusi overload memilih berdasarkan *banyaknya* argumen (Subbab 13.7), sehingga keempat fungsi dapat hidup berdampingan. Rancangan yang anggun membangun setiap overload di atas overload di bawahnya:

$$\text{add}(a_1, \ldots, a_k) = \text{add}(\text{add}(a_1, \ldots, a_{k-1}),\ a_k) \tag{16.2}$$

Ini gagasan yang sama dengan definisi rekursif (Bab 5), yang dibentangkan dengan tangan untuk $k = 3{,}4{,}5$: setiap versi hanya satu baris, dan perbaikan pada versi dua argumen otomatis sampai ke semua yang lain.

16.4.4 Penyelesaian C++

Kode 16.4 `add_overload.cpp`: `add` untuk dua sampai lima bilangan bulat, untuk dua `double`, dan sebuah template variadik.

```cpp
// add_overload.cpp - Problem 16.2: add() overloaded for 2, 3, 4 and 5 arguments.
#include <iostream>

int add(int a, int b) { return a + b; }        // given
int add(int a, int b, int c) { return add(add(a, b), c); }
int add(int a, int b, int c, int d) { return add(add(a, b, c), d); }
int add(int a, int b, int c, int d, int e) { return add(add(a, b, c, d), e); }

double add(double a, double b) { return a + b; }   // same count, other types

// Going further (C++17): one template for any number of arguments.
template <typename... Ts>
long long addAll(Ts... xs) {
    return (0LL + ... + xs);                   // a fold expression
}

int main() {
    std::cout << "add(3450, 1200)                  = " << add(3450, 1200) << '\n';
    std::cout << "add(3450, 1200, 5000)            = " << add(3450, 1200, 5000)
              << '\n';
    std::cout << "add(3450, 1200, 5000, 2750)      = "
              << add(3450, 1200, 5000, 2750) << '\n';
    std::cout << "add(3450, 1200, 5000, 2750, 4100) = "
              << add(3450, 1200, 5000, 2750, 4100) << '\n';
    std::cout << "add(2.5, 1.25)                   = " << add(2.5, 1.25) << '\n';
    std::cout << "addAll(1, 2, 3, 4, 5, 6, 7)      = "
              << addAll(1, 2, 3, 4, 5, 6, 7) << '\n';
    // add(2, 1.5);  would not compile: int->double and double->int tie
    return 0;
}
```

```
add(3450, 1200)                  = 4650
add(3450, 1200, 5000)            = 9650
add(3450, 1200, 5000, 2750)      = 12400
add(3450, 1200, 5000, 2750, 4100) = 16500
add(2.5, 1.25)                   = 3.75
addAll(1, 2, 3, 4, 5, 6, 7)      = 28
```

Kode 16.4 mengikuti Persamaan (16.2) secara harfiah. Overload `double` punya jumlah parameter yang sama dengan `add` yang diberikan, tetapi tipenya berbeda, sehingga `add(2.5, 1.25)` memilihnya.

Template di bagian akhir menjawab pertanyaan yang wajar: mengapa menulis empat fungsi bila satu-satunya perbedaan adalah banyaknya argumen? Karena overloading diselesaikan pada waktu kompilasi, satu tanda tangan fungsi demi satu, dan fungsi C tidak dapat menerima "berapa pun" argumen bertipe dengan aman. C++11 menambahkan *variadic template* tepat untuk ini: `template <typename... Ts>` menerima berapa pun argumen bertipe apa pun, dan *fold expression* C++17 `(0LL + ... + xs)` dibentangkan menjadi `0LL + x1 + x2 + ... + xn` pada waktu kompilasi. Template berada di luar cakupan buku ini, tetapi baik diketahui bahwa keempat overload itu adalah versi "tulis tangan" dari sebuah alat umum.

Baris yang dijadikan komentar di akhir `main` adalah jebakan yang layak dipahami: `add(2, 1.5)` tidak terkompilasi. Argumen pertama cocok persis dengan `add(int, int)` dan dengan `add(double, double)` hanya setelah konversi; argumen kedua cocok persis dengan `add(double, double)` dan dengan `add(int, int)` hanya setelah konversi. Tidak ada overload yang lebih baik untuk *kedua* argumen, sehingga pemanggilan itu ambigu: aturan resolusi overload menuntut satu kandidat setidaknya sama baik untuk setiap argumen dan lebih baik untuk salah satunya [3].

16.4.5 Penyelesaian Python

Python tidak punya overloading; `def add` yang kedua akan menggantikan yang pertama. Ia punya sesuatu yang lebih sederhana: parameter yang ditulis `*numbers` mengumpulkan berapa pun argumen ke dalam sebuah tuple, sehingga satu fungsi mencakup setiap jumlah argumen (**Kode 16.5**).

Kode 16.5 `add_overload.py`: satu fungsi Python untuk berapa pun argumen.

```python
# add_overload.py - Problem 16.2 in Python: no overloading, so one function
# with *numbers accepts any count (a second 'def add' would replace the first).


def add(*numbers):
    total = 0
    for x in numbers:
        total += x
    return total


print(f"add(3450, 1200)                  = {add(3450, 1200)}")
print(f"add(3450, 1200, 5000)            = {add(3450, 1200, 5000)}")
print(f"add(3450, 1200, 5000, 2750)      = {add(3450, 1200, 5000, 2750)}")
print(f"add(3450, 1200, 5000, 2750, 4100) = {add(3450, 1200, 5000, 2750, 4100)}")
print(f"add(2.5, 1.25)                   = {add(2.5, 1.25)}")
print(f"addAll(1, 2, 3, 4, 5, 6, 7)      = {add(1, 2, 3, 4, 5, 6, 7)}")
```

Keluarannya identik baris demi baris; label terakhir mempertahankan nama C++ `addAll` agar kedua keluaran dapat dibandingkan secara otomatis.

16.4.6 Penyelesaian keliru yang menggoda

Argumen bawaan tampak seperti jalan pintas: satu fungsi `add(int a, int b, int c = 0, int d = 0, int e = 0)` mencakup dua sampai lima argumen sekaligus. Bila dipakai *sebagai pengganti* fungsi yang diberikan, itu sebenarnya penyelesaian yang rapi. Kesalahan yang menggoda adalah

menambahkannya di samping `add(int, int)` yang diberikan, yang menurut soal harus tetap ada (**Kode 16.6**).

Kode 16.6 `wrong_add.cpp`: `add` berargumen bawaan di samping yang diberikan; pemanggilan dengan dua argumen menjadi ambigu.

```
// wrong_add.cpp - a tempting wrong solution to Problem 16.2: default arguments
// ON TOP of the given two-argument add(). This file does not compile.
#include <iostream>

int add(int a, int b) { return a + b; }        // given
int add(int a, int b, int c = 0, int d = 0, int e = 0) {   // defaults added
    return a + b + c + d + e;
}

int main() {
    std::cout << add(1, 2, 3) << '\n';         // fine: only the second fits
    std::cout << add(1, 2) << '\n';            // both fit equally well
    return 0;
}
```

`g++` menolak programnya:

```
tempting/wrong_add.cpp: In function 'int main()':
tempting/wrong_add.cpp:12:21: error: call of overloaded 'add(int, int)' is ambiguous
   12 |     std::cout << add(1, 2) << '\n';            // both fit equally well
      |                  ~~~^~~~~~
tempting/wrong_add.cpp:5:5: note: candidate: 'int add(int, int)'
    5 | int add(int a, int b) { return a + b; }        // given
      |     ^~~
tempting/wrong_add.cpp:6:5: note: candidate: 'int add(int, int, int, int, int)'
    6 | int add(int a, int b, int c = 0, int d = 0, int e = 0) {   // defaults added
```

Untuk `add(1, 2)`, kedua fungsi cocok sempurna: yang diberikan dengan kedua parameternya, yang baru dengan tiga argumen bawaan terisi. Resolusi overload hanya memeringkat kandidat menurut seberapa cocok *argumen* dengan parameternya, dan di sini keduanya cocok persis, sehingga tidak ada pemenang. Pesan galatnya menyebut kedua kandidat, dan dari situlah Anda mengenali kesalahan ini: setiap kali `g++` mengatakan "ambiguous", carilah dua overload yang sama cocoknya untuk pemanggilan yang sama.

> **PERIKSA DIRI ANDA.** Pertahankan **Kode 16.4** dan tambahkan `long long add(long long a, long long b)`. Fungsi mana yang dipanggil `add(3450, 1200)` sekarang, dan fungsi mana yang dipanggil `add(3450LL, 1200)`?
>
> *Jawaban.* `add(3450, 1200)` tetap memanggil `add(int, int)`: kedua argumennya `int`, cocok persis, lebih baik daripada konversi apa pun. `add(3450LL, 1200)` berisi sebuah `long long` dan sebuah `int`: overload baru cocok persis dengan yang pertama dan dengan yang kedua setelah promosi; `add(int, int)` memerlukan konversi penyempitan untuk yang pertama; `add(double, double)` memerlukan konversi untuk keduanya. Overload `long long` setidaknya sama baik untuk setiap argumen dan lebih baik untuk salah satunya, sehingga ia menang dan mengembalikan 4650 sebagai `long long`.

16.5 Soal 16.3: Mencari Anagram

16.5.1 Soalnya

> **SOAL 16.3.** Diberikan sebuah pola dan sebuah teks, temukan setiap posisi di dalam teks tempat dimulainya sebuah substring yang merupakan anagram dari pola itu, dan cetak setiap

> substring tersebut beserta indeksnya. Pakailah `std::is_permutation`. Misalnya, untuk pola `aaba` dan teks `aabaabaa`, anagram-anagramnya dimulai pada indeks 0, 1, 3, dan 4.

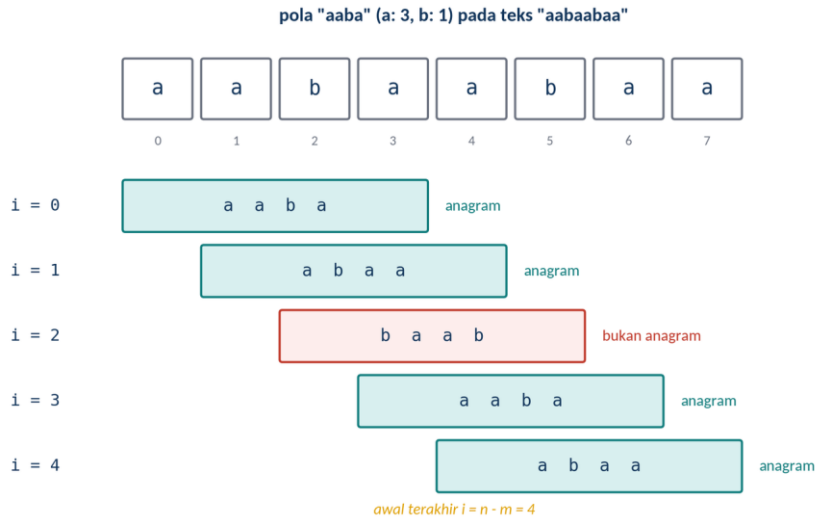
Anagram sebuah kata memakai huruf yang sama dengan banyak yang sama, dalam urutan apa pun: *kasur* dan *rusak*, atau *listen* dan *silent*. Keputusan kita: masukannya berupa pasangan `pattern text`, kata tanpa spasi, satu pasangan per baris, dibaca sampai akhir masukan; keluarannya menampilkan setiap temuan dan banyaknya temuan; huruf dibandingkan persis (huruf kapital adalah huruf yang berbeda); dan pola yang kosong atau lebih panjang daripada teks tidak punya anagram.

16.5.2 Memahaminya dengan tangan

Misalkan n panjang teks dan m panjang pola. Sebuah anagram panjangnya sama dengan pola, sehingga kita melihat setiap *jendela* berisi m karakter berurutan. Jendela pertama dimulai di indeks 0 dan yang terakhir di indeks $n - m$ (dan berakhir di indeks $n - 1$, karakter terakhir). Jadi ada

$$W = n - m + 1 \tag{16.3}$$

jendela, asalkan $m \leq n$. **Gambar 16.2** menunjukkan kelima jendela untuk $n = 8$ dan $m = 4$. Pola `aaba` punya tiga `a` dan satu `b`, demikian pula jendela di 0, 1, 3, dan 4; jendela di 2, `baab`, punya dua `b`. Jawaban soal terkonfirmasi.



Gambar 16.2 Kelima jendela sepanjang 4 dalam `aabaabaa`: empat di antaranya anagram dari `aaba`; jendela terakhir dimulai di $n - m = 4$.

Contoh-contoh kita sendiri: `abc` di dalam `cbabcacab` (temuan di 0, 2, 3, dan 6, termasuk yang saling tumpang-tindih); `wash` di dalam `washwash`, tempat *setiap* jendela adalah anagram, termasuk jendela terakhir di indeks 4, justru jendela yang terlewat oleh perulangan yang ceroboh; dan `aab` di dalam `ab`, pola yang lebih panjang daripada teks, yang tidak boleh menghasilkan temuan dan, terutama, tidak boleh mogok.

16.5.3 Merancangnya di atas kertas

Rancangan yang lugas mengikuti soalnya: untuk setiap awal i dari 0 sampai $n - m$, tanyakan kepada `std::is_permutation` apakah jendela `text[i .. i+m-1]` merupakan permutasi dari pola. `is_permutation(first1, last1, first2)` membandingkan rentang dari `first1` sampai `last1`

dengan rentang sama panjang yang dimulai di `first2` dan mengembalikan `true` bila keduanya memuat elemen yang sama dengan banyak yang sama [4].

Pseudokode 16.2 Mencari anagram dengan menguji setiap jendela memakai pemeriksaan permutasi.

```
FIND-ANAGRAMS(text, pattern)
1  n = length(text), m = length(pattern)
2  if m == 0 or m > n
3      return an empty list                 // guard BEFORE computing n - m
4  hits = empty list
5  for i = 0 to n - m                       // inclusive: W = n - m + 1 windows
6      if IS-PERMUTATION(text[i .. i+m-1], pattern)
7          append i to hits
8  return hits
```

Biayanya. Pseudokode 16.2 menguji W jendela. `is_permutation` pustaka memerlukan sampai sekitar m^2 perbandingan untuk satu jendela pada kasus terburuk, sehingga totalnya $O((n - m + 1)\, m^2)$ pada kasus terburuk. Untuk pola pendek ini tidak berarti apa-apa. Untuk pola panjang ada gagasan yang lebih cepat, *hitungan bergeser* (*sliding count*): simpan banyaknya setiap karakter dalam jendela saat ini di sebuah tabel berisi 256 penghitung; menggeser jendela satu langkah ke kanan menambahkan satu karakter dan membuang satu, dua pembaruan tabel; dan sebuah jendela adalah anagram bila tabelnya sama dengan tabel pola. Biayanya $O(n + m)$ pembaruan tabel ditambah satu perbandingan 256 penghitung per jendela, sebuah konstanta. Program kita melaksanakan keduanya dan memeriksa bahwa keduanya sepakat pada setiap masukan: dua metode independen yang memberi jawaban sama adalah bukti kebenaran yang paling murah.

16.5.4 Penyelesaian C++

Kode 16.7 `anagram.cpp`: pencarian anagram dengan `std::is_permutation` dan dengan hitungan bergeser.

```cpp
// anagram.cpp - Problem 16.3: every place in a text where an anagram of a
// pattern starts, with std::is_permutation and again with a sliding count.
#include <algorithm>
#include <array>
#include <iostream>
#include <string>
#include <vector>

// Straightforward: test every window of length m.  O((n - m + 1) * m^2) worst.
std::vector<std::size_t> findAnagrams(const std::string& text,
                                      const std::string& pattern) {
    std::vector<std::size_t> hits;
    const std::size_t n = text.size(), m = pattern.size();
    if (m == 0 || m > n) return hits;          // guard BEFORE computing n - m
    for (std::size_t i = 0; i + m <= n; ++i)   // i runs 0 .. n - m inclusive
        if (std::is_permutation(text.begin() + i, text.begin() + i + m,
                                pattern.begin()))
            hits.push_back(i);
    return hits;
}

// Faster: keep letter counts of the window and slide it.  O(n + m) steps.
std::vector<std::size_t> findAnagramsFast(const std::string& text,
                                          const std::string& pattern) {
    std::vector<std::size_t> hits;
    const std::size_t n = text.size(), m = pattern.size();
    if (m == 0 || m > n) return hits;
    std::array<int, 256> need{}, have{};
    for (unsigned char c : pattern) ++need[c];
    for (std::size_t i = 0; i < n; ++i) {
        ++have[static_cast<unsigned char>(text[i])];          // enters on the right
        if (i >= m) --have[static_cast<unsigned char>(text[i - m])];   // leaves
        if (i + 1 >= m && have == need) hits.push_back(i + 1 - m);
    }
    return hits;
}
```

Kode 16.7 (Lanjutan) *anagram.cpp: pencarian anagram dengan std::is_permutation dan dengan hitungan bergeser.*

```cpp
int main() {
    std::string pattern, text;
    while (std::cin >> pattern >> text) {
        std::vector<std::size_t> hits = findAnagrams(text, pattern);
        std::cout << "pattern \"" << pattern << "\" in \"" << text << "\"\n";
        for (std::size_t i : hits)
            std::cout << "  index " << i << ": " << text.substr(i, pattern.size())
                      << '\n';
        std::cout << "  " << hits.size() << " found; sliding count agrees: "
                  << (hits == findAnagramsFast(text, pattern) ? "yes" : "no")
                  << '\n';
    }
    return 0;
}
```

Berkas masukan `input/anagram.txt` memuat kelima pasangan dari Subbab 16.5.2:

```
aaba aabaabaa
abc cbabcacab
cat actcatac
aab ab
wash washwash
```

```
$ ./anagram < input/anagram.txt
```

```
pattern "aaba" in "aabaabaa"
  index 0: aaba
  index 1: abaa
  index 3: aaba
  index 4: abaa
  4 found; sliding count agrees: yes
pattern "abc" in "cbabcacab"
  index 0: cba
  index 2: abc
  index 3: bca
  index 6: cab
  4 found; sliding count agrees: yes
pattern "cat" in "actcatac"
  index 0: act
  index 2: tca
  index 3: cat
  index 5: tac
  4 found; sliding count agrees: yes
pattern "aab" in "ab"
  0 found; sliding count agrees: yes
pattern "wash" in "washwash"
  index 0: wash
  index 1: ashw
  index 2: shwa
  index 3: hwas
  index 4: wash
  5 found; sliding count agrees: yes
```

Tiga rincian dalam **Kode 16.7** patut diperhatikan. Pertama, perulangannya ditulis `for (std::size_t i = 0; i + m <= n; ++i)`, yaitu "$i \le n - m$" yang disusun ulang agar tidak muncul pengurangan; Subbab 16.5.6 menunjukkan mengapa hal itu penting. Kedua, versi bergeser mengubah setiap karakter menjadi `unsigned char` sebelum memakainya sebagai indeks, karena sebuah `char` bisa bernilai negatif untuk karakter di luar ASCII biasa, dan indeks negatif berada di luar array. Ketiga, `have == need` membandingkan dua `std::array` elemen demi elemen; `std::array` adalah pembungkus C++ untuk array C berukuran tetap, dan tidak seperti array C, ia dapat dibandingkan dan disalin dengan operator biasa.

16.5.5 Penyelesaian Python

Python tidak punya `is_permutation`, tetapi dua string merupakan permutasi satu sama lain tepat bila huruf-hurufnya yang sudah diurutkan sama, dan `collections.Counter` [5] memberi hitungan huruf untuk versi bergeser (**Kode 16.8**).

Kode 16.8 `anagram.py`: kedua metode yang sama dalam Python.

```python
# anagram.py - Problem 16.3: every place in a text where an anagram of a
# pattern starts, by sorting each window and again with a sliding count.
import sys
from collections import Counter


def is_permutation(a, b):
    return sorted(a) == sorted(b)              # same letters, same counts


def find_anagrams(text, pattern):
    n, m = len(text), len(pattern)
    if m == 0 or m > n:
        return []
    return [i for i in range(n - m + 1)        # i runs 0 .. n - m inclusive
            if is_permutation(text[i:i + m], pattern)]


def find_anagrams_fast(text, pattern):
    n, m = len(text), len(pattern)
    if m == 0 or m > n:
        return []
    need, have, hits = Counter(pattern), Counter(), []
    for i, c in enumerate(text):
        have[c] += 1                           # enters on the right
        if i >= m:
            have[text[i - m]] -= 1             # leaves on the left
            if have[text[i - m]] == 0:
                del have[text[i - m]]
        if i + 1 >= m and have == need:
            hits.append(i + 1 - m)
    return hits


words = sys.stdin.read().split()
for pattern, text in zip(words[0::2], words[1::2]):
    hits = find_anagrams(text, pattern)
    print(f'pattern "{pattern}" in "{text}"')
    for i in hits:
        print(f"  index {i}: {text[i:i + len(pattern)]}")
    agree = "yes" if hits == find_anagrams_fast(text, pattern) else "no"
    print(f"  {len(hits)} found; sliding count agrees: {agree}")
```

`range(n - m + 1)` adalah ejaan Python untuk "0 sampai $n - m$ inklusif": `range` berhenti *sebelum* argumennya. Bila sebuah hitungan turun ke nol, versi bergeser menghapus huruf itu dari `Counter`, agar dua penghitung dengan hitungan tak-nol yang sama terbandingkan sebagai setara.

16.5.6 Penyelesaian keliru yang menggoda

Kesalahan klasiknya ada pada batas perulangan. `range(n - m)` di Python, atau `i < n - m` di C++, tampak wajar tetapi berhenti satu jendela terlalu awal (**Kode 16.9**).

Kode 16.9 `wrong_anagram.py`: perulangan berhenti satu jendela terlalu awal.

```python
# wrong_anagram.py - a tempting wrong solution to Problem 16.3: the loop stops
# one window too early, because range(n - m) ends at n - m - 1.


def find_anagrams(text, pattern):
    n, m = len(text), len(pattern)
    return [i for i in range(n - m) if sorted(text[i:i + m]) == sorted(pattern)]


print(find_anagrams("aabaabaa", "aaba"))       # should be [0, 1, 3, 4]
print(find_anagrams("washwash", "wash"))       # should be [0, 1, 2, 3, 4]
```

```
[0, 1, 3]
[0, 1, 2, 3]
```

Temuan di indeks 4 hilang pada kedua teks, dan tidak ada yang memperingatkan kita: programnya berjalan dan mencetak daftar yang tampak masuk akal. Hanya contoh tangan dari soal, yang menyebut indeks 4, yang membongkar kesalahannya; karena itulah langkah 5 pada Subbab 16.2.1 meminta Anda menjalankan contoh soal lebih dulu.

Dalam C++ kesalahan yang sama punya bentuk kedua yang lebih buruk. `text.size()` bertipe *unsigned* `std::size_t`, yang tidak dapat menampung bilangan negatif. Bila pola lebih panjang daripada teks, `n - m` tidak menjadi -1; ia *berputar balik* ke `size_t` terbesar yang mungkin, sekitar 1.8×10^{19}, dan perulangan `for (i = 0; i < n - m; ++i)` lalu berjalan jauh melewati ujung string dan membaca memori yang bukan miliknya. Penyelesaian kita menghindari kedua bentuk itu: ia kembali lebih awal bila $m > n$, *sebelum* ada pengurangan apa pun, dan syarat perulangannya `i + m <= n` sama sekali tidak memuat pengurangan.

> **PERIKSA DIRI ANDA.** Ramalkan keluaran program yang benar untuk pasangan `aa aaaa`, dan banyaknya jendela menurut Persamaan (16.3).
>
> *Jawaban.* $n = 4$, $m = 2$, jadi $W = 4 - 2 + 1 = 3$ jendela, di 0, 1, dan 2, dan masing-masing berisi `aa`. Program mencetak tiga temuan, `index 0: aa`, `index 1: aa`, `index 2: aa`, lalu `3 found; sliding count agrees: yes`. Perulangan yang keliru hanya akan mencetak dua yang pertama.

16.6 Soal 16.4: Perkalian Matriks

16.6.1 Soalnya

> **SOAL 16.4.** Bacalah dua matriks A dan B. Bila banyaknya kolom A sama dengan banyaknya baris B, hitung dan cetak $C = A \times B$; bila tidak, cetak pesan bahwa kedua matriks tidak dapat dikalikan. Sebuah program parsial yang disediakan belum memberi hasil yang benar; perbaikilah.

Keputusan kita: masukannya memuat pasangan-pasangan matriks, masing-masing ditulis sebagai banyaknya baris dan kolom diikuti nilai-nilainya baris demi baris; program menangani setiap pasangan dalam berkas; nilainya bilangan bulat, dan hasil kalinya dihitung dalam `long long`.

16.6.2 Memahaminya dengan tangan

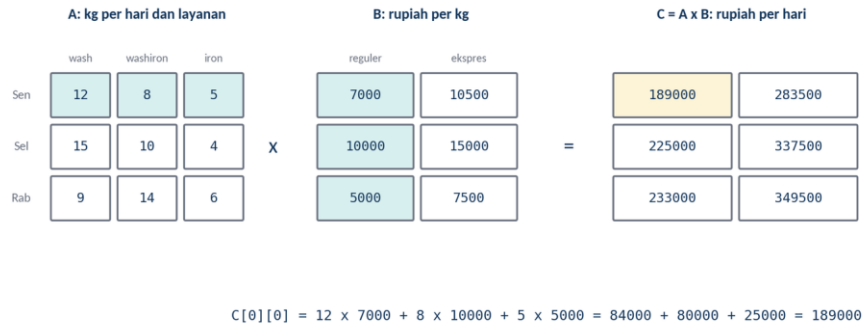
> **DEFINISI.** Bila A punya n baris dan m kolom dan B punya m baris dan p kolom, hasil kali keduanya $C = A \times B$ punya n baris dan p kolom, dan setiap elemennya adalah jumlah hasil kali baris i dari A dengan kolom j dari B:

$$C_{ij} = \sum_{k=0}^{m-1} A_{ik}\, B_{kj}, \qquad 0 \le i < n,\ 0 \le j < p \tag{16.4}$$

Ukurannya bekerja seperti kartu domino: $(n \times m) \times (m \times p) = (n \times p)$; kedua bilangan di tengah harus sama dan lenyap. Itulah pemeriksaan yang diminta soal:

$$\text{cols}(A) = \text{rows}(B) \tag{16.5}$$

Contoh pertama milik Profitina Laundry. A memuat kilogram yang dicuci pada Senin, Selasa, dan Rabu (baris) untuk ketiga layanan cuci, cuci setrika, dan setrika (kolom). B memuat harga per kilogram setiap layanan (baris) untuk pesanan reguler dan pesanan ekspres (kolom). Maka $C = A \times B$ punya satu baris per hari dan satu kolom per jenis harga: C_{00} adalah pendapatan Senin bila semua pesanan reguler. **Gambar 16.3** menghitungnya menurut Persamaan (16.4): $12 \times 7{,}000 + 8 \times 10{,}000 + 5 \times 5{,}000 = 189{,}000$. Untuk kolom kedua, $12 \times 10{,}500 + 8 \times 15{,}000 + 5 \times 7{,}500 = 126{,}000 + 120{,}000 + 37{,}500 = 283{,}500$.



Gambar 16.3 Kilogram harian per layanan dikali harga per kilogram menghasilkan pendapatan harian; C_{00} memadukan baris 0 dari A dengan kolom 0 dari B.

Contoh-contoh kita yang lain: hasil kali kecil yang dihitung sepenuhnya dengan tangan, $\begin{pmatrix}1 & 2 & 3\\ 4 & 5 & 6\end{pmatrix} \times \begin{pmatrix}7 & 8\\ 9 & 10\\ 11 & 12\end{pmatrix} = \begin{pmatrix}58 & 64\\ 139 & 154\end{pmatrix}$ (misalnya $C_{00} = 1 \cdot 7 + 2 \cdot 9 + 3 \cdot 11 = 58$); hasil kali dengan matriks identitas I, yang harus mengembalikan A tanpa perubahan; dan pasangan yang tidak cocok, 2×3 dikali 2×2, yang ditolak Persamaan (16.5) karena $3 \neq 2$.

16.6.3 Merancangnya di atas kertas

Sebuah matriks adalah `std::vector<std::vector<long long>>`, vektor berisi baris-baris; `a[i][k]` adalah baris i, kolom k, seperti array dua dimensi di Bab 6, tetapi ukurannya dipilih pada waktu jalan dan setiap vektor mengetahui ukurannya sendiri. Hasil kalinya adalah perulangan rangkap tiga dari Persamaan (16.4):

Pseudokode 16.3 Perkalian matriks dengan pemeriksaan ukuran lebih dulu.

```
MULTIPLY(A, B)                          // A is n x m, B is m' x p
1  if n == 0 or m != m'
2      return "cannot multiply"
3  C = an n x p matrix of zeros          // every sum starts at 0
4  for i = 0 to n - 1
5      for j = 0 to p - 1
6          for k = 0 to m - 1
7              C[i][j] = C[i][j] + A[i][k] * B[k][j]
8  return C
```

Baris terdalam berjalan $n \cdot p \cdot m$ kali, sehingga biayanya $O(nmp)$, atau $O(n^3)$ untuk matriks persegi; algoritma yang lebih cepat ada untuk matriks yang sangat besar, tetapi perulangan rangkap tiga adalah titik awal yang baku [6]. Bila program parsialnya "tidak memberi hasil yang benar", **Pseudokode 16.3** memberi tahu di mana harus mencari: baris 1 yang hilang (tanpa pemeriksaan), baris 3 yang hilang (jumlah yang dimulai dari sampah memori), batas perulangan yang salah, atau indeks yang salah di baris 7.

16.6.4 Penyelesaian C++

Kode 16.10 melaksanakan **Pseudokode 16.3** baris demi baris.

Kode 16.10 `matrix.cpp`: perkalian matriks dengan pemeriksaan ukuran.

```cpp
// matrix.cpp - Problem 16.4: matrix multiplication with a dimension check.
// Input: pairs of matrices, each written as "rows cols" followed by the values.
#include <iomanip>
#include <iostream>
#include <vector>

using Matrix = std::vector<std::vector<long long>>;

bool readMatrix(std::istream& in, Matrix& m) {
    std::size_t rows = 0, cols = 0;
    if (!(in >> rows >> cols)) return false;
    m.assign(rows, std::vector<long long>(cols, 0));
    for (auto& row : m)
        for (long long& x : row)
            if (!(in >> x)) return false;
    return true;
}

// C = A * B, allowed only when A's columns equal B's rows.
bool multiply(const Matrix& a, const Matrix& b, Matrix& c) {
    std::size_t n = a.size(), m = b.size(), p = b.empty() ? 0 : b[0].size();
    if (n == 0 || a[0].size() != m) return false;
    c.assign(n, std::vector<long long>(p, 0));          // every sum starts at 0
    for (std::size_t i = 0; i < n; ++i)
        for (std::size_t j = 0; j < p; ++j)
            for (std::size_t k = 0; k < m; ++k)         // sum of A[i][k] * B[k][j]
                c[i][j] += a[i][k] * b[k][j];
    return true;
}

void print(const Matrix& m) {
    for (const auto& row : m) {
        for (long long x : row) std::cout << std::setw(10) << x;
        std::cout << '\n';
    }
}
```

Kode 16.10 (Lanjutan) *matrix.cpp: perkalian matriks dengan pemeriksaan ukuran.*

```
int main() {
    Matrix a, b, c;
    int number = 0;
    while (readMatrix(std::cin, a) && readMatrix(std::cin, b)) {
        std::cout << "Case " << ++number << ": A is " << a.size() << " x "
                  << a[0].size() << ", B is " << b.size() << " x " << b[0].size()
                  << '\n';
        if (!multiply(a, b, c)) {
            std::cout << "  cannot multiply: A has " << a[0].size()
                      << " columns but B has " << b.size() << " rows\n";
            continue;
        }
        std::cout << "  A x B is " << c.size() << " x " << c[0].size() << ":\n";
        print(c);
    }
    return 0;
}
```

Berkas masukannya memuat keempat pasangan dari Subbab 16.6.2:

```
3 3
12  8 5
15 10 4
 9 14 6
3 2
 7000 10500
10000 15000
 5000  7500

2 3
1 2 3
4 5 6
3 2
 7  8
 9 10
11 12

2 2
2 3
5 7
2 2
1 0
0 1

2 3
1 2 3
4 5 6
2 2
1 2
3 4
```

```
Case 1: A is 3 x 3, B is 3 x 2
  A x B is 3 x 2:
    189000    283500
    225000    337500
    233000    349500
Case 2: A is 2 x 3, B is 3 x 2
  A x B is 2 x 2:
        58        64
       139       154
Case 3: A is 2 x 2, B is 2 x 2
  A x B is 2 x 2:
         2         3
         5         7
Case 4: A is 2 x 3, B is 2 x 2
  cannot multiply: A has 3 columns but B has 2 rows
```

Setiap bilangan sesuai dengan perhitungan tangan. `m.assign(rows, std::vector<long long>(cols, 0))` membuat matriks berukuran tepat yang berisi nol dalam satu baris, yaitu baris 3 dari **Pseudokode 16.3**; `readMatrix` mengembalikan `false` ketika masukan habis, yang mengakhiri perulangan di `main`. `multiply` mengembalikan `false` untuk pasangan yang tidak cocok *sebelum* menyentuh `c`, dan `main` mencetak alasannya beserta kedua ukurannya, sehingga orang yang mengetik berkas itu tahu apa yang harus diperbaiki.

16.6.5 Penyelesaian Python

Kode 16.11 sama, dengan list berisi list. `[[0] * p for _ in range(n)]` membangun n baris yang terpisah; bentuk yang lebih pendek `[[0] * p] * n` akan membangun *satu* baris dan n referensi kepadanya, sehingga menulis ke `c[0][0]` akan mengubah setiap baris, jebakan klasik Python.

Kode 16.11 `matrix.py`: perkalian matriks dalam Python.

```python
# matrix.py - Problem 16.4: matrix multiplication with a dimension check.
# Input: pairs of matrices, each written as "rows cols" followed by the values.
import sys


def read_matrix(tokens):
    """Takes one matrix from the front of the token list, or returns None."""
    if len(tokens) < 2:
        return None
    rows, cols = int(tokens.pop(0)), int(tokens.pop(0))
    if len(tokens) < rows * cols:
        return None
    return [[int(tokens.pop(0)) for _ in range(cols)] for _ in range(rows)]


def multiply(a, b):
    """C = A * B, or None when A's columns differ from B's rows."""
    n, m, p = len(a), len(b), len(b[0]) if b else 0
    if n == 0 or len(a[0]) != m:
        return None
    c = [[0] * p for _ in range(n)]            # every sum starts at 0
    for i in range(n):
        for j in range(p):
            for k in range(m):                 # C[i][j] = sum A[i][k] B[k][j]
                c[i][j] += a[i][k] * b[k][j]
    return c


tokens = sys.stdin.read().split()
number = 0
while True:
    a = read_matrix(tokens)
    b = read_matrix(tokens) if a is not None else None
    if b is None:
        break
    number += 1
    print(f"Case {number}: A is {len(a)} x {len(a[0])}, "
          f"B is {len(b)} x {len(b[0])}")
    c = multiply(a, b)
    if c is None:
        print(f"  cannot multiply: A has {len(a[0])} columns "
              f"but B has {len(b)} rows")
        continue
    print(f"  A x B is {len(c)} x {len(c[0])}:")
    for row in c:
        print("".join(f"{x:>10}" for x in row))
```

16.6.6 Penyelesaian keliru yang menggoda

Bila program perkalian matriks "hampir benar", kesalahan tertanam yang paling sering adalah indeks yang tertukar: `b[j][k]` alih-alih `b[k][j]`. **Kode 16.12** membuat kesalahan itu lalu menguji dirinya dengan cara yang dipakai banyak orang, yaitu dengan matriks identitas.

Kode 16.12 `wrong_matrix.cpp`: indeks *B* tertukar, diuji dengan matriks identitas dan dengan matriks biasa.

```cpp
// wrong_matrix.cpp - a tempting wrong solution to Problem 16.4: B's indices
// swapped, B[j][k] instead of B[k][j]. The identity test does not notice.
#include <iostream>
#include <vector>

using Matrix = std::vector<std::vector<long long>>;

Matrix multiplyWrong(const Matrix& a, const Matrix& b) {
    Matrix c(a.size(), std::vector<long long>(b[0].size(), 0));
    for (std::size_t i = 0; i < a.size(); ++i)
        for (std::size_t j = 0; j < b[0].size(); ++j)
            for (std::size_t k = 0; k < b.size(); ++k)
                c[i][j] += a[i][k] * b[j][k];  // should be b[k][j]
    return c;
}

void print(const char* title, const Matrix& m) {
    std::cout << title;
    for (const auto& row : m) std::cout << " [" << row[0] << ' ' << row[1] << ']';
    std::cout << '\n';
}

int main() {
    Matrix a = {{2, 3}, {5, 7}}, identity = {{1, 0}, {0, 1}}, b = {{1, 2}, {3, 4}};
    print("A x I =", multiplyWrong(a, identity));    // looks right
    print("A x B =", multiplyWrong(a, b));           // should be [11 16] [26 38]
    return 0;
}
```

```
A x I = [2 3] [5 7]
A x B = [8 18] [19 43]
```

Uji identitasnya *lolos*: $A \times I$ keluar sebagai A. Uji itu lolos karena menukar indeks B sama dengan mengalikan dengan *transpos* B (baris dan kolom ditukar), dan matriks identitas adalah transposnya sendiri, sehingga kesalahannya tidak tampak. Uji kedua membongkarnya: hasil kali yang benar dari $\begin{pmatrix}2 & 3\\5 & 7\end{pmatrix}$ dan $\begin{pmatrix}1 & 2\\3 & 4\end{pmatrix}$ adalah $\begin{pmatrix}11 & 16\\26 & 38\end{pmatrix}$ ($C_{00} = 2\cdot 1 + 3\cdot 3 = 11$), tetapi program mencetak $\begin{pmatrix}8 & 18\\19 & 43\end{pmatrix}$. Pelajarannya melampaui matriks: sebuah uji hanya sebaik kemampuannya untuk gagal. Ujilah dengan data yang *tidak simetris*, tempat setiap kesalahan mengubah hasil, dan bandingkan dengan nilai yang dihitung tangan. Dan untuk matriks yang tidak persegi, indeks yang tertukar lebih buruk daripada sekadar salah: `b[j][k]` dapat berjalan melewati ujung sebuah baris, yang merupakan perilaku tak terdefinisi.

> **PERIKSA DIRI ANDA.** Tanpa menjalankan program, hitunglah kedua pendapatan Rabu (C_{20} dan C_{21}) pada contoh laundry, dan sebutkan apa yang dicetak program untuk matriks A berukuran 3×3 dikali matriks B berukuran 2×3.
>
> *Jawaban.* Baris 2 dari A adalah (9,14,6). $C_{20} = 9\times 7{,}000 + 14\times 10{,}000 + 6\times 5{,}000 = 63{,}000 + 140{,}000 + 30{,}000 = 233{,}000$, dan $C_{21} = 9\times 10{,}500 + 14\times 15{,}000 + 6\times 7{,}500 = 94{,}500 + 210{,}000 + 45{,}000 = 349{,}500$, baris terakhir Case 1. Untuk 3×3 dikali 2×3, Persamaan (16.5) tidak terpenuhi ($3 \neq 2$), dan program mencetak `cannot multiply: A has 3 columns but B has 2 rows`.

16.7 Soal 16.5: Kapan Bisa Diambil?

16.7.1 Soalnya

> **SOAL 16.5.** Profitina Laundry buka pukul 08.00 sampai 20.00 setiap hari. Mencuci memakan waktu 6 jam, cuci setrika 12 jam, setrika saja 3 jam; pesanan ekspres memakan separuh waktunya. Cucian yang selesai ketika toko tutup dapat diambil pukul 08.00 pada saat buka berikutnya. Tulislah kelas `Time` untuk sebuah saat dalam seminggu, dengan `+` (menambah menit), `-` (menit di antara dua waktu), `<`, dan `==`, serta pencetakan dalam bentuk `Mon 09:15`. Untuk setiap pesanan, cetak waktu penitipan, waktu cucian selesai, waktu pengambilan, dan lama menunggu; lalu daftar pesanan menurut urutan pengambilan.

Keputusan kita: hari 0 adalah hari Senin, dan hari dihitung terus tanpa batas (hari 7 adalah Senin berikutnya); waktu dicetak hanya dengan nama hari (dalam singkatan bahasa Inggris seperti pada program); cucian yang selesai tepat pukul 20.00 masih dapat diambil pukul 20.00; `Time` dengan jam di luar 0–23 atau menit di luar 0–59 ditolak.

16.7.2 Memahaminya dengan tangan

Kunci kelas ini adalah *invariannya*: kita menyimpan waktu sebagai nomor hari dan menit sesudah tengah malam, $0 \le \text{minute} < 1440$ (ada $24 \times 60 = 1440$ menit dalam sehari). Menambahkan Δ menit ke (hari, menit) menghasilkan

$$\text{day}' = \text{day} + \left\lfloor \frac{\text{minute} + \Delta}{1440} \right\rfloor, \qquad \text{minute}' = (\text{minute} + \Delta) \bmod 1440 \tag{16.6}$$

yang memulihkan invarian setelah setiap penjumlahan. Dengan invarian itu, selisih dua waktu cukup $1440 \cdot (\text{day}_a - \text{day}_b) + (\text{minute}_a - \text{minute}_b)$, dan membandingkan dua waktu berarti membandingkan selisih itu dengan nol.

Sekarang pesanan-pesanannya, dengan tangan. Pak Budi menitipkan cuci setrika, ekspres, pada Senin pukul 14.40: 720 menit dibagi dua menjadi 360; $880 + 360 = 1240$ menit adalah 20.40, sesudah tutup, sehingga waktu pengambilannya Selasa 08.00, dan lama menunggunya $1440 + 480 - 880 = 1040$ menit, 17 jam 20 menit. Pak Hendra menitipkan cuci setrika pada Minggu (hari 6) pukul 18.00: $1080 + 720 = 1800$, sehingga menurut Persamaan (16.6) harinya menjadi $6 + \lfloor 1800/1440 \rfloor = 7$ dan menitnya $1800 \bmod 1440 = 360$, pukul 06.00 pada Senin berikutnya; itu sebelum buka, jadi ia mengambil pukul 08.00, setelah 14 jam.

16.7.3 Merancangnya di atas kertas

Pseudokode 16.4 Menghitung waktu pengambilan dari waktu penitipan dan layanannya.

```
PICKUP-TIME(order)
1  duration = 360 for wash, 720 for washiron, 180 for iron
2  if order is express
3      duration = duration / 2
4  ready = order.in + duration              // Time + minutes, normalized
5  if ready.hour < 8
6      return (ready.day, 08:00)            // wait for opening, same day
7  if ready.hour * 60 + ready.minute > 20 * 60
8      return (ready.day + 1, 08:00)        // closed: next morning
9  return ready
```

Operator-operatornya mengikuti Bab 15: `+=` adalah anggota yang menerapkan Persamaan (16.6); `+` adalah non-anggota yang dibangun di atasnya; `-`, `<`, dan `==` membandingkan melalui selisih.

Untuk mengurutkan pesanan menurut waktu pengambilan kita memakai `std::stable_sort`, yang mempertahankan urutan semula bagi pesanan dengan waktu pengambilan yang sama.

16.7.4 Penyelesaian C++

Kode 16.13 `pickup_time.cpp`: kelas `Time` dengan operator-operatornya, dan waktu pengambilan lima pesanan.

```
// pickup_time.cpp - Problem 16.5: a Time class with operators, used to tell
// customers when their laundry can be picked up (shop open 08:00 to 20:00).
#include <algorithm>
#include <iomanip>
#include <iostream>
#include <stdexcept>
#include <string>
#include <vector>

class Time {
public:
    Time(int day, int hour, int minute) : day_(day), minute_(hour * 60 + minute) {
        if (day < 0) throw std::invalid_argument("day must be 0 or more");
        if (hour < 0 || hour > 23)
            throw std::invalid_argument("hour must be 0..23");
        if (minute < 0 || minute > 59)
            throw std::invalid_argument("minute must be 0..59");
    }
    int day() const { return day_; }
    int hour() const { return minute_ / 60; }
    int minute() const { return minute_ % 60; }

    Time& operator+=(int minutes) {            // keeps 0 <= minute_ < 1440
        int total = minute_ + minutes;
        day_ += total / 1440;
        minute_ = total % 1440;
        return *this;
    }
    // Minutes from b to a; the invariant makes this one line.
    friend int operator-(const Time& a, const Time& b) {
        return (a.day_ - b.day_) * 1440 + (a.minute_ - b.minute_);
    }
    friend bool operator<(const Time& a, const Time& b) { return (a - b) < 0; }
    friend bool operator==(const Time& a, const Time& b) { return (a - b) == 0; }

private:
    int day_;                                  // 0 = the first Monday
    int minute_;                               // minutes after midnight
};

Time operator+(Time t, int minutes) { return t += minutes; }
```

Kode 16.13 (Lanjutan) *pickup_time.cpp: kelas Time dengan operator-operatornya, dan waktu pengambilan lima pesanan.*

```
std::ostream& operator<<(std::ostream& os, const Time& t) {
    static const char* names[] = {"Mon", "Tue", "Wed", "Thu", "Fri", "Sat", "Sun"};
    char oldFill = os.fill('0');
    auto oldFlags = os.flags();
    os << names[t.day() % 7] << ' ' << std::right << std::setw(2) << t.hour()
       << ':' << std::setw(2) << t.minute();
    os.fill(oldFill);                          // leave the stream as we found it
    os.flags(oldFlags);
    return os;
}

// Laundry that is ready while the shop is closed waits for 08:00.
Time pickupTime(const Time& ready) {
    if (ready.hour() < 8) return Time(ready.day(), 8, 0);
    if (ready.hour() * 60 + ready.minute() > 20 * 60)
        return Time(ready.day() + 1, 8, 0);
    return ready;
}

int processingMinutes(const std::string& service, bool express) {
    int m = service == "wash" ? 360 : service == "washiron" ? 720 : 180;
    return express ? m / 2 : m;                // express takes half the time
}

struct Ticket {
    int number;
    std::string customer, service;
    bool express;
    Time in, pickup;
};

int main() {
    std::vector<Ticket> tickets = {
        {1001, "Bu Rina", "wash", false, Time(0, 9, 15), Time(0, 0, 0)},
        {1002, "Pak Budi", "washiron", true, Time(0, 14, 40), Time(0, 0, 0)},
        {1003, "Mbak Sari", "iron", false, Time(0, 19, 30), Time(0, 0, 0)},
        {1004, "Mas Joko", "wash", true, Time(0, 10, 0), Time(0, 0, 0)},
        {1005, "Pak Hendra", "washiron", false, Time(6, 18, 0), Time(0, 0, 0)},
    };
    for (Ticket& t : tickets) {
        Time ready = t.in + processingMinutes(t.service, t.express);
        t.pickup = pickupTime(ready);
        int wait = t.pickup - t.in;
        std::cout << t.number << ' ' << std::left << std::setw(11) << t.customer
                  << "in " << t.in << "  ready " << ready << "  pick up "
                  << t.pickup << "  (" << wait / 60 << " h " << wait % 60
```

Kode 16.13 (Lanjutan) *pickup_time.cpp: kelas Time dengan operator-operatornya, dan waktu pengambilan lima pesanan.*

```
                  << " min)\n";
    }
    std::cout << "1002 and 1003 together: "
              << (tickets[1].pickup == tickets[2].pickup ? "yes" : "no") << '\n';
    std::stable_sort(tickets.begin(), tickets.end(),   // earliest pick-up first
                     [](const Ticket& a, const Ticket& b) {
                         return a.pickup < b.pickup;
                     });
    std::cout << "Pick-up order:";
    for (const Ticket& t : tickets) std::cout << ' ' << t.number;
    std::cout << '\n';
    try {
        Time bad(0, 24, 0);
        std::cout << bad << '\n';
    } catch (const std::invalid_argument& e) {
        std::cout << "Time(0, 24, 0) refused: " << e.what() << '\n';
    }
    return 0;
}
```

```
1001 Bu Rina    in Mon 09:15  ready Mon 15:15  pick up Mon 15:15  (6 h 0 min)
1002 Pak Budi   in Mon 14:40  ready Mon 20:40  pick up Tue 08:00  (17 h 20 min)
1003 Mbak Sari  in Mon 19:30  ready Mon 22:30  pick up Tue 08:00  (12 h 30 min)
1004 Mas Joko   in Mon 10:00  ready Mon 13:00  pick up Mon 13:00  (3 h 0 min)
1005 Pak Hendra in Sun 18:00  ready Mon 06:00  pick up Mon 08:00  (14 h 0 min)
1002 and 1003 together: yes
Pick-up order: 1004 1001 1002 1003 1005
Time(0, 24, 0) refused: hour must be 0..23
```

Hasilnya sesuai dengan pekerjaan tangan. Tiga fitur dalam **Kode 16.13** baru. `operator-`, `operator<`, dan `operator==` dideklarasikan `friend` di dalam kelas: fungsi *friend* bukan anggota tetapi boleh membaca data *private*, praktis untuk operator simetris yang memerlukan isi kedua objek. `operator<<` menyimpan lalu memulihkan karakter pengisi dan *flag* stream, karena ia beralih ke pengisi `'0'` untuk jam dan menit dan tidak boleh meninggalkan stream dalam keadaan itu untuk keluaran berikutnya (versi awal program ini mencetak `Mon 90:15` untuk 09.15, karena `std::left` yang dipasang untuk nama pelanggan masih aktif; memulihkan flag dan memasang `std::right` secara lokal memperbaikinya). Dan perbandingan yang diberikan kepada `std::stable_sort` adalah *lambda*, fungsi kecil tanpa nama yang ditulis di tempat, `[](const Ticket& a, const Ticket& b) { return a.pickup < b.pickup; }`, yang memakai `operator<` milik `Time`.

16.7.5 Penyelesaian Python

Kode 16.14 adalah kelas yang sama dalam Python; `list.sort` selalu stabil, dan `key=lambda t: t[5]` mengurutkan menurut waktu pengambilan dengan memakai `Time.__lt__`.

Kode 16.14 `pickup_time.py`: kelas `Time` dalam Python.

```python
# pickup_time.py - Problem 16.5: a Time class with operators, used to tell
# customers when their laundry can be picked up (shop open 08:00 to 20:00).
from functools import total_ordering

NAMES = ["Mon", "Tue", "Wed", "Thu", "Fri", "Sat", "Sun"]


@total_ordering
class Time:
    def __init__(self, day, hour, minute):
        if day < 0:
            raise ValueError("day must be 0 or more")
        if not 0 <= hour <= 23:
            raise ValueError("hour must be 0..23")
        if not 0 <= minute <= 59:
            raise ValueError("minute must be 0..59")
        self._day, self._minute = day, hour * 60 + minute

    day = property(lambda self: self._day)
    hour = property(lambda self: self._minute // 60)
    minute = property(lambda self: self._minute % 60)

    def __add__(self, minutes):                # keeps 0 <= _minute < 1440
        total = self._minute + minutes
        t = Time(self._day, 0, 0)
        t._day += total // 1440
        t._minute = total % 1440
        return t

    def __sub__(self, other):                  # minutes from other to self
        return (self._day - other._day) * 1440 + (self._minute - other._minute)

    def __lt__(self, other):
        return self - other < 0

    def __eq__(self, other):
        return self - other == 0

    def __str__(self):
        return f"{NAMES[self.day % 7]} {self.hour:02d}:{self.minute:02d}"
```

Kode 16.14 (Lanjutan) *pickup_time.py: kelas Time dalam Python.*

```
def pickup_time(ready):
    """Laundry that is ready while the shop is closed waits for 08:00."""
    if ready.hour < 8:
        return Time(ready.day, 8, 0)
    if ready.hour * 60 + ready.minute > 20 * 60:
        return Time(ready.day + 1, 8, 0)
    return ready


def processing_minutes(service, express):
    m = {"wash": 360, "washiron": 720}.get(service, 180)
    return m // 2 if express else m            # express takes half the time


tickets = [
    [1001, "Bu Rina", "wash", False, Time(0, 9, 15)],
    [1002, "Pak Budi", "washiron", True, Time(0, 14, 40)],
    [1003, "Mbak Sari", "iron", False, Time(0, 19, 30)],
    [1004, "Mas Joko", "wash", True, Time(0, 10, 0)],
    [1005, "Pak Hendra", "washiron", False, Time(6, 18, 0)],
]
for t in tickets:
    number, customer, service, express, time_in = t
    ready = time_in + processing_minutes(service, express)
    pickup = pickup_time(ready)
    t.append(pickup)
    wait = pickup - time_in
    print(f"{number} {customer:<11}in {time_in}  ready {ready}  pick up {pickup}"
          f"  ({wait // 60} h {wait % 60} min)")
together = tickets[1][5] == tickets[2][5]
print(f"1002 and 1003 together: {'yes' if together else 'no'}")
tickets.sort(key=lambda t: t[5])               # Python's sort is stable
print("Pick-up order:" + "".join(f" {t[0]}" for t in tickets))
try:
    bad = Time(0, 24, 0)
    print(bad)
except ValueError as e:
    print(f"Time(0, 24, 0) refused: {e}")
```

16.7.6 Penyelesaian keliru yang menggoda

Rancangan yang menggoda menyimpan jam dan menit terpisah, seperti saat dicetak, lalu menambahkan menit ke kolom menit (**Kode 16.15**).

Kode 16.15 `wrong_time.py`: jam dan menit disimpan terpisah, tanpa simpanan (*carry*).

```
# wrong_time.py - a tempting wrong solution to Problem 16.5: hours and minutes
# kept apart, and minutes added without carrying into hours and days.


class WrongTime:
    def __init__(self, hour, minute):
        self.hour, self.minute = hour, minute

    def add(self, minutes):
        return WrongTime(self.hour, self.minute + minutes)   # no carry!

    def __lt__(self, other):
        return (self.hour, self.minute) < (other.hour, other.minute)

    def __str__(self):
        return f"{self.hour:02d}:{self.minute:02d}"


a = WrongTime(14, 40).add(360)                  # should be 20:40
b = WrongTime(19, 30).add(180)                  # should be 22:30
closing = WrongTime(20, 0)
print(a, b)
print("b ready before closing?", b < closing)  # 22:30 is after 20:00: False
```

```
14:400 19:210
b ready before closing? True
```

Cucian Pak Budi "selesai pukul 14:400", dan cucian Mbak Sari pukul "19:210", yang lalu dianggap program masih sebelum tutup, karena perbandingannya melihat jam lebih dulu dan 19 lebih kecil daripada 20. Invarian dari Subbab 16.7.2 tidak pernah ditegakkan, sehingga setiap operasi yang mengandalkannya salah. Perbaikannya bukan perbandingan yang lebih pintar; perbaikannya adalah representasi yang membuat nilai tidak sah mustahil ada, dan itulah yang diberikan Persamaan (16.6).

> **PERIKSA DIRI ANDA.** Mbak Sari membawa setrikaan, ekspres, pada hari Jumat (hari 4) pukul 18.30. Kapan ia dapat mengambilnya, dan berapa lama ia menunggu? Bagaimana bila ia datang pukul 18.31?
>
> *Jawaban.* Setrika memakan 180 menit, ekspres 90. Pukul 18.30 ditambah 90 menit adalah 20.00; itu tidak *sesudah* 20.00, sehingga ia mengambil pada Jumat pukul 20.00 setelah 1 jam 30 menit. Pada 18.31 cuciannya selesai pukul 20.01, sesudah tutup, sehingga ia mengambil pada Sabtu pukul 08.00, setelah $1440 + 480 - 1111 = 809$ menit, 13 jam 29 menit. Satu menit di meja kasir mengubah jawabannya hampir setengah hari, dan karena itulah kasus batas harus ada dalam setiap pengujian.

16.8 Kelima Soal Berdampingan

Tabel 16.1 mengumpulkan kelima soal: alat utamanya, biayanya, jebakannya, dan uji yang menangkap jebakan itu.

Tabel 16.1 Kelima soal bab ini berdampingan.

Soal	Alat utama	Biaya	Jebakan	Uji yang menangkapnya
16.1 Bidak catur	pewarisan, fungsi virtual	$O(1)$ per gerakan	*atau* logika sebagai ganti `^`; diam di tempat	gerakan diagonal benteng; gerakan ke petak yang sama
16.2 Overloading `add`	overloading, pemakaian ulang	$O(k)$ untuk k argumen	argumen bawaan di samping overload	mengompilasi pemanggilan dua argumen
16.3 Anagram	string, `is_permutation`	$O((n-m+1)m^2)$, atau $O(n+m)$ bergeser	jendela terakhir terlewat; *unsigned* berputar balik	contoh soal; pola lebih panjang daripada teks
16.4 Matriks	vektor berisi vektor	$O(nmp)$	`b[j][k]`; tanpa pemeriksaan ukuran	contoh tangan tak simetris, bukan hanya I
16.5 Waktu pengambilan	invarian kelas, operator	$O(1)$ per operasi	menit tanpa simpanan	waktu yang melintasi jam dan hari

Bacalah kolom terakhir dari atas ke bawah: setiap jebakan tertangkap oleh contoh *tertentu* yang dipilih pada langkah 2 metode ini. Itulah pelajaran terpenting dari keempat bab evaluasi buku ini. Contoh-contoh yang Anda pilih sebelum menulis kode menentukan kesalahan mana yang akan Anda temukan.

16.9 Memakainya Sungguhan

Kelima soal ini kecil, tetapi masing-masing adalah inti dari tugas nyata di Profitina Laundry atau usaha kecil mana pun. Hierarki catur adalah bentuk setiap "beberapa jenis dari satu hal" dalam program: layanan per kilogram atau per potong, pelanggan anggota atau bukan, pembayaran tunai atau lewat transfer QRIS. Overload `add` adalah pemakaian sehari-hari satu nama untuk satu gagasan lintas tipe. Jendela bergeser adalah cara program menemukan pola dalam aliran data yang panjang: jendela tiga jam tersibuk dalam sehari, atau rentang tujuh hari dengan pesanan ekspres terbanyak. Perkalian matriks mengubah kilogram menjadi uang untuk setiap hari dan setiap daftar harga sekaligus; dengan matriks harga yang diubah, perkalian yang sama menjawab "bagaimana bila harga setrika kita naikkan 10%?". Dan kelas `Time`, diperluas dengan jam buka toko yang sebenarnya (lebih pendek pada hari Minggu, tutup saat Idulfitri), adalah yang mencetak waktu pengambilan pada setiap struk.

Ketika menyesuaikannya, pertahankan metodenya. Nyatakan ulang soalnya beserta keputusan-keputusan yang tertulis, kerjakan contoh dengan tangan termasuk contoh batas, rancang sebelum menulis kode, dan simpan contoh-contoh tangan itu sebagai uji yang dijalankan setelah setiap perubahan.

16.10 Kesalahan yang Sering Terjadi, dan Cara Menghindarinya

- **Menerjemahkan aturan dari kalimat yang longgar.** "Bergerak sepanjang baris atau kolom" bukanlah `||`. Tuliskan aturannya sebagai syarat pada d_x dan d_y dan periksa pada contoh-contoh, termasuk "diam di tempat".
- **Mengubah kode yang diberikan alih-alih menirunya.** Bila soal memberikan kelas model, kelas-kelas baru harus mengikutinya dengan tepat: pola konstruktor yang sama, format keluaran yang sama, nama-nama yang sama. Kreativitas tempatnya di syarat-syaratnya, bukan di strukturnya.

- **Mencampur argumen bawaan dan overload.** Dua fungsi yang sama-sama dapat menerima pemanggilan yang sama membuat setiap pemanggilan seperti itu ambigu. Pilih satu mekanisme untuk satu keluarga fungsi.
- **Salah satu (*off-by-one*) pada perulangan jendela.** Ada $n - m + 1$ jendela (Persamaan (16.3)); awal terakhir adalah $n - m$, inklusif. Dalam C++, jaga kasus $m > n$ sebelum ada pengurangan ukuran *unsigned*.
- **Menguji hanya dengan data simetris.** Matriks identitas menyembunyikan kesalahan indeks. Pakailah contoh tak simetris yang dihitung dengan tangan.
- **Lupa menginisialisasi jumlah.** Setiap C_{ij} harus dimulai dari 0. `assign(n, std::vector<long long>(p, 0))` melakukannya dalam satu baris.
- **Menyimpan nilai dalam bentuk yang memungkinkan keadaan tidak sah.** Jam dan menit yang disimpan terpisah memungkinkan 14:400. Pilih representasi dengan invarian, seperti menit sejak tengah malam, dan pulihkan invarian itu di setiap operasi.
- **Meninggalkan stream dalam keadaan berubah.** `operator<<` yang memasang `fill('0')` atau `std::left` harus memulihkan keadaan lamanya, atau keluaran berikutnya yang tidak berhubungan menjadi rusak.

16.11 Ringkasan Bab

- Metode tujuh langkah (nyatakan ulang, contoh dengan tangan, rancang, tulis kode, jalankan contohnya, coba rusak, ubah satu hal dan ramalkan) menyelesaikan lima soal yang memakai C++ dari Bab 13 sampai 15 dan dasar-dasar seluruh buku.
- Soal 16.1 melengkapi hierarki kelas dengan meniru; aturan setiap bidak menjadi syarat yang tepat, dengan atau-eksklusif untuk benteng (Persamaan (16.1)) dan `move` bersama yang memanggil `moveCondition` virtual.
- Soal 16.2 me-*overload* `add` untuk dua sampai lima argumen, masing-masing dibangun di atas yang di bawahnya (Persamaan (16.2)); argumen bawaan yang ditambahkan di samping overload yang sudah ada membuat pemanggilan ambigu.
- Soal 16.3 menemukan anagram dalam $W = n - m + 1$ jendela (Persamaan (16.3)) dengan `std::is_permutation`, dan sekali lagi dengan hitungan bergeser berwaktu linear; batas perulangan dan pengurangan *unsigned* adalah jebakannya.
- Soal 16.4 mengalikan matriks menurut Persamaan (16.4) setelah memeriksa Persamaan (16.5); matriks identitas tidak membongkar indeks yang tertukar, contoh tangan yang tak simetris membongkarnya.
- Soal 16.5 membangun kelas `Time` yang invariannya (Persamaan (16.6)) membuat penjumlahan, selisih, dan perbandingan masing-masing cukup satu baris, lalu memakainya untuk menghitung waktu pengambilan di sekitar jam buka.
- Setiap penyelesaian ada dalam C++ dan Python dengan keluaran identik yang diperiksa otomatis; setiap penyelesaian keliru yang menggoda gagal dengan cara yang terungkap oleh contoh yang dipilih dengan baik.

16.12 Soal Latihan

1. Tambahkan kelas `King` (raja) ke `chess.cpp` dan `chess.py` (dengan syarat dari kotak “periksa diri Anda” yang pertama) dan ujilah dari (4,4) dengan satu gerakan sah, satu tidak sah, dan satu gerakan sah kedua.

2. Perluas Soal 16.1 dengan kelas `Board` yang memegang beberapa bidak dan menolak gerakan benteng, gajah, atau ratu bila ada bidak lain di petak yang terletak tepat di antara petak awal dan tujuan. Jelaskan struktur data dan pemeriksaannya; kodenya opsional.
3. Tulislah overload `add` untuk dua sampai lima argumen `Rupiah` (Bab 15), masing-masing dibangun di atas `operator+`. Apakah overloading `add` masih berguna bila `+` sudah tersedia?
4. Ubahlah `anagram.cpp` agar huruf kapital dan huruf kecil dihitung sebagai huruf yang sama (`Wash` adalah anagram dari `shaw`). Metode yang mana memerlukan perubahan yang mana?
5. Tambahkan fungsi `transpose(const Matrix&)` ke `matrix.cpp` dan pakailah untuk menunjukkan, pada contoh Subbab 16.6.6, bahwa program yang keliru menghitung $A \times B^{\mathsf{T}}$.
6. Beri kelas `Time` pada Soal 16.5 jam buka sungguhan per hari: 08.00 sampai 20.00 pada Senin sampai Sabtu, 09.00 sampai 15.00 pada Minggu. Kapan waktu pengambilan Pak Hendra sekarang?

16.13 Penyelesaian Soal Latihan

1. `class King : public Piece` dengan pola konstruktor seperti yang lain dan `moveCondition` yang mengembalikan `std::max(std::abs(xDest - x_), std::abs(yDest - y_)) == 1` (`std::max` memerlukan `<algorithm>`); dalam Python, `max(abs(x_dest - self._x), abs(y_dest - self._y)) == 1`. Memulainya di (4,4) memerlukan koordinat awal opsional: `King(1, 4, 4)`. Gerakan (5,5), (5,7), (4,4) menghasilkan: berpindah ke (5,5); tidak sah, karena $|d_y| = 2$; berpindah kembali ke (4,4).

2. `Board` memegang `std::vector<Piece*>` (atau array 8×8 berisi `Piece*` dengan `nullptr` untuk petak kosong, yang membuat "apakah petak ini terisi?" cukup satu kali lihat). Untuk gerakan lurus atau diagonal, petak-petak yang tepat di antara awal dan tujuan ditemukan dengan melangkah dengan $s_x = \text{sign}(d_x)$ dan $s_y = \text{sign}(d_y)$ (masing-masing -1, 0, atau $+1$) dari petak awal, sebanyak $\max(|d_x|, |d_y|) - 1$ kali; bila salah satunya terisi, gerakannya tidak sah. Kuda melompat dan tidak memerlukan pemeriksaan ini. Bidak-bidak lalu harus dapat bertanya kepada papan, sehingga `moveCondition` mendapat parameter `const Board&`: perubahan rancangan kecil yang menunjukkan mengapa "bidak sendirian di papan" adalah keputusan penting di Subbab 16.3.1.

3. `Rupiah add(const Rupiah& a, const Rupiah& b) { return a + b; }` dan ketiga overload yang lebih besar dibangun di atasnya seperti Persamaan (16.2). Dengan `operator+` tersedia, `a + b + c` sudah lebih pendek daripada `add(a, b, c)`, sehingga overload ini tidak menambahkan apa pun; ia hanya berguna bila nama fungsi diperlukan, misalnya untuk memberikan "penjumlahan" kepada fungsi lain. Memilih notasi yang wajar adalah bagian dari rancangan yang baik.

4. Ubah kedua string menjadi huruf kecil lebih dulu dengan `std::tolower` (dan *cast* `unsigned char` dari Subbab 13.4), sekali saja, sebelum mencari. Kedua metode lalu tidak memerlukan perubahan apa pun, dan substring yang dicetak sebaiknya diambil dari teks *asli* agar keluarannya menunjukkan huruf yang sebenarnya. Dalam Python, `text.lower()` dan `pattern.lower()` melakukan hal yang sama.

5. `transpose` membangun matriks `t` berukuran $p \times n$ dengan `t[j][i] = m[i][j]`. `multiply(a, transpose(b), c)` pada $A = \begin{pmatrix} 2 & 3 \\ 5 & 7 \end{pmatrix}$ dan $B = \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}$, dengan $B^{\mathsf{T}} = \begin{pmatrix} 1 & 3 \\ 2 & 4 \end{pmatrix}$, menghasilkan $\begin{pmatrix} 8 & 18 \\ 19 & 43 \end{pmatrix}$, persis keluaran program yang keliru ($C_{00} = 2 \cdot 1 + 3 \cdot 2 = 8$). Dan untuk identitas, $I^{\mathsf{T}} = I$, itulah sebabnya uji tersebut lolos.

6. Ganti kedua konstanta 8 dan 20 dengan tabel kecil berisi menit buka dan tutup per hari, dicari dengan `day % 7`. Aturan **Pseudokode 16.4** menjadi: bila waktu selesai sebelum jam buka hari itu, ambil saat buka; bila sesudah jam tutup hari itu, ambil saat buka *hari berikutnya*. Cucian Pak Hendra selesai pukul 06.00 pada hari 7, hari Senin, yang buka pukul 08.00, sehingga tidak ada yang berubah baginya: Senin 08.00. Namun penitipan setrikaan pada hari Minggu pukul 13.00 akan berubah: selesai pukul 16.00, sesudah jam tutup Minggu pukul 15.00, sehingga pengambilannya pindah ke Senin pukul 08.00.

Enam belas bab yang lalu, program pertama Anda mencetak satu baris. Sejak itu Anda telah belajar membuat komputer memutuskan, mengulang, mengingat, membaca berkas, dan kini memodelkan benda-benda dunia nyata sebagai kelas yang menjaga aturannya sendiri. Soal-soal bab ini kecil, tetapi metode yang Anda pakai untuk menyelesaikannya berlaku untuk program sebesar apa pun: katakan dengan tepat apa yang Anda inginkan, cobalah dengan tangan, rancang sebelum mengetik, dan biarkan contoh-contoh yang dipilih dengan baik menguji setiap langkah.

Daftar Pustaka

[1] B. Stroustrup, *The C++ Programming Language*, 4th ed. Upper Saddle River, NJ, USA: Addison-Wesley, 2013.

[2] G. Pólya, *How to Solve It: A New Aspect of Mathematical Method*, 2nd ed. Princeton, NJ, USA: Princeton University Press, 1957.

[3] cppreference.com, “Overload resolution,” accessed Sep. 2026. [Online]. Available: https://en.cppreference.com/w/cpp/language/overload_resolution

[4] cppreference.com, “std::is_permutation,” accessed Sep. 2026. [Online]. Available: https://en.cppreference.com/w/cpp/algorithm/is_permutation

[5] Python Software Foundation, “collections — Container datatypes: Counter,” Python 3.14 documentation, accessed Sep. 2026. [Online]. Available: https://docs.python.org/3.14/library/collections.html

[6] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*, 4th ed. Cambridge, MA, USA: MIT Press, 2022.