Kuliah 1 — Komputer, C, dan Program Pertama Anda

Mata kuliah: Pemrograman Dasar (EF234101)

Buku: Irfan Subakti, *Pemrograman Dasar dengan C, C++, dan Python*, Bab 1

Dosen: Irfan Subakti — yifana@gmail.com

Hari ini kita berangkat dari nol sampai program yang berjalan: apa yang dikerjakan komputer terhadap program, mengapa kita belajar C dan Python berdampingan, meja kerja yang gratis, dan program-program pertama, diakhiri dengan kasir Profitina Laundry versi 0.1. Buku: Bab 1, Subbab 1.1–1.14.

Cakupan Pertemuan Ini

1. **Mengapa program** — meja kasir *laundry* tempat hitungan tangan meleset.
2. **Apa yang dikerjakan komputer** — perangkat keras dan perangkat lunak, bahasa pemrograman, dikompilasi dan diinterpretasi.
3. **Meja kerja** — VS Code, GCC, dan Python 3.14 di Windows, macOS, dan Linux.
4. **Program pertama** — `hello`, variabel dan memori, `scanf` dan `input`, empat tipe dasar.
5. **Aritmetika dan keputusan** — lima operator, pembagian bilangan bulat, presedensi, `if`; lalu kasir v0.1 dan perbaikannya.

Setiap program ada dalam C dan Python, dan kedua versi mencetak keluaran yang persis sama. Setiap keluaran yang ditampilkan adalah keluaran yang sungguh dicetak program. Buku: Bab 1.

Titik Berangkat: Struk yang Ditulis Tangan

Profitina Laundry adalah *laundry* kecil dekat kampus. Mbak Rina menimbang setiap kantong dan menulis harganya di struk kertas.

Cuci, kering, dan lipat Rp 8.000 per kilogram; kilat satu hari Rp 12.000; setrika saja Rp 6.000.

Pada Senin yang ramai ia menulis Rp 34.000 untuk 4,3 kg alih-alih Rp 34.400, lupa tarif minimum tiga kilogram, dan total harian tidak cocok dengan isi laci.

Komputer tidak lelah dan tidak lupa aturan yang sudah dituliskan. Menuliskan aturan itu adalah **pemrograman**; aturan yang tertulis adalah **program**.

Laundry ini menemani kita sepanjang semester: satu struk hari ini, antrean pelanggan di Bab 2, laporan harian yang tervalidasi di Bab 3, lalu berkas, pengurutan, dan kelas C++. Buku: Subbab 1.1.

Apa yang Dikerjakan Komputer

Perangkat keras adalah mesinnya: prosesor yang menjalankan instruksi, memori yang menyimpan data, media simpan, serta perangkat masukan dan keluaran.

Perangkat lunak adalah instruksinya. Memori adalah deretan byte bernomor; nomor sebuah byte adalah **alamatnya**.

Prosesor hanya memahami **bahasa mesin**. **Bahasa tingkat tinggi** seperti C atau Python memungkinkan kita menulis dalam istilah masalahnya, dan penerjemah mengerjakan sisanya.

Gagasan alamat tampak kecil hari ini. Ia kembali di `scanf` minggu ini, di array pada Bab 6, dan di pointer pada Bab 7. Buku: Subbab 1.2.

C dan Python: Dua Bahasa, Satu Cara Berpikir

C (Dennis Ritchie, Bell Labs, 1969–1973) menjalankan kernel Linux, Windows, dan macOS, basis data, bahkan interpreter Python. Kita memakai standar C17, ISO/IEC 9899:2018.

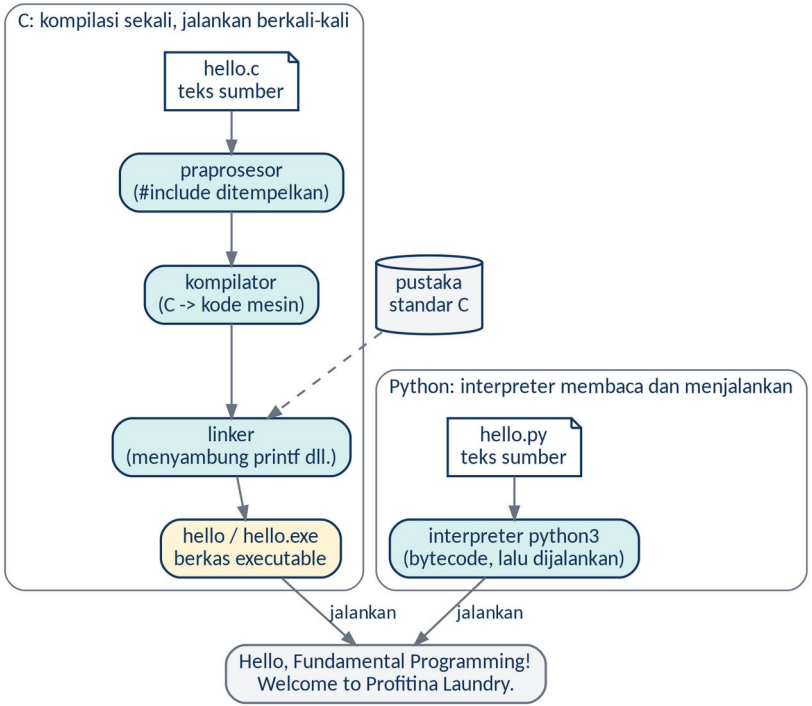
C **hanya menyembunyikan sedikit**: Anda memilih jumlah byte sebuah bilangan, Anda meneruskan alamat untuk mengubah variabel, dan tidak ada yang menghalangi pembacaan di luar array.

Python (Guido van Rossum, 1991) mengurus pembukuannya: bilangan tumbuh sesuai kebutuhan, list tumbuh, memori dilepaskan otomatis.

*Gagasan*nya (variabel, keputusan, perulangan, fungsi) sama di keduanya; *perbedaan*nya memperlihatkan apa yang dikerjakan Python untuk Anda.

Dalam indeks TIOBE September 2026 Python di tempat pertama dengan 17,76% dan C kedua dengan 10,28%. Python terkini adalah 3.14, rilis pemeliharaan 3.14.7. Buku: Subbab 1.3.1 dan 1.3.2.

Dikompilasi dan Diinterpretasi



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.

*Gambar 1.1: C **dikompilasi** sekali menjadi berkas executable berisi bahasa mesin, yang kemudian berjalan tanpa kompilator. Python **diinterpretasi**: `python3` membaca sumbernya setiap kali dijalankan, mengubahnya menjadi bytecode, dan langsung menjalankannya. Buku: Subbab 1.3.3.*

Meja Kerja

Tabel 1.1 Meja kerja buku ini; semua alat gratis dan berjalan di Windows, macOS, dan Linux.

Alat	Versi (September 2026)	Kegunaan
Visual Studio Code	1.139	editor: mewarnai kode, menjalankannya, membantu mencari galat
GCC (Windows, Linux)	16.2 terbaru; versi 9 ke atas cukup	kompilator C, perintah `gcc`
Apple clang (macOS)	bersama Xcode Command Line Tools	kompilator C yang juga menjawab `gcc`
Python	3.14.7	interpreter, `python3` (`python` atau `py` di Windows)
Ekstensi VS Code	*C/C++* dan *Python*, dari Microsoft	pewarnaan, pemeriksaan galat, *debugging*

Tabel 1.1: di Windows GCC berasal dari MSYS2, di macOS dari Xcode Command Line Tools, di Linux dari `build-essential`. Semua orang memasang VS Code beserta kedua ekstensinya. Buku: Subbab 1.4.

Memasang di Setiap Sistem

Windows — MSYS2 ke `C:\msys64`; di **MSYS2 UCRT64** jalankan `pacman -S --needed base-devel mingw-w64-ucrt-x86_64-toolchain`; tambahkan `C:\msys64\ucrt64\bin` ke Path; Python install manager, lalu `py install 3.14`.

macOS — `xcode-select --install` untuk clang; pemasang python.org untuk Python 3.14; VS Code ke folder Applications, lalu *Install 'code' command in PATH*.

Linux — `sudo apt install build-essential gdb python3`. Ubuntu 26.04 LTS membawa GCC 15.2 dan Python 3.14; Python 3.10 ke atas menjalankan semua program buku ini.

Periksa di terminal *baru*: `gcc --version`, `python3 --version` (atau `python`), `code --version`.

> **WAWASAN KUNCI:** Simpan satu folder per bab dan buka folder itu, bukan satu berkas, di VS Code. Terminal (**View → Terminal**) lalu mulai di tempat yang tepat dan perintah-perintah buku berjalan persis seperti tercetak.

Terminal membaca Path hanya ketika dimulai: tutup semua terminal yang terbuka sesudah mengubahnya. Di Mac, `gcc --version` menjawab "Apple clang", yang menerima opsi yang sama. Buku: Subbab 1.4.1–1.4.5.

Satu Perintah Kompilasi untuk Seluruh Mata Kuliah

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

`-std=c17` — ikuti standar C17. `-Wall -Wextra` — tampilkan peringatan yang lazim dan beberapa tambahan.

`-O2` — optimalkan kode mesin. `-o hello` — beri nama *executable* `hello` (`hello.exe` di Windows).

Program yang memakai pustaka matematika menambahkan `-lm` di akhir. Jalankan dengan `./hello` (Linux, macOS) atau `.\hello.exe` (PowerShell).

Setiap program buku ini juga dijalankan dengan `-fsanitize=address,undefined`, yang menghentikan program begitu ia menyentuh memori yang bukan miliknya. Buku: Subbab 1.4 dan 1.6.1.

Program Pertama 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 */
}
```

Kode 1.1 terdiri atas sembilan baris dengan enam gagasan yang muncul lagi di setiap program C. Kita membacanya di halaman berikut. Buku: Subbab 1.5.

Enam Gagasan dalam Sembilan Baris

`/* ... */` adalah **komentar**, hanya untuk manusia; komentar `//` berlaku sampai akhir baris.

`#include <stdio.h>` menyuruh **praprosesor** menempelkan deklarasi `printf` dan `scanf`.

`int main(void)` — setiap program mulai di baris pertama `main`; ia mengembalikan sebuah `int` dan tidak menerima apa pun.

Kurung kurawal mengapit **badan**; pernyataan berjalan dari atas ke bawah dan diakhiri `;`. Urutan *escape* `\n` berarti "baris baru".

`return 0;` melaporkan keberhasilan kepada sistem operasi; bilangan lain berarti ada yang salah.

Tanyakan ke kelas: apa yang terjadi bila `\n` pada `printf` pertama dibuang? Kedua sapaan jatuh di satu baris. Buku: Subbab 1.5.

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.")
```

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

Kode 1.2 tidak punya `#include`, `main`, kurung kurawal, ataupun titik koma: Python menjalankan berkas dari atas ke bawah, dan `print` menambahkan baris baru sendiri. Jalankan dengan `python3 hello.py`. Bila kedua baris ini muncul, meja kerja Anda berfungsi. Buku: Subbab 1.5.

Galat, Peringatan, dan Galat Logika

Galat (*error*) — tidak ada *executable* yang dihasilkan. `hello.c:6:48: error: expected ';' before 'printf'` berarti berkas, baris 6, kolom 48.

Kompilator sering menyadari kesalahan satu baris *terlambat*: bila pesan menunjuk baris yang tampak sempurna, lihat baris di atasnya.

Peringatan (*warning*) — program diterjemahkan, tetapi sangat mungkin bukan yang Anda maksud. Anggap setiap peringatan sebagai galat yang menyamar.

Galat logika — sah dan berjalan, tetapi menghitung hal yang salah. Hanya pengujian terhadap jawaban yang sudah diketahui yang menemukannya.

> **PERINGATAN:** Program yang berjalan belum tentu program yang benar. Jangan abaikan peringatan dengan alasan "programnya tetap jalan".

Python melaporkan kurung yang tidak ditutup sebagai `SyntaxError: '(' was never closed` sebelum satu baris pun berjalan. Buku: Subbab 1.6.2.

Variabel adalah Kotak Bernama

DEFINISI. *Variabel* adalah tempat bernama di memori yang menyimpan satu nilai. Ia memiliki **nama**, **tipe**, **nilai**, dan **alamat**.

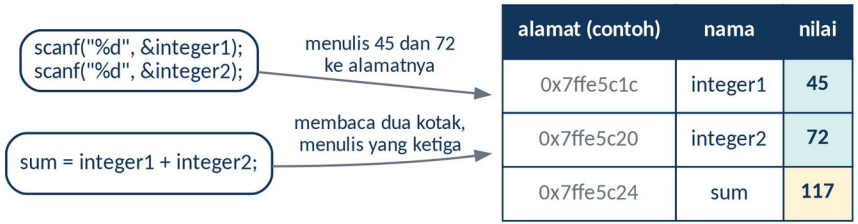
`int integer1;` adalah **deklarasi**: sebuah kotak yang cukup untuk satu bilangan bulat, bernama `integer1`.

Nama memakai huruf, angka, dan `_`, tidak diawali angka, dan membedakan huruf besar-kecil. `weightKg` lebih baik daripada `w`.

`sum = integer1 + integer2;` adalah **penugasan**: hitung ruas kanan, simpan di kotak sebelah kiri. `=` berarti "menjadi".

`count = count + 1;` *omong kosong sebagai matematika, tetapi masuk akal sebagai instruksi. Buku: Subbab 1.7.1.*

Tiga Variabel di Memori



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`.

Gambar 1.2 memperlihatkan nama, nilai, dan alamat sekaligus. Alamatnya hanya contoh; berbeda di setiap komputer dan setiap kali dijalankan. Buku: Subbab 1.7.1.

Membaca Masukan: `scanf` dan `input`

Pada `scanf("%d", &integer1)`, **format** `"%d"` membaca bilangan bulat desimal; `&integer1` adalah **alamat** kotaknya.

`scanf` memerlukan alamat karena ia *menulis* ke dalam kotak; `printf` hanya membaca nilainya, jadi menerima `integer1` tanpa `&`.

`scanf` mengembalikan banyaknya nilai yang terbaca. `if (scanf(...) != 1) return 1;` menghentikan program bila pengguna mengetik `abc`.

`input` di Python mengembalikan **teks**; `int(...)` atau `float(...)` mengubahnya menjadi bilangan.

Pemeriksaan itu juga diperlukan agar kompilasi bersih: di banyak sistem Linux GCC memperingatkan bila hasil `scanf` diabaikan. Buku: Subbab 1.7.2.

Empat Tipe Dasar

Kode 1.3 Cuplikan `types.c`: sebuah `int`, sebuah `double`, dan sebuah `char`, serta `int` terbesar.

```c
    int kilos = 5;           /* bilangan bulat */
    double price = 8000.0;   /* bilangan yang punya bagian pecahan */
    char grade = 'A';        /* satu karakter, disimpan sebagai bilangan kecil */

    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);
```

```
bill  = 40000.00
grade = A, stored as the number 65
largest int  = 2147483647
```

`int`: empat byte, -2147483648 sampai 2147483647. `double`: delapan byte, sekitar 15 sampai 16 digit bermakna. `char`: satu byte. `long long`: delapan byte, sampai sekitar 9.2×10^{18}.

Kode 1.3: `%d`, `%f`, dan `%c` adalah spesifikasi konversi; dicetak dengan `%d`, huruf A menampakkan bilangan ASCII-nya, 65. `INT_MAX` berasal dari `limits.h`. Python tidak punya `char` dan tidak membatasi `int`. Buku: Subbab 1.7.3.

Lima Operator dan Satu Jebakan

`+`, `-`, `*`, `/`, dan `%` (sisa bagi). Bila kedua ruas `/` bilangan bulat, C melakukan **pembagian bilangan bulat**: `7 / 3` bernilai 2. `%` memberikan apa yang dibuang pembagian bulat: `7 % 3` bernilai 1. Keduanya selalu cocok:

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

Untuk $a=7$, $b=3$, Persamaan (1.1) terbaca $7 = 2\cdot 3 + 1$. Untuk bilangan negatif C memotong ke arah nol: `-7 / 2` bernilai -3 dan `-7 % 2` bernilai -1.

Uang kembalian, menit menjadi jam, "setiap giliran ketiga" dengan `turn % 3 == 0`: semuanya pembagian bulat dan sisa bagi. Buku: Subbab 1.8.1.

Contoh Terurai 1: Program Aritmetika

Kode 1.4 Cuplikan `arithmetic.c`: pembagian bilangan bulat, sisa bagi, sebuah `double`, dan presedensi.

```c
    printf("7 / 3 = %d\n", 7 / 3);        /* pembagian bilangan bulat */
    printf("7 %% 3 = %d\n", 7 % 3);       /* sisa pembagian */
    printf("7.0 / 2 = %.2f\n", 7.0 / 2);  /* satu double membuatnya tepat */
    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);
```

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

Kerjakan Kode 1.4 di papan sebelum menampilkan keluarannya. `%%` mencetak satu tanda persen. Satu `double` di salah satu ruas mempertahankan pecahan. Buku: Subbab 1.8.1.

Presedensi: Siapa Lebih Dulu

Tabel 1.2 Presedensi operator yang sudah kita kenal; bila ragu, tambahkan tanda kurung.

Operator	Arti	Urutan
`( )`	tanda kurung	pertama; pasangan terdalam lebih dulu
`* / %`	perkalian, pembagian, sisa bagi	berikutnya; dari kiri ke kanan
`+ -`	penjumlahan, pengurangan	berikutnya; dari kiri ke kanan
`=`	penugasan	terakhir

Menurut Tabel 1.2, `a + b + c / 3` hanya membagi `c`: 85/3=28, ditambah 170, menjadi 198, yang bahkan tidak berada di antara ketiga nilai.

Di Python `/` selalu mempertahankan pecahan dan `//` adalah pembagian bulat yang membulatkan ke bawah, bukan ke arah nol. Tanda kurung gratis dan menyatakan maksud Anda. Buku: Subbab 1.8.2 dan 1.8.3.

Keputusan dengan `if`

`if (kondisi) pernyataan` menjalankan pernyataan hanya bila kondisinya benar.

Enam operator pembanding: `==`, `!=`, `<`, `>`, `<=`, `>=`.

`=` menugaskan; `==` membandingkan. `if (x = 5)` menugaskan 5 dan selalu benar; `-Wall` memperingatkannya.

Di Python badannya adalah blok berindentasi sesudah `if kondisi:`; indentasi adalah tata bahasa, bukan hiasan.

`compare.c` di buku menjaga satu `printf` dengan masing-masing dari enam operator; jalankan dengan 7 dan 7, lalu 3 dan 9, dan tanyakan baris mana yang muncul. Buku: Subbab 1.9.

Contoh Terurai 2: Kasir v0.1

Kode 1.5 Cuplikan `receipt.c`: kasir v0.1 membaca berat dan mencetak total.

```c
    int weightKg;               /* berat cucian, kilogram bulat */
    int pricePerKg = 8000;      /* cuci, kering & lipat: rupiah per kilogram */
    int total;                  /* jumlah yang harus dibayar, dalam rupiah */

    printf("Weight (kg): ");
    if (scanf("%d", &weightKg) != 1) return 1;
    total = weightKg * pricePerKg;
    printf("TOTAL    : Rp %d\n", total);
```

Komentar setiap variabel menyebutkan **satuannya**. Uang disimpan sebagai rupiah bulat: tepat, tanpa sen.

Kode 1.5 adalah inti program lengkap di buku, yang juga mencetak nama layanan dan harga per kilogram. Untuk 5 kg ia mencetak `TOTAL : Rp 40000`. Buku: Subbab 1.10.

Kasir v0.1 dalam Python

Kode 1.6 Cuplikan `receipt.py`: kasir v0.1 dalam Python.

```
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(f"TOTAL    : Rp {total}")
```

Kode 1.6: `int(input(...))` mengubah teks ketikan menjadi bilangan, dan f-string menyisipkan nilai ke dalam teks. `print()` saja mencetak baris kosong, padanan `\n` di awal. Buku: Subbab 1.10.

Lebih Baik: Desimal dan Tarif Minimum

Kode 1.7 Cuplikan `receipt_min.c`: berat desimal, minimum tiga kilogram, dan pembulatan ke rupiah utuh.

```
    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 */
```

`double weight` dibaca dengan `%lf` dan dicetak dengan `%.1f`, satu desimal, seperti yang ditampilkan timbangan.

Tagihkan berat sebenarnya; *bila* di bawah 3.0, tagihkan 3.0. Menambahkan 0.5 sebelum *cast* ke `int` membulatkan ke rupiah terdekat.

Kode 1.7 adalah jantung kasir yang diperbaiki. Kurung kurawal di sekitar badan `if` memperjelas pernyataan mana yang termasuk. Buku: Subbab 1.11.

Dua Kali Jalan, Dua Jalur

4,3 kg di atas minimum: $4.3 \times 8000 = 34400$.

2,2 kg di bawahnya: struk menulis `Minimum  : 3.0 kg charged` dan pelanggan membayar $3 \times 8000 = 24000$.

```
Weight (kg): 2.2

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

Tulis struk yang diharapkan di kertas sebelum program dijalankan, lalu bandingkan. Uji kantong terkecil, terbesar, satu tepat 3.0 kg, dan satu berat desimal. Buku: Subbab 1.11 dan 1.12.

Memakainya Sungguhan

Bentuk yang sama cocok untuk toko fotokopi (per halaman), depot isi ulang air (per galon), parkir (per jam), penjual beras (per kilogram).

Untuk mengadaptasi `receipt_min`: ganti nama layanan, harga per satuan, dan minimumnya (atau hapus).

Program hanya sebenar aturannya: tulis harga dan tanggal berlakunya dalam komentar di bagian atas.

Bab 11 memindahkan harga dari program ke berkas yang dapat disunting pemilik. Buku: Subbab 1.12.

Kesalahan yang Sering Terjadi

Lupa `;` — dilaporkan di baris berikutnya. **Lupa `&` pada `scanf`** — peringatan, lalu mogok atau sampah.

Konversi tertukar — `double` dicetak dengan `%d` atau dibaca dengan `%f` alih-alih `%lf`. **`=` alih-alih `==`**.

Menjalankan *executable* lama sesudah menyunting tanpa mengompilasi ulang. **Tab dan spasi bercampur** di Python.

Pembagian bulat padahal yang dimaksud pecahan — Kode 1.8 lolos kompilasi tanpa satu peringatan pun.

Kode 1.8 Cuplikan `wrong_average.c`: `1 / 2` bernilai 0, sehingga separuh dari 7 kg menjadi 0.00 kg; `kilos / 2.0` memberi 3.50 kg.

```c
    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);
```

Obatnya: jadikan salah satu ruas pembagian `double`, atau kalikan dengan `0.5`. Buku: Subbab 1.13.

Ringkasan: Lima Hal untuk Diingat

1. C dikompilasi sekali menjadi *executable*; Python diinterpretasi setiap kali (Gambar 1.1). Kompilasi dengan `gcc -std=c17 -Wall -Wextra -O2`.
2. Program C dimulai di `main`; pernyataan diakhiri `;`; `return 0;` melaporkan keberhasilan. Python berjalan dari atas ke bawah, dan indentasi adalah tata bahasa.
3. Variabel memiliki nama, tipe, nilai, dan alamat (Gambar 1.2); `scanf` memerlukan `&` karena ia menulis ke dalam kotak.
4. Pembagian bulat membuang pecahan dan `%` menyimpan sisanya (Persamaan (1.1)); `*`, `/`, `%` didahulukan sebelum `+`, `-`.
5. `if` menjalankan pernyataan hanya bila kondisinya benar; `=` menugaskan, `==` membandingkan. Anggap setiap peringatan sebagai galat.

Kasir v0.1 membaca berat dan mencetak struk; versi perbaikannya membaca desimal, menerapkan minimum tiga kilogram, dan membulatkan ke rupiah utuh. Buku: Subbab 1.14.

Pertemuan Berikutnya

Bab 2 — Pengembangan Program Terstruktur. Algoritma, pseudokode, tiga struktur kontrol, `if`…`else`, dan perulangan `while` dengan pencacah dan dengan *sentinel*.

Kasir v0.2 akan melayani seluruh antrean pelanggan dan mencetak ringkasan akhir hari.

Sebelum itu: pasang meja kerja, jalankan `hello` dalam kedua bahasa, dan kerjakan Soal Latihan 1.2, 1.3, 1.4, dan 1.6.

Soal Latihan 1.6, kembalian dengan lembaran sesedikit mungkin, adalah `/` dan `%` yang bekerja, dan penyelesaiannya ada di buku. Buku: Subbab 1.15 dan 1.16.

Kuliah 2 — Pengembangan Program Terstruktur

Mata kuliah: Pemrograman Dasar (EF234101)

Buku: Irfan Subakti, *Pemrograman Dasar dengan C, C++, dan Python*, Bab 2

Dosen: Irfan Subakti — yifana@gmail.com

Program tidak diketik; program dirancang. Hari ini: algoritma dan pseudokode, tiga struktur kontrol, `if`…`else` beserta rantainya, perulangan `while` dengan pencacah dan dengan sentinel, penghalusan bertahap dari atas ke bawah, dan kasir v0.2. Buku: Bab 2, Subbab 2.1–2.14.

Cakupan Pertemuan Ini

1. **Algoritma dan pseudokode** — tindakan dalam urutan yang benar, ditulis untuk manusia.
2. **Tiga struktur sudah cukup** — runtunan, seleksi, perulangan; diagram alir dan blok.
3. **Seleksi** — `if`…`else`, rantai keputusan, operator kondisional `?:`.
4. **Perulangan** — `while` dengan pencacah; dengan *sentinel*, dirancang lewat penghalusan bertahap; tabel penelusuran dan *cast*.
5. **Merangkai semuanya** — struktur bersarang, `+=` dan `++`, kasir v0.2, dan pengaman terhadap masukan mustahil.

Setiap program kembali hadir dalam C dan Python dengan keluaran identik. Buku: Bab 2.

Pertemuan Sebelumnya

Kasir v0.1 mencetak **satu** struk lalu berhenti. Versi perbaikannya membaca desimal dan menerapkan minimum tiga kilogram.

Di meja kasir antrean tidak berhenti sesudah satu pelanggan. Pemiliknya, Pak Hendra, bertanya saat tutup: berapa pelanggan, berapa uang, berapa rata-rata tagihan?

Untuk menjawabnya program harus **mengulang** dan **memilih**, dan ia harus **dirancang** sebelum diketik.

Dari Kuliah 1 kita membawa `if`, enam operator pembanding, pembagian bulat, dan `%`. Buku: Subbab 2.1.

Algoritma: Tindakan dalam Urutan yang Benar

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

Tepat — "tambahkan 30 ml deterjen", bukan "sedikit". **Berhingga** — berhenti untuk setiap masukan. **Benar** — hasil yang benar untuk setiap masukan yang diizinkan, termasuk hari tanpa pelanggan.

Timbang kantongnya, lalu masukkan pakaian ke mesin. Tukar urutannya dan Anda menimbang kantong basah: tindakan sama, urutan salah.

Es teh untuk pelanggan: "celupkan kantong teh, tuang air, tambah gula, aduk" tidak benar (air dingin), tidak tepat (berapa gula?), dan tidak berhingga (aduk sampai kapan?). Komputer mengikuti persis apa yang tertulis. Buku: Subbab 2.2.

Pseudokode

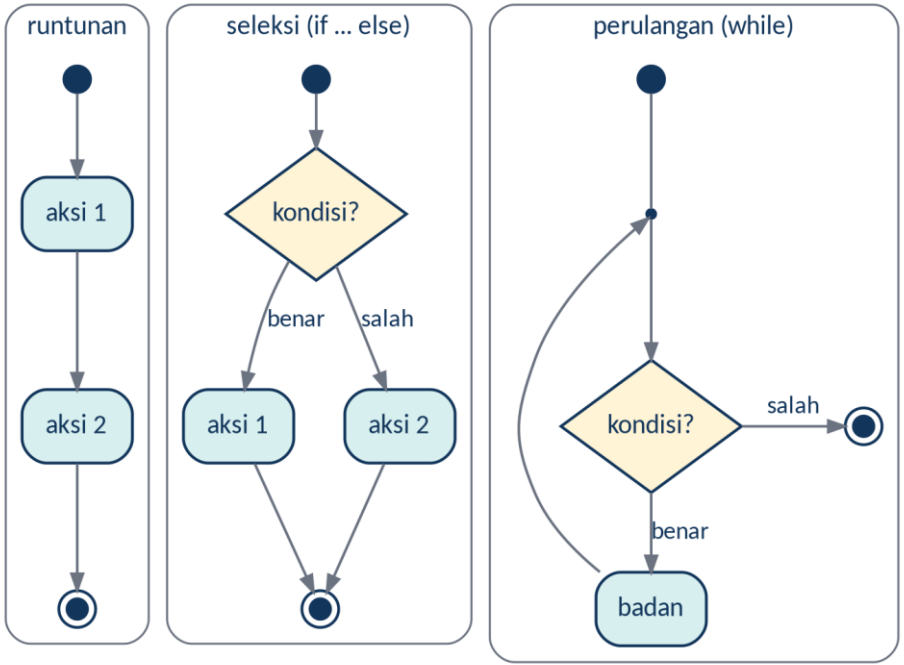
Pseudokode 2.1 Pseudokode untuk program struk dari Bab 1.

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

Pseudokode 2.1: nama di puncak, baris bernomor, `←` untuk penugasan, indentasi untuk struktur, hanya tindakan yang dapat dijalankan (tanpa deklarasi).

Pseudokode tidak pernah dikompilasi; ia dibaca manusia pada saat merancang, ketika mengubah satu baris hanya butuh beberapa detik. Habiskan sepuluh menit pertama jauh dari papan ketik. Buku: Subbab 2.3.

Tiga Struktur Sudah Cukup



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

Gambar 2.1*: Böhm dan Jacopini (1966) membuktikan bahwa setiap algoritma dapat ditulis dengan tiga struktur ini, digabung dengan cara ditumpuk dan disarangkan; surat Dijkstra "Go To Statement Considered Harmful" (1968) menyusul. Buku: Subbab 2.4.1.*

Satu Jalan Masuk, Satu Jalan Keluar

Runtunan — tindakan satu demi satu. **Seleksi** — kondisi memilih jalur: `if`, `if`…`else`, rantai, dan `switch` di Bab 3.

Perulangan — tindakan diulang selama kondisi berlaku: `while` hari ini; `for` dan `do`…`while` di Bab 3.

Setiap persegi dapat diganti seluruh struktur (**bersarang**) atau disusul struktur lain (**bertumpuk**); program tetap terbaca dari atas ke bawah.

Blok: kurung kurawal di C, indentasi di Python. Tanpa kurung kurawal, hanya pernyataan *pertama* sesudah `if` atau `while` yang menjadi miliknya.

Diagram alir: belah ketupat adalah keputusan dengan tepat dua jalan keluar, benar dan salah; persegi bersudut bulat adalah tindakan. Letakkan jari di jalan masuk dan ikuti panahnya. Buku: Subbab 2.4.2 dan 2.4.3.

Rantai Keputusan: Diskon

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)
```

***Pseudokode 2.2**: 10% mulai Rp 100.000, 5% mulai Rp 50.000, anggota mendapat lima poin lebih. Kondisi benar pertama yang menang, jadi uji yang paling menuntut diletakkan pertama. Buku: Subbab 2.5.2.*

Kalikan Dulu, Baru Bagi

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

Persamaan (2.1): $\lfloor x \rfloor$ adalah bilangan bulat terbesar yang tidak melebihi x. Untuk bilangan positif itulah pembagian bulat C: `bill * percent / 100`; di Python `bill * percent // 100`.

`bill / 100 * percent` membuang dua digit terakhir tagihan sebelum mengalikan.

Dikerjakan dengan tangan: $b = 120000$ lolos kedua uji; uji 10% di depan, jadi $p = 10$; anggota mendapat $p = 15$ dan $d = 18000$.

Tanyakan: bila uji 5% diletakkan pertama, tagihan Rp 120.000 mendapat berapa? Hanya 5%, begitu pula setiap tagihan besar. Buku: Subbab 2.5.2.

Rantai dalam C

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

```c
    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");
```

*Dalam **Kode 2.1**, `member == 1 ? "member" : "guest"` adalah **operator kondisional**: sebuah `if`…`else` yang menghasilkan nilai. Python menulis rantainya dengan `elif` dan ekspresinya sebagai `'member' if member == 1 else 'guest'`. Buku: Subbab 2.5.2 dan 2.5.3.*

Perulangan `while`

`while (kondisi) { pernyataan }` — uji dulu; bila salah, lewati perulangan; bila benar, jalankan badan sekali lalu uji lagi.

Satu kali badan berjalan disebut **iterasi**. Sesuatu di badan harus akhirnya membuat kondisi salah, atau perulangannya **tak berujung**.

Perulangan terkendali pencacah: banyaknya iterasi diketahui. Pencacah mulai dari 1 dan berjalan selama paling banyak 5.

Akumulator seperti `total` mulai dari 0 dan bertambah satu nilai setiap iterasi. Tanpa inisialisasi, ia mulai dengan sampah.

Bila program berhenti merespons, tekan Ctrl+C dan cari pernyataan yang seharusnya mengubah kondisi perulangan. Buku: Subbab 2.6.1 dan 2.6.2.

Menghitung sampai Lima

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
```

Pseudokode 2.3 *membaca tepat lima berat. Baris 8 membagi dua bilangan bulat, jadi rata-ratanya kehilangan pecahan; versi sentinel memperbaikinya dengan cast. Buku: Subbab 2.6.2.*

Masalah yang Lebih Sulit: Berapa Pesanan?

Pak Hendra ingin rata-rata berat **semua** pesanan hari itu. Pagi hari tidak ada yang tahu berapa banyaknya.

Nilai sentinel adalah nilai khusus yang tidak mungkin menjadi data sungguhan sehingga berarti "data habis". Tidak ada kantong seberat 0 kg, jadi 0 mengakhiri hari.

Penghalusan bertahap dari atas ke bawah: mulai dari satu kalimat, **puncak**, lalu pecah setiap pernyataan menjadi yang lebih kecil sampai setiap baris langsung dapat diterjemahkan ke C atau Python.

Pseudokode 2.4 Puncak: satu kalimat yang menyatakan seluruh tugas.

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

Pseudokode 2.4 *lengkap tetapi terlalu samar untuk diprogram. Buku: Subbab 2.7.1 dan 2.7.2.*

Penghalusan Pertama

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
```

Pseudokode 2.5: tiga tahap yang dimiliki hampir setiap program.

Menghaluskan baris 2 menyingkap satu hal halus: perulangan harus menguji berat **sebelum** menjumlahkannya, supaya *sentinel* tidak pernah terhitung. Baca sekali sebelum perulangan, dan sekali lagi di akhir badan.

Menghaluskan baris 3 menyingkap hal lain: hari tanpa pesanan memiliki $n = 0$, dan rata-rata tidak boleh membagi dengan nol.

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

Persamaan (2.2) adalah rata-rata biasa yang kita terapkan. Kedua keputusan kunci, membaca lebih dulu dan melindungi pembagian, ditemukan di atas kertas. Buku: Subbab 2.7.3.

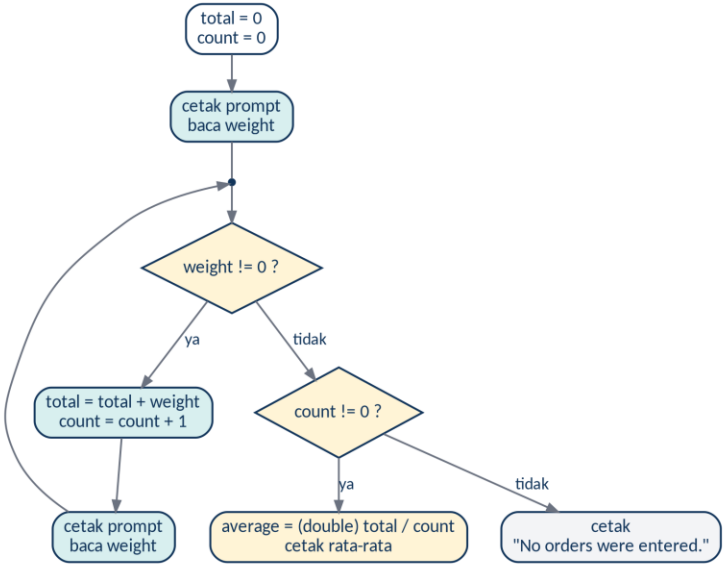
Penghalusan Kedua

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."
```

*Setiap baris **Pseudokode 2.6** kini langsung dapat diterjemahkan. Buku: Subbab 2.7.3.*

Perulangan Sentinel sebagai Diagram Alir



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.

*Ikuti **Gambar 2.2** dengan jari: satu kali baca, belah ketupat, jumlahkan, hitung, baca, kembali ke belah ketupat; belah ketupat kedua melindungi pembagian. Buku: Subbab 2.7.3.*

Programnya dalam C

Kode 2.2 Cuplikan `average_sentinel.c`: perulangan terkendali sentinel, sebuah *cast*, dan pengaman terhadap pembagian dengan nol.

```
    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("Average weight: %.2f kg\n", average);
    } else {
        printf("No orders were entered.\n");
    }
```

Kode 2.2 *mengikuti pseudokode baris demi baris. Dengan 4, 7, 3, 0 ia mencetak* `Orders: 3, total 14 kg` *dan* `Average weight: 4.67 kg`*; dengan 0 langsung ia mencetak* `No orders were entered.` *Buku: Subbab 2.7.4.*

Contoh Terurai 1: Tabel Penelusuran

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

*Isi **Tabel 2.1** di papan baris demi baris. Badan berjalan tiga kali, sentinel tidak pernah dijumlahkan, `count` bernilai 3. Dua menit penelusuran menemukan sebagian besar galat off-by-one. Buku: Subbab 2.7.5.*

Cast

`total` dan `count` bertipe `int`; `total / count` akan membuang pecahan.

`(double) total` membuat salinan `double` sementara; variabelnya tetap `int`. **Promosi** lalu mengubah `count` juga, dan $14/3 = 4.666\ldots$ selamat.

`(double) (total / count)` sudah terlambat: pembagian bulat telah membuang pecahan, dan *cast* hanya mengubah 4 menjadi 4.0. *Cast* operannya, bukan hasil jadinya.

Python tidak memerlukan cast: `/` selalu memberi pecahan, `//` adalah pembagian bulat. Buku: Subbab 2.7.6.

Algoritma yang Sama dalam Python

Kode 2.3 Cuplikan `average_sentinel.py`: algoritma yang sama dalam Python.

```
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"Average weight: {average:.2f} kg")
else:
    print("No orders were entered.")
```

*Dalam **Kode 2.3** indentasi saja yang menunjukkan baris mana milik `while`. Buku: Subbab 2.7.6.*

Struktur Kontrol Bersarang

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!"
```

***Pseudokode 2.7**: sepuluh pelanggan menjawab 1 (puas) atau 2 (tidak); lebih dari delapan puas dan Pak Hendra mentraktir tim makan siang. Sebuah `if`…`else` berada di dalam `while`, dan sebuah `if` lain ditumpuk sesudahnya. Buku: Subbab 2.8.*

Membaca Struktur Bersarang

`while` berjalan tepat sepuluh kali; di dalamnya, tepat satu dari dua pencacah bertambah setiap iterasi, sehingga `satisfied + unhappy` selalu 10 di akhir.

Keputusan "traktir tim" **ditumpuk sesudah** perulangan, tidak bersarang di dalamnya: keputusan itu diambil sekali, ketika kesepuluh jawaban sudah masuk.

Di C kurung kurawal menunjukkan apa yang menjadi milik `while`; di Python indentasi saja yang menunjukkannya. Geser `if` terakhir satu tingkat ke dalam dan ia berjalan sepuluh kali.

Dikerjakan dengan tangan: jawaban 1, 1, 2, 1, 1, 1, 1, 1, 1, 1 memberi 9 puas dan 1 tidak puas, dan $9 > 8$, jadi makan siang ditanggung pemilik.

Mintalah kelas memberi masukan yang menghasilkan tepat 8 puas: kondisinya salah dan tidak ada makan siang, sebuah batas yang layak diuji. Buku: Subbab 2.8.

Penugasan Gabungan dan `++`

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

`++c` menambah 1 lalu memakai nilai baru; `c++` memakai nilai lama lalu menambah 1 sesudahnya. Dengan `c = 5`: `c++` menampilkan 5, `++c` menampilkan 6; keduanya menyisakan 6.

***Tabel 2.2**: variabel disebut sekali saja, jadi `total = totl + weight` tidak mungkin terjadi. Python tidak punya `++`; `c += 1` adalah pernyataan. Jangan pernah menulis `c += c++;`. Buku: Subbab 2.9.*

Kasir v0.2: Rancangannya

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)
```

Pseudokode 2.8 *menumpuk semua bahan hari ini: perulangan sentinel, rantai keputusan, akumulator, dan rata-rata yang dilindungi. Buku: Subbab 2.10.*

Kasir v0.2: Perulangannya dalam C

Kode 2.4 Cuplikan `cashier_v02.c`: perulangan kasir v0.2, dengan `continue` untuk layanan yang tidak dikenal.

```
    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;
```

*Dalam **Kode 2.4**, `continue` melewati sisa badan dan melompat kembali ke uji perulangan, sehingga tidak ada berat yang ditanyakan untuk layanan yang tidak ada. Buku: Subbab 2.10.*

Contoh Terurai 2: Satu Hari yang Singkat

Masukan: layanan 1 dengan 5 kg, layanan 2 dengan 3 kg, salah ketik 7, layanan 3 dengan 2 kg, lalu 0.

Dengan tangan: $5 \times 8000 = 40000$; $3 \times 12000 = 36000$; angka 7 ditolak dan tidak dihitung; $2 \times 6000 = 12000$.

Total Rp 88.000 dari tiga pelanggan; rata-rata $88000/3 = 29333.33$.

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

Kerjakan di papan lebih dulu; program harus sepakat dengan hitungan tangan sampai digit terakhir. Buku: Subbab 2.10.

Lebih Baik: Masukan yang Mustahil

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
```

*v0.2 menerima 0 atau −3 kg, dan tagihan negatif mengurangi pendapatan. **Pseudokode 2.9** bertanya lagi; batas 50 kg adalah keputusan bisnis. Bab 3 memberi `do`…`while` dan `||` untuk tugas ini. Buku: Subbab 2.11.*

Memakainya Sungguhan

Baca permintaan, periksa, tangani sesuai jenisnya, tambahkan ke total: bentuk setiap program kasir.

Depot isi ulang air: 1 = isi ulang biasa Rp 6.000, 2 = mineral Rp 8.000, 3 = galon baru Rp 45.000, 0 = tutup; berat menjadi banyaknya galon.

Simpan uang dalam rupiah bulat; pilih *sentinel* yang tidak mungkin menjadi data sungguhan; uji hari yang kosong.

0 aman untuk nomor layanan, tetapi tidak untuk besaran diskon, yang memang bisa 0. Buku: Subbab 2.12.

Kesalahan yang Sering Terjadi

Akumulator tanpa inisialisasi; **memproses *sentinel***; **perulangan tak berujung** karena pembaruan terlupa.

`(double) (total / count)` — terlambat; **membagi dengan nol** tanpa `if (count != 0)`.

`while (weight != 0);` — titik koma nyasar; **urutan salah** dalam rantai `else if`; **indentasi tanpa kurung kurawal** di C.

***Sentinel* yang bisa menjadi data**; `==` **pada pecahan**; **hanya menguji jalur mulus**: coba kosong, tunggal, biasa, dan setiap batas.

Batas untuk diskon: tepat Rp 50.000 dan Rp 100.000. Buku: Subbab 2.13.

Ringkasan: Lima Hal untuk Diingat

1. Algoritma itu tepat, berhingga, dan benar; pseudokode menuliskannya untuk manusia dengan `←` dan indentasi.

2. Runtunan, seleksi, dan perulangan, masing-masing dengan satu jalan masuk dan satu jalan keluar, ditumpuk dan disarangkan, membangun setiap program (**Gambar 2.1**).

3. Dalam rantai keputusan kondisi benar pertama yang menang; `?:` memilih di antara dua nilai.

4. Perulangan *sentinel* membaca sebelum perulangan dan di akhir badan, serta melindungi pembagian (**Gambar 2.2**); penghalusan bertahap menemukan keputusan itu di atas kertas.

5. *Cast* operannya, `(double) total / count`, bukan hasilnya; `+=` dan `++` menyingkat pembaruan.

Kasir v0.2 melayani pelanggan sampai 0, menolak layanan tak dikenal dengan `continue`, dan mencetak jumlah pelanggan, pendapatan, serta rata-rata tagihan. Buku: Subbab 2.14.

Pertemuan Berikutnya

Bab 3 — Kontrol Program. `for`, `do`…`while`, `switch`, `break` dan `continue`, serta operator logika `&&`, `||`, dan `!`.

Kasir v0.3 memvalidasi setiap berat, memilih layanan dengan `switch`, dan mencetak laporan harian berformat.

Sebelum itu: Soal Latihan 2.1, 2.2, 2.5, dan 2.6.

Soal Latihan 2.5 adalah tabel penelusuran untuk masukan 5, 0: kerjakan dengan tangan sebelum menjalankan apa pun. Buku: Subbab 2.15 dan 2.16.

Kuliah 3 — Kontrol Program: for, do...while, switch, dan Operator Logika

Mata kuliah: Pemrograman Dasar (EF234101)

Buku: Irfan Subakti, *Pemrograman Dasar dengan C, C++, dan Python*, Bab 3

Dosen: Irfan Subakti — yifana@gmail.com

Hari ini kita melengkapi kotak perkakas struktur kontrol: `for`, `do...while`, `switch`, `break` dan `continue`, serta operator logika, masing-masing di samping bentuk Python-nya, diakhiri dengan kasir v0.3 dan daftar harga untuk dinding. Buku: Bab 3, Subbab 3.1–3.13.

Cakupan Pertemuan Ini

1. `for` — perulangan berhitung dalam satu baris; `for` dan `range` di Python; bunga majemuk dan lebar kolom.
2. `do`…`while` — paling sedikit sekali; “tanya sampai jawabannya sah”; `while True` dengan `break` di Python.
3. `switch` — satu keputusan, banyak kasus; *fall-through*; `match` di Python.
4. `break`, `continue`, `&&`, `||`, `!` — evaluasi hubung-singkat, hukum De Morgan, aturan bisnis sebagai kondisi.
5. **Memilih struktur** — dan kasir v0.3 dengan laporan harian berformat.

Sesudah hari ini Anda mengenal semua struktur kontrol C; Bab 4 memakai semuanya untuk lima soal utuh. Buku: Bab 3.

Pertemuan Sebelumnya

Kasir v0.2 melayani antrean sampai *sentinel* 0 dan mencetak jumlah pelanggan, pendapatan, dan rata-rata tagihan.

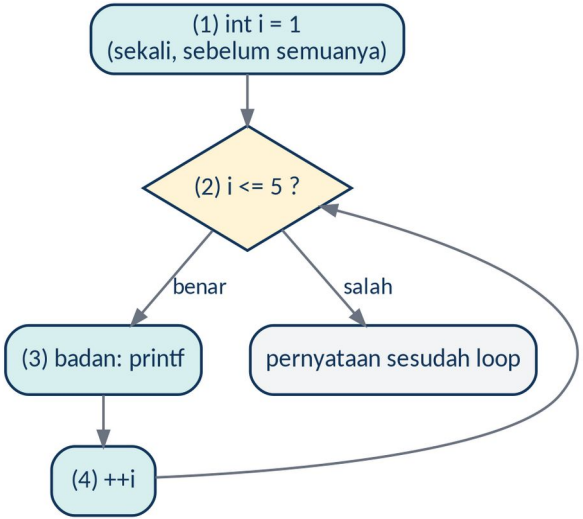
Titik lemahnya: Mbak Rina mengetik 60 alih-alih 6, dan pendapatan melonjak Rp 432.000 yang tidak pernah dibayar siapa pun. Program diam saja.

Pak Hendra menginginkan lebih: jumlah per layanan, tagihan terbesar, tabel yang dapat difoto, dan rencana tabungan untuk mesin cuci baru.

Setiap keinginan cocok dengan satu alat: menghitung sampai 10 adalah `for`; bertanya sampai masuk akal adalah `do`…`while`; memilih layanan adalah `switch`; "paling sedikit 1 dan paling banyak 50" adalah operator logika. Buku: Subbab 3.1.

Pernyataan `for`

```
for ( int i = 1 ; i <= 5 ; ++i )
        (1)       (2)    (4)
    printf("%d ", i);   (3)
```



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

Gambar 3.1: urutannya 1, 2, 3, 4, 2, 3, 4, 2, … sampai uji pada langkah 2 gagal. Ketiga bagian perulangan pencacah dari Bab 2 kini duduk di satu kepala, tempat tidak satu pun dapat terlupa. Buku: Subbab 3.2.1.

`for` adalah `while` dalam Satu Baris

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

`for (int i = 1; i <= 5; ++i)` menghitung naik; `for (int i = 5; i >= 1; --i)` menghitung mundur; `i += 7` menghitung berlangkah.

Pencacah yang dideklarasikan di kepala `for` hanya ada di dalam perulangan.

Python: `for i in range(start, stop, step)`; `stop` **tidak ikut**, jadi `range(1, 11)` memberi 1 sampai 10.

Pilih `for` bila pencacah mengendalikan perulangan: pembaca langsung melihat dari mana ia mulai, di mana berhenti, dan bagaimana bergerak. Buku: Subbab 3.2.1 dan 3.2.2.

Bunga Majemuk

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

Persamaan (3.1): setoran awal p, suku bunga tahunan r sebagai pecahan, n tahun. Sesudah satu tahun $p + p \cdot r = p(1+r)$; setiap tahun berikutnya mengalikan lagi dengan $(1+r)$.

Untuk $p = 1000000$ dan $r = 0.05$: 1.050.000 sesudah satu tahun dan 1.102.500 sesudah dua, karena $1.05^2 = 1.1025$.

Suku bunganya hanya contoh untuk aritmetika; suku bunga dan ketentuan deposito sungguhan berasal dari bank. Mintalah kelas menghitung tahun kedua dengan tangan sebelum program melakukannya. Buku: Subbab 3.3.1.

Contoh Terurai 1: Tabel Tabungan

Kode 3.1 Cuplikan `savings.c`: setoran yang tumbuh dengan bunga majemuk, dicetak sebagai tabel yang rata.

```c
    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 */
    }
```

```
Year    Amount on deposit
   1           1050000.00
   2           1102500.00
   8           1477455.44
   9           1551328.22
  10           1628894.63
```

Kode 3.1 memakai `pow` dari `math.h`; di Linux tautkan dengan `-lm`. Baris 3 sampai 7 dihilangkan di sini. Buku: Subbab 3.3.2.

Lebar Kolom

`%4d` — bilangan bulat rata kanan dalam 4 karakter, diisi spasi di kiri.

`%21.2f` — `double` dengan 2 desimal, rata kanan dalam 21 karakter. `%4s` dan `%21s` menaruh judul tepat di atas kolomnya.

`%-18s` — tanda minus membuat rata kiri; label laporan kasir v0.3 memakainya.

Python memakai angka yang sama di f-string: `{year:4d}`, `{amount:21.2f}`, dengan `>` untuk rata kanan dan `<` untuk rata kiri.

Lebar kolom mengubah sederet angka menjadi tabel yang dapat difoto dan dikirim pemilik. Buku: Subbab 3.3.3.

Berapa Tahun sampai Target?

Mesinnya seharga Rp 1.500.000. Sesudah delapan tahun deposito berisi Rp 1.477.455,44, belum cukup; sesudah sembilan, Rp 1.551.328,22.

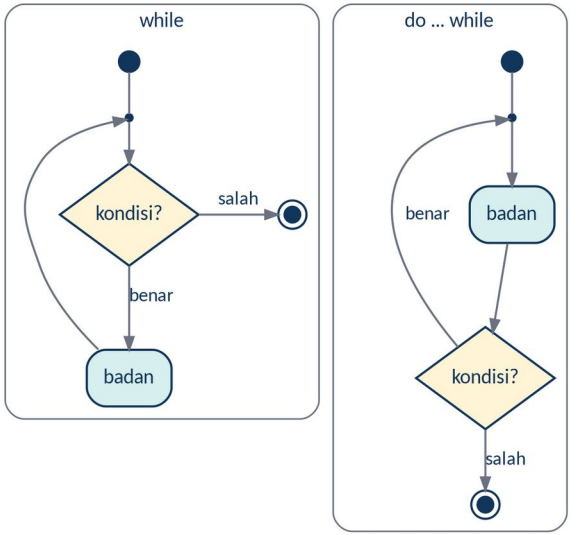
“Sesudah berapa tahun?” bukan perulangan berhitung: banyaknya tahun adalah *jawabannya*, tidak diketahui sebelumnya.

Bentuk alaminya `while`: kalikan jumlahnya dengan $1+r$ dan hitung satu tahun *selama* jumlahnya masih di bawah target.

Aturan memilih: apakah banyaknya pengulangan diketahui sebelum perulangan dimulai? Ya → `for`. Tidak → `while`.

Sembilan tahun terlalu lama untuk mesin yang sudah bocor; itulah kesimpulan sungguhan pemilik dari tabel itu. Buku: Subbab 3.3.4.

`do`...`while`: Paling Sedikit Sekali



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.

Gambar 3.2: kita tidak dapat memeriksa berat sebelum membacanya, jadi ujinya tempatnya sesudah badan. Perhatikan titik koma sesudah `while (condition);`*, yang mengakhiri pernyataan* `do`*. Buku: Subbab 3.4.1.*

Tanya sampai Jawabannya Sah

Kode 3.2 Cuplikan `validate.c`: `do`…`while` terus bertanya sampai beratnya berada dalam rentang.

```
    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 */
```

Jawaban 0, 75, 12: dua yang pertama mencetak pesan dan bertanya lagi; 12 membuat kondisi salah dan diterima. Badan berjalan tiga kali.

Dengan jawaban pertama yang baik badan berjalan sekali, tidak pernah nol kali. Kode 3.2 adalah pengaman yang diminta Bab 2. Buku: Subbab 3.4.2.

Python: `while True` dengan `break`

Kode 3.3 Cuplikan `validate.py`: bentuk Python untuk "tanya sampai sah" adalah `while True` dengan `break`.

```
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
```

Python boleh menulis `1 <= weight <= 50`, yang terbaca seperti matematika.

Di C, `1 <= weight <= 50` berarti `(1 <= weight) <= 50`: ia membandingkan 0 atau 1 dengan 50 dan **selalu benar**.

Kode 3.3 menaruh uji di tempatnya, sesudah masukan; `break` adalah satu-satunya jalan keluar. Buku: Subbab 3.4.2.

`switch`: Satu Keputusan, Banyak Kasus

Kode 3.4 Cuplikan `service_count.c`: `switch` dengan kasus bersama, `break`, dan `default`.

```
        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;
        }
```

Dalam Kode 3.4 ekspresinya dievaluasi sekali, kendali melompat ke `case` yang cocok atau ke `default`, dan berjalan sampai `break`. Perulangan di sekitarnya membaca kode dengan `scanf(" %c", &code)`; spasinya melewati karakter kosong. Buku: Subbab 3.5.1.

Fall-through

Begitu kendali melompat ke sebuah label, ia terus bergerak ke bawah melewati label berikutnya sampai bertemu `break`: ***fall-through*** (jatuh terus).

Label bertumpuk `case 'W': case 'w':` memakainya dengan sengaja. `break` yang *terlupa* memakainya tanpa sengaja; `-Wextra` memperingatkan.

`switch` hanya menguji nilai bulat (`int`, `char`, enumerasi): tanpa rentang, pecahan, atau string. Pakai `if`…`else if` untuk itu.

Mengapa `switch` alih-alih rantai: ia menyebut variabel yang diuji sekali saja dan mendaftar kasusnya seperti menu.

Rantai `if (code == 'W' || code == 'w') ++wash; else if ...` mengerjakan hal yang sama; pembaca harus memeriksa setiap kondisi untuk tahu bahwa semuanya menguji `code`. Buku: Subbab 3.5.2 dan 3.5.3.

`match` di Python

Kode 3.5 Cuplikan `service_count.py`: `match`, dengan `case "W" | "w"` mendaftar dua alternatif dan `case _` menangkap sisanya.

```
        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
```

Kode 3.5: Python 3.10 ke atas. `|` menggantikan label bertumpuk, `case _` adalah `default`, dan tidak ada fall-through maupun `break`; `break` di dalam `match` meninggalkan perulangan yang membungkusnya. Buku: Subbab 3.5.4.

`break` dan `continue`

`break` langsung mengakhiri perulangan. `continue` melewati sisa iterasi saat ini; di `for`, *increment* tetap berjalan.

`break` di 5 mencetak `1 2 3 4`; `continue` di 5 mencetak `1 2 3 4 6 7 8 9 10`.

Beberapa `break` berarti beberapa jalan keluar: sifat satu jalan masuk, satu jalan keluar hilang. Pakai `break` untuk kasus yang jelas, seperti keluar dari `while True`.

`break` hanya meninggalkan perulangan atau `switch` **terdalam**. Untuk menghentikan dua perulangan bersarang, pakai penanda seperti `found`.

`break_continue.c` di buku dan kembarannya di Python mencetak dua baris yang sama. Buku: Subbab 3.6.

Operator Logika

Tabel 3.1 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

Tabel 3.1: `&&` memerlukan keduanya, `||` memerlukan paling sedikit satu, `!` membalik. Python menulis `and`, `or`, `not`.

Presedensi: `!` lebih dulu, lalu pembanding, lalu `&&`, dan `||` terakhir. `a || b && c` berarti `a || (b && c)`; tetap tulis tanda kurungnya. Buku: Subbab 3.7.1.

Evaluasi Hubung-Singkat

Kode 3.6 Cuplikan `logic.c`: pembagian yang dilindungi evaluasi hubung-singkat.

```c
    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");
    }
```

`&&` berhenti bila ruas kirinya salah; `||` berhenti bila ruas kirinya benar. Ruas kanan **sama sekali tidak dievaluasi**.

Bila dibalik, `kilos / orders > 5 && orders != 0` membagi dengan nol lebih dulu. Urutkan bagian-bagiannya supaya masing-masing melindungi yang berikutnya.

Kode 3.6 mencetak `No verdict: no orders yet.` Python juga hubung-singkat; `and` dan `or` mengembalikan salah satu operannya, sehingga buku membungkusnya dengan `int(...)`. Buku: Subbab 3.7.2.

Hukum De Morgan

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

Berat sah: `weight >= 1 && weight <= 50`. Menurut Persamaan (3.2), "tidak sah" adalah `!(weight >= 1) || !(weight <= 50)`, yaitu `weight < 1 || weight > 50`.

Tebakan keliru `weight < 1 && weight > 50` tidak pernah benar: tidak ada berat yang memenuhi keduanya.

Bila harus menegasikan kondisi, terapkan hukumnya di atas kertas alih-alih menebak. Buku: Subbab 3.7.3.

Contoh Terurai 2: Aturan menjadi Kondisi

“Kilat hanya sampai pukul 10 pagi, kecuali untuk anggota.” → `service != 2 || hour < 10 || member == 1`.

“Pewangi gratis untuk paling sedikit 5 kg yang bukan setrika saja.” → `weight >= 5 && service != 3`.

“Tutup ketika kasir mengetik 0, atau lewat pukul 21.00.” Perulangan *berlanjut* selama `!(service == 0 || hour >= 21)`, yang menurut De Morgan adalah `service != 0 && hour < 21`.

Bacakan setiap kondisi dengan suara keras; uji pada tabel kecil, satu baris per kombinasi: empat baris untuk dua bagian, delapan untuk tiga.

Kerjakan aturan pertama di papan: pesanan bukan kilat selalu diterima, pesanan kilat di pagi hari, pesanan kilat anggota kapan saja. Buku: Subbab 3.7.4.

Memilih Struktur yang Tepat

Tabel 3.2 Memilih struktur kontrol: pertanyaan yang diajukan dan pernyataan yang menjawabnya.

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

Tabel 3.2: nomor nota 1 sampai 40 → `for`; kantong sampai 0 → `while`; nomor telepon sampai 10 hingga 13 digit → `do…while`; nomor menu → `switch`. Struktur yang menyatakan maksud Anda adalah dokumentasi termurah. Buku: Subbab 3.8.

Kasir v0.3: Rancangannya

Perulangan utama: `do`…`while`, karena kasir ditanya layanan paling sedikit sekali.

`switch` atas layanan: `case 0: break;` hanya meninggalkan `switch`, dan perulangan berakhir karena `service != 0` salah.

`case 1: case 2: case 3:` berbagi satu blok lewat *fall-through*; `do`…`while` bersarang menanyakan berat sampai 1 hingga 50 kg.

Tagihan terbesar: `largest` mulai dari 0, dan setiap tagihan yang lebih besar menggantikannya.

Di Python perulangan utamanya `while True` dengan `break`, `switch` menjadi `match` dengan `case 1 | 2 | 3`, dan pemeriksaan berat memakai `1 <= weight_kg <= 50`. Buku: Subbab 3.9.

Kasir v0.3: Satu Hari dengan Salah Ketik

Masukan: cuci 5 kg; kilat diketik 0, lalu 60, lalu 3 kg; layanan tak dikenal 4; setrika 2 kg; cuci 7 kg; 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
```

Periksa dengan tangan: 40.000 + 56.000 + 36.000 + 12.000 = 144.000 dari empat pelanggan; rata-rata 36.000; terbesar 56.000. Angka 0 dan 60 ditolak, angka 4 dilaporkan dan tidak dihitung. Buku: Subbab 3.9.

Lebih Baik: Daftar Harga

```
 kg      Wash   Express   Ironing
  1      8000     12000      6000
  2     16000     24000     12000
  3     24000     36000     18000
 10     80000    120000     60000
```

Tabel adalah perulangan di dalam perulangan: `for` luar atas kilogram, `for` dalam atas tiga layanan; badan dalam berjalan $10 \times 3 = 30$ kali.

Harganya berasal dari kondisional bersarang: `service == 1 ? 8000 : service == 2 ? 12000 : 6000`.

Baris 4 sampai 9 dihilangkan di sini. Setiap isian harus sama dengan cetakan kasir v0.3 untuk berat yang sama: dua program yang menghitung hal yang sama saling memeriksa. Buku: Subbab 3.10.

Memakainya Sungguhan

"Tanya sampai sah" cocok di mana pun orang mengetik angka: usia 0 sampai 120, bulan 1 sampai 12. Ambil batasnya dari bisnis.

Huruf yang diketik di tempat angka masih mengakhiri program (Bab 9 memperbaikinya); laporan terlupa di akhir (Bab 11 menyimpannya).

Sementara itu: `./cashier_v03 > report.txt` menyimpan keluaran program apa pun ke berkas.

50 kg tepat untuk laundry ini dan salah untuk grosir beras. Buku: Subbab 3.11.

Kesalahan yang Sering Terjadi

`for (...);` — badan kosong; batas **off-by-one**: `i < 10` berjalan 9 kali, dan `range(1, 10)` berhenti di 9.

`break` **terlupa** di `switch`; **tanpa** `;` sesudah `do`…`while`; `1 <= x <= 50` di C.

`&` **atau** `|` alih-alih && atau ||; **menegasikan tanpa De Morgan**; mengira `break` meninggalkan `if`.

Mengubah pencacah di badan `for`; `case 2.5:` atau `case "W":`; **lupa** `-lm` (`undefined reference to 'pow'`).

Menghitunglah dengan bilangan bulat; hitung pecahan dari pencacah di dalam badan. Buku: Subbab 3.12.

Ringkasan: Lima Hal untuk Diingat

1. `for` menghimpun inisialisasi, kondisi, dan *increment* di satu kepala (Gambar 3.1); `range` di Python tidak menyertakan batas akhirnya.
2. `do`…`while` berjalan paling sedikit sekali (Gambar 3.2) dan menjadi perulangan untuk “tanya sampai sah”; Python menulis `while True` dengan `break`.
3. `switch` melompat ke `case` yang cocok dan jatuh terus sampai `break`; `match` di Python punya `|`, `case _`, dan tanpa *fall-through*.
4. `&&`, `||`, dan `!` bersifat hubung-singkat, sehingga ruas kiri dapat melindungi ruas kanan; hukum De Morgan (Persamaan (3.2)) menegasikan dengan benar.
5. Pilih struktur yang menyatakan maksud Anda (Tabel 3.2); lebar kolom seperti `%4d` dan `%-18s` merapikan laporan.

Kasir v0.3 menggabungkan semuanya: perulangan utama `do`…`while`, sebuah `switch`, berat yang divalidasi, pencacah per layanan, tagihan terbesar, dan laporan yang rata. Buku: Subbab 3.13.

Pertemuan Berikutnya

Bab 4 — Evaluasi dan Pembahasan I. Lima soal utuh, masing-masing diselesaikan dari nol: bangun datar, permainan tebak angka, 100 pangkat *n*, batu-kertas-gunting, dan pertarungan melawan tiga goblin.

Tanpa pernyataan baru: setiap alat dari Bab 1 sampai 3, dipakai dengan sebuah metode.

Sebelum itu: Soal Latihan 3.2, 3.3, 3.4, dan 3.6.

Soal Latihan 3.4 adalah hukum De Morgan dalam praktik; kerjakan di kertas lebih dulu. Buku: Subbab 3.14 dan 3.15.

Kuliah 4 — Evaluasi dan Pembahasan I: Lima Soal Diselesaikan Langkah demi Langkah

Mata kuliah: Pemrograman Dasar (EF234101)

Buku: Irfan Subakti, *Pemrograman Dasar dengan C, C++, dan Python*, Bab 4

Dosen: Irfan Subakti — yifana@gmail.com

Tidak ada pernyataan baru hari ini. Kita mengambil alat-alat dari Bab 1 sampai 3 dan menaklukkan lima soal utuh dengan satu metode, dan untuk setiap soal kita membongkar sebuah solusi keliru yang menggoda. Buku: Bab 4, Subbab 4.1–4.11.

Cakupan Pertemuan Ini

0. Metodenya — tujuh langkah dari pernyataan soal yang singkat sampai program yang teruji.

1. Luas bangun datar — runtunan murni dengan satu pengaman. **2. Tebak angka** — perulangan *sentinel* yang menyamar.

3. 100 pangkat n — batas bilangan di dalam komputer.

4. Batu, kertas, gunting — validasi dan aturan ringkas. **5. Pertarungan goblin** — variabel keadaan, satu perulangan, urutan yang cermat.

6. Berdampingan — lima jebakan, dan di mana bentuk yang sama muncul dalam perangkat lunak sungguhan.

Setiap soal berjalan dengan cara yang sama: rumuskan ulang, kerjakan dengan tangan, rancang di kertas, C, Python dengan keluaran identik, solusi keliru yang menggoda, dan varian periksa diri. Buku: Bab 4.

Pertemuan Sebelumnya: Kotak Perkakas

Bab 1 — `gcc -std=c17 -Wall -Wextra -O2`; nama, tipe, nilai, alamat; `int` berakhir di 2.147.483.647; pembagian bulat dan `%`; `if`.

Bab 2 — pseudokode dan penghalusan bertahap; perulangan `while` pencacah dan *sentinel*, membaca lebih dahulu; pembagian yang dilindungi; `+=`, `++`, `(double)`.

Bab 3 — `for`, `do`…`while`, `switch` dan `match`, `break`, `continue`, `&&`, `||`, `!`, hubung-singkat, De Morgan.

Itu cukup untuk kelima soal. Fungsi, array, dan string datang di Bab 5, 6, dan 9, dan beberapa program ini menjadi lebih pendek nanti; hari ini kita menyelesaikannya dengan jalan panjang dan memahami setiap baris. Buku: Subbab 4.1.

Tujuh Langkah

1. **Rumuskan ulang** dengan kata-kata sendiri: apa yang masuk, apa yang keluar, setiap keputusan yang dibiarkan terbuka. **2. Kerjakan contoh dengan tangan**, termasuk satu yang tidak biasa.
3. **Rancang di kertas**: struktur dan pseudokode. **4. Telusuri** rancangan pada satu contoh tangan.
5. **Tulis** dari pseudokode dengan peringatan menyala. **6. Uji** terhadap contoh tangan, karakter demi karakter.
7. **Serang program Anda sendiri**: batas, nilai tidak sah, perulangan yang berjalan nol kali atau tanpa akhir.

> **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.

Tujuh langkah ini adalah versi pemrogram dari nasihat Pólya: pahami, rencanakan, laksanakan, tinjau kembali. Kode ada di `Chapter04/c`, `python`, dan `python/tempting` untuk versi yang keliru. Buku: Subbab 4.2.

Soal 4.1: Luas Bangun Datar

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.

Keputusan kita: satu *prompt* per nilai; trapesium memakai `height` yang sama; dua desimal; panjang negatif mana pun mencetak pesan tanpa luas; 0 diizinkan.

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

Dengan tangan memakai Persamaan (4.1): $b = 10$, $h = 5$ memberi 25; $b_1 = 8$, $b_2 = 4$ memberi 30; $s = 3$ memberi 9; $r = 2$ memberi $4\pi \approx 12.57$. Lingkaran berjari-jari 1 yang tercetak 3.00 membongkar aritmetika bilangan bulat. Buku: Subbab 4.3.1 dan 4.3.2.

4.1: Rancangan dan Jebakannya

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
```

Kode 4.1 Cuplikan `geometry_wrong.c`: `1 / 2` dievaluasi lebih dahulu, sebagai pembagian bulat.

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

Pseudokode 4.1 *adalah runtunan murni dengan satu pengaman; C17 tidak punya π, jadi kita mendefinisikan sebuah* `const double`*.* ***Kode 4.1*** *mencetak* `Triangle area = 0.00` *untuk setiap segitiga, tanpa peringatan. Obatnya:* `0.5`*,* `1.0 / 2`*, atau* `base * height / 2`*. Buku: Subbab 4.3.3–4.3.6.*

Soal 4.2: Tebak Angka

> **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.

Pesan persis seperti diberikan; ucapan selamat menyebutkan angka rahasia dan banyaknya tebakan, “1 try” atau “3 tries”.

Dengan tangan, rahasia 37: 50 terlalu tinggi, 25 terlalu rendah, 37 benar sesudah 3 tebakan. Kasus tidak biasa: benar pada tebakan **pertama**, hanya ucapan selamat.

Pemain cerdas yang membagi dua rentang memerlukan paling banyak 7 tebakan untuk 1 sampai 100, tetapi program tidak peduli cara pemain menebak. Buku: Subbab 4.4.1 dan 4.4.2.

4.2: Perulangan *Sentinel* yang Menyamar

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
```

*Dalam **Pseudokode 4.2** sentinel-nya bukan nilai tetap melainkan jawaban itu sendiri. Di dalam perulangan tebakan sudah pasti salah, jadi `if`…`else` sudah cukup. Buku: Subbab 4.4.3.*

4.2: Perulangannya dalam C

Kode 4.2 Cuplikan `guess.c`: permainan tebak angka sebagai perulangan `while` baca-lebih-dahulu.

```
    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");
```

Jebakannya: `do`…`while` "baca, nilai, ulangi sampai benar" ikut menilai tebakan yang menang, dan `else` mencetak "too high" untuk "sama".

*Dalam **Kode 4.2** badan hanya pernah melihat tebakan yang salah. Versi keliru benar untuk setiap tebakan salah dan gagal pada satu tebakan yang paling penting. Buku: Subbab 4.4.4 dan 4.4.6.*

> **PERIKSA DIRI.** Batasi permainan menjadi 5 tebakan. *Jawaban:* ulangi selama `guess != answer && tries < 5`; sesudah perulangan uji `guess == answer`, bukan `tries == 5`, karena tebakan tepat pada percobaan kelima juga memiliki `tries == 5`.

Soal 4.3: 100 Pangkat n

> **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.

Petunjuknya adalah seluruh soal: $100^5 = 10^{10}$ sudah lebih dari empat kali `int` terbesar.

Keputusan kita: cetak 100^n dengan **tepat** untuk $-100000 \leq n \leq 100000$; n negatif mencetak pecahan desimal yang tepat; laporkan apakah nilainya muat di `int` dan `long long`.

Soal ini bertanya apakah Anda menyadari batasnya, dan apa yang Anda lakukan terhadapnya. Buku: Subbab 4.5.1.

4.3: Cetak, Jangan Hitung

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

Menurut Persamaan (4.2), 100^n adalah angka 1 diikuti $2n$ nol: 100^3 adalah 1 dan enam nol. Untuk $n = -m$: "0.", lalu $2m - 1$ nol, lalu 1; $100^{-2} = 0.0001$.

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`?	Muat di `long long`?
4	10^8	ya	ya
5	10^{10}	tidak	ya
9	10^{18}	tidak	ya
10	10^{20}	tidak	tidak

***Tabel 4.1**: `int` menampung sampai sekitar 2.1×10^9, `long long` sampai sekitar 9.2×10^{18}. Buku: Subbab 4.5.2.*

4.3: Rancangan dan Program

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"
```

Kode 4.3 Cuplikan `power100.c`: 100 pangkat n, dicetak digit demi digit, tanpa pernah meluap.

```
    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");
    }
```

Pseudokode 4.3 *memakai* `for` *karena banyaknya nol diketahui sebelum perulangan dimulai;* ***Kode 4.3*** *mengikutinya baris demi baris. Buku: Subbab 4.5.3 dan 4.5.4.*

4.3: Jebakan `pow`

Kode 4.4 Cuplikan `power100_wrong.c`: `pow` mengembalikan `double`, yang hanya memiliki sekitar 16 angka penting.

```
    printf("100^%d = %.0f\n", n, pow(100.0, n));
```

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

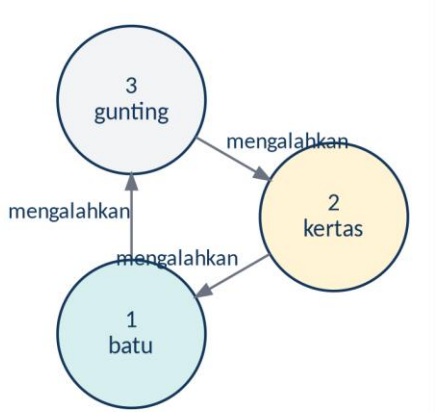
Untuk $n = 3$ jawabannya benar, dan penguji yang hanya mencoba nilai kecil akan percaya. 10^{24} tidak punya `double` yang tepat; `%.0f` dengan setia mencetak yang terdekat.

> **PERIKSA DIRI.** Cetak 1000^n dengan tepat. *Jawaban:* $1000^n = 10^{3n}$, jadi $3n$ nol; `long long` menampung $1000^6 = 10^{18}$ tetapi tidak 1000^7, jadi $n \leq 6$.

***Kode 4.4**: `100.0 ** n` di Python mencetak digit keliru yang sama, karena memakai `double` yang sama. Buku: Subbab 4.5.6.*

Soal 4.4: Batu, Kertas, Gunting

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.



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

***Gambar 4.1**: sembilan kombinasi, tiga putusan. Mengelilingi siklus, nomor pemenang selalu satu lebih besar daripada nomor yang kalah, modulo 3. Buku: Subbab 4.6.1 dan 4.6.2.*

4.4: Satu Sisa Bagi yang Memutuskan

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

Tabel 4.2 Kombinasi yang diperiksa terhadap Persamaan (4.3) (dua hasil seri lainnya juga bersisa 0): 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
1 batu	1 batu	3	0	seri
1 batu	2 kertas	2	2	pemain 2
1 batu	3 gunting	1	1	pemain 1
2 kertas	1 batu	4	1	pemain 1
2 kertas	3 gunting	2	2	pemain 2
3 gunting	1 batu	5	2	pemain 2
3 gunting	2 kertas	4	1	pemain 1

Rumus harus diperiksa, bukan dipercaya: ***Tabel 4.2*** *memeriksa Persamaan (4.3) kasus demi kasus.* `+ 3` *menjaga ruas kiri tetap positif, karena* `%` *di C mempertahankan tanda bilangan yang dibagi. Buku: Subbab 4.6.2.*

4.4: Putusannya dalam C

Kode 4.5 Cuplikan `rps.c`: pemenang ditemukan dengan satu sisa pembagian.

```
    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");
```

Rancangan: dua perulangan `do`…`while` menanyai setiap pemain sampai 1, 2, atau 3; sebuah `switch` mencetak nama; putusannya adalah **Kode 4.5**.

Jebakannya: tanpa `+ 3`, batu melawan gunting memberi $1 - 3 = -2$ dan `-2 % 3` bernilai -2 di C, sehingga gunting "mengalahkan" batu.

Hanya kombinasi dengan pemenang bernomor lebih kecil yang gagal, itulah sebabnya uji cepat kertas melawan batu lolos. Di Python `(p1 - p2) % 3` bernilai 1 dan benar. Buku: Subbab 4.6.3–4.6.6.

Soal 4.5: Pertarungan Goblin

> **SOAL 4.5.** A boy fights three goblins, turn by turn. The boy has 200 HP and an attack of 10; every third turn his skill deals 20. Each goblin has 165 HP; their attacks are 3, 4 and 1. The boy attacks first, then the goblins. Print the battle until one side is wiped out, then the number of turns and the winner's remaining HP.

Keputusan kita: si anak menyerang goblin hidup yang pertama; hanya goblin yang masih hidup **sesudah** serangannya yang membalas; HP tidak pernah di bawah 0; satu baris tabel per giliran.

Setiap keputusan terbuka mengubah hasilnya, jadi setiap keputusan dituliskan. Buku: Subbab 4.7.1.

4.5: Ramalkan Hasilnya dengan Tangan

Giliran 1, 2, 3: goblin 1 dari 165 → 155 → 145 → 125; si anak 200 → 192 → 184 → 176. Setiap tiga giliran goblin 1 kehilangan 40, si anak 24.

Goblin 1 tumbang pada giliran 13, goblin 2 pada giliran 26, goblin 3 pada giliran 39.

Si anak menerima $8 \times 12 + 5 \times 13 + 1 \times 13 = 96 + 65 + 13 = 174$ dan menang sesudah 39 giliran dengan $200 - 174 = 26$ HP.

Kerjakan ini di papan: ramalan yang dibuat sebelum program ada adalah uji terbaik yang mungkin. Buku: Subbab 4.7.2.

4.5: Satu Giliran

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.

***Gambar 4.2**: keadaannya lima bilangan, HP si anak, HP tiga goblin, dan nomor giliran. Perulangan berlanjut selama `boy > 0 && (g1 > 0 || g2 > 0 || g3 > 0)`, tanda kurungnya wajib. Buku: Subbab 4.7.3.*

4.5: Rancangannya

Pseudokode 4.4 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
```

***Pseudokode 4.4**: $[g_1 > 0]$ bernilai 1 bila goblinnya hidup dan 0 bila tidak. Satu iterasi adalah satu giliran utuh. Buku: Subbab 4.7.3.*

4.5: Pembaruan yang Dilindungi dalam C

Kode 4.6 Cuplikan `battle.c`: hanya goblin yang hidup yang membalas, dan HP tidak pernah turun di bawah 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;
```

Keluarannya berakhir dengan `The boy wins after 39 turns with 26 HP left.`, sesuai ramalan.

Jebakannya: tanpa pemeriksaan "masih hidup?". Goblin 1 tertahan di −5 HP dan terus memukul; si anak tumbang pada giliran 25 dan goblin menang. Dua pengaman yang hilang membalik putusan.

*Dalam **Kode 4.6** setiap pukulan memakai bentuk terlindung yang sama, `g1 = g1 > damage ? g1 - damage : 0`. Di versi keliru bahkan `g1 + g2 + g3 > 0` gagal: dengan HP negatif, jumlah positif tidak berarti ada yang hidup. Buku: Subbab 4.7.4 dan 4.7.6.*

> **PERIKSA DIRI.** Berapa HP awal terkecil yang masih membuat si anak menang? *Jawaban:* ia kehilangan tepat 174 HP sepanjang pertarungan, jadi 175: ia menang sesudah 39 giliran dengan sisa 1 HP.

Kelima Soal Berdampingan

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	`1 / 2` adalah pembagian bulat
4.2 Tebak angka	`while` baca-lebih-dahulu	`do`…`while` menilai tebakan yang menang
4.3 100^n	`if`…`else`, `for` atas digit	`pow` mengembalikan `double`
4.4 Batu, kertas, gunting	`do`…`while`, `switch`, satu sisa bagi	tanpa `+ 3`, `%` bernilai negatif
4.5 Pertarungan goblin	variabel keadaan, `while` dengan AND dan OR	tanpa pemeriksaan “hidup”

***Tabel 4.3**: tidak ada jebakan yang aneh. Pembagian bulat, menilai sesudah kejadian, batas tipe, tanda sisa bagi, dan pengaman yang hilang muncul di program sungguhan setiap hari. Buku: Subbab 4.8.*

Memakainya Sungguhan

4.1 adalah setiap halaman kalkulator: toko keramik yang menghitung jumlah dus untuk sebuah lantai. 4.2 adalah setiap perulangan coba-lagi: papan PIN dengan batas percobaan.

4.3: uang dalam rupiah bulat muat dengan lega di `long long`; kenali batasnya sebelum memilih tipe.

4.5 adalah simulasi: tiga mesin 60 menit per muatan, satu kantong setiap 15 menit, satu iterasi per menit. Sepanjang apa antrean menjelang siang?

Dengan alat-alat Bab 1 sampai 3 simulasi laundry itu sudah dapat ditulis. Buku: Subbab 4.9.

Kesalahan yang Sering Terjadi

Memutuskan rincian diam-diam — tuliskan di bagian atas program. **Hanya menguji kasus biasa**.

Teks keluaran tidak cocok — tanda seru yang hilang menggagalkan pembandingan otomatis.

Memercayai rumus tanpa memeriksa setiap kasus; **luapan yang tidak terlihat kompilator** — jalankan dengan `-fsanitize=undefined`.

Mencetak di cabang yang salah; **masukan yang setengah divalidasi** — periksa pemain 2 secermat pemain 1.

Setiap solusi keliru yang menggoda lolos uji biasa dan gagal pada uji yang tidak biasa. Buku: Subbab 4.10.

Ringkasan: Lima Hal untuk Diingat

1. Tujuh langkah: rumuskan ulang, contoh dengan tangan, rancang, telusuri, tulis, uji, serang. Sebagian besar waktu dipakai sebelum kode.

2. Rumus seperti Persamaan (4.1) memerlukan konstanta `double`: `0.5`, bukan `1 / 2`.

3. `while` baca-lebih-dahulu hanya memperlihatkan tebakan salah kepada badan; `do`…`while` menilai sebelum menguji.

4. Tipe punya batas (**Tabel 4.1**); mencetak digit 10^{2n} itu tepat, `pow` dibulatkan.

5. Periksa aturan ringkas pada setiap kasus (**Tabel 4.2**); lindungi setiap pembaruan keadaan (**Gambar 4.2**).

Kelima solusi keliru yang menggoda adalah halaman paling berguna di bab ini: masing-masing memperlihatkan uji yang memisahkan yang benar dari yang hampir benar. Buku: Subbab 4.11.

Pertemuan Berikutnya

Bab 5 — Fungsi: Bagi, Beri Nama, dan Taklukkan. Mendefinisikan dan memanggil fungsi, prototipe, *pass-by-value*, pustaka standar, cakupan, keacakan yang dapat diulang, dan rekursi.

Kasir versi 4 dipecah menjadi berkas header, berkas aturan bisnis, `main` yang pendek, dan program uji.

Sebelum itu: kerjakan ulang Soal 4.5 dengan serangan goblin 2 bernilai 6, ramalkan hasilnya dengan tangan, lalu jalankan.

Jawaban buku: goblin menang sesudah 24 giliran dengan total sisa 180 HP. Buku: Subbab 4.12 dan 4.13.

Kuliah 5 — Fungsi: Bagi, Beri Nama, dan Taklukkan

Mata kuliah: Pemrograman Dasar (EF234101)

Buku: Irfan Subakti, *Pemrograman Dasar dengan C, C++, dan Python*, Bab 5

Dosen: Irfan Subakti — yifana@gmail.com

Hari ini kita memotong program menjadi bagian-bagian bernama: mendefinisikan dan memanggil fungsi, prototipe, pass-by-value, pustaka standar, cakupan dan masa hidup, keacakan yang dapat diulang, rekursi, dan kasir versi 4 yang dipecah menjadi modul lengkap dengan ujinya. Buku: Bab 5, Subbab 5.1–5.14.

Cakupan Pertemuan Ini

1. **Gagasannya** — fungsi adalah pekerjaan bernama; apa yang terjadi saat pemanggilan; mengapa bagian kecil menang.
2. **Menulis fungsi** — satu aturan ditulis sekali, prototipe, `void`, *pass-by-value*, cara pandang Python.
3. **Meminjam dan menyembunyikan** — pustaka standar dan `-lm`; cakupan dan masa hidup; pembangkit acak dengan keadaan `static` tersembunyi.
4. **Rekursi** — kasus dasar dan masalah yang lebih kecil, tumpukan panggilan, rupiah bertitik, harga Fibonacci.
5. **Kasir v4** — berkas header, berkas aturan, `main` yang pendek, dan program uji.

Setiap program kembali dalam C dan Python, dengan keluaran identik. Buku: Bab 5.

Pertemuan Sebelumnya

Bab 4 menyelesaikan lima soal, masing-masing di dalam satu `main` yang panjang. Kasir pun tumbuh dengan cara yang sama.

Pak Hendra mengubah daftar harga: cuci lipat Rp 7.000 per kg, cuci setrika baru Rp 10.000, setrika saja Rp 6.000, kilat +50%, minimum 2 kg.

Dimas mencari `8000` dan mengubahnya di tagihan, tetapi angka yang sama juga duduk di laporan harian. Empat hari laporan keliru menyusul.

Buku resep menulis sambal sekali, di halaman 12, dan setiap masakan menunjuk ke sana. Program memerlukan kebiasaan yang sama.

Anda sudah memakai fungsi sejak hari pertama: `printf` dan `scanf` ditulis orang lain, dan `main` milik Anda. Hari ini Anda menulis yang lainnya. Buku: Subbab 5.1.

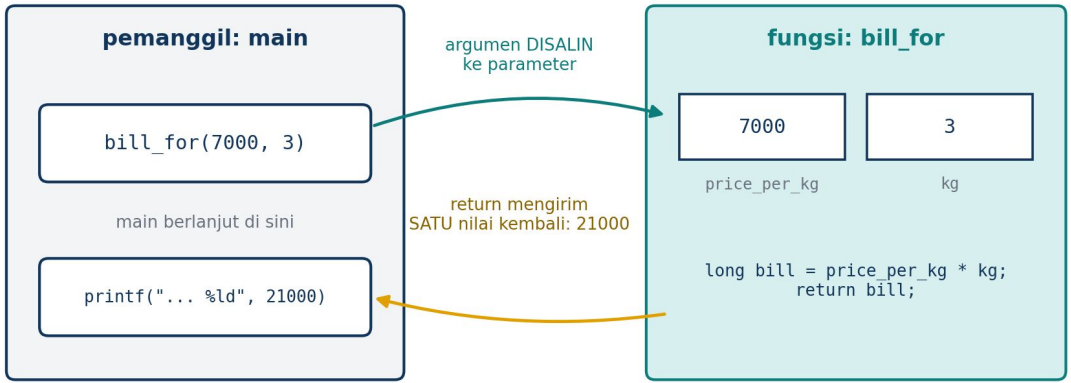
Fungsi adalah Pekerjaan Bernama

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*. Menulis fungsi disebut *mendefinisikan*; memakainya dengan namanya diikuti tanda kurung disebut *memanggil*. Nilai-nilai di dalam tanda kurung pemanggilan adalah *argumen*.

```
long   bill_for   (int price_per_kg, int kg)   { ... }
^^^^   ^^^^^^^^   ^^^^^^^^^^^^^^^^^^^^^^^^^   ^^^^^^^
type   name        parameters                   body
```

Sebuah mesin kecil berlabel: labelnya menyebut nama dan apa yang masuk. `void` berarti tidak ada yang keluar; `(void)` berarti tidak ada yang masuk, seperti pada `main`. Buku: Subbab 5.2.1.

Apa yang Terjadi saat Pemanggilan



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.

*Gambar 5.1 dalam empat langkah: evaluasi argumen; buat parameter dan **salin** setiap argumen ke dalamnya; jalankan badan sampai `return`; kirim 21000 kembali, hapus parameter dan variabel lokal, dan lanjutkan di pemanggil. Buku: Subbab 5.2.2.*

Mengapa Bagian Kecil Menang

Satu aturan, satu tempat — ketika harga berubah, satu baris yang berubah.

Nama menjelaskan — `charged_kg(weight)` menyatakan apa yang dikerjakannya; `main` yang dibangun dari pemanggilan bernama baik terbaca seperti cerita program.

Bagian dapat diuji sendirian — panggil dengan angka hitungan tangan. **Bagian dapat dipakai ulang** — pencetak rupiah melayani struk, laporan, dan daftar harga.

*Memotong masalah besar menjadi masalah-masalah kecil disebut **bagi dan taklukkan** (divide and conquer), penghalusan bertahap Wirth yang diterapkan pada kode. Buku: Subbab 5.2.3.*

Satu Aturan, Ditulis Sekali

Kode 5.1 Cuplikan `first_function.c`: satu aturan tagihan, didefinisikan sekali di atas `main` dan dipanggil empat kali.

```c
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;
}
```

Kode 5.1 mencetak Rp 21000, Rp 50000, dan Rp 40000. Cast `(long)` menjadikan hasil kali `long` sebelum sempat meluap. Di Python aturannya `def bill_for(price_per_kg, kg): return price_per_kg * kg`. Buku: Subbab 5.3.1.

Prototipe: Janji sebelum Pemanggilan

Kode 5.2 Cuplikan `prototypes.c`: empat prototipe di bagian atas berkas, sebelum `main`.

```c
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);
```

Prototipe adalah kepala fungsi diikuti `;`: nama, tipe parameter, tipe kembalian. Dengannya `main` dapat diletakkan pertama, sebagai daftar isi.

`print_rule` bertipe `void`: ia ada untuk apa yang dikerjakannya. `max3` dibangun dari `max2`: fungsi boleh memanggil fungsi.

> **PERINGATAN:** Sejak C99 pemanggilan tanpa prototipe tidak diizinkan, dan `gcc` melaporkan *implicit declaration of function*. Selalu berikan prototipe, atau definisikan fungsinya sebelum pemakaian pertama.

Kode 5.2 adalah bagian atas berkas; definisinya menyusul sesudah `main`. Buku: Subbab 5.3.2.

Contoh Terurai 1: *Pass-by-Value*

`main` memiliki `bill = 30000` dan memanggil `with_fee = add_express_fee(bill);`.

Fungsinya menerima **salinan**, menambahkan separuhnya, dan mencetak parameternya: 45000.

Kembali di `main`, `bill` tetap 30000. Jumlah baru hanya tiba lewat nilai kembalian, yang disimpan di `with_fee`.

```
bill in function : 45000
bill in main     : 30000
with express fee : 45000
```

Kerjakan di papan dengan dua kotak bernama `bill`, satu per fungsi. `add_express_fee(bill);` saja tidak mengubah apa pun; hasilnya terbuang. Buku: Subbab 5.3.2.

Cara Pandang Python

Python tidak punya prototipe: sebuah nama harus ada **ketika dipanggil**, bukan ketika dibaca.

Jadi `main` boleh didefinisikan pertama dan dipanggil di bawah berkas, sesudah semua definisi.

Parameter berperilaku sama: mengikat ulang parameter di dalam fungsi tidak mengubah variabel pemanggil.

`prototypes.py` *di buku mencetak delapan baris yang sama dengan program C. Buku: Subbab 5.3.3.*

Meminjam Fungsi: Pustaka Standar

`<stdio.h>` — `printf`, `scanf`, `putchar`. `<math.h>` — `sqrt`, `pow`, `ceil`, `floor`, `round`, `fabs` (tautkan dengan `-lm` di Linux).

`<stdlib.h>`, `<string.h>`, `<time.h>`, dan lainnya: fungsi-fungsi teruji, dikelompokkan menurut pokok bahasan, dideklarasikan di header berisi prototipe.

Laundry menagih **setiap kilogram yang dimulai**, paling sedikit 2:

$$k = \max(2, \lceil w \rceil) \tag{5.1}$$

Menurut Persamaan (5.1): 3,4 kg ditagih 4, 0,8 kg ditagih 2, tepat 2,0 kg ditagih 2. `ceil` *adalah pembulatan ke atas* $\lceil w \rceil$. `round(2.5)` *di Python bernilai 2 dan di C 3.0: sebutkan aturan pembulatan yang Anda maksud. Buku: Subbab 5.4.*

`floor`, `round`, dan `ceil`

```
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
  7.25     7.0     7.0     8.0      8 kg
```

Baris 2.10: `floor` dan `round` memberi 2, tetapi kilogram ketiga sudah dimulai, jadi `ceil` memberi 3 dan tagihannya 3 kg.

Baris 0.80: `ceil` memberi 1, dan `charged_kg` menaikkannya ke minimum 2.

`round` bawaan Python membulatkan setengah ke tetangga **genap** (*banker's rounding*): `round(2.5)` bernilai 2, `round(3.5)` bernilai 4. C membulatkan setengah menjauhi nol.

`rounding.c` di buku mencetak tabel ini (satu baris dihilangkan di sini); `rounding.py` mendefinisikan `c_round` agar mencetak persis sama. `-lm` diletakkan di akhir perintah, karena linker membaca pustaka secara berurutan. Buku: Subbab 5.4.

Cakupan dan Masa Hidup

Tabel 5.1 Cakupan dan masa hidup empat jenis variabel dalam C.

Jenis	Dideklarasikan	Terlihat (cakupan)	Ada (masa hidup)
lokal	di dalam fungsi	deklarasi sampai akhir blok	masuk sampai keluar blok
parameter	di kepala fungsi	seluruh badan	selama satu pemanggilan
global	di luar semua fungsi	deklarasi sampai akhir berkas	seluruh jalannya program
lokal `static`	di dalam, dengan `static`	seperti lokal	seluruh jalannya program

Variabel global terlihat di mana-mana, jadi ketika nilainya salah setiap fungsi menjadi tersangka. Lokal `static` mempertahankan nilainya di antara pemanggilan dan tetap pribadi.

Tabel 5.1: penghitung nota `next_receipt_number` adalah lokal `static`, terbaik dari kedua sisi. Sebuah blok dapat mendeklarasikan `kg` kedua yang menyembunyikan yang di luar. Python tidak punya cakupan blok maupun lokal `static`; ia memakai `global` dan variabel modul. Buku: Subbab 5.5.

Keacakan yang Dapat Diulang

$$x_{k+1} = (1103515245\, x_k + 12345) \bmod 2^{32} \tag{5.2}$$

Promosi Sabtu: tagihan paling sedikit Rp 50.000 melempar dua dadu; dadu kembar memenangkan voucer setrika gratis.

`rand()` berbeda antarsistem, jadi kita menulis Persamaan (5.2) sendiri: keadaan `static`, `uint32_t` yang berputar di 2^{32}, dan bit 16 sampai 30 dikembalikan sebagai 0 sampai 32767.

`1 + my_rand() % 6` memberi mata dadu 1 sampai 6. Python meniru perputarannya dengan `& 0xFFFFFFFF`.

Pseudo-acak berarti seed yang sama memberi deret yang sama: sebuah keunggulan, karena program seperti itu dapat diuji. Mata 1 dan 2 sedikit lebih mungkin (5462 lawan 5461 dari 32768), tidak masalah untuk voucer, tetapi tidak untuk undian. Buku: Subbab 5.6.

Rekursi: Kasus Dasar dan Masalah yang Lebih Kecil

DEFINISI. Fungsi *rekursif* menyelesaikan masalah dengan memanggil dirinya sendiri pada contoh yang lebih kecil dari masalah yang sama. Fungsi itu memiliki *kasus dasar*, yang cukup kecil untuk langsung dijawab, dan *langkah rekursif*, yang mengubah setiap contoh lain menjadi yang lebih kecil lalu menggabungkan jawabannya.

$$n! = \begin{cases} 1 & \text{if } n \le 1, \\ n \cdot (n-1)! & \text{if } n > 1. \end{cases} \tag{5.3}$$

Kode 5.3 Cuplikan `recursion.c`: faktorial, diterjemahkan baris demi baris dari definisinya.

```c
long factorial(int n)
{
    if (n <= 1)                         /* kasus dasar: berhenti di sini */
        return 1;
    return n * factorial(n - 1);        /* langkah rekursif: masalah lebih kecil */
}
```

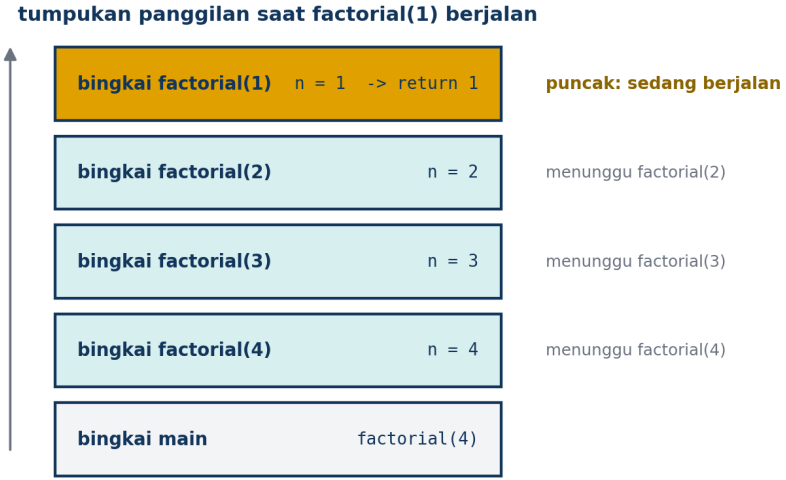
Kode 5.3 adalah Persamaan (5.3) dalam C. Buku: Subbab 5.7.1.

Contoh Terurai 2: Menelusuri `factorial(4)`

`factorial(4)` memerlukan `4 * factorial(3)` dan menunggu; `factorial(3)` menunggu `factorial(2)`, yang menunggu `factorial(1)`.

`factorial(1)` adalah kasus dasar dan langsung mengembalikan 1.

Pemanggilan yang menunggu selesai dalam urutan terbalik: $2 \times 1 = 2$, $3 \times 2 = 6$, $4 \times 6 = 24$.



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.

*Gambar 5.2: setiap pemanggilan memiliki **bingkai** di **tumpukan panggilan**. Tumpukan itu beberapa megabita; rekursi tanpa kasus dasar yang tercapai berakhir dengan stack overflow; Python berhenti di kedalaman 1000 dengan `RecursionError`. Buku: Subbab 5.7.1.*

Rekursi yang Berguna: Rupiah Bertitik

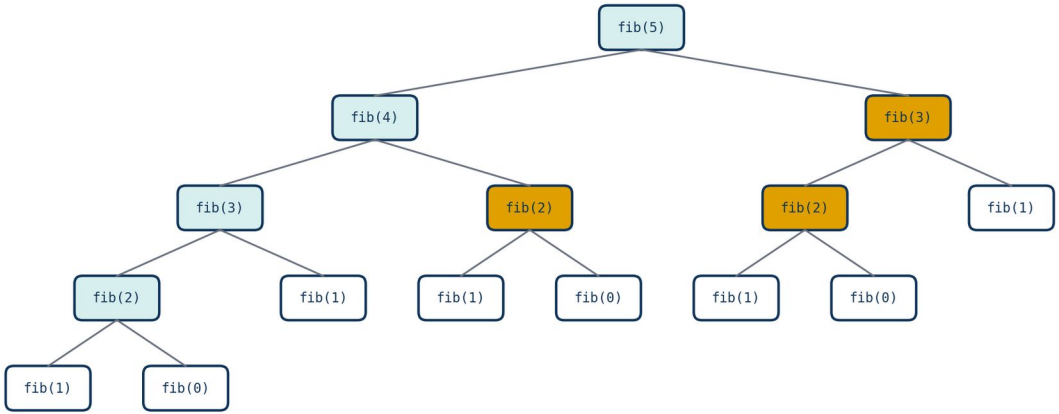
Kode 5.4 Cuplikan `recursion.c`: jumlah uang dicetak dengan titik di antara kelompok tiga digit.

```c
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 */
}
```

Kode 5.4: untuk mencetak 1234567, cetak dulu 1234 (masalah yang sama, lebih kecil), lalu `.567`. `%03ld` menjaga tiga digit, sehingga 5007 menjadi 5.007, bukan 5.7. Ia mencetak `Rp 7`, `Rp 28.000`, `Rp 187.500`, `Rp 1.234.567`. Buku: Subbab 5.7.2.

Harga Rekursi: Fibonacci

$$F(0)=0, F(1)=1, F(n)=F(n-1)+F(n-2) \text{ for } n \geq 2 \tag{5.4}$$



fib(5): 15 panggilan; kuning = kerja yang sudah dilakukan di tempat lain

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.

Persamaan (5.4) menjadi fungsi tiga baris, benar tetapi mahal: dalam Gambar 5.3 seluruh subpohon `fib(3)` dihitung dua kali. Buku: Subbab 5.7.3.

Menghitung Pemanggilan

$$C(n)=C(n-1)+C(n-2)+1, C(0)=C(1)=1 \tag{5.5}$$

Dari Persamaan (5.5): $C(2)=3$, $C(3)=5$, $C(4)=9$, $C(5)=15$, kelima belas kotak itu. Secara umum $C(n)=2F(n+1)-1$.

```
fib(10) =     55 after      177 calls
fib(20) =   6765 after    21891 calls
fib(25) =  75025 after   242785 calls
fib(30) = 832040 after  2692537 calls
```

Setiap kenaikan 5 mengalikan kerja dengan sekitar $1.618^5 \approx 11$*. Perulangan yang menyimpan dua bilangan terakhir memerlukan 30 langkah untuk* `fib(30)`*; Bab 8 mengukur keduanya. Pilih rekursi bila masalahnya bersarang, perulangan bila rekursi mengulang kerja. Buku: Subbab 5.7.3 dan 5.7.4.*

Kasir v4: Header

Kode 5.5 Cuplikan `laundry.h`: janji-janji modul aturan bisnis, dengan konstanta dan *include guard*.

```
#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
```

Kode 5.5: setiap janji memiliki komentar satu baris, termasuk jawaban untuk masukan buruk. `#define` memberi setiap konstanta satu rumah; include guard membuat `#include` kedua tidak berbahaya. Buku: Subbab 5.8.1.

Kasir v4: Aturannya

Kode 5.6 Cuplikan `laundry.c`: dua aturan bisnis Profitina Laundry, satu fungsi per aturan.

```c
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;
}
```

`charged_kg` adalah Persamaan (5.1); `bill_for` menggabungkan harga dan kilogram lalu menambahkan biaya kilat. Ia menghitung dan mengembalikan; ia tidak pernah mencetak.

Kode 5.6: `price_per_kg` adalah `switch` yang mengembalikan 0 untuk layanan tak dikenal; fungsi bantu `print_thousands` ditandai `static`, pribadi bagi berkasnya. Buku: Subbab 5.8.1.

Mengompilasi Beberapa Berkas

```
gcc -std=c17 -Wall -Wextra -o cashier cashier.c laundry.c -lm
```

Sebutkan setiap berkas `.c`; jangan headernya, yang dibawa masuk oleh `#include`.

`gcc` mengompilasi setiap berkas menjadi *object file*; ***linker*** menggabungkannya, mencocokkan setiap pemanggilan di `cashier.c` dengan definisinya di `laundry.c`.

Dijanjikan di header tetapi tidak pernah didefinisikan: kompilator puas, *linker* berkata `undefined reference to ...`, "sebuah janji tidak ditepati".

Python: `python3 cashier.py` menemukan `laundry.py` di folder yang sama lalu mengimpornya.

`laundry.h` yang sama akan melayani laporan harian Bab 6 dan ringkasan tutup toko Bab 7 tanpa diubah. Buku: Subbab 5.8.2.

Satu Pagi di Meja Kasir

```
#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
```

Periksa dengan tangan: 1,2 kg menjadi minimum 2 kg, $2 \times 10000 = 20000$*, ditambah separuh menjadi 30.000; 12,75 kg ditampilkan 12.8, ditagih 13 kg,* $91000 + 45500 = 136500$*. Buku: Subbab 5.9.*

Mengujinya

Kode 5.7 Cuplikan `test_laundry.c`: fungsi bantu yang membandingkan setiap aturan dengan jawaban hitungan tangan.

```
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);
    }
}
```

Uji yang berharga adalah tepinya: 0,8 kg → 2, tepat 2,0 kg → 2, 2,1 kg → 3; penghitung nota mulai dari 1.

Ketika kilat turun menjadi 40%, `check(bill_for(2, 1.2, 1), 30000, ...)` gagal dengan `got 28000`: kabar baik, ujinya menyadari.

Kode 5.7 mengambil nomor baris dari `__LINE__` dan program mengembalikan 1 bila ada kegagalan, sehingga skrip menyadarinya. Layanan keempat mengubah tepat dua fungsi, masing-masing satu baris `case`. Buku: Subbab 5.10 dan 5.11.

Memakainya Sungguhan

Penjahit — `price_of(job)`. **Kios fotokopi** — satu fungsi lagi, `discount_for(pages)`.

Bengkel motor — ongkos servis ditambah suku cadang. **Dapur katering** — “paling sedikit 10 kotak, dengan kelipatan 5”, bentuk Persamaan (5.1).

Tulis header janji lebih dulu, lalu uji dengan jawaban yang dikerjakan pemilik di kertas, baru kemudian `main`.

Ubah aturannya, bukan ceritanya: pemilik jauh lebih mudah membaca dan memeriksa uji daripada kode. Buku: Subbab 5.12.

Kesalahan yang Sering Terjadi

Mengira fungsi mengubah argumennya; `return` **terlupa** (*control reaches end of non-void function*).

Tanpa prototipe (*implicit declaration*); **tanpa** `-lm` (*undefined reference to* `ceil`).

Luapan dalam `price * kg` — *cast* satu faktor ke `long`; **kasus dasar yang tak tercapai**; **rekursi pohon** untuk n besar.

`round` **Python bukan** `round` **C**; **lokal yang menyembunyikan global** — `gcc -Wshadow` memperingatkan.

Setiap kesalahan di buku disertai pesan kompilator dan perbaikannya. Buku: Subbab 5.13.

Ringkasan: Lima Hal untuk Diingat

1. Fungsi adalah pekerjaan bernama; pemanggil hanya memerlukan kepalanya (Gambar 5.1).

2. Argumen disalin ke parameter: jawabannya kembali lewat `return`.

3. Prototipe adalah janji; header adalah berkas janji; *undefined reference* dari *linker* berarti janji tidak ditepati.

4. Lokal hidup selama satu pemanggilan, global dan lokal `static` selama program berjalan (Tabel 5.1); utamakan lokal.

5. Rekursi memerlukan kasus dasar; setiap pemanggilan punya bingkai sendiri (Gambar 5.2); `fib` membuat $2\,F(n+1)-1$ pemanggilan.

Kasir v4: aturan di modul, cerita di `main`, dan uji yang memeriksa setiap aturan terhadap jawaban hitungan tangan. Buku: Subbab 5.14.

Pertemuan Berikutnya

Bab 6 — Array: Banyak Nilai, Satu Nama. Mendeklarasikan dan menelusuri array, penghitung, meneruskan array ke fungsi, bubble sort, pencarian linear dan biner, tabel dua dimensi.

Kasir versi 5 mengingat seluruh hari dan mencetak total per layanan, rata-rata, median, dan tiga tagihan terbesar.

Sebelum itu: Soal Latihan 5.1, 5.3, 5.4, dan 5.7.

Soal Latihan 5.4 adalah algoritma Euclid, $\gcd(a, b) = \gcd(b, a \bmod b)$*, rekursi yang lebih pendek daripada perulangan mana pun. Buku: Subbab 5.15 dan 5.16.*

Kuliah 6 — Array: Banyak Nilai, Satu Nama

Mata kuliah: Pemrograman Dasar (EF234101)

Buku: Irfan Subakti, *Pemrograman Dasar dengan C, C++, dan Python*, Bab 6

Dosen: Irfan Subakti — yifana@gmail.com

Hari ini program belajar mengingat banyak nilai sejenis: array dan list Python, penghitung, array sebagai parameter, bubble sort, pencarian linear dan biner, tabel dua dimensi, dan kasir versi 5 dengan laporan hariannya. Buku: Bab 6, Subbab 6.1–6.15.

Cakupan Pertemuan Ini

1. **Gagasannya** — sederet kotak bernomor; mengapa indeks mulai dari 0; tidak ada yang menjaga pagar; list Python.
2. **Memakai array** — deklarasi, pengisian, penelusuran; array sebagai penghitung; meneruskan array ke fungsi dengan ukuran dan `const`.
3. **Pengurutan** — bubble sort, invariannya, dan biayanya.
4. **Pencarian** — pencarian linear dan biner, dan kapan memakai yang mana.
5. **Tabel dan kasir** — array dua dimensi dalam urutan baris; versi 5 dengan total, rata-rata, median, dan tiga teratas.

Setiap program kembali dalam C dan Python dengan keluaran identik. Buku: Bab 6.

Pertemuan Sebelumnya

Kasir v4 mencetak nota yang benar lalu melupakannya begitu pelanggan berikutnya maju. Ia hanya menyimpan total berjalan.

Saat tutup Mbak Rina bertanya: rata-rata tagihan, pesanan terbesar, berapa per layanan, hari tersibuk dalam seminggu.

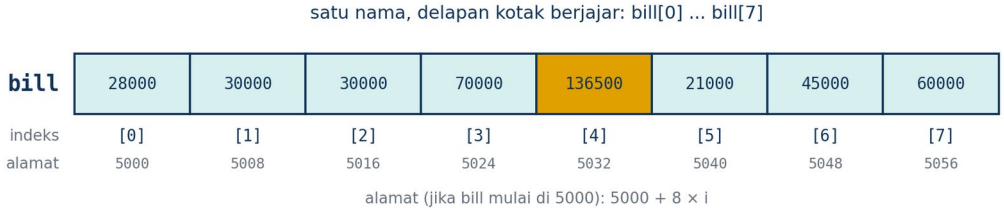
Total memberi jumlah dan rata-rata, tetapi bukan median, bukan tiga teratas, dan bukan daftar terurut: semua itu memerlukan **setiap** tagihan di akhir hari.

`long bill1, bill2, ..., bill100;` gagal: perulangan tidak dapat berkata “variabel berikutnya”. Kita memerlukan satu nama dan sebuah nomor.

Kebutuhan yang sama muncul pada penjualan harian warung, curah hujan di kebun buah dan durian, nilai satu kelas. Buku: Subbab 6.1.

Array adalah Sederet Kotak Bernomor

DEFINISI. *Array* adalah sejumlah tetap variabel bertipe sama yang disimpan berdampingan di memori dengan satu nama. Setiap variabel adalah *elemen*, dipilih dengan *indeks* 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 $n - 1$.



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.

Gambar 6.1: `bill[4]` adalah kotak kelima dan berisi 136500. Kekuatannya: indeks boleh berupa variabel, sehingga satu perulangan atas `i` mengunjungi setiap kotak. Buku: Subbab 6.2.1.

Mengapa Indeks Mulai dari 0

$$\text{address}(\text{bill}[i]) = a + i \cdot s \tag{6.1}$$

Indeks adalah **jarak** dari awal, diukur dalam elemen; Persamaan (6.1) tidak memerlukan koreksi untuk elemen pertama.

Dengan $a = 5000$ dan $s = 8$: `bill[4]` berada di $5000 + 4 \cdot 8 = 5032$. Satu perkalian dan satu penjumlahan, sepanjang apa pun arraynya.

Rumus yang sama menetapkan ukurannya: semua elemen dipesan sekaligus, dan bita sesudah `bill[7]` mungkin milik variabel lain.

Program yang harus tumbuh memesan ruang lebih banyak daripada perkiraannya, atau menyalin ke blok yang lebih besar, yang dilakukan list Python diam-diam. Buku: Subbab 6.2.2.

Tidak Ada yang Menjaga Pagar

`bill[8]` pada array berisi delapan: kompilator tidak menghentikannya, program tidak memeriksanya. Persamaan (6.1) menghitung alamat sesudah ujung, dan program menyentuh apa pun yang tinggal di sana.

Standar C menyebutnya **perilaku tak terdefinisi** (*undefined behavior*): apa pun dapat terjadi, termasuk tampak berjalan sampai suatu hari tidak.

Pertahanannya adalah kebiasaan: `for (i = 0; i < n; i++)` dengan `<`, tidak pernah `<=`; periksa setiap indeks dari masukan; namai ukurannya sebagai konstanta.

Kembangkan dengan `-fsanitize=address,undefined`: akses pertama di luar batas menghentikan program dengan laporan yang jelas.

Python memeriksa: `bill[8]` melempar `IndexError: list index out of range`. List mengetahui panjangnya (`len`), tumbuh dengan `append`, dan menyimpan referensi ke objek bilangan, yang memakan memori dan kecepatan. Buku: Subbab 6.2.3 dan 6.2.4.

Array C dan List Python

Tabel 6.1 Array C dan list Python berdampingan.

Pertanyaan	Array C	List Python
indeks pertama	0	0
tahu panjangnya?	tidak: teruskan `n`	ya: `len(bill)`
dapat tumbuh?	tidak: tetap sejak dibuat	ya: `append`, `pop`
indeks di luar?	perilaku tak terdefinisi	`IndexError`
isi sebuah kotak	nilainya sendiri, rapat	referensi ke sebuah objek

Tabel 6.1: list membayar kenyamanannya dengan memori dan kecepatan, beberapa kali memori array C untuk sejuta bilangan; untuk beberapa ratus pesanan sehari perbedaannya tidak berarti. Buku: Subbab 6.2.4.

Mendeklarasikan, Mengisi, dan Menelusuri

Kode 6.1 Cuplikan `day_bills.c`: delapan tagihan dalam satu array, dijumlahkan dan dicari yang terbesar.

```
    long bill[N] = {28000, 30000, 30000, 70000, 136500, 21000, 45000, 60000};

    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;
```

Nilai yang tidak ada di inisialisasi menjadi 0: `int count[4] = {0};`. Bila ukurannya dihilangkan, kompilator menghitung nilainya.

`best` menyimpan **indeks** tagihan terbesar, sehingga nilai dan letaknya sama-sama diketahui. `sizeof bill / sizeof bill[0]` memberi 8, tetapi hanya di tempat deklarasinya terlihat.

Kode 6.1 mencetak total 420500, rata-rata 52562.50, dan yang terbesar, 136500 di indeks 4. Array yang dideklarasikan di dalam fungsi tanpa inisialisasi berisi sampah. Buku: Subbab 6.3.

Contoh Terurai 1: Menelusuri Penjumlahan

Sebelum perulangan `total` bernilai 0. $i=0$: 28000. $i=1$: 58000. Lalu 88000, 158000, 294500, 315500, 360500, dan 420500 pada $i=7$.

`i++` membuat `i` bernilai 8; `i < N` salah; perulangan berakhir tanpa menyentuh `bill[8]`.

$$\dot{x} = \frac{1}{n} \sum_{i=0}^{n-1} x_i \tag{6.2}$$

Persamaan (6.2) adalah persis perulangan itu diikuti `(double) total / N`: 420500/8 = 52562.5.

Telusuri satu perulangan dengan tangan sekali, seperti komputer menjalankannya. Versi Python berjalan dengan `range(N)`; Python yang idiomatis memakai `sum(bill)` dan `max(bill)`, dan hari ini kita menulis apa yang mereka kerjakan di dalamnya. Buku: Subbab 6.3.

Array sebagai Penghitung

Kode 6.2 Cuplikan `service_count.c`: array penghitung, diindeks dengan kode layanan.

```
    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++;
    }
```

Nilai datanya menjadi indeks: tidak ada yang dibandingkan atau dicari.

Dua puluh pesanan memberi 11, 5, dan 3 untuk ketiga layanan dan 1 kode tak dikenal, salah ketik 9. Tanpa penjaga, `count[9]++` menulis melewati ujung.

Kode 6.2 mengorbankan `count[0]` agar kodenya langsung menjadi indeks: pertukaran yang baik. Program menggambar setiap hitungan sebagai batang bintang; Python menulis `"*" * count[s]`. Buku: Subbab 6.4.

Meneruskan Array ke Fungsi

Kode 6.3 Cuplikan `array_functions.c`: fungsi yang berjanji tidak mengubah array, dan fungsi yang mengubah array milik pemanggil.

```c
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;
}

void add_to_all(long a[], int n, long amount)   /* mengubah array milik PEMANGGIL */
{
    for (int i = 0; i < n; i++)
        a[i] += amount;
}
```

Parameternya menerima **alamat** array pemanggil, bukan salinan: fungsi dapat mengubahnya.

Fungsi tidak tahu panjangnya, jadi panjang datang sebagai parameter `n`; `const` menandai array yang hanya dibaca fungsi.

Dalam Kode 6.3, `add_to_all(evening, 3, 5000)` menambahkan ongkos antar ke setiap kantong sore di array milik `main` sendiri. Bab 7 menjelaskan sebabnya: alamat adalah kuncinya. Buku: Subbab 6.5.

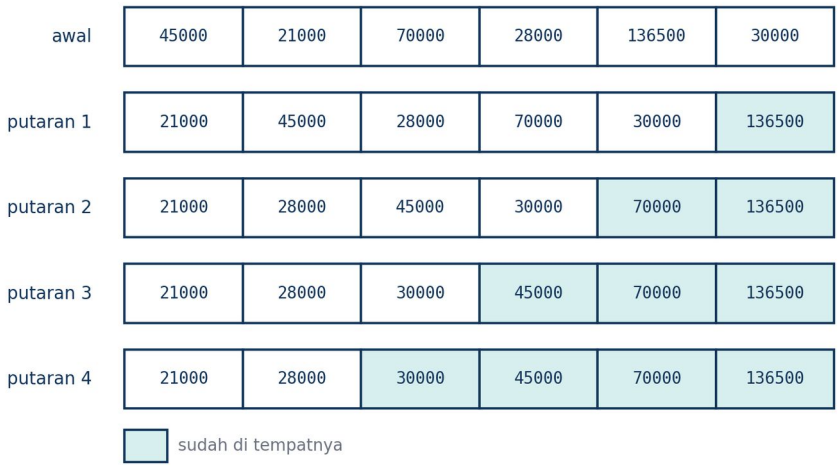
Bubble Sort

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            // no swap in a whole pass: sorted
```

Pseudokode 6.1: bandingkan tetangga, tukar bila urutannya salah; nilai terbesar naik ke ujung seperti gelembung. Daftar terurut memberi median, tiga teratas, dan pencarian cepat. Buku: Subbab 6.6.

Contoh Terurai 2: Enam Tagihan, Putaran demi Putaran



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.

Kerjakan putaran 1 Gambar 6.2 di papan: 45000 > 21000, tukar; 45000 < 70000, tidak; 70000 > 28000, tukar; 70000 < 136500, tidak; 136500 > 30000, tukar. Kini 136500 berada di ujung kanan. Buku: Subbab 6.6.

Invarian dan Biayanya

WAWASAN KUNCI: Setelah putaran p, p posisi terakhir memuat p nilai terbesar, dalam urutan akhirnya. *Invarian perulangan* ini menjelaskan mengapa perulangan dalam berhenti di `n - pass`, dan mengapa $n-1$ putaran selalu cukup.

$$(n-1)+(n-2)+\cdots+1=\frac{n(n-1)}{2} \tag{6.3}$$

Menurut Persamaan (6.3): 15 perbandingan untuk 6 tagihan; hampir 200 juta untuk 20.000; empat kali lipat setiap kali daftarnya berlipat dua.

Bubble sort bersifat **stabil**: ia menukar hanya bila benar-benar lebih besar, sehingga dua tagihan Rp 30.000 mempertahankan urutan notanya.

Metode yang tepat untuk dipelajari pertama, dan hampir tidak pernah tepat untuk data besar; pengurutan pustaka jauh lebih cepat. Buku: Subbab 6.6.

Pencarian Linear dan Biner

Kode 6.4 Cuplikan `sort_search.c`: pencarian biner pada array terurut (versi buku juga mencetak setiap tilikan).

```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;
        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;
}
```

Pencarian linear melihat dari depan sampai ketemu atau habis, dan mengembalikan -1 untuk "tidak ada": sampai n tilikan, pada array apa pun.

Pencarian biner memerlukan data terurut dan membagi dua bagian yang masih dipertanyakan pada setiap tilikan, seperti membuka kamus di tengah.

Dalam Kode 6.4, `low + (high - low) / 2` tidak pernah meluap; `(low + high) / 2` bersembunyi di pustaka Java selama sembilan tahun sebelum ditemukan pada 2006. Buku: Subbab 6.7.

Menelusuri Pencarian Biner

```
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
```

$$k_{\max} = \lfloor \log_2 n \rfloor + 1 \tag{6.4}$$

Untuk $n = 6$, $\log_2 6 \approx 2.58$ dan Persamaan (6.4) memberi 3 tilikan, persis yang diperlukan 50000. Untuk sejuta nota: 20 tilikan lawan sampai sejuta. Urutkan sekali dan cari berkali-kali: biner; cari sekali: linear. Buku: Subbab 6.7.

Array Dua Dimensi

Kilogram per hari dan layanan: `int kg[7][3];`, baris `d`, kolom `s`, elemen `kg[d][s]`. Inisialisasinya punya satu pasang kurung kurawal per baris.

Dua perulangan bersarang mengunjungi setiap sel: luar atas baris, dalam atas kolom. Total baris diatur ulang untuk setiap baris; total kolom hidup melintasi semua baris.

$$\text{address}(\text{kg}[r][c]) = a + (r \cdot C + c) \cdot s \tag{6.5}$$

Persamaan (6.5): r baris utuh berisi C elemen datang lebih dulu, lalu c elemen. Itulah sebabnya parameter harus menyebut banyaknya kolom: `int kg[][3]`. Buku: Subbab 6.8.

Urutan Baris

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`).

Gambar 6.3: tabel seperti kita bayangkan dan seperti memori menyimpannya. Python memakai list berisi list; jangan membangunnya dengan `[[0] * 3] * 7`, yang membuat tujuh nama untuk satu baris. Buku: Subbab 6.8.

Kasir v5: Array Paralel

Kode 6.5 Cuplikan `day_report.c`: hari itu dibaca ke dalam array paralel, paling banyak `MAX_ORDERS` pesanan.

```
    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++;
    }
```

`service[i]`, `weight[i]`, dan `bill[i]` bersama-sama menggambarkan pesanan *i*: **array paralel**.

`n < MAX_ORDERS` diletakkan pertama dalam kondisi, sehingga hari yang memecahkan rekor tidak pernah menulis melewati ujung; `n` hanya menghitung pesanan yang sah.

Total per layanan adalah array yang diindeks kode layanan: `count[service[i]]++`, `kg[service[i]] += weight[i]`, pola penghitung tiga kali sekaligus.

Kode 6.5 memakai ulang `laundry.h` dari Bab 5 tanpa perubahan: `price_per_kg` menyatakan apakah layanan ada dan `bill_for` menghitung tagihan. 100 pesanan adalah tiga kali hari tersibuk dan hanya memakan 800 bita. Buku: Subbab 6.9.

Kasir v5: Median

Kode 6.6 Cuplikan `bills.c`: median dari sebuah salinan, sehingga urutan milik pemanggil tidak tersentuh.

```
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;
}
```

Kode 6.6 termasuk modul kedua, `bills.h` dan `bills.c`, di samping `laundry.h` dari Bab 5 yang tidak berubah. Buku: Subbab 6.9.

Laporan Hari Sabtu

```
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
```

Verifikasi dengan tangan: $3.4+12.75+4.2+7.9+5.5=33.75$ *kg;* $523500/11=47590.9$*, tercetak 47.590 karena pembagian bulat. Layanan tak dikenal 7 dilewati. Buku: Subbab 6.10.*

Rata-rata atau Median?

Rata-rata, Rp 47.590, lebih tinggi daripada tujuh dari sebelas tagihan.

Satu pesanan Rp 136.500 menarik rata-rata ke atas tetapi hanya menggeser median satu tempat.

“Berapa belanja pelanggan biasa?” → median, Rp 35.000. “Berapa uang masuk per pesanan?” → rata-rata.

Uji di bab ini mencakup array kosong dan janji bahwa `median` tidak mengubah masukannya. Buku: Subbab 6.10 dan 6.11.

Menguji dan Memperbaiki

Sembilan pemeriksaan pada array kecil yang dihitung dengan tangan: lima nilai dengan total 150 dan median 30, satu elemen, panjang genap yang mediannya rata-rata bulat dari 2 dan 3, yaitu 2.

Dua pemeriksaan tentang perilaku: sesudah `median(a, 5)` arraynya tidak berubah, dan hari kosong memberi total 0, bukan mogok.

Lebih baik: beri peringatan bila pesanan diabaikan karena kapasitas; urutkan array *indeks* agar data tidak berpindah; bulatkan rata-rata dengan sengaja memakai `(sum + n / 2) / n`.

Bab 11 menyimpan hari itu ke berkas; fungsi-fungsi `bills.c` tidak akan berubah, karena mereka tidak peduli dari mana arraynya diisi. Buku: Subbab 6.11 dan 6.12.

Memakainya Sungguhan

Daftar apa pun yang sejenis: penjualan harian warung, curah hujan harian di kebun, karung beras dalam satu kiriman.

Penghitung per kode, pengurutan untuk tiga teratas, pencarian biner pada daftar nomor nota yang terurut, tabel mingguan per hari dan produk.

Pesan ukuran maksimum yang bernama dan periksa setiap indeks yang berasal dari masukan.

Lima alat kecil yang sama menjawab hampir setiap pertanyaan pemilik saat tutup toko. Buku: Subbab 6.13.

Kesalahan yang Sering Terjadi

`i <= n` — satu melewati ujung; **mulai dari 1**; **penghitung tanpa inisialisasi**; **indeks dari masukan yang tidak diperiksa**.

`sizeof` **pada parameter array** — ia mengukur sebuah alamat; teruskan panjangnya.

Mengurutkan data pemanggil tanpa sengaja; **pencarian biner pada data tak terurut**; `(low + high) / 2`.

`[[0] * 3] * 7` di Python; `a = b` untuk array: tidak terkompilasi di C, nama kedua di Python; salinlah.

Setiap kesalahan di buku disertai perbaikannya. Buku: Subbab 6.14.

Ringkasan: Lima Hal untuk Diingat

1. Elemen i berjarak i dari awal (Persamaan (6.1), Gambar 6.1): indeks berjalan dari 0 sampai $n-1$, dan elemen mana pun hanya satu langkah jauhnya.

2. C tidak memeriksa indeks: ulangi dengan `<`, periksa masukan, pakai *sanitizer*. Python melempar `IndexError`.

3. Parameter array adalah array milik pemanggil: teruskan panjangnya, tandai array baca-saja dengan `const`.

4. Bubble sort memerlukan sampai $n(n-1)/2$ perbandingan; pencarian biner paling banyak $\lfloor \log_2 n \rfloor + 1$ tilikan, pada data terurut.

5. Tabel disimpan baris demi baris (Gambar 6.3); median menggambarkan pelanggan yang biasa.

Kasir v5 menyimpan hari itu dalam array paralel dan melaporkan total per layanan, rata-rata, median, dan tiga teratas. Buku: Subbab 6.15.

Pertemuan Berikutnya

Bab 7 — Pointer: Alamat sebagai Nilai. `&` dan `*`, `swap`, parameter keluaran, aritmetika pointer, `const` dan pointer, pointer fungsi dan `qsort`.

Mengapa `add_to_all` dapat mengubah array milik `main`, dan mengapa `scanf` memerlukan `&`: jawabannya adalah alamat.

Sebelum itu: Soal Latihan 6.1, 6.2, 6.3, dan 6.6.

Soal Latihan 6.2 adalah Persamaan (6.1) dan Persamaan (6.5) dengan tangan. Buku: Subbab 6.16 dan 6.17.

Kuliah 7 — Pointer: Alamat sebagai Nilai

Mata kuliah: Pemrograman Dasar (EF234101)

Buku: Irfan Subakti, *Pemrograman Dasar dengan C, C++, dan Python*, Bab 7

Dosen: Irfan Subakti — yifana@gmail.com

Hari ini alamat menjadi nilai: `&` dan `*`, `swap`, parameter keluaran, array dan aritmetika pointer, `const` bersama pointer, pointer ke fungsi dan `qsort`, keamanan pointer, dan kasir versi 6. Kita membahas pointer pelan-pelan, dengan gambar. Buku: Bab 7, Subbab 7.1–7.15.

Cakupan Pertemuan Ini

1. **Gagasannya** — setiap variabel punya alamat; mengikuti panah; `NULL`; pointer yang sudah pernah Anda pakai.
2. **Mengubah pemanggil** — `swap`, *pass-by-reference*, parameter keluaran dengan status.
3. **Array dan pointer** — peluruhan, aritmetika pointer, `a[i]` adalah `*(a + i)`, menelusuri dengan pointer.
4. **`const` dan pointer fungsi** — apa yang dilindungi `const`; perilaku sebagai parameter; `qsort`.
5. **Kasir v6 dan keamanan** — empat jawaban dari satu pemanggilan; kesalahan pointer klasik dan *sanitizer*.

Python tidak punya pointer; di setiap langkah kita menyebutkan apa yang dilakukannya sebagai gantinya. Buku: Bab 7.

Pertemuan Sebelumnya: Dua Janji

Fungsi mengembalikan **satu** nilai, padahal saat tutup Mbak Rina menginginkan total, rata-rata, terkecil, dan terbesar dari satu penelusuran.

Fungsi menerima **salinan**, padahal `scanf` mengisi variabel kita dan `add_to_all` di Bab 6 mengubah array milik `main`.

`swap(left, right)` untuk dua kantong di rak tidak dapat bekerja dengan nilai: Bab 5 sudah membuktikannya.

Laundry itu sudah punya jawabannya: **label** bernomor rak. Kecil, mudah disalin dan diteruskan, dan siapa pun yang memegangnya sampai ke kantong yang sama.

Nomor pada nota menuntun ke tepat satu kantong di satu rak. Nota hilang: kantongnya ada tetapi tidak cepat ditemukan. Dua nota bernomor sama: dua pelanggan mengaku satu kantong. Itulah persis kesalahan pointer hari ini. Buku: Subbab 7.1.

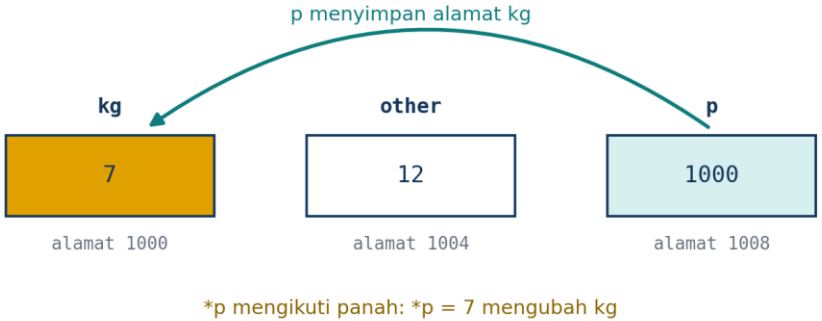
Setiap Variabel Punya Alamat

> **DEFINISI.** *Pointer* adalah variabel yang nilainya adalah alamat variabel lain. Jika `p` menyimpan alamat `kg`, `p` *menunjuk* `kg`. `int *p;` menjadikan `p` pointer ke `int`. Operator `&` memberikan alamat sebuah variabel; `*`, bila dikenakan pada pointer, memberikan variabel yang ditunjuknya, langkah yang disebut *dereferensi* atau *mengikuti* pointer.

Memori adalah deretan panjang loker bernomor; `int` memakai empat, `double` delapan. Nomor loker pertama adalah **alamatnya**.

Anda sudah menulis `&` sejak `scanf("%d", &kg)` pertama. Buku: Subbab 7.2.1.

Mengikuti Panah



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

$$*(\&x) \equiv x \tag{7.1}$$

***Gambar 7.1**: `p` di 1008 menyimpan 1000, alamat `kg`, digambar sebagai panah. Menurut Persamaan (7.1), `&` dan `*` saling membatalkan; ≡ berarti "adalah hal yang sama dengan", jadi `*(&x)` dapat diberi nilai. Alamatnya hanya bilangan kecil rekaan. Buku: Subbab 7.2.2.*

Dua Bintang, Dua Tugas

Dalam **deklarasi**, `int *p;` berarti “p adalah pointer ke int”. Dalam **ekspresi**, `*p = 7;` berarti “ikuti p”.

Bacalah `int *p` sebagai “`*p` adalah `int`”. Maka `double *price;` berarti `*price` adalah `double`.

Jebakan: `long *a, b;` hanya menjadikan `a` pointer. Tulis satu deklarasi per baris.

Pointer bertipe: tipe menyatakan berapa bita dibaca dan bagaimana. `NULL` tidak menunjuk ke mana pun; mengikutinya adalah perilaku tak terdefinisi dan biasanya membuat program mogok.

`int **pp;` berarti `**pp` adalah `int`: pointer ke pointer, yang diperlukan linked list di Bab 10. Buku: Subbab 7.2.2 dan 7.2.3.

Pointer Pertama Anda

Kode 7.1 Cuplikan `first_pointer.c`: mengambil alamat, menulis melaluinya, mengarahkan ulang pointer, dan menyalinnya.

```
    int kg = 5;
    int *p = &kg;                       /* p menyimpan alamat kg */
    *p = 7;                             /* tulis MELALUI pointer */

    int other = 12;
    p = &other;                         /* arahkan p ke variabel lain */
    *p += 1;

    int *q = p;                         /* menyalin pointer: dua panah */
    *q = 20;
```

Sesudah `*p = 7;` nilai `kg` menjadi 7, walaupun pernyataan itu tidak pernah menyebut `kg`: itulah intinya.

`p = &other;` tanpa bintang memindahkan panah; `*p += 1` membuat `other` bernilai 13. `int *q = p;` memberi dua panah ke satu kotak: `*q = 20` membuat `*p` juga 20.

*Gambarkan **Kode 7.1** di papan sebagai kotak dan panah, satu pernyataan demi satu. Program tidak mencetak alamat, karena alamat sungguhan berubah setiap kali dijalankan. Python tidak punya pointer, hanya referensi: list satu elemen berperan sebagai kotak yang dipakai bersama dua nama. Buku: Subbab 7.3.*

Pointer yang Sudah Pernah Anda Pakai

`scanf("%d", &kg)` menerima pointer ke `kg` dan menulis melaluinya.

Teks dalam `printf("Hello\n")` adalah array karakter; `printf` menerima pointer ke karakter pertamanya.

Parameter `const long a[]` di Bab 6 adalah pointer, itulah sebabnya fungsi-fungsi itu dapat mengubah array pemanggil dan tidak tahu panjangnya.

Lupa `&` berarti memberi `scanf` nilai `kg`, bilangan acak yang lalu dipakainya sebagai alamat. Buku: Subbab 7.2.4 dan 7.4.

Mengubah Variabel Pemanggil

Kode 7.2 Cuplikan `swap.c`: menukar salinan tidak mengubah apa pun di `main`; menukar melalui pointer mengubah variabel milik `main`.

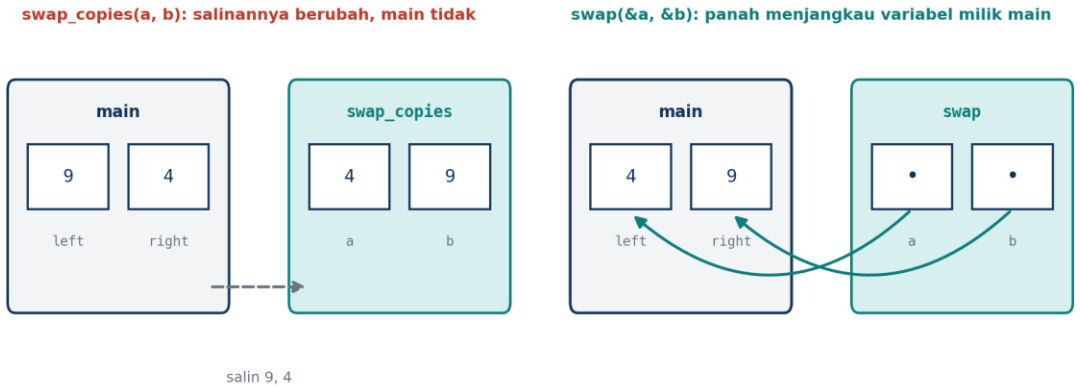
```c
void swap_copies(int a, int b)          /* hanya menukar salinannya sendiri */
{
    int t = a;
    a = b;
    b = t;
}

void swap(int *a, int *b)               /* menukar apa yang ditunjuk panah */
{
    int t = *a;
    *a = *b;
    *b = t;
}
```

```
after swap_copies: left 9, right 4
after swap:        left 4, right 9
```

Kode 7.2 *dipanggil sebagai* `swap(&shelf_left, &shelf_right)`*. Halaman berikut menggambar kedua pemanggilan. Buku: Subbab 7.4.*

Contoh Terurai 1: Dua Penukaran di Memori



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.

*Langkahi **Gambar 7.2** di papan: `int t = *a;` membaca kiri milik `main` (9); `*a = *b;` menyalin kanan milik `main` (4) ke kiri; `*b = t;` menaruh 9 di kanan. Tepatnya C tetap pass-by-value: salinan alamat, yang menuntun ke variabel yang sama, seperti fotokopi label rak. Buku: Subbab 7.4.*

Mengurutkan Tiga Kantong, dan Python

`if (bag[0] > bag[1]) swap(&bag[0], &bag[1]);` lalu `bag[1]`, `bag[2]`, lalu `bag[0]`, `bag[1]` lagi: sebuah *jaringan pengurut*.

Pada `{7, 3, 5}`: tukar menjadi `{3, 7, 5}`; tukar menjadi `{3, 5, 7}`; tidak ditukar. Keluarannya `3 5 7`.

Python tidak dapat meneruskan alamat variabel. Ia menukar di pemanggil dengan `left, right = right, left`, atau mengubah list bersama di dalam fungsi.

Coba lima urutan lain dari tiga bilangan di kertas; semuanya berakhir terurut, bukti kecil bahwa tiga langkah cukup. Buku: Subbab 7.4.

Parameter Keluaran: Satu Pemanggilan, Beberapa Jawaban

Kode 7.3 Cuplikan `out_params.c`: tiga jawaban melalui pointer, dan status yang menyatakan apakah jawabannya sah.

```c
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;
}
```

*Dalam **Kode 7.3** pemanggil meneruskan alamat tiga variabel penerima jawaban; nilai kembalian menyatakan apakah jawabannya sah. Hari kosong mengembalikan 0 dan tidak menyentuhnya. `scanf` mengikuti pola yang sama. Buku: Subbab 7.5.*

Contoh Terurai 2: 3,45 kg dalam kg dan Gram

$$g = \text{round}(1000\,w), \qquad \text{kg} = \lfloor g/1000 \rfloor, \qquad \text{grams} = g \bmod 1000 \tag{7.2}$$

$w = 3.45$: $g = 3450$, kg $=3$, gram $=450$. Programnya mencetak `3.45 kg is 3 kg and 450 g`.

Pembulatannya penting: 0,29 kg kali 1000 adalah 289.99999999999994 dalam biner, yang oleh konversi biasa diubah menjadi 289 gram. `lround` memberi 290.

`split_weight(3.45, &kg, &grams)` menulis kedua jawaban melalui `*kg` dan `*grams`.

Persamaan (7.2) di papan, lalu pemanggilannya. Python mengembalikan tuple, `kg, grams = split_weight(3.45)`, dan `None` untuk "tidak ada jawaban sah". Buku: Subbab 7.5.

Nama Array adalah Alamat

Di hampir setiap ekspresi, `bill` berarti `&bill[0]`: *peluruhan array menjadi pointer*. Jadi `long *p = bill;` tidak perlu `&`.

Pengecualiannya `sizeof`: `sizeof bill` adalah 40 bita (lima `long`), `sizeof p` adalah 8, ukuran sebuah alamat.

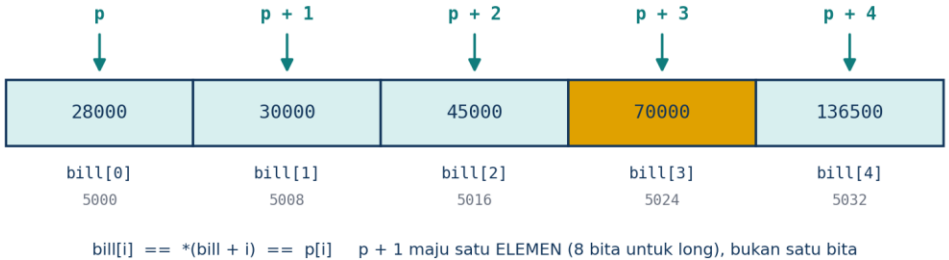
Array *adalah* elemen-elemennya; pointer hanya *tahu di mana* elemen itu berada.

Inilah sebabnya parameter array tidak dapat mengetahui panjangnya. Buku: Subbab 7.6.1.

Aritmetika Pointer

$$\text{address}(p + k) = \text{address}(p) + k \cdot \text{sizeof}(*p) \tag{7.3}$$

$$a[i] \equiv *(a + i) \tag{7.4}$$



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.

Persamaan (7.3) bergerak per elemen, bukan per bita; standar C mendefinisikan kurung siku dengan Persamaan (7.4). Dalam ***Gambar 7.3*** *array mulai di 5000:* `p + 3` *adalah* $5000 + 3 \cdot 8 = 5024$*, dan* `*(p + 3)` *bernilai 70000. Dengan* `int` *hasilnya 5012, tetap elemen 3. Buku: Subbab 7.6.2.*

Menelusuri dengan Pointer

Kode 7.4 Cuplikan `pointer_walk.c`: penelusuran dengan pointer, selisih pointer, dan indeks negatif dari pointer yang sudah bergeser.

```
    long bill[5] = {28000, 30000, 45000, 70000, 136500};
    long *p = bill;                     /* sama dengan &bill[0] */

    long sum = 0;
    for (long *q = bill; q < bill + 5; q++)   /* telusuri array dengan pointer */
        sum += *q;

    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]);
```

`bill + 5` adalah satu lewat ujung: pointer boleh menunjuk ke sana, tetapi tidak pernah diikuti di sana.

Kode 7.4 *mencetak* `sum by walking: 309500`, `last - p = 4 elements apart`, *serta* `*p = 70000`, `p[-1] = 45000`. *Selisih pointer bertipe* `ptrdiff_t`, *dicetak dengan* `%td`. *Program di buku juga membalik "laundry" dengan dua pointer yang berjalan saling mendekat. Buku: Subbab 7.6.3.*

Mengapa Pointer, Bukan Hanya Indeks?

Kecepatan bukan lagi alasannya: kompilator masa kini menghasilkan kode yang sama untuk `a[i]` dan penelusuran pointer.

Pointer dapat menunjuk ke **tengah** array: `total(bill + 3, 2)` menjumlahkan `bill[3]` dan `bill[4]` dengan fungsi yang tidak berubah.

Pointer adalah cara C meneruskan array, string, dan parameter keluaran, dan kelak menautkan struktur.

Python menyatakan penelusuran yang sama dengan indeks dan slice. Buku: Subbab 7.6.4.

`const` dan Pointer

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 khas
`long *p`	ya	ya	parameter keluaran; penelusuran yang menulis
`const long *p`	tidak	ya	parameter baca-saja; penelusuran yang hanya membaca
`long *const p`	ya	tidak	pointer yang terpaku di satu tempat
`const long *const p`	tidak	tidak	pandangan tetap yang baca-saja

*Baca **Tabel 7.1** dari kanan ke kiri: `const long *p` adalah "p adalah pointer ke long yang const". Baris kedua paling sering Anda tulis; `const long a[]` dalam daftar parameter berarti sama. Buku: Subbab 7.7.*

Pointer ke Fungsi

Kode 7.5 Cuplikan `sort_by.c`: satu bubble sort yang bertanya kepada sebuah fungsi nilai mana yang lebih dulu.

```c
int cheaper_first(long a, long b) { return a < b; }
int dearer_first(long a, long b) { return a > b; }

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;
            }
}
```

*Dalam **Kode 7.5**, `int (*before)(long, long)` adalah pointer ke fungsi; tanda kurungnya wajib. Pengurutan ditulis sekali; aturannya menjadi parameter. Array `rule[3]` berisi pointer semacam itu memungkinkan menu memilih aturan dengan satu indeks yang diperiksa. Buku: Subbab 7.8.*

Tiga Aturan, Satu Pengurutan, dan `qsort`

```
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
```

$$\mathrm{cmp}(a, b) = [a > b] - [a < b] \in \{-1,\ 0,\ 1\} \tag{7.5}$$

"Round first" mempertahankan urutan asli di dalam setiap kelompok: pengurutannya ***stabil****.* `qsort` *menerima array, banyaknya, ukuran elemen, dan pembanding atas* `const void *`*; tulis sebagai Persamaan (7.5), jangan* `a - b`*, yang dapat meluap.* `qsort` *tidak wajib stabil. Buku: Subbab 7.8.*

Kasir v6: Empat Jawaban dari Satu Pemanggilan

Kode 7.6 Cuplikan `stats.c`: `day_stats` dengan empat jawaban dan penelusuran pointer `const`.

```c
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;
}
```

Kode 7.6 *menolak pointer null dan hari kosong, dan menulis jawaban hanya di akhir, sehingga pemanggilan yang ditolak membiarkan jawabannya tak tersentuh.* `stats.py` *di Python mengembalikan tuple atau* `None`*. Buku: Subbab 7.9.*

Ringkasan saat Tutup

```
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)
```

Sabtu yang sama dengan Bab 6: $33.75 + 10.7 + 8.5 = 52.95$ *kg, ditampilkan oleh Persamaan (7.2) sebagai 52 kg 950 g. Rata-ratanya kini 47.591, dibulatkan ke rupiah terdekat, sedangkan Bab 6 memotong pecahannya: pilih satu aturan dan nyatakan. Buku: Subbab 7.10.*

Mengujinya

`split_weight(0.9996, ...)` harus dibulatkan ke 1 kg dan 0 g, tidak pernah melaporkan 0 kg dan 1000 g.

`day_stats` harus memberi empat jawaban yang benar untuk empat tagihan kecil, dan menolak hari kosong serta pointer null.

Pemanggilan yang ditolak harus membiarkan jawabannya: uji itu lebih dulu menaruh nilai yang benar di `total`, memanggil dengan `n = 0`, lalu memeriksa bahwa `total` masih memegangnya.

Seluruhnya empat belas pemeriksaan, diakhiri `qsort` dan `compare_long`. Menguji perilaku selain hasil itulah yang membuat parameter keluaran aman dipakai. Buku: Subbab 7.11.

Keamanan Pointer

Inisialisasi setiap pointer, ke alamat sungguhan atau `NULL`. **Uji sebelum mengikuti** pointer dari luar (SEI CERT EXP34-C).

Jangan kembalikan alamat variabel lokal: ia mati saat `return` dan pointernya *menggantung*; `gcc` memperingatkan *function returns address of local variable*.

Ketahui siapa pemilik memori: fungsi meminjamnya selama pemanggilan dan tidak boleh menyimpan pointernya untuk nanti.

Kembangkan dengan `-fsanitize=address,undefined`: kerusakan diam-diam menjadi pesan yang keras dan tepat.

```
    return &best;          /* WRONG: best disappears at the return */
```

Kembalikan nilainya, pointer ke dalam array pemanggil (`return &bill[k];` aman), atau tulis melalui parameter keluaran. Buku: Subbab 7.12.

Memakainya Sungguhan

Parameter keluaran dengan status — timbangan pasar: `int read_weight(double *kg)` mengembalikan 0 bila timbangan tidak mengirim apa pun.

Beberapa statistik dari satu penelusuran — seminggu di kebun buah dan durian: total, rata-rata per pohon, pohon terlemah dan terkuat.

Aturan yang dipilih saat berjalan — kios fotokopi mengurutkan menurut harga, waktu, atau nama: satu pengurutan, tiga pembanding.

Pointer ke tengah — `total(sales + 7, 7)` adalah minggu kedua dalam sebulan penjualan warung, tanpa menyalin.

`split_weight` menjadi `split_duration(int minutes, int *hours, int *mins)` untuk rental motor, dan penelusuran dua pointer menjadi pemeriksa palindrom di Bab 8. Buku: Subbab 7.13.

Kesalahan yang Sering Terjadi

Lupa `&` dalam pemanggilan; **lupa** `*` di dalam fungsi (`a = b;` menukar alamat lokal).

Pointer tanpa inisialisasi; **mengikuti** `NULL`; **mengembalikan alamat variabel lokal**.

Aritmetika dalam bita; **berjalan melewati ujung**; `sizeof` **pada pointer**; `a - b` **dalam pembanding**.

`long *a, b;`; `p == q` **lawan** `*p == *q`; **mengira Python mengubah bilangan milik pemanggil**.

`swap(left, right)` memancing makes pointer from integer without a cast: anggap peringatan itu galat. Buku: Subbab 7.14.

Ringkasan: Lima Hal untuk Diingat

1. Pointer menyimpan alamat; `*p` mengikutinya; `*(&x)` adalah `x` (Persamaan (7.1), **Gambar 7.1**); `NULL` tidak pernah diikuti.

2. Meneruskan alamat memungkinkan fungsi mengubah variabel pemanggil (**Gambar 7.2**); parameter keluaran ditambah status memberi beberapa jawaban.

3. Nama array meluruh menjadi pointer; `p + k` maju *k* elemen; `a[i]` adalah `*(a + i)` (Persamaan (7.4)).

4. `const` sebelum tipe melindungi nilainya, sesudah bintang melindungi pointernya (**Tabel 7.1**); pointer fungsi meneruskan perilaku; `qsort` memerlukan Persamaan (7.5).

5. Inisialisasi, uji `NULL`, jangan kembalikan alamat lokal, kembangkan dengan *sanitizer*.

Kasir v6 mendapat empat statistik dari satu pemanggilan, memecah berat harian dengan Persamaan (7.2), dan mengurutkan dengan `qsort`. Buku: Subbab 7.15.

Pertemuan Berikutnya

Bab 8 — Evaluasi dan Pembahasan II. Enam masalah dengan fungsi, array, dan pointer: fungsi statistik, Fibonacci dua cara, palindrom, determinan 3×3, nota dengan pencarian biner rekursif, dan penggabungan dua sif.

Masing-masing dengan penyelesaian tangan, rancangan, C dan Python, solusi salah yang menggoda, dan pertanyaan periksa diri.

Sebelum itu: Soal Latihan 7.1, 7.2, 7.5, dan 7.7.

Soal Latihan 7.1: gambarkan kotak dan panah sesudah setiap pernyataan; itu persiapan terbaik untuk Bab 8. Buku: Subbab 7.16 dan 7.17.

Kuliah 8 — Evaluasi dan Pembahasan II: Fungsi, Array, dan Pointer, Diselesaikan Langkah demi Langkah

Mata kuliah: Pemrograman Dasar (EF234101)

Buku: Irfan Subakti, *Pemrograman Dasar dengan C, C++, dan Python*, Bab 8

Dosen: Irfan Subakti — yifana@gmail.com

Tidak ada fitur baru C hari ini. Kita mengambil fungsi, array, dan pointer lalu menyelesaikan enam masalah utuh pelan-pelan, masing-masing dengan penyelesaian tangan, rancangan, program, dan solusi salah yang menggoda yang dibongkar. Buku: Bab 8, Subbab 8.1–8.11.

Cakupan Pertemuan Ini

0. Metodenya — delapan langkah, dan mengapa contoh hitungan tangan menentukan arti “benar”.

1. Fungsi statistik di dalam kerangka tetap. **2. Fibonacci** dua cara, diukur.

3. Palindrom tiga cara. **4. Determinan** 3×3 dan invers.

5. Mencari nota dengan pencarian biner rekursif. **6. Menggabungkan dua sif** dengan pointer yang berjalan.

7. Berdampingan — enam jebakan dan uji tepi yang menyingkapnya.

Setiap program dikompilasi dengan `-Wall -Wextra` *dan sekali lagi dengan* `-fsanitize=address,undefined`*; versi Python mencetak teks yang persis sama. Buku: Bab 8.*

Pertemuan Sebelumnya: Kotak Perkakas

Bab 5 — fungsi, prototipe, *pass-by-value*, `<math.h>` dengan `-lm`, cakupan, rekursi dengan kasus dasar.

Bab 6 — array dari 0 sampai $n-1$, perulangan dengan `i < n`, array diteruskan bersama panjangnya, bubble sort, pencarian linear dan biner, tabel.

Bab 7 — `&` dan `*`, parameter keluaran, `a[i]` adalah `*(a + i)`, dua pointer yang berjalan saling mendekat, `const`, fungsi sebagai parameter.

Tabel 8.1 Enam masalah dalam bab ini dan alat dari Bab 5 sampai 7 yang paling banyak dilatih masing-masing.

Masalah	Tugas	Alat utama
8.1	Lima fungsi statistik dalam kerangka yang diberikan	parameter array, perulangan, *cast*, `sqrt`
8.2	Fibonacci dengan perulangan dan rekursi	parameter keluaran, rekursi, menghitung pemanggilan
8.3	Apakah sebuah kata palindrom?	string sebagai array, dua pointer, rekursi, XOR
8.4	Determinan 3×3	array dua dimensi, satu rumus
8.5	Mencari nomor nota	pencarian biner rekursif
8.6	Menggabungkan dua sif	dua pointer yang berjalan, array keluaran

Tabel 8.1: menamai alat yang dibutuhkan sebuah masalah sudah separuh penyelesaiannya. Buku: Subbab 8.1.

Delapan Langkah

1. Rumuskan ulang, tuliskan setiap keputusan yang terbuka. **2.** Selesaikan contoh kecil dengan tangan, sampai digit terakhir. **3.** Rancang di kertas: fungsi, array, perulangan atau rekursi.

4. Perkirakan biayanya: 2^n langkah tidak berguna untuk $n = 60$. **5.** Tulis kode satu fungsi demi satu, peringatan menyala.

6. Uji contohnya dan tepinya. **7.** Temukan solusi salah yang menggoda dan masukan tempat ia gagal. **8.** Tinjau kembali.

> **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.

Empat tahap Pólya, pahami, rencanakan, laksanakan, tinjau kembali, diuraikan untuk program. Cobalah langkah 1 sampai 3 sendiri sebelum membaca setiap penyelesaian. Buku: Subbab 8.2.

Soal 8.1: Fungsi Statistik

> **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.

Kode yang diberikan adalah kesepakatan: mengganti nama `max` atau mengubah prototipe merusaknya, meskipun program masih berjalan.

Data kita: kilogram yang dibawa sebuah indekos kecil pada sepuluh kunjungan: 12, 7, 15, 9, 20, 11, 8, 14, 10, 18.

Dalam tim sungguhan prototipe adalah kesepakatan dengan kode orang lain. Buku: Subbab 8.3.1.

8.1: dengan Tangan

$$\acute{x} = \frac{1}{n}\sum_{i=0}^{n-1} x_i = \frac{124}{10} = 12.4 \tag{8.1}$$

$$\sigma = \sqrt{\frac{1}{n}\sum_{i=0}^{n-1} \left(x_i - \acute{x}\right)^2} \tag{8.2}$$

Tabel 8.2 Simpangan baku kesepuluh nilai yang dihitung dengan tangan: simpangan-simpangannya berjumlah nol, dan kuadratnya berjumlah 166,40.

x_i	12	7	15	9	20	11	8	14	10	18
$x_i - \acute{x}$	−0,4	−5,4	2,6	−3,4	7,6	−1,4	−4,4	1,6	−2,4	5,6
$\left(x_i - \acute{x}\right)^2$	0,16	29,16	6,76	11,56	57,76	1,96	19,36	2,56	5,76	31,36

Persamaan (8.1) memberi 12,4. Untuk Persamaan (8.2), Tabel 8.2: simpangan-simpangannya berjumlah 0, pemeriksaan aritmetika; $166.40/10 = 16.64$ *dan* $\sqrt{16.64} \approx 4.079$*. Jadi program harus mencetak 20, 7, 124, 12.400, dan 4.079. Buku: Subbab 8.3.2.*

8.1: Rancangan dan Penyelesaian

Kode 8.1 Cuplikan `stats.c`: rata-rata, dengan *cast* sebelum pembagian.

```c
double average(int *a, unsigned int aSize)
{
    return (double) sum(a, aSize) / aSize;   /* cast SEBELUM membagi */
}
```

`max` mulai dari `INT_MIN`, `min` dari `INT_MAX`, sehingga elemen pertama selalu menggantikannya.

`sDeviation` memakai ulang `average`, yang memakai ulang `sum`, lalu mengambil `sqrt` dari rata-rata kuadrat simpangan: tautkan dengan `-lm`.

Ukurannya `unsigned int`, jadi pencacahnya juga `unsigned int i`; mencampurnya memancing *comparison of integer expressions of different signedness*.

Kode 8.1 adalah salah satu dari lima badan yang dilengkapi; keluarannya cocok dengan hitungan tangan baris demi baris. Buku: Subbab 8.3.3 dan 8.3.4.

8.1: Solusi Salah yang Menggoda

`min` dimulai dari 0 “karena pencacah mulai dari 0”; rata-rata ditulis `sum(a, aSize) / aSize`, dua bilangan bulat.

```
Minimum value      : 0
Average            : 12.000
Standard deviation : 4.099
```

Minimumnya salah hanya untuk array bernilai positif, persis uji yang banyak dilupakan orang. Rata-rata bulat juga merusak simpangan bakunya.

> **PERIKSA DIRI.** Program lain mencetak 4.300 alih-alih 4.079. *Jawaban:* ia membagi dengan $n-1=9$: $\sqrt{166.40/9}\approx 4.300$, simpangan baku *sampel*. Tidak ada yang keliru; kekeliruannya adalah tidak menyebutkan mana yang dihitung program.

`statistics.stdev` di Python membagi dengan $n-1$, `statistics.pstdev` dengan n. Buku: Subbab 8.3.6.

Soal 8.2: Fibonacci Dua Cara

SOAL 8.2. Bacalah n. Hitunglah $F(n)$, dimulai dengan $F(0)=0$ dan $F(1)=1$, serta jumlah dari $F(0)$ sampai $F(n)$, dengan perulangan dan dengan rekursi. Kemudian bandingkan kerja kedua cara untuk n di atas 30. Untuk $n=9$: 0 1 1 2 3 5 8 13 21 34, sehingga $F(9)=34$ dan jumlahnya 88.

$$\sum_{i=0}^{n} F(i) = F(n+2) - 1 \tag{8.3}$$

Dengan tangan: $F(10)=55$, $F(11)=89$, *dan* $89-1=88$. *Persamaan (8.3) dibuktikan dengan induksi, kembaran matematis rekursi. Nilai di atas 90 ditolak: jumlah sampai* $F(90)$ *adalah yang terbesar yang muat di* `long long`. *Buku: Subbab 8.4.1 dan 8.4.2.*

8.2: Perulangannya

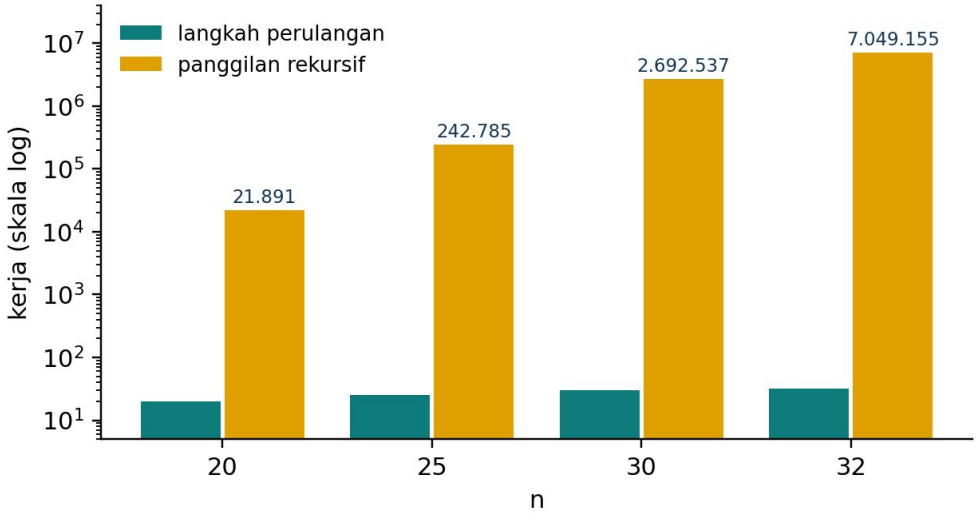
Pseudokode 8.1 Fibonacci iteratif: dua tetangga dalam barisan bergerak maju n kali, dan jumlahnya mengumpulkan setiap nilai yang pernah dipegang `prev`.

```
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
```

Telusuri Pseudokode 8.1 untuk $n=2$: jumlahnya mengumpulkan 0, lalu 1, dan sesudah perulangan `prev` bernilai $F(2)=1$, sehingga $0+1+1=2=F(4)-1$. Fungsi C mengembalikan $F(n)$ dan memberi jumlahnya lewat `long long *sum`. Rekursinya adalah `fib` dari Bab 5 dengan penghitung pemanggilan. Buku: Subbab 8.4.3.

8.2: Kerja Dua Cara

```
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
```



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

Gambar 8.1: $C(32) = 2 \cdot 3524578 - 1 = 7049155$ *pemanggilan lawan 32 langkah. Setiap kenaikan 5 mengalikan pemanggilan dengan sekitar 11. Buku: Subbab 8.4.4.*

8.2: Diukur, dan Jebakannya

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 $1.618^5 \approx 11$.

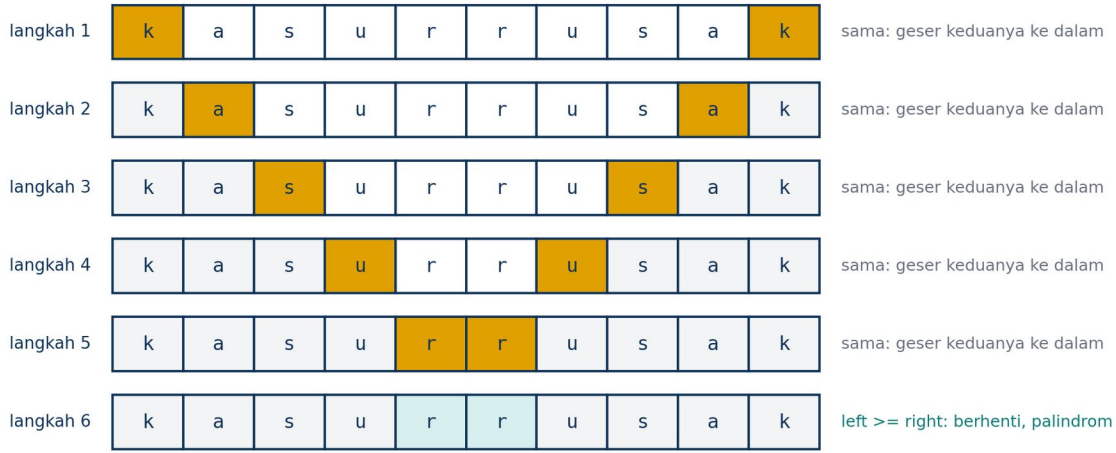
Cara	$F(35)$	$F(40)$
rekursi C, `gcc -O0`	0,069 s	0,807 s
rekursi C, `gcc -O2`	0,014 s	0,162 s
rekursi Python	1,05 s	11,3 s
perulangan, C atau Python	di bawah 0,0001 s	di bawah 0,0001 s

Jebakannya: mulai dari 1, 1 mencetak sepuluh bilangan Fibonacci yang benar, tetapi bergeser: $F(9)=55$, jumlah 143. Contoh yang diberikan ada untuk menangkapnya.

Tabel 8.3: pengoptimal membuat C lima kali lebih cepat tetapi tidak dapat mengubah pertumbuhannya; Python sekitar tujuh puluh kali lebih lambat; perulangan mengalahkan semuanya. Algoritma yang lebih baik mengalahkan bahasa yang lebih cepat. Dengan `int` jumlahnya meluap pada $n=45$. Buku: Subbab 8.4.6 dan 8.4.7.

Soal 8.3: Palindrom

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`. Kata-katanya ditulis dalam huruf kecil tanpa spasi. Sebagai tantangan tambahan, putuskan juga dengan rekursi atau operasi bit.



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.

Gambar 8.2: dua jari pada `kasurrusak`; lima pasangan sama, jari-jarinya bersilang, palindrom. Untuk `laundry`, `l` lawan `y` langsung memutuskan. Satu huruf tidak punya pasangan dan merupakan palindrom. Paling banyak $\lfloor n/2 \rfloor$ perbandingan. Buku: Subbab 8.5.1 dan 8.5.2.

8.3: Tiga Cara yang Harus Sepakat

Kode 8.2 Cuplikan `palindrome.c`: pemindaian dua pointer dan kembaran rekursifnya.

```c
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);
}
```

Kode 8.2: pemindaiannya adalah penelusuran pembalik dari Bab 7 dengan perbandingan alih-alih penukaran. Cara ketiga meng-OR-kan `x ^ y` setiap pasangan: 0 di akhir berarti palindrom, tanpa `if` di dalam perulangan. Baca dengan `scanf("%99s", word)`: tanpa `&`, dan dengan lebar. Buku: Subbab 8.5.3 dan 8.5.4.

8.3: Jebakan `n - i`

Kode 8.3 Cuplikan `wrong_palindrome.c`: indeks pasangannya `n - i`, bukan `n - 1 - i`.

```c
    for (size_t i = 0; i < n / 2; i++)
        if (s[i] != s[n - i])            /* SALAH: s[n] adalah '\0' */
            return 0;
```

Setiap palindrom dua huruf atau lebih ditolak: `s[0]` dibandingkan dengan `s[n]`, yaitu `'\0'` penutup. Hanya `x` yang lolos.

Di C program tidak mogok, karena `s[n]` masih di dalam array: *sanitizer* pun diam. Python tanpa nol eksplisit akan melempar `IndexError`.

> **PERIKSA DIRI.** Terima “Kasur rusak” dan “Never odd or even”. *Jawaban:* lewati yang bukan huruf dengan `isalpha`, bandingkan `tolower` keduanya; dan baca per baris, bukan dengan `%99s`, yang berhenti di spasi pertama.

Kode 8.3: pasangan dari `i` adalah `n - 1 - i`. Buku: Subbab 8.5.6.

Soal 8.4: Determinan

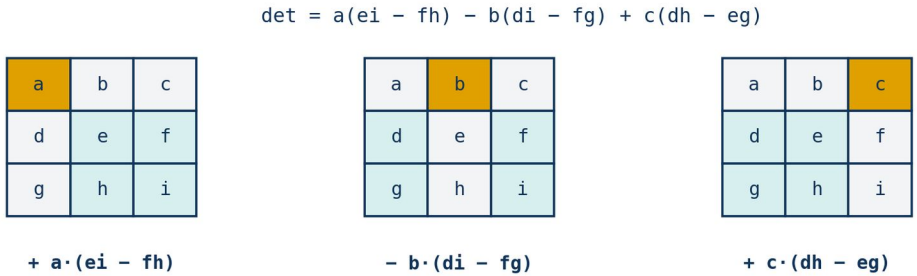
SOAL 8.4. Bacalah matriks 3×3 berisi bilangan bulat, hitung determinannya, dan laporkan apakah matriks itu punya invers: matriks itu punya invers tepat ketika determinannya tidak nol.

$$A = \begin{pmatrix} 2 & 1 & 1 \\ 1 & 3 & 2 \\ 1 & 0 & 0 \end{pmatrix} \tag{8.4}$$

Mengapa laundry peduli: tiga nota untuk deterjen, pelembut, dan kanji setrika, Rp 60.000, Rp 72.000, dan Rp 20.000, dengan jumlah barang Persamaan (8.4). Harganya dapat dipulihkan tepat ketika A punya invers. Nota yang berulang memberi dua baris sama dan determinan 0. Buku: Subbab 8.6.1.

8.4: Ekspansi Sepanjang Baris Pertama

$$\det A = a(ei - fh) - b(di - fg) + c(dh - eg) \tag{8.5}$$



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 $+, -, +$.

Gambar 8.3: entrinya a, b, c; d, e, f; g, h, i baris demi baris. Tanda minus pada suku tengah adalah bagian yang mudah terlupa. Buku: Subbab 8.6.2.

8.4: dengan Tangan, dan dalam C

Baris $(2,-3,1)$, $(2,0,-1)$, $(1,4,5)$: $2\cdot 4+3\cdot 11+1\cdot 8=8+33+8=49$. Punya invers.

Persamaan (8.4): $2\cdot 0-1\cdot(-2)+1\cdot(-3)=-1$. Tidak nol: harganya dapat dipulihkan.

Kode 8.4 Cuplikan `determinant.c`: Persamaan (8.5) disalin dengan `m[0][0]` untuk a, `m[0][1]` untuk b, dan seterusnya.

```c
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]);
}
```

Kode 8.4 memakai `long long`, karena hasil kali tiga entri cepat membesar; `read_matrix` membaca dengan dua perulangan bersarang dan mengembalikan 0 bila masukan habis. Buku: Subbab 8.6.3 dan 8.6.4.

8.4: Jebakan Tanda

Jebakannya: ketiga suku dijumlahkan. Untuk matriks identitas hasilnya tetap 1, karena suku tengah dikalikan $b = 0$.

Dua baris sama harus memberi 0; program keliru berkata 24. Contoh tangan memberi -17 alih-alih 49; nota-nota memberi -5 alih-alih -1.

> **PERIKSA DIRI.** Baris $(1, 2, 3)$, $(2, 4, 6)$, $(0, 1, 5)$? *Jawaban:* nol, karena baris kedua dua kali baris pertama: $1 \cdot 14 - 2 \cdot 10 + 3 \cdot 2 = 0$.

Uji yang dicoba semua orang lebih dulu, matriks identitas, tidak dapat menangkap kekeliruan ini; pilih uji yang setiap entrinya berpengaruh. Buku: Subbab 8.6.6.

Soal 8.5: Mencari Nota

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.

Kode 8.5 Cuplikan `find_receipt.c`: pencarian biner rekursif (versi buku juga mencetak setiap pemanggilan, menjorok sesuai kedalamannya).

```c
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;
    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);
}
```

Kode 8.5: dua kasus dasar, bagian kosong dan tepat kena; setiap pemanggilan rekursif bekerja pada bagian yang benar-benar lebih kecil. Buku: Subbab 8.7.1–8.7.3.

8.5: Menelusuri Pemanggilan

```
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
```

Dua belas nota dari 1003 sampai 1066. 1047 ditemukan dalam dua pemanggilan; 1003 dalam tiga; 1050, yang tidak ada, sesudah empat.

Jebakannya: rekursi pada `mid` alih-alih `mid - 1` atau `mid + 1`. Dengan bagian dua elemen dan kunci di kanan, bagiannya tidak pernah menyusut: *stack overflow*.

Tumpukan sedalam banyaknya pemanggilan, paling banyak $\lfloor \log_2 n \rfloor + 1$*: 20 bingkai untuk sejuta nota.* `fib` *berbahaya karena total pemanggilannya, bukan kedalamannya. Buku: Subbab 8.7.3 dan 8.7.4.*

Soal 8.6: Menggabungkan Dua Sif

> **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.

Dua tumpukan menghadap ke atas, yang terkecil di atas: ambil nota teratas yang lebih kecil, ulangi; bila satu tumpukan habis, taruh sisa tumpukan lainnya ke hasil.

Pagi 12, 28, 30, 45, 136,5 dan sore 18, 30, 56, 70 (ribuan) memberi 12, 18, 28, 30, 30, 45, 56, 70, 136,5.

Kerjakan penggabungan di papan dengan dua tumpuk kartu sebelum menampilkan kodenya. Buku: Subbab 8.8.1 dan 8.8.2.

8.6: Tiga Pointer yang Berjalan

Kode 8.6 Cuplikan `merge_shifts.c`: menggabungkan dua array terurut dengan tiga pointer yang berjalan.

```c
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);
}
```

$$n_a + n_b - 1 \tag{8.6}$$

Kode 8.6: `<=` *mengambil tagihan pagi bila sama, sehingga penggabungannya stabil;* `po - out` *menghitung hasilnya.* $n_a + n_b$ *langkah dan paling banyak Persamaan (8.6) perbandingan. Langkah ini adalah jantung merge sort.* ***Jebakannya:*** *hanya perulangan pertama; tagihan hilang kecuali kedua tumpukan habis bersamaan. Buku: Subbab 8.8.2–8.8.4.*

Keenam Masalah Berdampingan

Tabel 8.4 Keenam masalah berdampingan: biaya setiap penyelesaian, jebakan solusi salah yang menggoda, dan masukan uji yang menyingkap jebakan itu.

Masalah	Biaya	Jebakan	Uji yang menyingkapnya
8.1 Statistik	n langkah per fungsi	`min` dari 0; rata-rata bulat	hanya nilai positif
8.2 Fibonacci	perulangan n; rekursi $2F(n+1)-1$	mulai dari 1, 1	contoh yang diberikan
8.3 Palindrom	paling banyak $n/2$ perbandingan	pasangan di `n - i`	palindrom dua huruf atau lebih
8.4 Determinan	tetap 14 operasi	tanda suku tengah	dua baris sama
8.5 Nota	paling banyak $\lfloor \log_2 n \rfloor + 1$ pemanggilan	rekursi pada `mid`	bagian dua elemen
8.6 Penggabungan	$n_a + n_b$ langkah	melupakan sisanya	array kosong

Tabel 8.4: dalam setiap kasus uji yang menyingkap adalah sebuah tepi: batas, masukan kosong, nilai berulang, contoh yang dihitung dengan tangan. Kernighan dan Pike menyebutnya pengujian di batas. Buku: Subbab 8.9.

Kesalahan yang Sering Terjadi

Mengubah kode yang diberikan; **meng-*cast* hasil** alih-alih operan; **minimum yang mulai dari 0**; **bertanda dan tak bertanda tercampur**.

Melewatkan contoh; **rekursi pada masukan besar tanpa menghitung pemanggilannya**.

Indeks pasangan meleset satu; **`&` pada array di `scanf`**, dan tanpa lebar; **matriks uji yang lemah**.

Rekursi yang tidak menyusut; **melupakan sisa sesudah penggabungan**.

Setiap butir adalah salah satu dari enam jebakan dalam bentuk umumnya. Buku: Subbab 8.10.

Ringkasan: Lima Hal untuk Diingat

1. Delapan langkah; contoh hitungan tangan menentukan arti “benar” (Tabel 8.2).

2. Algoritma yang lebih baik mengalahkan bahasa yang lebih cepat: n langkah perulangan lawan $2F(n+1)-1$ pemanggilan (Gambar 8.1).

3. Dua pointer dari kedua ujung memutuskan palindrom; pasangan dari `i` adalah `n - 1 - i` (Gambar 8.2).

4. Salin rumus dengan persis (Persamaan (8.5)); matriks identitas tidak dapat menangkap kekeliruan tanda (Gambar 8.3).

5. Setiap pemanggilan rekursif harus menyusutkan bagiannya; penggabungan harus menyalin sisanya. Uji tepinya.

Fungsi, array, dan pointer bersama-sama menyelesaikan masalah yang tidak dapat diselesaikan masing-masing sendirian. Buku: Subbab 8.11.

Pertemuan Berikutnya

Bab 9 — Karakter, String, dan I/O Terformat. Teks sebagai array `char`, membaca baris utuh, dan seluruh kemampuan `printf` dan `scanf`.

Pemeriksa palindrom belajar membaca “Kasur rusak” sebagai satu baris.

Sebelum itu: Soal Latihan 8.1, 8.3, 8.5, dan 8.6.

Soal Latihan 8.5 memakai aturan Cramer untuk memulihkan tiga harga dari nota-nota Mbak Rina; jawaban buku memberi deterjen Rp 20.000 dan pelembut Rp 12.000. Buku: Subbab 8.12 dan 8.13.

Kuliah 9 — Karakter, String, dan I/O Terformat

Mata kuliah: Pemrograman Dasar (EF234101)

Buku: Irfan Subakti, *Pemrograman Dasar dengan C, C++, dan Python*, Bab 9

Dosen: Irfan Subakti — yifana@gmail.com

Hari ini kasir belajar menangani nama: bagaimana C menyimpan teks, bagaimana menggolongkan dan mengubah karakter, bagaimana merapikan kolom keluaran, bagaimana membaca satu baris dengan aman, dan kasir v0.7. Buku: Bab 9, Subbab 9.1–9.13.

Cakupan Pertemuan Ini

1. **Teks adalah bilangan** — karakter adalah bilangan bulat kecil; string adalah larik `char` yang diakhiri `'\0'`.
2. **Satu karakter pada satu waktu** — `<ctype.h>`: menggolongkan, membersihkan nomor telepon, merapikan nama.
3. **Pustaka string dan sisi tajamnya** — menyalin, menyambung, membandingkan, mencari, memotong; satu aturan aman untuk masing-masing.
4. **Seluruh tombol pengatur `printf`** — *flag*, lebar, presisi, tipe; struk yang kolomnya rata.
5. **Membaca teks secara aman, dan kasir v0.7** — `fgets`, baris yang terlalu panjang, kata-kata yang dipotong dari satu baris, *read–evaluate–print loop*.

Setiap program ada dalam C dan Python, dan keduanya mencetak keluaran yang persis sama. Buku: Bab 9.

Posisi Kita: Nama-nama di Buku Mbak Rina

Sampai Bab 7 kasir sudah menghitung tagihan, menyimpan pesanan sehari dalam larik, dan mengembalikan beberapa hasil lewat pointer; Bab 8 memakai semua itu pada soal-soal utuh.

Tetapi pesanan diberi nomor 1, 2, 3, dan Mbak Rina menyimpan buku catatan kertas untuk mengingat nomor mana milik pelanggan mana.

Buku itu memuat “andi”, “ANDI PRATAMA”, “Andi Pratama”, dan “andi p.” untuk satu orang, dan nomor telepon dalam tiga bentuk berbeda.

Kolom struk berpindah-pindah: nama yang panjang mendorong harga ke kanan.

Semuanya adalah soal teks: merapikan huruf, menyisakan angka, mencari satu teks di dalam teks lain, memberi tahu `printf` berapa kolom yang dipakai, memotong baris ketikan menjadi kata. Buku: Subbab 9.1.

Karakter Adalah Bilangan Kecil

C menyimpan huruf `A` sebagai bilangan yang disepakati untuknya: dalam **ASCII**, `'A'` adalah 65, `'a'` 97, `'0'` 48, spasi 32.

Angka-angka berurutan, jadi nilai sebuah karakter angka adalah `c - '0'`: `'7' - '0'` adalah $55 - 48 = 7$.

Setiap huruf kecil terletak 32 tempat di atas huruf besarnya: `'a' - 'A'` adalah $97 - 65 = 32$.

Huruf di luar bahasa Inggris, seperti “Chloé”, memakai lebih dari satu byte dalam UTF-8; bab ini bekerja dengan ASCII, yang mencakup semua yang diketik di laundry.

Sebuah `char` dapat dijumlahkan, dibandingkan, dan dikurangkan seperti bilangan bulat lain. Itulah seluruh “keajaiban” di balik pengubahan huruf besar-kecil. Buku: Subbab 9.2.1.

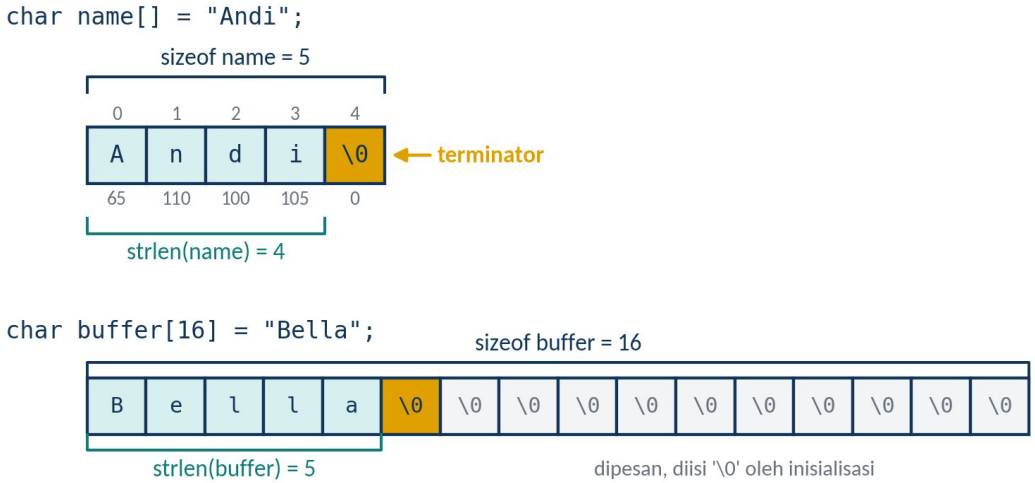
String Diakhiri dengan Nol

DEFINISI. *String* dalam C adalah deretan karakter yang disimpan dalam byte-byte berurutan sebuah larik `char` dan diakhiri karakter nol `'\0'` (byte bernilai 0). *String literal*, seperti `"Andi"`, disimpan kompilator dengan penutup yang sudah ditambahkan. *Panjang* string adalah banyaknya karakter sebelum penutup.

Penutup itulah seluruh triknya: ia menandai akhir teks, sehingga panjang tidak disimpan di mana pun.

C tidak punya tipe string bawaan yang tumbuh sendiri seperti Python, tetapi gagasan di balik string C cukup kecil untuk dipahami seluruhnya hari ini. Buku: Subbab 9.2.2.

Dua String di Memori



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.

Gambar 9.1: `char name[] = "Andi";` membuat larik yang pas, lima byte; `char buffer[16] = "Bella";` memesan enam belas dan mengisi sisanya dengan nol. Buku: Subbab 9.2.2.

Byte Lawan Karakter

`sizeof` — berapa byte yang ditempati larik; kompilator mengetahuinya dari deklarasi.

`strlen` — berapa panjang teksnya; sebuah perulangan yang menghitung karakter sampai bertemu `'\0'`.

$$\text{sizeof name} = n + 1\,, \text{strlen}(\text{name}) = n \tag{9.1}$$

Persamaan (9.1) berlaku untuk larik yang dideklarasikan dari literal berisi n karakter. Lupa "+ 1" adalah cara paling umum membuat larik kurang satu byte.

Untuk `name` jawabannya 5 dan 4; untuk `buffer` 16 dan 5. Buku: Subbab 9.2.2.

Setiap Fungsi String Adalah Perulangan

Kode 9.1 Dari `string_basics.c`: `strlen` yang ditulis sendiri; sisa program mencetak larik pada Gambar 9.1 byte demi byte.

```
static size_t my_strlen(const char *s)
{
    size_t n = 0;
    while (s[n] != '\0')
        n++;
    return n;
}
```

Kode 9.1 adalah pola setiap fungsi dalam `<string.h>`: berjalan sampai penutup. Buku: Subbab 9.2.2.

Apa yang Dicetak `string_basics`

```
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
```

`name[0] = 'a'` diperbolehkan: larik menyimpan **salinannya sendiri** atas huruf-huruf itu.

`buffer[3] = '\0'` tidak menghapus apa pun, tetapi setiap fungsi kini berhenti di indeks 3 dan hanya melihat `Bel`.

Isi byte `65 110 100 105 0` adalah gambar tadi dalam bentuk bilangan. Buku: Subbab 9.2.2.

Larik Lawan Pointer ke Literal

> **PERINGATAN:** `const char *shop = "Profitina Laundry";` tidak membuat larik; ia membuat *pointer* ke literal yang disimpan di memori yang mungkin tidak boleh ditulisi. `shop[0] = 'p';` adalah perilaku tak terdefinisi dan biasanya membuat program jatuh. Deklarasikan pointer seperti itu dengan `const`, maka kompilator menolak penugasannya. Teks yang akan diubah disimpan di larik.

Di Python, string mengetahui panjangnya dan tidak dapat diubah di tempat; `len("Andi")` bernilai 4 dan “perubahan” membangun string baru (`string_basics.py` di buku).

Pointer dari Bab 7 bertemu teks dari Bab 9 di sini: nama larik dan pointer tampak serupa dalam ekspresi, tetapi keduanya berbeda. Buku: Subbab 9.2.2–9.2.3.

Satu Karakter pada Satu Waktu: `<ctype.h>`

Menggolongkan — `isalpha`, `isdigit`, `isspace`, `isupper`, `islower`, `isalnum`, `ispunct`.

Mengubah — `toupper`, `tolower`; karakter yang bukan huruf dikembalikan tanpa perubahan.

Satu aturan — kirimkan `(unsigned char)c`; `char` bernilai negatif adalah perilaku tak terdefinisi.

Tiga tugas untuk laundry: menghitung jenis karakter, menyisakan angka nomor telepon (`digits_only`), dan menulis nama dalam *title case* (`tidy_name`).

`valid_mobile` lalu memeriksa bahwa nomor yang sudah bersih diawali `08` dan terdiri atas 10 sampai 13 angka. Buku: Subbab 9.3.

Contoh Terurai 1: Merapikan Nama

Kode 9.2 Dari `char_classes.c`: `tidy_name` memangkas spasi di ujung, merapatkan spasi di tengah menjadi satu, dan menulis setiap kata dalam *title case*.

```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++] = ' ';          /* one space between words */
        dst[n++] = (char)(new_word ? toupper(c) : tolower(c));
        new_word = 0;
    }
    dst[n] = '\0';
}
```

Kode 9.2: variabel keadaan `new_word` mengingat apakah karakter sebelumnya spasi. Buku: Subbab 9.3.

Menelusuri `"  aNDi   pratama "`

Dua spasi di depan — masing-masing mengeset `new_word = 1` dan tidak menyalin apa pun; `n` masih 0.

`a` — kata dimulai, tetapi `n` masih 0, jadi tidak ada spasi yang ditulis; `A` ditulis kapital. `n` menjadi 1.

`NDi` — di dalam kata: ditulis `ndi`. Hasil sejauh ini `Andi`.

Tiga spasi — hanya `new_word = 1`. Lalu **`p`**: satu spasi ditulis, lalu `P`; sisanya menjadi `ratama`.

Spasi di belakang — tidak ada karakter sesudahnya, jadi ia tidak menghasilkan apa pun. Hasil: `Andi Pratama`.

“Apa yang baru saja saya lihat, dan apa artinya bagi karakter sekarang?” adalah inti semua pemrosesan teks. Di Python seluruh fungsinya menjadi `" ".join(w.capitalize() for w in src.split())`, sampai batas ukuran. Buku: Subbab 9.3.

Gagasan yang Sama di Python

Karakter — `ord("A")` bernilai 65 dan `chr(97)` adalah `"a"`; string satu huruf berperan sebagai `char`.

Menggolongkan — `c.isalpha()`, `c.isdigit()`, `c.isspace()` berperilaku seperti saudaranya di C untuk teks ASCII.

Membersihkan nomor telepon — `"".join(c for c in phone if c.isdigit())` hanya menyisakan angka.

Merapikan nama — `src.split()` tanpa argumen memecah pada setiap deretan spasi dan membuang potongan kosong: pemangkasan dan perapatan perulangan C dalam satu panggilan.

Mencari — `part.lower() in name.lower()` adalah `strstr` tanpa membedakan huruf besar-kecil milik kasir.

Python membangun string baru alih-alih mengisi larik, jadi tidak ada penutup yang bisa terlupa dan tidak ada ukuran yang harus dikirim; harganya, setiap perubahan membuat salinan. Keluaran kedua bahasa sama, baris demi baris. Buku: Subbab 9.3.

Pustaka String: Enam Keluarga

Panjang — `strlen`. **Menyalin** — `strcpy`, `strncpy`; lebih baik `snprintf(dst, sizeof dst, "%s", src)`.

Menyambung — `strcat`; penyambungan berbatas dengan `snprintf` di ujung teks sekarang.

Membandingkan — `strcmp(a, b)` bernilai negatif, nol, atau positif; *sama* berarti `== 0`.

Mencari — `strchr` untuk karakter, `strstr` untuk teks; keduanya mengembalikan `NULL` bila tidak ketemu.

Memotong — `strtok` memecah di tempat, mengubah masukannya, dan mengingat posisinya dalam variabel tersembunyi.

Tak satu pun fungsi ini memeriksa apa-apa. Aturannya: beri setiap salinan ukuran, uji hasil pencarian terhadap `NULL`, bandingkan dengan `== 0`, dan ingat apa yang dilakukan `strtok` pada masukannya. `string_library.c` dan `string_library.py` memperlihatkannya berdampingan. Buku: Subbab 9.4.

Sisi Tajam: Kelebihan Satu Byte

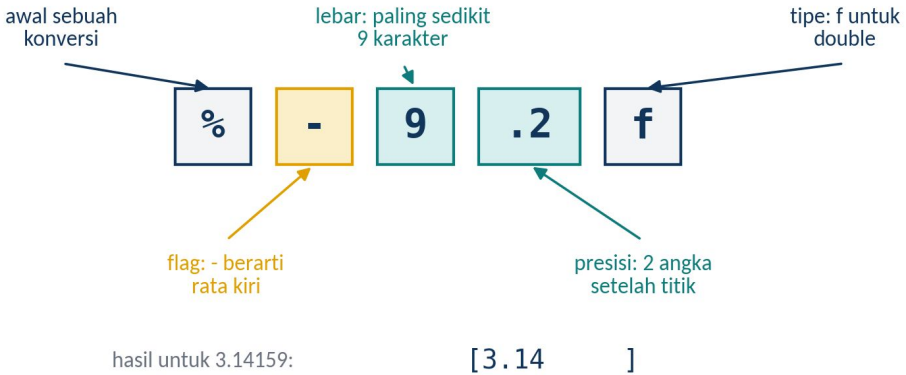
> **PERINGATAN:** `char code[7]; strcpy(code, "PL-0042");` menulis 8 byte ke dalam 7: tujuh huruf dan penutup. `strcpy` tidak tahu ukuran `code`, jadi ia menulis melewati ujungnya ke apa pun yang ada berikutnya di memori. Programnya bahkan bisa tampak berjalan benar.

Perbaikannya: tentukan ukuran menurut Persamaan (9.1), atau salin dengan `snprintf(code, sizeof code, "%s", text)`, yang tidak pernah menulis lebih dari `sizeof code` byte dan selalu memberi penutup.

Mengompilasi dengan `-fsanitize=address,undefined` mengubah luapan diam-diam itu menjadi laporan yang jelas.

Inilah sumber bug paling terkenal dalam sejarah C; satu kebiasaan menghilangkannya. Buku: Subbab 9.4.

Seluruh Tombol Pengatur `printf`



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`.

Gambar 9.2 membongkar satu spesifikasi. Buku: Subbab 9.5.

Spesifikasi Konversi

$$\%\,[\text{flags}][\text{width}][.\,\text{precision}]\,\text{type} \tag{9.2}$$

Lebar — batas minimum: nilai pendek diberi pengisi (di kiri kecuali ada `-`); nilai panjang tidak pernah dipotong.

Presisi — angka di belakang titik untuk `f`; angka bermakna untuk `g`; banyak karakter *maksimum* untuk `s`.

Flag — `-` rata kiri, `0` pengisi nol, `+` selalu bertanda, spasi untuk celah di depan bilangan positif.

`*` — lebar diambil dari daftar argumen: `printf("%*s", w, text)`.

Persamaan (9.2) adalah seluruh tata bahasanya. `format_dial.c` memutar satu tombol pada satu waktu dan mencetak setiap medan di antara kurung siku; operator `%` Python membaca spesifikasi yang sama. Buku: Subbab 9.5.

Contoh Terurai 2: Membaca Tombol Pengatur

CONTOH. Apa yang dicetak `printf("[%-6.3s][%06.1f][%+5d]\n", "Profitina", 3.14159, 42);`?

`%-6.3s` — paling banyak 3 karakter, `Pro`, rata kiri dalam 6: `[Pro   ]`.

`%06.1f` — satu desimal, `3.1`, diisi nol sampai lebar 6: `[0003.1]`.

`%+5d` — selalu bertanda, `+42`, rata kanan dalam 5: `[  +42]`.

```
[Pro   ][0003.1][  +42]
```

Kerjakan di papan sebelum menjalankannya; inilah Soal Latihan 9.1 di buku. Buku: Subbab 9.5 dan 9.14.

Struk yang Kolomnya Rata

Lebar tetap untuk setiap kolom: `%-20s` untuk nama, `%-10s` untuk layanan, `%6.1f` untuk berat, jumlah uang rata kanan.

`rupiah` menulis 1250000 sebagai `1.250.000`: `snprintf` membuat deretan angka, lalu titik diletakkan sebelum angka ke-`i` bila `len - i` kelipatan tiga dan `i` bukan 0.

`%*s` meletakkan nama toko di tengah: lebarnya dihitung dari panjang teks.

Uang tetap disimpan sebagai **bilangan bulat**. `%.2f` dari 2.675 mencetak 2.67, karena 2.675 tidak dapat dinyatakan tepat dalam biner.

`receipt.c` dan `receipt.py` mencetak struk yang sama; `rupiah` dan fungsi penengah dipakai sampai akhir buku. Buku: Subbab 9.6.

Membaca Teks secara Aman

`scanf("%s", ...)` berhenti di spasi pertama dan tidak tahu ukuran larik; `%31s` hanya memperbaiki masalah kedua.

Langkah 1 — baca satu baris utuh dengan `fgets(buf, size, stdin)`: tidak pernah lebih dari `size` byte, selalu diberi penutup, `NULL` di akhir masukan.

Langkah 2 — periksa yang datang: `'\n'` di akhir berarti barisnya muat; buang (bersama `'\r'` dari berkas Windows). Larik penuh tanpa baris baru berarti baris terlalu panjang; baca dan buang sisanya.

Langkah 3 — baru kemudian potong baris menjadi kata dan tentukan artinya.

Pisahkan membaca dari memahami. Satu baris `buf[strcspn(buf, "\r\n")] = '\0';` membuang akhir baris. Buku: Subbab 9.7.

`read_line` dan `next_word`

`read_line` mengembalikan 1 untuk baris yang baik, 0 di akhir masukan, −1 untuk baris yang terlalu panjang.

Dengan 80 byte, satu baris boleh 79 karakter: bila larik penuh, `getchar()` mengintip; baris baru berarti barisnya tepat muat.

`next_word(&p, out, size)` melompati spasi, menyalin satu kata (dipotong menjadi `size - 1`, tetapi seluruh kata selalu dihabiskan), dan memajukan `p` melewatinya.

Parameter `const char **p` adalah teknik Bab 7 untuk fungsi yang mengubah pointer milik pemanggil.

Mengapa bukan `sscanf` dengan `%15s`? Ia membaca 15 huruf dari kata 20 huruf dan meninggalkan 5 untuk dianggap kata berikutnya, sehingga arti baris berubah diam-diam. Buku: Subbab 9.7.

Kasir v0.7: *Read–Evaluate–Print Loop*

Cetak *prompt* `>`, baca satu baris, jalankan perintahnya, cetak hasilnya, mulai lagi.

`add <service> <kg> <name>` — layanan `fold`, `wash-iron`, atau `iron`; berat seperti `3` atau `3.5`; nama beberapa kata dengan huruf besar-kecil sembarang.

`find <part of a name>` — `strstr` tanpa membedakan huruf besar-kecil atas nama yang sudah rapi. **`list`**, **`help`**, **`quit`**.

Setiap baris yang salah mendapat pesan yang sopan, tidak pernah membuat program jatuh.

`cashier_v7.c` dan `cashier_v7.py` adalah program lengkapnya, bersama fungsi-fungsi pembantu bab ini. Buku: Subbab 9.8.

Satu Pagi di Meja Kasir

```
> 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 iron 4.25 citra
  weight must look like 3 or 3.5 (0.1 to 50.0 kg)
> add fold 1.5
  the customer's name is missing
> find ari
  #3 Citra Dewi Lestari, iron, 4.0 kg = Rp 24.000
```

Dari jalannya program pada `data/session9.txt` dengan `--echo`; C dan Python mencetak hal yang sama. `find ari` menemukan "Lestari": pencarian dengan sebagian nama. Buku: Subbab 9.9.

Tabel dan Baris yang Terlalu Panjang

```
> 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
> 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.
```

Kolom tidak lagi berpindah-pindah, dan baris yang panjang ditolak utuh, bukan dibaca sebagai dua perintah. Program penguji membandingkan keluaran kedua program dan menjalankan C di bawah sanitizer. Buku: Subbab 9.9.

Membuatnya Lebih Baik, Memakainya secara Nyata

Tiga larik sejajar — medan baru berarti larik baru di semua tempat; Bab 10 mengelompokkan satu pesanan dalam satu `struct`.

Ukuran tetap, hilang saat listrik padam — Bab 10 menumbuhkan data dengan `malloc`; Bab 11 menyimpannya ke berkas.

Secara nyata — penjahit, klinik, atau kebun buah dan durian yang berjualan lewat obrolan dapat memakai ulang `tidy_name`, `digits_only`, dan tata letak struk.

Nama dan nomor telepon adalah data pribadi (UU PDP, UU No. 27 Tahun 2022): simpan hanya yang dibutuhkan usaha, dan cetak hanya empat angka terakhir.

Nyatakan batasnya (`NAME_LEN` 32, baris 79 karakter) di dalam pesan. Nama UTF-8 lewat dengan aman, tetapi `toupper` membiarkan é apa adanya dan panjang dihitung dalam byte. Buku: Subbab 9.10–9.11.

Kesalahan yang Sering Terjadi

Lupa byte penutup; `==` **pada string** (membandingkan alamat); `if (strcmp(a, b))` dibaca sebagai “sama”.

`sizeof` **pada parameter pointer** memberi 8, bukan ukuran larik pemanggil: kirim ukurannya terpisah.

`char` **biasa untuk** `<ctype.h>`; **menulis ke string literal**; `%s` **tanpa lebar**.

`fgets` **menyimpan baris baru**, sehingga `"quit\n"` bukan `"quit"`; `strtok` pada medan kosong; **uang dalam** `double`.

Setiap kesalahan di buku disertai perbaikannya. Buku: Subbab 9.12.

Ringkasan: Lima Hal yang Perlu Diingat

1. Karakter adalah bilangan bulat kecil; nilai karakter angka adalah `c - '0'`.

2. String C adalah larik `char` yang diakhiri `'\0'`; `sizeof` menghitung byte, `strlen` menghitung karakter (Persamaan (9.1)).

3. `<ctype.h>` menerima `unsigned char`; `<string.h>` tidak memeriksa apa pun, jadi beri setiap salinan ukuran dan uji setiap pencarian terhadap `NULL`.

4. `%[flags][width][.precision]type` (Persamaan (9.2)) merapikan setiap kolom; uang tetap bilangan bulat.

5. Baca satu baris utuh dengan `fgets`, tangani baris yang terlalu panjang, buang akhir baris, lalu potong menjadi kata.

Kasir v0.7 menyatukan kelimanya, dalam C dan Python dengan keluaran yang sama. Buku: Subbab 9.13.

Pertemuan Berikutnya

Bab 10 — Struktur, Enum, dan Linked List Pertama. Satu pesanan, satu nama: `struct`, `enum`, `typedef`, `malloc`, dan `free`.

Tiga larik sejajar di v0.7 menjadi satu struktur, dan buku pesanan tumbuh satu pesanan demi satu pesanan.

Sebelum itu: kerjakan Soal Latihan 9.1–9.4, terutama kolom telepon pada Soal 9.4, dan catat baris mana yang tidak diperlukan lagi bila memakai `struct`.

Program `word_stats` di bagian penyelesaian adalah pemanasan yang baik. Buku: Subbab 9.14–9.15.

Kuliah 10 — Struktur, Enum, dan Linked List Pertama

Mata kuliah: Pemrograman Dasar (EF234101)

Buku: Irfan Subakti, *Pemrograman Dasar dengan C, C++, dan Python*, Bab 10

Dosen: Irfan Subakti — yifana@gmail.com

Hari ini sebuah pesanan menjadi satu kesatuan dengan satu nama, memori diminta selagi program berjalan, dan pesanan-pesanan di toko dirangkai menjadi linked list. Buku: Bab 10, Subbab 10.1–10.12.

Cakupan Pertemuan Ini

1. **Satu rekaman, satu nama** — mendeklarasikan `struct`, operator titik, inisialisasi, tata letak di memori dengan *padding*.
2. **Struktur dan fungsi** — lewat nilai dan lewat pointer, operator `->`, larik struktur.
3. **`enum` dan `typedef`** — nama sebagai ganti angka, dan tabel layanan yang diindeks dengan nama.
4. **Memori sesuai permintaan** — `malloc`, `free`, dan tiga aturannya.
5. ***Linked list*, dan kasir v0.8** — menyusuri, menambah, menghapus, membebaskan; pesanan yang bergerak dari menunggu ke dicuci ke siap.

Setiap program ada dalam C dan Python dengan keluaran yang sama, dan versi C bersih di bawah AddressSanitizer. Buku: Bab 10.

Posisi Kita: Tiga Larik yang Harus Sepakat

Kasir v0.7 menyimpan pesanan dalam tiga **larik sejajar**: `names[i]`, `service_of[i]`, `tenths_of[i]`. Hanya indeks `i` yang mengikat ketiganya.

Perulangan pengurutan yang ditulis tergesa-gesa menukar nama tetapi tidak menukar berat: nama Andi bersebelahan dengan tagihan Budi. Tidak ada yang jatuh; tabelnya hanya salah.

Ukuran larik tetap 50: ruang terbuang di hari Selasa, pelanggan ke-51 ditolak sesudah libur panjang.

Pesanan yang keluar dari tengah berarti menggeser setiap pesanan sesudahnya, di ketiga larik.

Jawaban C: struktur untuk masalah pertama; memori yang diminta satu pesanan demi satu pesanan dan dirangkai dengan pointer, yaitu linked list, untuk masalah kedua dan ketiga. Buku: Subbab 10.1.

Mendeklarasikan Struktur

Kode 10.1 Deklarasi `struct Order`: empat anggota dengan tipe berbeda di bawah satu nama.

```
struct Order {
    int id;
    char customer[22];
    int tenths;
    long long amount;
};
```

Kode 10.1 mendeklarasikan sebuah *tipe*; belum ada variabel. `struct Order a;` membuatnya, dan `a.id`, `a.customer`, `a.amount` mencapai anggota-anggotanya.

Inisialisasi berurutan, `{2, "Budi Santoso", 20, 20000}`, atau dengan nama, `{.customer = "Citra", .id = 3}`; anggota yang tidak diberi nilai menjadi nol.

Titik koma sesudah kurung kurawal penutup wajib ada; bila terlupa, kompilator melaporkan galat membingungkan di baris berikutnya. Buku: Subbab 10.2.1.

Apa Itu Struktur

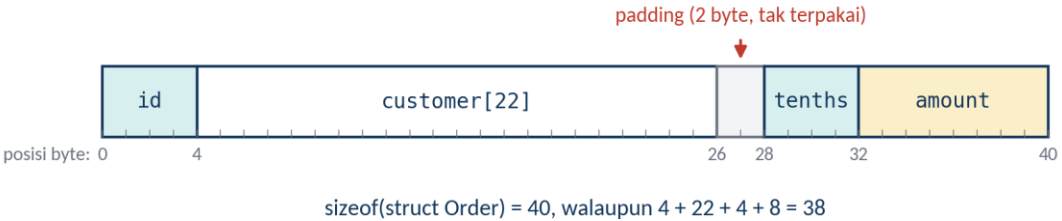
DEFINISI. *Struktur* (`struct`) adalah tipe yang terdiri atas *anggota* bernama yang tipenya boleh berbeda-beda. Variabel bertipe struktur menyimpan satu nilai untuk setiap anggota, tersimpan bersama dalam satu blok memori. Anggota `m` dari variabel struktur `s` ditulis `s.m`; bila `p` adalah pointer ke struktur, anggotanya adalah `(*p).m`, yang di C boleh ditulis `p->m`.

Penugasan `b = a;` menyalin seluruh struktur, termasuk larik di dalamnya.

Tidak ada `==` untuk struktur: bandingkan anggota demi anggota.

Di Python rekaman yang sama adalah data class, `@dataclass class Order`, yang objek-objeknya selalu ditangani lewat referensi. Buku: Subbab 10.2.1 dan 10.2.3.

Struktur di Memori



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.

***Gambar 10.1**: jumlah anggotanya 38 byte, tetapi `sizeof(struct Order)` bernilai 40. Buku: Subbab 10.2.2.*

Contoh Terurai 1: Di Mana Setiap Anggota Dimulai

$$o' = a \cdot \left\lceil \frac{o}{a} \right\rceil \tag{10.1}$$

Persamaan (10.1): bila byte bebas berikutnya di o dan anggota berikutnya berperataan a, anggota itu dimulai di kelipatan a pertama yang tidak lebih kecil dari o.

`id` — posisi 0, byte 0–3. `customer` — 22 byte, posisi 4–25.

`tenths` — $o = 26$, $a = 4$: $o' = 4 \cdot \lceil 6.5 \rceil = 28$. Dua byte *padding*.

`amount` — $o = 32$, $a = 8$: $o' = 32$, tanpa *padding*. Berakhir di byte 39; ukurannya, 40, kelipatan 8.

Ukuran keseluruhan dibulatkan ke atas menjadi kelipatan perataan terbesar, sehingga setiap elemen larik struktur juga rata. Ukur dengan `sizeof` dan `offsetof`; jangan menebak. Buku: Subbab 10.2.2.

Rekaman yang Sama di Python

`@dataclass` menuliskan konstruktor, pencetakan, dan `==` sebuah kelas rekaman untuk kita: `Order(1, "Andi Pratama", 35, 24500)`.

`b = a` di Python **tidak** menyalin: kedua nama merujuk satu objek. Salinan dibuat dengan `dataclasses.replace(a)` atau `copy.copy(a)`.

`==` pada *data class* membandingkan medan demi medan, hal yang di C harus kita tulis sendiri.

`ctypes.Structure` menggambarkan tata letak C, sehingga `order_struct.py` dapat mencetak ukuran dan posisi yang sama, 40 byte dengan `tenths` di posisi 28.

Perbedaan dalam menyalin inilah yang perlu diingat: di C, penugasan menyalin struktur; di Python, penugasan berbagi objek yang sama. Buku: Subbab 10.2.3.

Struktur dan Fungsi

Lewat nilai — fungsi menerima salinan; struktur milik pemanggil tidak pernah berubah. Struktur 1.000 byte berarti 1.000 byte disalin setiap panggilan.

Lewat pointer — fungsi menerima 8 byte, berapa pun ukurannya, dan dapat mengubah struktur pemanggil lewat pointer itu.

`o->amount` berarti persis `(*o).amount`; `*o.amount` berarti `*(o.amount)` dan tidak dapat dikompilasi.

Cara yang lazim — `const struct Order *o` untuk membaca: tanpa salinan, dan kompilator menegakkan janji untuk tidak mengubah.

`order_functions.c` memakai `struct Order day[4]`, larik struktur: `day[i]` adalah seluruh pesanan, dan `qsort` memindahkan setiap pesanan sebagai satu kesatuan, sehingga kekacauan di Subbab 10.1 tidak mungkin terjadi. Buku: Subbab 10.3.

`enum` dan `typedef`

Kode 10.2 Enumerasi layanan: `FOLD` bernilai 0, `WASH_IRON` 1, `IRON_ONLY` 2, dan `N_SERVICES` menghitung banyaknya.

```
typedef enum { FOLD, WASH_IRON, IRON_ONLY, N_SERVICES } Service;
```

Kode 10.2: `if (service == WASH_IRON)` lebih mudah dibaca daripada `if (service == 2)`; `N_SERVICES` menentukan ukuran tabel dan batas perulangan.

`typedef` memberi tipe nama kedua yang lebih pendek; ia tidak membuat tipe baru.

Inisialisasi bernama untuk larik — `[WASH_IRON] = {"wash-iron", 10000, 48}` menaruh entri itu di indeks 1, bagaimanapun urutan barisnya ditulis.

> **PERINGATAN:** Variabel `enum` tetap dapat menampung `int` apa saja. Periksa nilai dari masukan atau berkas sebelum memakainya sebagai indeks.

`service_enum.c` membangun tabel layanan dengan cara ini, dan enumerasi kedua, `Status`, menamai perjalanan sebuah pesanan; Python memakai `IntEnum` dan dictionary. Buku: Subbab 10.4.

Memori sesuai Permintaan: `malloc` dan `free`

Variabel global hidup sepanjang program; variabel lokal hidup selama fungsinya berjalan (di *stack*). Keduanya tidak memungkinkan program memutuskan *selagi berjalan* berapa banyak pesanan yang disimpan.

`malloc(n)` memesan `n` byte di ***heap*** dan mengembalikan pointer, atau `NULL`. Isinya tidak diinisialisasi.

`free(p)` mengembalikan blok itu; `free(NULL)` tidak melakukan apa-apa.

Idiomnya: `Node *n = malloc(sizeof *n);` lalu uji `n == NULL`. `sizeof *n` tetap benar bila tipe `n` diubah.

Blok itu hidup sampai `free`, tidak bergantung pada fungsi yang membuatnya: fungsi dapat membuat pesanan, mengembalikan pointer, dan pesanan itu tetap ada. Buku: Subbab 10.5.

Tiga Aturan `malloc`

1. **Periksa `NULL`.** Menulis lewat pointer nol membuat program jatuh.
2. **Bebaskan setiap blok tepat satu kali.** Tidak pernah dibebaskan berarti *memory leak*; dibebaskan dua kali merusak *heap*.
3. **Jangan memakai blok sesudah dibebaskan.** Pointer masih menyimpan alamat lama, tetapi memorinya mungkin sudah milik yang lain.

> **WAWASAN KUNCI.** *Use after free* dan *NULL pointer dereference* sama-sama ada dalam daftar MITRE 2025 tentang kelemahan perangkat lunak paling berbahaya, dan keduanya persis pelanggaran aturan-aturan ini. `-fsanitize=address` menangkap ketiganya.

Python tidak punya `malloc` dan `free`: interpreter membebaskan objek bila tidak ada lagi yang merujuknya (garbage collection, di CPython sebagian besar lewat penghitungan referensi). Buku: Subbab 10.5.

Struktur yang Menunjuk Dirinya Sendiri

Kode 10.3 Struktur yang merujuk dirinya sendiri: setiap simpul menyimpan satu pesanan dan alamat simpul berikutnya.

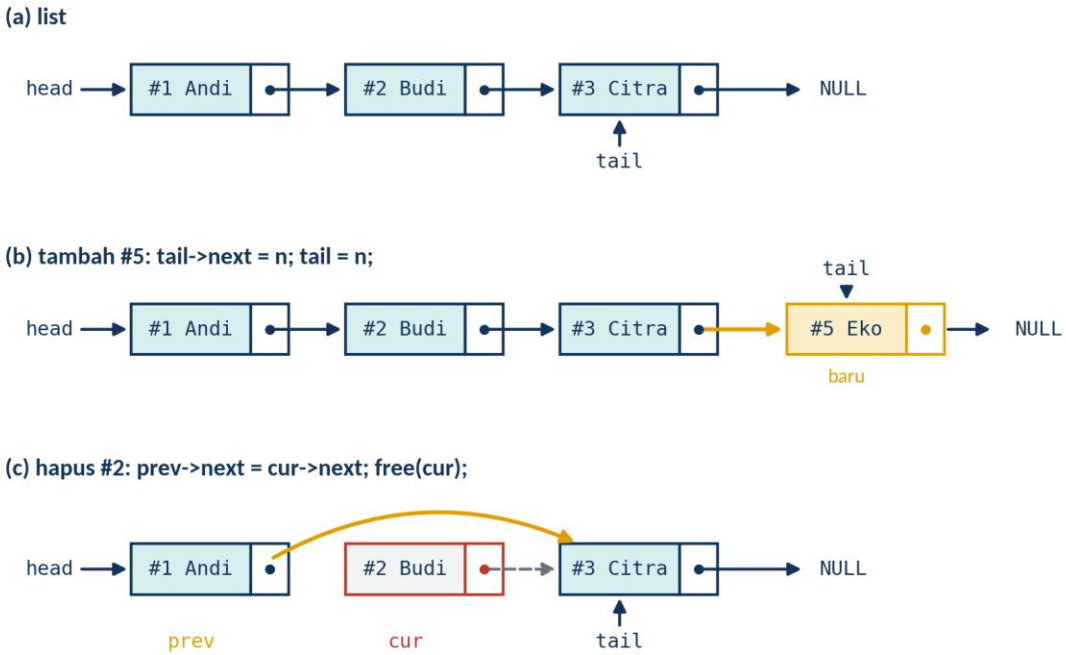
```c
typedef struct Node {
    int id;
    char customer[24];
    long amount;
    struct Node *next;
} Node;
```

Kode 10.3: mengikuti `next` dari simpul ke simpul seperti perburuan harta karun, setiap petunjuk memberi tahu di mana petunjuk berikutnya disembunyikan.

`next` simpul terakhir bernilai `NULL`; alamat simpul pertama disimpan di `head`, yang bernilai `NULL` untuk senarai kosong.

Di dalam kurung kurawal nama pendek `Node` belum ada, jadi anggotanya ditulis `struct Node *next`. Buku: Subbab 10.6.1.

Linked List Berisi Pesanan



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.

Gambar 10.2 *adalah gambar yang perlu diingat untuk setiap operasi senarai hari ini. Buku: Subbab 10.6.1.*

Larik Lawan *Linked List*

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

***Tabel 10.1**: senarai melepaskan akses langsung dan mendapatkan pertumbuhan, penyisipan murah, dan penghapusan murah. Untuk antrean laundry, tempat pesanan datang di akhir dan pergi dari mana saja, senarai menang. Buku: Subbab 10.6.1.*

Menyusuri dan Menambah

Penyusuran — `for (const Node *p = head; p != NULL; p = p->next)`; untuk senarai kosong badan perulangan berjalan nol kali, tanpa kasus khusus.

Mencetak, menghitung, menjumlah, dan mencari semuanya adalah perulangan ini dengan badan yang berbeda.

Menambah di depan — `n->next = head;` *lalu* `head = n;`. Urutan sebaliknya menghilangkan seluruh senarai.

Menambah di akhir — simpan pointer `tail`: `tail->next = n; tail = n;`; pada senarai kosong `n` sekaligus menjadi `head`.

`head`, `tail`, dan `count` disimpan bersama dalam struktur `List`, sehingga fungsi menerima seluruh senarai lewat satu pointer. Buku: Subbab 10.6.2–10.6.3.

Menghapus Simpul

Kode 10.4 Dari `order_list.c`: `remove_id`, penyusuran dua pointer dengan tiga kasusnya.

```c
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 10.4**: `prev` selalu satu langkah di belakang `cur`. Simpul pertama: `head` bergeser. Simpul terakhir: `tail` mundur. Simpul baru dibebaskan sesudah dilepas dari rangkaian. Buku: Subbab 10.6.4.*

Contoh Terurai 2: Menghapus #2 dengan Tangan

CONTOH. Senarainya Dewi (#4), Andi (#1), Budi (#2), Citra (#3). Hapus #2.

Awal — `prev` bernilai `NULL`, `cur` adalah Dewi. Id 4, bukan 2: lanjut.

Langkah 2 — `prev` adalah Dewi, `cur` adalah Andi. Id 1: lanjut.

Langkah 3 — `prev` adalah Andi, `cur` adalah Budi. Id 2: berhenti.

Lepaskan — `prev` bukan `NULL`, jadi `prev->next = cur->next`: Andi kini menunjuk Citra. `tail` adalah Citra, bukan Budi, jadi tetap.

Bebaskan — simpul Budi dibebaskan; `count` turun menjadi 3. Senarai menjadi Dewi, Andi, Citra.

Menghapus simpul yang sudah ditemukan hanya perlu satu penugasan, sepanjang apa pun senarainya; mencarinya bisa menyusuri seluruh n simpul, dan larik sesudah itu masih harus menggeser sisanya. Buku: Subbab 10.6.4.

Membebaskan Seluruh Senarai

> **PERINGATAN:** `for (p = head; p != NULL; p = p->next) free(p);` melanggar aturan 3: sesudah `free(p)`, pembaruan `p = p->next` membaca dari simpul yang sudah tidak ada.

Kode 10.5 Dari `order_list.c`: `free_all` menyimpan simpul berikutnya sebelum membebaskan.

```
Node *p = l->head;
while (p != NULL) {
    Node *next = p->next;            /* read next BEFORE free */
    free(p);
    p = next;
}
```

Kode 10.5 *adalah perulangan yang harus ditulis setiap kali; pemeriksaan kebocoran oleh program penguji memastikan tidak ada yang tertinggal. Buku: Subbab 10.6.5.*

Senarai Bekerja

```
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
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
head -> [#1 Andi] -> [#5 Eko] -> NULL   count=2
```

Baris-baris pilihan dari `order_list`: simpul pertama, tengah, dan terakhir dihapus. Menambah Eko sesudahnya menaruhnya tepat di belakang Andi, bukti bahwa `tail` sudah mundur; bila tidak, sanitizer melaporkan heap-use-after-free. Buku: Subbab 10.6.6.

Senarai yang Sama di Python

Kelas `Node` dengan `id`, `customer`, `amount`, dan `next`; `None` berperan sebagai `NULL`.

Penyusurannya perulangan yang sama: `p = head`, lalu `while p is not None: ...; p = p.next`.

`remove_id` menyimpan `prev` dan `cur` persis seperti di C, termasuk tiga kasus untuk simpul pertama, simpul tengah, dan simpul terakhir.

Tidak ada `free`: begitu tidak ada variabel yang merujuk simpul yang dihapus, Python mengambilnya kembali. Tidak ada `free_all` juga; mengeset `head` dan `tail` menjadi `None` sudah cukup.

Dalam Python sehari-hari `list` atau `collections.deque` sudah mengerjakan tugas ini; kita membangun senarai sendiri untuk melihat pointer yang diperlihatkan C. `order_list.py` mencetak persis apa yang dicetak `order_list.c`. Buku: Subbab 10.6.6.

Kasir v0.8: Satu Struktur per Pesanan

Sebuah pesanan menyimpan `id`, `customer`, `service` (sebuah `Service`), `tenths`, `amount`, `status` (sebuah `Status`), dan `next`.

Pesanan-pesanan di toko membentuk satu *linked list* di dalam struktur `Shop`, bersama nomor pesanan berikutnya dan pendapatan hari itu.

`add` — `malloc`, isi, tambahkan di akhir; tanpa batas banyaknya pesanan.

`advance <id>` — menunggu → dicuci → siap. **`pickup <id>`** — pesanan yang siap dilepas, dibayar, dan dibebaskan. **`list`**, **`quit`**.

Fungsi-fungsi teks dari Bab 9 disalin tanpa perubahan; Bab 11 memindahkannya ke berkas tersendiri. Buku: Subbab 10.7.

Perjalanan Sebuah Pesanan di Meja Kasir

```
> 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 7
  no order #7 in the shop
```

Dari jalannya program pada `data/session10.txt`; C dan Python mencetak hal yang sama. Pesanan tidak dapat diambil sebelum siap, tidak dapat maju melewati siap, dan nomor yang tidak dikenal mendapat pesan yang jelas. Buku: Subbab 10.8.

Membuatnya Lebih Baik, Memakainya secara Nyata

Jaga urutan yang berguna — `insert_sorted` menyisipkan pesanan ekspres sebelum pesanan yang lebih lambat, dengan penyusuran dua pointer.

Semua pesanan seorang pelanggan — penyusuran yang tidak berhenti di kecocokan pertama.

Jangan kehilangan satu hari — listrik padam masih menghapus semuanya; Bab 11 menyimpan setiap kali ada perubahan.

Secara nyata — bengkel sepeda motor: nomor polisi, pemilik, keluhan, enumerasi `BikeStatus`; `advance` menjadi langkah berikutnya sang mekanik.

Kebocoran kecil tidak berbahaya pada program yang dijalankan ulang setiap pagi, tetapi fatal pada program yang berjalan berbulan-bulan: periksa setiap adaptasi di bawah sanitizer. Buku: Subbab 10.9–10.10.

Kesalahan yang Sering Terjadi

Tanpa titik koma sesudah `}` struktur; `.` **pada pointer atau** `->` **pada struktur**.

Struktur besar lewat nilai tanpa sengaja; `memcmp` **pada struktur** (*padding* berbeda); **mengira ukurannya 38**.

`malloc` **tidak diperiksa**; **membebaskan lalu memakai**; **kehilangan senarai** karena menulis `head = n` lebih dulu.

Lupa `tail` saat menghapus simpul terakhir atau menambah ke senarai kosong; **memercayai** `enum` dari masukan.

Uji penghapusan simpul terakhir lalu penambahan: `tail` *yang terlupa langsung tertangkap. Buku: Subbab 10.11.*

Ringkasan: Lima Hal yang Perlu Diingat

1. `struct` mengelompokkan anggota bernama; `.` pada struktur, `->` lewat pointer; penugasan menyalin, perbandingan per anggota.
2. Anggota diratakan dengan *padding* (Persamaan (10.1), **Gambar 10.1**); `sizeof` dan `offsetof` mengatakan yang sebenarnya.
3. Kirim struktur lewat pointer `const` untuk membaca dan lewat pointer untuk mengubah; larik struktur menggantikan larik sejajar.
4. Memori `malloc` hidup sampai `free`: periksa `NULL`, bebaskan sekali, jangan pakai sesudah dibebaskan.
5. *Linked list* dengan `head` dan `tail` menambah di kedua ujung dalam satu langkah dan menghapus dengan satu penyusuran dan satu penugasan (**Tabel 10.1**, **Gambar 10.2**).

`enum` menamai kode, `typedef` menamai tipe, dan kasir v0.8 memakai semuanya, bersih di bawah AddressSanitizer. Buku: Subbab 10.12.

Pertemuan Berikutnya

Bab 11 — Pemrosesan Berkas: Data yang Bertahan Saat Listrik Padam. *Stream*, berkas teks dan biner, CSV, log, dan menyimpan tanpa kehilangan.

Kasir v0.9 memuat senarai saat mulai dan menyimpannya sesudah setiap perubahan, dan programnya dipecah menjadi beberapa berkas.

Sebelum itu: Soal Latihan 10.1–10.6, terutama ukuran `struct Item` dengan tangan dan pembalikan senarai di tempat dalam C dan Python.

Pukul empat sore listrik padam; lain kali pesanan-pesanan selamat. Buku: Subbab 10.13–10.14.

Kuliah 11 — Pemrosesan Berkas: Data yang Bertahan Saat Listrik Padam

Mata kuliah: Pemrograman Dasar (EF234101)

Buku: Irfan Subakti, *Pemrograman Dasar dengan C, C++, dan Python*, Bab 11

Dosen: Irfan Subakti — yifana@gmail.com

Hari ini kasir berhenti lupa: aliran dan berkas, mode-mode `fopen`, jebakan `feof`, berkas CSV nyata yang berisi kesalahan, log, rekaman biner, menyimpan tanpa kehilangan, dan kasir v0.9 yang dipecah menjadi modul. Buku: Bab 11, Subbab 11.1–11.14.

Cakupan Pertemuan Ini

1. **Berkas dan aliran** — `FILE *`, `fopen`, `fclose`, *buffering*, dan enam mode.
2. **Berkas teks dan jebakan `feof`** — `fprintf`, `fscanf`, dan mengapa perulangan membaca menguji hasil pembacaan itu sendiri.
3. **Data nyata** — berkas CSV dibaca baris demi baris, baris rusak dilaporkan beserta nomornya.
4. **Log dan berkas biner** — mode tambah; rekaman berukuran tetap yang dicapai dengan `fseek`.
5. **Menyimpan tanpa kehilangan, dan kasir v0.9** — tulis, tutup, ganti nama; *header* dan *include guard*; listrik padam yang disimulasikan.

Seperti biasa, setiap program ada dalam C dan Python dengan keluaran yang sama. Buku: Bab 11.

Posisi Kita: Pukul Empat Sore

Kasir v0.8 punya nama yang rapi, pesanan yang bergerak dari *menunggu* ke *dicuci* ke *siap*, dan senarai yang tumbuh sepanjang hari.

Pukul empat sore pada hari Sabtu yang ramai ia menyimpan tiga puluh pesanan. Lalu listrik padam, dan ketiga puluh pesanan itu hilang.

RAM hanya menyimpan isinya selama ada listrik; variabel, larik, dan *linked list* semuanya tinggal di sana.

Data yang harus hidup lebih lama daripada programnya disimpan di **media penyimpanan** (SSD, *hard disk*, kartu memori) melalui **berkas**.

Menulis ke berkas itu mudah; menulis dengan aman perlu pemikiran lebih. Menyimpan hanya saat `quit` tetap kehilangan satu hari, dan menulis ulang sesudah setiap pesanan dapat meninggalkan berkas setengah jadi bila listrik padam di tengah proses menulis. Buku: Subbab 11.1.

Aliran dan Pointer `FILE`

> **DEFINISI.** *Berkas* adalah deretan byte bernama yang disimpan sistem operasi di media 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 aliran dinyatakan dengan pointer bertipe `FILE *`, diperoleh dari `fopen` dan dilepas dengan `fclose`.

`fopen(name, mode)` mengembalikan `NULL` bila berkas tidak dapat dibuka; laporkan dengan `perror(name)`.

Setiap program dimulai dengan tiga aliran yang sudah terbuka: `stdin`, `stdout`, `stderr`.

Aliran memakai ***buffer***: keluaran masuk ke berkas dalam blok-blok, dan blok terakhir baru masuk pada `fflush` atau `fclose`.

Program tidak pernah melihat ke dalam objek `FILE`; ia hanya meneruskan pointernya, seperti menunjukkan tiket tanpa membaca huruf-huruf kecilnya. Buku: Subbab 11.2.1.

Mode-mode `fopen`

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

Tabel 11.1: mode yang berbahaya adalah `"w"`, yang mengosongkan berkas yang sudah ada tepat saat dibuka. `open` di Python memakai huruf yang sama, melempar `FileNotFoundError` alih-alih mengembalikan `NULL`, dan di dalam blok `with` menutup berkas secara otomatis. Buku: Subbab 11.2.2.

Menulis dan Membaca Berkas Teks

Menulis — `fprintf(fp, "%d %ld %s\n", id, amount, name)`: persis `printf`, dengan aliran di depannya.

Membaca — `fscanf(fp, "%d %ld %31[^\n]", &id, &amount, name)`; *scan set* membaca sisa baris, spasi ikut, paling banyak 31 karakter.

Letakkan medan yang mungkin berisi spasi di **akhir** baris.

Perulangannya — `while (fscanf(...) == 3)`: lanjut tepat selama satu rekaman utuh berhasil dibaca.

`write_read_text.c` menulis tiga pesanan ke `orders.txt` dan membacanya kembali, dengan total 68500. Buku: Subbab 11.3.

Jebakan `feof`

Kode 11.1 Dari `feof_trap.c`: perulangan `while (!feof(fp))` berjalan satu kali terlalu banyak.

```
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);
}
```

Kode 11.1 tampak benar bagi hampir setiap pemula. Tebak keluarannya untuk berkas berisi tiga pesanan sebelum membuka halaman berikutnya. Buku: Subbab 11.4.

Contoh Terurai 1: Empat Putaran untuk Tiga Pesanan

```
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)
```

Sesudah putaran 3 aliran berada tepat sebelum akhir berkas, tetapi belum ada pembacaan yang *gagal*, jadi `feof` masih bernilai salah.

`fscanf` di putaran 4 gagal dan mengembalikan -1 (`EOF`); variabel masih berisi nilai Citra, dan 24000 miliknya dijumlahkan dua kali.

Baru sekarang `feof` bernilai benar. Perbaikannya: ulangi berdasarkan hasil pembacaan, `fscanf`, `fgets`, `fgetc`, atau `fread`.

`feof` tidak melihat ke depan; ia melaporkan akhir berkas hanya sesudah sebuah pembacaan mencoba melewatinya. Ia berguna sesudah perulangan, untuk membedakan akhir berkas dari galat pembacaan. Buku: Subbab 11.4.

Data Nyata: Berkas CSV

CSV, *comma-separated values*: satu rekaman per baris, medan dipisahkan koma, baris kepala yang menamai kolom. Setiap program lembar kerja dapat membaca dan menulisnya.

`data/week.csv` berisi 13 pesanan yang diketik pemilik, dan beberapa kesalahan: layanan yang tidak dikenal, medan yang hilang, medan berlebih, baris kosong, berat yang bukan bilangan.

Pembaca yang baik tidak jatuh, tidak berhenti di kesalahan pertama, dan tidak diam-diam menerima isian ngawur. Ia melaporkan setiap baris rusak **beserta nomor barisnya** dan menyimpan yang baik.

Perlakukan setiap berkas sebagai masukan dari orang asing. Buku: Subbab 11.5.

Membaca Satu Baris dalam Tiga Langkah

1. `fgets` membaca seluruh baris, sampai 255 karakter.
2. `line[strcspn(line, "\r\n")] = '\0';` memotong `"\n"` atau `"\r\n"` dari Windows.
3. `split_csv` menulis penutup di atas setiap koma dan mengingat di mana setiap medan dimulai; medan kosong **dipertahankan**, dan banyaknya medan dihitung.

Setiap medan lalu diperiksa: layanan yang dikenal, berat yang diterima utuh oleh `parse_tenths` (angka, paling banyak satu desimal) dan berada dalam rentang.

> **PERINGATAN:** `strtok` melompati medan kosong, sehingga nilai yang hilang menggeser setiap medan sesudahnya satu tempat ke kiri dan kesalahannya tidak terlihat. Pakai pemecah yang mempertahankan medan kosong.

`csv_report.c` menjumlahkan per layanan dan per hari; `csv_report.py` melakukan hal yang sama dengan `line.split(",")`. Buku: Subbab 11.5.

Contoh Terurai 2: Isi Laporannya

```
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
```

Empat pesan untuk tepat empat baris yang rusak; baris kosong 11 dilewati begitu saja. Medan berlebih di baris 13 adalah `4,5`, berat yang diketik dengan koma desimal, kebiasaan Indonesia, dan itulah sebabnya menghitung medan penting. Berkas yang tidak ada mencetak `No such file or directory` di `stderr` dan keluar dengan status 1. Buku: Subbab 11.5.

Mode Tambah: Cara Log Tumbuh

Mode `"a"` mengirim setiap operasi tulis ke akhir dan tidak pernah mengosongkan berkas: persis yang dibutuhkan **log**. Bila ada yang salah, pertanyaan pertama adalah “apa yang terjadi, dan kapan?”; log menjawabnya.

`activity_log.c` menghitung baris yang sudah ada (berkas yang belum ada berarti nol baris), menambahkan satu entri bernomor, dan mencetak seluruh log; setiap kali dijalankan bertambah satu baris.

Setiap server, mesin kasir, dan bank menyimpan log. Versi Python membuka dengan `"a"` dengan cara yang sama. Buku: Subbab 11.6.

Berkas Biner dan Akses Acak

Berkas teks harus dibaca dari depan: baris ke-5.000 dimulai di mana pun 4.999 karakter baris baru sebelumnya berakhir.

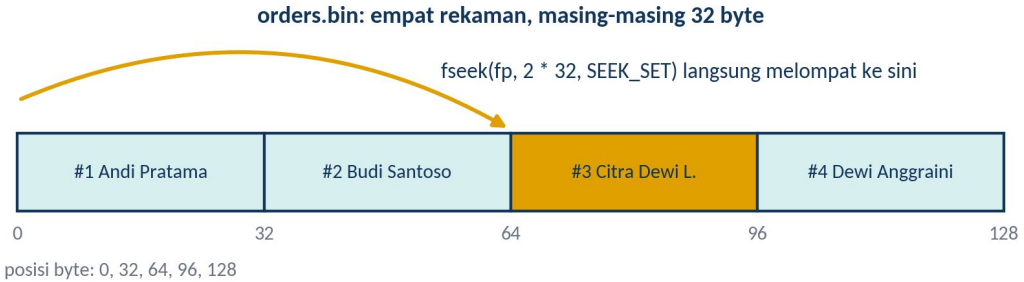
Berkas **biner** menyimpan byte data apa adanya seperti di memori: `fwrite(ptr, size, count, fp)` dan `fread(...)` mengembalikan banyaknya objek utuh yang dipindahkan.

$$\text{offset}(i) = i \cdot s \tag{11.1}$$

Dengan rekaman s byte, rekaman i dimulai di byte Persamaan (11.1), dan `fseek(fp, offset, SEEK_SET)` langsung ke sana tanpa membaca apa pun di jalan: **akses acak**.

`fseek(fp, 0, SEEK_END)` *diikuti* `ftell(fp)` *memberi ukuran berkas dalam byte. Buku: Subbab 11.7.*

Empat Rekaman, Satu Lompatan



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.

Gambar 11.1: `binary_records.c` *menulis empat rekaman, melompat ke rekaman 2, mengubahnya di tempat dengan* `r+b`*, dan membacanya kembali; Python melakukan hal yang sama dengan modul* `struct`*. Buku: Subbab 11.7.*

Harga Berkas Biner

Berkas biner terikat pada satu platform: *padding* struktur dan urutan byte berbeda antara kompilator dan mesin.

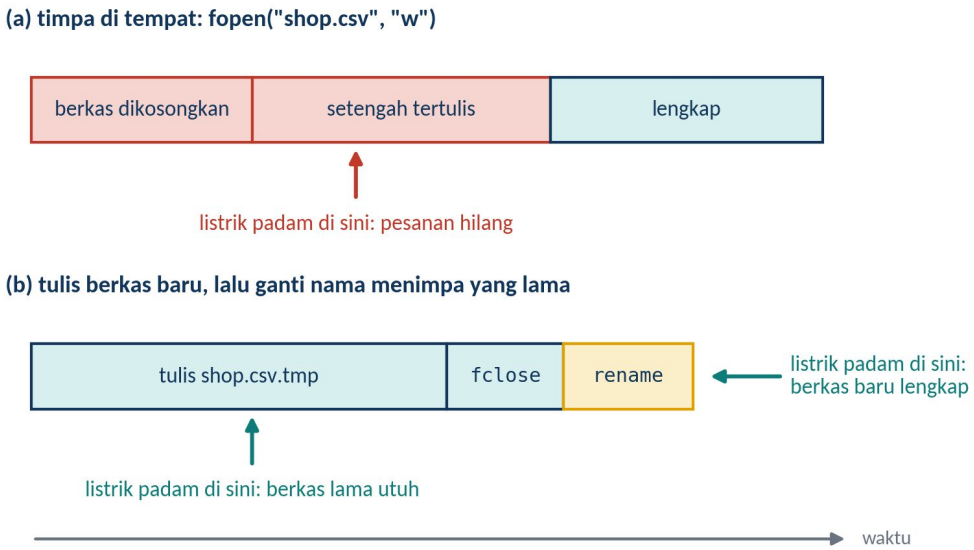
Tidak ada yang dapat membacanya di penyunting teks atau membukanya di lembar kerja.

Pegangan praktis — teks untuk data yang dibaca orang atau berpindah antarprogram; biner hanya untuk rekaman berukuran tetap di satu mesin yang harus dicapai dengan cepat.

> **PERINGATAN:** Di Windows berkas biner harus dibuka dengan `b` dalam modenya, atau setiap byte 10 yang ditulis menjadi dua byte 13 10 dan rekaman-rekamannya bergeser.

Kasir memilih teks: `shop.csv` *dapat dibuka di lembar kerja mana pun bila ada yang tampak salah. Buku: Subbab 11.7.*

Menyimpan Tanpa Kehilangan



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.

Gambar 11.2: pada (a), `"w"` langsung mengosongkan berkas, sehingga selama beberapa milidetik satu-satunya salinan kosong atau setengah tertulis. Buku: Subbab 11.8.

Tulis, Tutup, Ganti Nama

1. Tulis semua pesanan ke berkas *baru*, `shop.csv.tmp`.

2. Tutup dan periksa: `ferror(fp)` untuk operasi tulis yang gagal, dan hasil `fclose`, yang melaporkan byte terakhir di *buffer* (misalnya disk penuh).

3. Hanya bila semuanya berhasil, `rename` berkas baru itu menimpa `shop.csv`.

Listrik padam saat langkah 1 atau 2 membiarkan berkas lama utuh. Di Linux dan macOS, POSIX menjamin `rename` mengganti target secara **atomik**.

Cara ini pun masih memercayai cache sistem operasi; basis data juga memanggil `fsync` (POSIX) atau `FlushFileBuffers` (Windows) dan menunggu disk. Buku: Subbab 11.8.

`save_shop` dalam C

Kode 11.2 Dari `storage.c`: bagian akhir `save_shop`, yang memeriksa operasi tulis dan mengganti nama berkas sementara.

```c
    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;
```

Kode 11.2 adalah Subbab 11.8 dalam bentuk kode, dengan percobaan kedua untuk Windows. `os.replace` di Python mengganti nama menimpa berkas yang ada di semua sistem. Buku: Subbab 11.9.2.

Satu Program, Beberapa Berkas

`shop.h` / `shop.c` — pesanan, fungsi teks, *linked list*, dan perintah-perintah: v0.8 tanpa `main`.

`storage.h` / `storage.c` — memuat CSV, memeriksa setiap medan di setiap baris, dan menyimpan dengan aman.

`cashier_v9.c` — `main` dan perulangan perintah, yang memanggil `save_shop` sesudah setiap perubahan.

Header hanya berisi deklarasi; `#include "shop.h"` dengan tanda kutip untuk *header* buatan sendiri; `extern` mendeklarasikan data yang didefinisikan di berkas lain.

Include guard, `#ifndef SHOP_H` / `#define SHOP_H` / `#endif`, membuat *header* aman di-*include* dua kali.

Kompilasi semua sumber bersama: `gcc -std=c17 -O2 -Wall -Wextra -o cashier_v9 cashier_v9.c shop.c storage.c`. *Buku: Subbab 11.9.1.*

Listrik Padam yang Disimulasikan

```
$ step 1
Profitina Laundry cashier v0.9 - 0 order(s) loaded from shop.csv
> 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
```

Baris-baris pilihan. Sesi pertama dimatikan tanpa pamit dan tanpa penyimpanan terakhir, tetapi sesi kedua menemukan ketiga pesanan beserta statusnya. Itulah seluruh bab dalam satu layar. Buku: Subbab 11.10.

`shop.csv` yang Rusak

```
  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
```

Baris 4 mengulang id 3; baris 5 berisi layanan tak dikenal `dry-clean`; baris 6 tidak punya nama.

Pesanan yang baik, #1, #3, dan #6, dimuat; kasir tetap bekerja.

Pemuat yang menolak seluruh berkas karena satu baris buruk akan meninggalkan Mbak Rina tanpa apa-apa. Buku: Subbab 11.10.

Membuatnya Lebih Baik, Memakainya secara Nyata

Simpan hari kemarin — tambahkan satu baris per pengambilan ke `pickups.log` dengan mode `"a"`.

Buat cadangan — salinan di *flashdisk* atau penyimpanan awan setiap sore. **Ketahui kapan berkas tidak cukup** — banyak pengguna atau banyak rekaman memerlukan basis data.

Secara nyata — pesanan terbuka seorang penjahit dalam CSV; log panen kebun buah dan durian dengan mode tambah, satu baris per peti.

Berkas berisi nama dan nomor telepon adalah data pribadi (UU PDP): beri kata sandi pada komputernya, dan hapus yang tidak diperlukan lagi.

Periksa setiap `fopen`, baca berdasarkan hasil pembacaan, perlakukan berkas sebagai masukan dari orang asing, simpan lewat berkas sementara, utamakan teks. Buku: Subbab 11.11–11.12.

Kesalahan yang Sering Terjadi

`fopen` tidak diperiksa; **`"w"` padahal maksudnya `"r"` atau `"a"`**; **`while (!feof(fp))`**.

Hasil `fgetc` disimpan di `char` (`EOF` tidak muat); **nama dibaca dengan `%s`**; **`'\r'` dari berkas Windows**.

CSV dipecah dengan `strtok`; **`fclose` terlupa atau hasilnya diabaikan**.

Berkas biner dipakai bersama antarmesin; ***header* tanpa *include guard***.

Masing-masing punya satu baris dalam program-program di buku yang menghindarinya. Buku: Subbab 11.13.

Ringkasan: Lima Hal yang Perlu Diingat

1. RAM lupa; berkas tidak. `fopen` dapat mengembalikan `NULL`; `fclose` mengosongkan *buffer*. `"w"` langsung mengosongkan berkas (Tabel 11.1).

2. Perulangan membaca menguji hasil setiap pembacaan, tidak pernah `feof`.

3. Berkas nyata dibaca baris demi baris, dipecah menjadi medan yang boleh kosong, diperiksa medan demi medan; baris rusak dilaporkan beserta nomornya.

4. Log tumbuh dengan mode tambah; rekaman biner dicapai dengan `fseek` (Persamaan (11.1), Gambar 11.1) tetapi terikat pada satu platform.

5. Simpan dengan menulis berkas baru, memeriksa `ferror` dan `fclose`, lalu mengganti nama (Gambar 11.2).

Kasir v0.9, dalam modul-modul dengan header yang dijaga, selamat dari listrik padam dalam C dan Python dengan keluaran yang sama. Buku: Subbab 11.14.

Pertemuan Berikutnya

Bab 12 — Evaluasi dan Pembahasan III: Senarai, Pohon, dan Seekor Kuda Catur. Tiga soal utuh yang diselesaikan langkah demi langkah.

Mengurutkan *linked list*, menumbuhkan pohon biner dari berkas perintah, dan lintasan terpendek kuda catur dengan *breadth-first search*.

Sebelum itu: Soal Latihan 11.1–11.6, terutama `find_customer`, yang mencari di berkas CSV baris demi baris.

Lain kali kita memakai semua yang ada di Bab 9 sampai 11 pada soal-soal yang lebih besar daripada satu gagasan. Buku: Subbab 11.15–11.16.

Kuliah 12 — Evaluasi dan Pembahasan III: Senarai, Pohon, dan Seekor Kuda Catur

Mata kuliah: Pemrograman Dasar (EF234101)

Buku: Irfan Subakti, *Pemrograman Dasar dengan C, C++, dan Python*, Bab 12

Dosen: Irfan Subakti — yifana@gmail.com

Hari ini kita menelusuri tiga soal struktur data yang utuh: bagaimana masing-masing dipahami dengan tangan, dirancang di atas kertas, diprogram, diuji, dan bagaimana sebuah penyelesaian keliru yang menggoda gagal. Buku: Bab 12, Subbab 12.1–12.9.

Cakupan Pertemuan Ini

1. **Metodenya** — delapan langkah untuk soal struktur data, dari menyatakan ulang soal sampai memeriksa diri dengan variasi.
2. **Soal 12.1** — mengurutkan *singly linked list* dengan *bubble sort*, lalu menyisipkan di akhir.
3. **Soal 12.2** — menumbuhkan pohon biner dari berkas perintah navigasi.
4. **Soal 12.3** — lintasan terpendek kuda catur dengan *breadth-first search* dan antrean berupa *linked list*.
5. **Berdampingan** — gagasan tambahan yang diperlukan setiap soal, dan uji yang menangkap setiap penyelesaian keliru.

Setiap program ada dalam C dan Python dengan keluaran yang sama, dan setiap program C dijalankan di bawah sanitizer. Buku: Bab 12.

Posisi Kita

Bab 9 mengajarkan teks, Bab 10 struktur dan *linked list*, Bab 11 berkas.

Ketiga soal ini satu tingkat lebih tinggi: seluruh kesulitannya terletak pada menjaga sekumpulan pointer tetap benar selagi strukturnya berubah.

Ketiganya saling lepas dan cocok untuk tim kecil: setiap orang memimpin satu soal, tetapi semua menggambar ketiganya dan meninjau program teman.

Program yang hanya pernah Anda baca dan tidak pernah Anda pertanyakan bukanlah program yang Anda pahami. Buku: Subbab 12.1.

Delapan Langkah

1. Nyatakan ulang dengan kata-kata sendiri: masukan, keluaran, setiap keputusan yang dibiarkan terbuka. **2. Kerjakan contoh kecil dengan tangan**, satu gambar per langkah.

3. Rancang di atas kertas — *pseudocode* dan biaya. **4. Programkan** satu fungsi demi satu fungsi.

5. Uji dengan kasus yang disiapkan — kosong, satu elemen, terurut, terbalik, kembar, masukan terbesar, masukan salah; ketahui jawabannya *sebelum* menjalankan.

6. Jalankan di bawah *sanitizer*. 7. Bongkar penyelesaian keliru yang menggoda. 8. Periksa diri dengan variasi.

Untuk soal pointer, langkah 2 bukan kemewahan: bila Anda tidak dapat menggambar yang terjadi, Anda tidak dapat memprogramnya. `python3 tests/run_tests.py` *mengompilasi, menjalankan kedua bahasa, dan membandingkannya dengan keluaran yang diharapkan. Buku: Subbab 12.2.*

Soal 12.1: Urutkan, Lalu Sisipkan di Akhir

> **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.

Keputusan kita: bilangan-bilangan dari baris masukan pertama, bilangan baru dari baris kedua; program harus berjalan untuk senarai apa pun, termasuk yang kosong dan yang berisi satu elemen.

"Sisipkan di akhir" berarti di akhir, bukan di tempat terurutnya. Buku: Subbab 12.3.1.

Contoh Terurai 1: *Bubble Sort* dengan Tangan

```
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
```

Bubble sort hanya membandingkan **tetangga**, dan tetangga sebuah simpul berjarak satu `->next`. Kita menukar `data`, bukan simpulnya: tiga penugasan, tanpa mengubah pointer.

Garis tegak menandai ekor yang terurut: sesudah lintasan ke-k, k bilangan terbesar sudah di tempatnya. Lintasan tanpa pertukaran mengakhiri pengurutan.

Empat lintasan, $5+3+2=10$ *pertukaran. Lalu 23 disambungkan sesudah 42. Buku: Subbab 12.3.2.*

Sepuluh Bukan Kebetulan

$$\text{swaps of bubble sort} = \#\left\{(i, j) : i < j \text{ and } a_i > a_j\right\} \tag{12.1}$$

Inversi adalah pasangan posisi $i<j$ dengan $a_i>a_j$. Setiap pertukaran tetangga menghapus tepat satu inversi dan tidak menciptakan inversi baru.

Untuk 29, 4, 17, 4, 42, 8, 15: 29 memberi 5, 17 memberi 3, 42 memberi 2, sisanya nol. Totalnya 10.

Persamaan (12.1) memberi banyak pertukaran yang diharapkan untuk setiap uji *sebelumnya*, persis yang diminta langkah 5.

Program mencetak banyak lintasan dan pertukarannya, sehingga ramalan itu dapat diperiksa. Buku: Subbab 12.3.2.

Bubble Sort pada Senarai, di Atas Kertas

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
```

Pseudokode 12.1, baris 6 yang halus: pada lintasan 1, `last` bernilai `NIL`, sehingga perulangan berhenti pada simpul terakhir dan tidak pernah membaca `p.next.data` lewat `NIL`. Biaya: paling banyak $\frac{n(n-1)}{2}$ *perbandingan,* $O(n^2)$*; satu lintasan bila sudah terurut; penyisipan di akhir* $O(n)$*. Buku: Subbab 12.3.3.*

Menjalankan Soal 12.1

```
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, sesuai ramalan penelusuran tangan dan Persamaan (12.1).

Uji tepi: senarai yang sudah terurut (satu lintasan, nol pertukaran), senarai terbalik, satu bilangan, senarai kosong.

`list_sort.c` dan `list_sort.py` *mencetak baris yang sama pada kelima uji. Buku: Subbab 12.3.6.*

Penyelesaian Keliru yang Menggoda: Satu Lintasan

```
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 hampir terurut satu lintasan kebetulan cukup, dan pemrogramnya percaya programnya benar.

Satu lintasan hanya menjamin bilangan *terbesar* sampai di akhir. **Selalu uji dengan senarai terbalik.**

Kesalahan klasik kedua: `p = head; while (p->next != NULL)` jatuh pada senarai kosong; uji `list_sort_empty` dan *sanitizer* menangkapnya.

Dari `wrong_list_sort.c`. Periksa diri: untuk urutan menurun hanya `>` di baris 7 yang berubah; menyisipkan di depan memakai `push_front` dari Bab 10. Buku: Subbab 12.3.7–12.3.8.

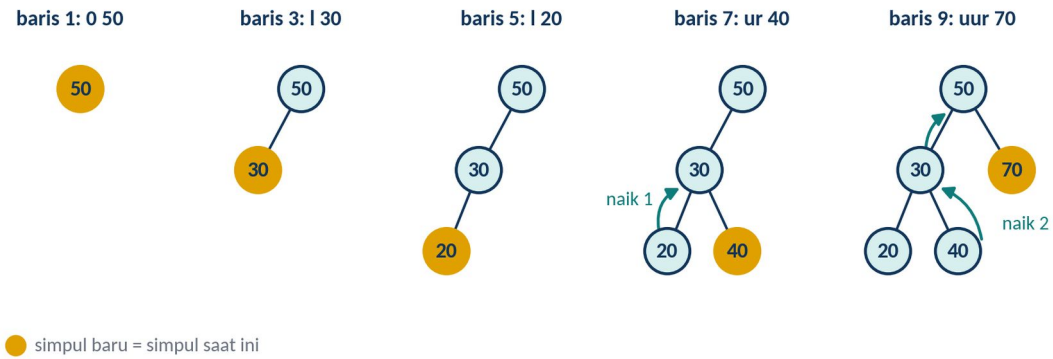
Soal 12.2: Pohon yang Ditumbuhkan dari Berkas

> **SOAL 12.2.** Sebuah berkas teks menggambarkan pohon biner dalam pasangan baris: perintah, lalu bilangan bulat. `0` membuat akar. `l` menyisipkan simpul baru sebagai anak *kiri* simpul saat ini. Perintah `u…ur` lebih dulu naik satu tingkat ke arah akar untuk setiap `u`, lalu menyisipkan simpul baru sebagai anak *kanan*. Simpul baru menjadi simpul saat ini.

Pohon biner: setiap simpul punya paling banyak dua anak, dan setiap simpul kecuali akar punya tepat satu orang tua. Simpul senarai punya satu `next`; simpul pohon punya dua.

Dengan kata-kata kita: resep untuk berjalan di dalam gedung dan menaruh kotak. Keputusan kita: nama berkas di baris perintah, baris kosong diabaikan, `r` saja berarti anak kanan simpul saat ini, dan galat disebut beserta barisnya. Buku: Subbab 12.4.1.

Contoh Terurai 2: Mengikuti `tree5.txt`



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.

Gambar 12.1: `0 50` membuat akar. `l 30`, `l 20` turun ke kiri. `ur 40`: dari 20 naik satu tingkat ke 30; 40 menjadi anak kanan 30. `uur 70`: dari 40 naik ke 30, lalu ke 50; 70 menjadi anak kanan 50. Buku: Subbab 12.4.2.

Pengamatan Kunci: Jalan ke Atas

Pendakian dimulai dari simpul **saat ini**, yaitu simpul yang baru saja disisipkan.

Simpul pohon biasanya hanya mengenal anak-anaknya, jadi setiap simpul juga menyimpan pointer `parent`, diisi saat dibuat. “Naik satu tingkat” adalah `at = at->parent`.

Orang tua akar bernilai `NULL`: begitulah pendakian melewati akar terdeteksi.

Untuk memperlihatkan bentuknya: ***preorder*** 50 30 20 40 70, ***inorder*** 20 30 40 50 70, ***postorder*** 20 40 30 70 50, dan cetakan menyamping.

Ketiga urutan itu rekursif (Bab 5). Dua pohon yang berbeda dapat berbagi satu urutan, karena itu program mencetak ketiganya beserta gambarnya. Buku: Subbab 12.4.2.

Membangun Pohon, di Atas Kertas

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
```

Pseudokode 12.2: setiap simpul dibuat sekali; pendakian k tingkat berbiaya k langkah; penelusurannya $O(n)$. Pembebasan memori berupa penelusuran postorder: anak lebih dulu, lalu simpulnya. Buku: Subbab 12.4.3.

Menjalankan Soal 12.2

```
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
```

Satu baris jejak per perintah: baris pertama tempat program dan kertas berbeda menunjuk kesalahannya. Uji lain mencakup pendakian melewati akar, akar kedua, anak yang sudah terisi, dan perintah tak dikenal. Buku: Subbab 12.4.6.

Penyelesaian Keliru yang Menggoda: Naik dari Orang Tua

Kesalahan selisih satu: memulai pendakian dari *orang tua* simpul saat ini, “karena kita sudah satu tingkat di bawah”.

Ulangi: baris 7 `ur` berjalan dari 20 ke 30, lalu ke 50; 40 menjadi anak kanan **50**.

Baris 9 `uur` berjalan dari 40 ke 50 dan mencoba naik dua tingkat lagi, melewati akar: galat pada berkas yang sepenuhnya benar.

Dengan berkas yang sedikit berbeda tidak ada galat sama sekali, hanya pohon yang bentuknya salah, dan itu lebih buruk.

Pertahanannya: telusuri contoh dari soal itu sendiri dengan tangan sebelum menulis kode. Periksa diri: tulis berkas untuk pohon seimbang dengan preorder 8 4 2 6 12 10 14 dan bandingkan inorder-nya dengan 2 4 6 8 10 12 14. Buku: Subbab 12.4.7–12.4.8.

Soal 12.3: Lintasan Terpendek Kuda Catur

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.

Kuda bergerak membentuk huruf “L”: dua petak ke satu arah, satu petak tegak lurus; paling banyak delapan langkah dari satu petak, lebih sedikit di dekat tepi.

Keputusan kita: pertanyaan `r1 c1 r2 c2` *dari masukan standar, satu per baris; cetak banyak langkah dan satu lintasan terpendek; tolak petak di luar papan. Buku: Subbab 12.5.1.*

Breadth-First Search: Riak di Kolam

Petak awal berjarak 0; setiap petak satu langkah darinya berjarak 1; setiap petak baru satu langkah dari petak-petak itu berjarak 2; cincin demi cincin sampai tujuan.

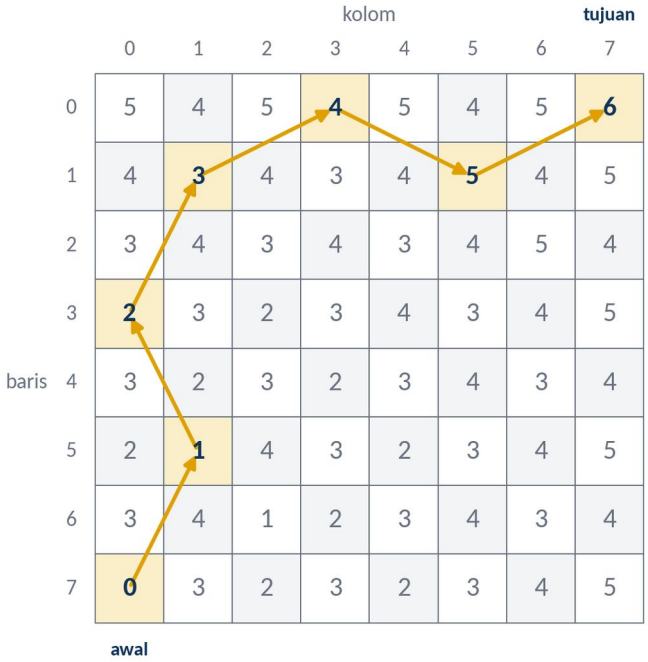
Antrean menjaga urutan cincin: semua petak berjarak d keluar sebelum petak mana pun yang berjarak $d+1$.

$$\text{dist}(v) = \text{dist}(u) + 1 \tag{12.2}$$

Persamaan (12.2): saat u mencapai petak baru v; dan u dicatat sebagai asal v, untuk menyusun lintasan dari belakang.

Karena cincin tumbuh satu langkah demi satu langkah, kedatangan pertama di tujuan adalah yang terpendek. Buku: Subbab 12.5.2.

Jarak dari (7,0)



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.

Gambar 12.2: (6,1), yang bersebelahan diagonal dengan petak awal, memerlukan 4 langkah. Jarak kuda hampir tidak ada hubungannya dengan jarak penggaris, karena itu soal ini memerlukan algoritma, bukan rumus. Buku: Subbab 12.5.2.

Pencariannya, di Atas Kertas

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]
```

Pseudokode 12.3: antreannya adalah `List` *dari Bab 10; enqueue adalah* `append`*, dequeue menghapus simpul pertama. Baris 8 menandai petak saat masuk antrean, sehingga paling banyak 64 petak masuk dan paling banyak* $64 \cdot 8 = 512$ *langkah dicoba. Buku: Subbab 12.5.3.*

Menjalankan Soal 12.3

```
(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
```

Baris-baris awal `knight` pada `data/knight_queries.txt`: lintasannya enam langkah, masing-masing satu langkah kuda dan satu cincin lebih jauh. Pertanyaan lain: awal sama dengan tujuan, satu langkah, lompatan pendek tiga langkah, pojok ke pojok, dan petak di luar papan. Buku: Subbab 12.5.6.

Penyelesaian Keliru yang Menggoda: *Depth-First Search*

```
(0,0) -> (7,7): DFS found a path of 52 move(s)
(3,3) -> (4,3): DFS found a path of 25 move(s)
```

Rekursi menggoda kita untuk menyelam mengikuti langkah pertama yang berhasil dan mundur hanya bila buntu: ***depth-first search***.

Pada contoh dari soal DFS menjawab 6, angka yang benar, dan pemrogram yang hanya menguji contoh itu merasa puas.

Dari pojok ke pojok ia menjawab 52; lompatan tiga langkah, 25. DFS menemukan *sebuah* lintasan, jarang yang terpendek.

Dari `knight_dfs.c`. Periksa diri: untuk raja hanya tabel `DR` dan `DC` yang berubah, dan (7,0) ke (0,7) memerlukan $\max(7,7)=7$ *langkah. Buku: Subbab 12.5.7–12.5.8.*

Ketiga Soal Berdampingan

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

Tabel 12.1: pada masing-masing, struktur dipilih agar satu operasi yang penting menjadi murah, dan masing-masing memerlukan satu pointer atau tanda tambahan yang tidak disebut soal. Buku: Subbab 12.6.

Memakainya secara Nyata

Senarai terurut dengan penyisipan — daftar tunggu klinik menurut jam janji, pasien tanpa janji ditambahkan di akhir.

Pohon — bagan organisasi koperasi, folder di laptop, bagan akun tempat “Beban” beranak “Sewa” dan “Gaji”.

BFS — transit paling sedikit antara dua halte bus, belokan paling sedikit di gudang antara dua rak, perhentian paling sedikit untuk mencapai setiap alamat antar dengan sepeda motor.

Bila langkah-langkahnya tidak sama, misalnya jalan dengan panjang berbeda, BFS biasa tidak cukup; mata kuliah berikutnya memperumumnya, tetapi antrean, tanda visited, dan pointer asal tetap sama. Buku: Subbab 12.7.

Kesalahan yang Sering Terjadi

Menulis kode sebelum menggambar; `p->next->data` **padahal** `p->next` **bisa** `NULL`; **struktur kosong terlupa**.

Hanya satu ujung antrean yang diperbarui: *enqueue* pada antrean kosong mengisi `head`; *dequeue* simpul terakhir mengeset `tail` menjadi `NULL`.

Selisih satu dalam navigasi; **menandai terlambat di BFS**; **memercayai contoh**.

Urutan pembebasan yang salah (pohon secara *postorder*); **tidak menjalankan *sanitizer***.

Setiap galat pointer di sini terlihat di kertas dalam satu menit dan tersembunyi di layar selama satu jam. Buku: Subbab 12.8.

Ringkasan: Lima Hal yang Perlu Diingat

1. Nyatakan ulang, gambar, rancang di kertas, tulis kode satu fungsi demi satu fungsi, uji kasus yang disiapkan di bawah *sanitizer*.

2. *Bubble sort* pada senarai menukar data tetangga dan berhenti sesudah lintasan tanpa pertukaran (Pseudokode 12.1); banyak pertukaran sama dengan banyak inversi (Persamaan (12.1)).

3. Pointer `parent` mengubah "naik k tingkat" menjadi k langkah (Pseudokode 12.2); bebaskan pohon secara *postorder*.

4. BFS dengan antrean menjelajah cincin demi cincin, sehingga kedatangan pertama adalah yang terpendek (Pseudokode 12.3); tandai saat dimasukkan.

5. Ketiga penyelesaian keliru lolos pada contoh soalnya: uji melampaui contoh (Tabel 12.1).

Buku: Subbab 12.9.

Pertemuan Berikutnya

Bab 13 — C++ sebagai C yang Lebih Baik. *Stream*, `std::string`, `std::vector`, referensi, *overloading*, argumen bawaan, `new` dan `delete`.

Kasir v1.0 ditulis ulang dalam C++, dan banyak sisi tajam dari Bab 9–11 menghilang.

Sebelum itu: Soal Latihan 12.1–12.6, terutama *insertion sort* dengan menyambung ulang simpul (`list_insertion_sort`).

Kita meninggalkan C dengan semua alat dari dua belas bab pertama di tangan. Buku: Subbab 12.10–12.11.

Kuliah 13 — C++ sebagai C yang Lebih Baik

Mata kuliah: Pemrograman Dasar (EF234101)

Buku: Irfan Subakti, *Pemrograman Dasar dengan C, C++, dan Python*, Bab 13

Dosen: Irfan Subakti — yifana@gmail.com

Hari ini kita berpindah dari C ke C++: semua yang membuat C cepat tetap ada, sedangkan pembukuan yang berbahaya diserahkan kepada bahasa. Stream, string, vector, referensi, overloading, argumen bawaan, `new` dan `delete`, serta kasir v1.0. Buku: Bab 13, Subbab 13.1–13.15.

Cakupan Pertemuan Ini

1. **Satu keluarga, dua dialek** — asal-usul C++, C++17, mengompilasi dengan `g++`.
2. ***Stream*, `std::string`, `std::vector`** — masukan dan keluaran tanpa *format string*; teks dan daftar yang tumbuh sendiri.
3. **Referensi** — mengubah variabel pemanggil tanpa pointer; referensi `const` untuk masukan yang besar.
4. ***Overloading* dan argumen bawaan** — satu nama, beberapa fungsi; argumen yang boleh dihilangkan.
5. **`new` dan `delete`, dan kasir v1.0** — alokasi bertipe, dan mengapa `vector` biasanya membuatnya tidak perlu.

Mulai bab ini program ditulis dalam C++17 dan Python, tetap dengan keluaran yang sama. Buku: Bab 13.

Posisi Kita: Struk-struk Bu Rina

Sepanjang Bab 1–12 kasir tumbuh dalam C: perulangan, larik, fungsi, string, struktur, *linked list*, berkas.

Ia bekerja, tetapi punya riwayat kecelakaan. Pelanggan bernama *Raden Ajeng Kusumawardhani Sulistyaningrum* meluap dari larik 30 karakter; huruf-huruf sisanya menimpa berat, dan struknya menunjukkan jutaan rupiah.

Setiap kecelakaan adalah pembukuan yang harus dikerjakan pemrogram dengan tangan: ukuran, penutup, `free`, `&` dan `*`.

C++ mempertahankan apa yang membuat C cepat dan dekat dengan mesin, lalu menambahkan alat yang mengerjakan pembukuan itu untuk Anda. Buku: Subbab 13.1.

Satu Keluarga, Dua Dialek

1979: Bjarne Stroustrup di Bell Labs menambahkan *class* ke C; “C with Classes” menjadi **C++** pada 1983, “C, selangkah lebih jauh”.

Dibakukan oleh ISO; C++26 selesai pada Maret 2026. Buku ini memakai **C++17**: lengkap di setiap kompilator, cukup untuk Bab 13–16.

C++ *hampir* merupakan superset C; `gcc` yang dipasang di Bab 1 sudah berisi `g++`.

`g++ -std=c++17 -O2 -Wall -Wextra hello_laundry.cpp -o hello_laundry`. Selalu tulis `-std=c++17`: GCC 16.1 memakai C++20 secara bawaan.

Di macOS, `g++` adalah nama kedua `clang++` dengan flag yang sama; di Windows perintah MSYS2 sama. Buku: Subbab 13.2.

Seluruh Bab dalam Satu Tabel

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`

Baca Tabel 13.1 sekali sekarang dan sekali lagi di akhir; saat itu setiap barisnya sudah akrab. Buku: Subbab 13.2.2.

Stream: `cout` dan `cin`

Kode 13.1 Dari `hello_laundry.cpp`: membaca nama dan berat dengan `cin` dan mencetak dengan `cout`.

```
    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";
```

`<<` (*insertion*) mengirim nilai ke *stream*; `>>` (*extraction*) mengambilnya. Tipe menentukan formatnya: tanpa `%d`, tanpa `&`. `std::getline` membaca satu baris utuh; `>>` akan berhenti di spasi dan hanya membaca “Bu”.

Kode 13.1: semua isi pustaka standar tinggal di namespace `std`. `hello_laundry.py` memakai `input` dan `int`. Buku: Subbab 13.3.

Contoh Terurai 1: Harga dalam Rupiah Bulat

$$P = \left\lfloor \frac{g \cdot r + 500}{1000} \right\rfloor \tag{13.1}$$

Persamaan (13.1): berat g dalam gram, tarif r dalam rupiah per kilogram; menambahkan setengah pembagi membulatkan ke rupiah terdekat.

$g = 3450$, $r = 7000$: $3450 \times 7000 = 24{,}150{,}000$.

Tambah 500 dan bagi 1000: $24{,}150{,}500 / 1000 = 24{,}150$; pecahan ,5 dibuang.

Pembagian bilangan bulat sama dengan pembulatan ke bawah untuk bilangan positif, di C++ maupun dengan `//` di Python. Tanpa `double`: uang tidak boleh kehilangan serupiah pun.

Rumus yang sama menghitung harga setiap baris kasir. Buku: Subbab 13.3.

Jebakan di Antara `>>` dan `getline`

JEBAKAN: Setelah `std::cin >> grams;`, karakter baris baru yang mengakhiri bilangan masih menunggu di *stream*. `std::getline` yang langsung menyusul membaca sisa baris yang kosong itu dan mengembalikan string kosong. Lewati dulu dengan `std::cin.ignore(1000, '\n')`, atau baca setiap baris dengan `getline` lalu ubah.

Itulah sebabnya `hello_laundry.cpp` membaca nama sebelum bilangan. Buku: Subbab 13.3.

`std::string`: Teks yang Pas

Sebuah *class* yang menampung teks **sepanjang apa pun** dan mengelola memorinya sendiri; tidak ada penutup yang bisa terlupa, tidak ada ukuran yang harus dipilih.

Menyambung dengan `+`; **membandingkan** dengan `==` dan `<`; **menyalin** dengan `=`.

`size()`, **`find(text)`** (mengembalikan `std::string::npos` bila tidak ada), **`substr(pos, len)`**, dan `s[i]` untuk satu karakter.

Nama panjang di Subbab 13.1 kini muat begitu saja.

`strings_demo.cpp` membangun "Rina Wulandari" dengan `+`, mencari, memotong, dan mengubah huruf besar-kecil; `strings_demo.py` melakukan hal yang sama dengan `str` Python. Buku: Subbab 13.4.

`std::vector`: Larik yang Tumbuh

`std::vector<int> grams = {3450, 1200, 5000, 2750};` lalu `grams.push_back(4100);`: *vector* tumbuh sendiri.

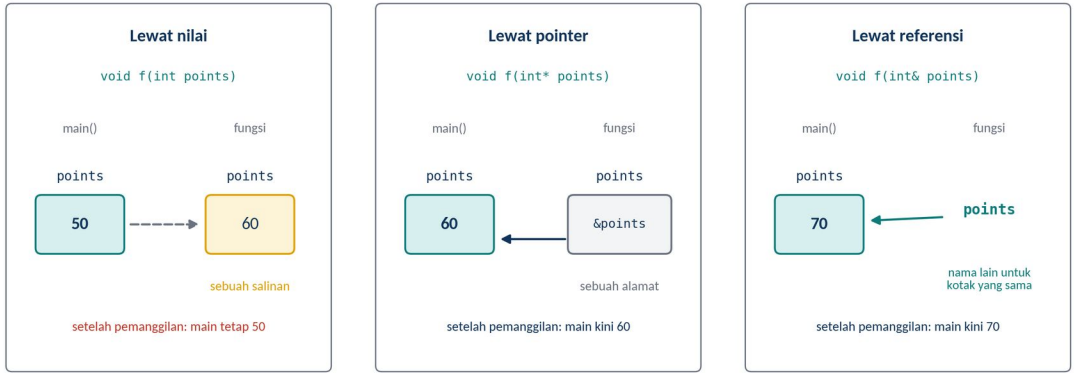
Ia tahu `size()`-nya, membebaskan memorinya sendiri, dan bekerja dengan perulangan berbasis rentang: `for (int g : grams)`.

`v[i]` tidak memeriksa; `v.at(i)` memeriksa dan melempar `std::out_of_range`.

`std::sort(grams.begin(), grams.end())` mengurutkannya; tipe elemen ditulis di dalam kurung sudut: `std::vector<std::string>`, nanti `std::vector<Order>`.

`vector_demo.cpp` dan `vector_demo.py`: `list` di Python mengerjakan apa yang dikerjakan `std::vector`. Buku: Subbab 13.5.

Tiga Cara Mengirim Variabel



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.

Gambar 13.1: lewat nilai (Bab 5), lewat pointer (Bab 7), dan yang baru di C++, lewat referensi. Buku: Subbab 13.6.

Referensi

DEFINISI. *Referensi* adalah nama lain bagi variabel yang sudah ada. Ia 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, dan tidak pernah dapat dibuat merujuk variabel lain.

Parameter referensi `int& points` menulis `points += 10;` tanpa `*`; pemanggil menulis `f(points)` tanpa `&`.

Di balik layar kompilator biasanya mengirim alamat; bedanya, referensi tidak dapat bernilai nol dan tidak dapat dialihkan. Buku: Subbab 13.6.

Contoh Terurai 2: Nilai, Pointer, Referensi

Kode 13.2 Dari `references.cpp`: parameter nilai, pointer, dan referensi.

```
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
```

Mulai dengan `points = 50`. **Lewat nilai**: salinannya menjadi 60; `points` tetap **50**.

Lewat pointer, `addBonusByPointer(&points)`: `*points` adalah kotak milik pemanggil; **60**.

Lewat referensi, `addBonusByReference(points)`: `points` di dalam fungsi *adalah* kotak milik pemanggil; **70**.

`int& alias = points; alias = 99;` menaruh **99** di `points`.

```
by value     : 50
by pointer   : 60
by reference : 70
alias = 99   : points is now 99
summary      : 12400 g in total, heaviest 5000 g
```

Kode 13.2: kerjakan keempat baris itu di papan sebelum memperlihatkan keluarannya. Buku: Subbab 13.6.

Dua Pemakaian Referensi yang Lazim

Kode 13.3 Dari `references.cpp`: fungsi dengan dua hasil dan masukan berupa referensi `const`.

```
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;
    }
}
```

Parameter keluaran — `total` dan `heaviest` memungkinkan satu fungsi mengembalikan dua hasil (Bab 7 melakukannya dengan pointer).

Referensi `const` untuk masukan — *vector* dikirim tanpa disalin, dan `grams.push_back(1)` di dalam fungsi tidak akan dapat dikompilasi.

Kode 13.3. Python tidak punya referensi: fungsi mengembalikan tuple, `return total, heaviest`, dan pemanggil membongkarnya. Buku: Subbab 13.6.

Overloading

> **DEFINISI.** Dua fungsi adalah *overload* satu sama lain bila namanya sama dan daftar parameternya berbeda (banyak parameter berbeda, atau tipenya berbeda). Pada setiap panggilan, kompilator memilih *overload* yang parameternya paling cocok dengan argumennya; inilah *overload resolution*, dan seluruhnya terjadi saat kompilasi. Tipe kembalian saja tidak dapat membedakan *overload*.

`printLine(text)`, `printLine(label, rupiah)`, `printLine(ch, count)`: tiga fungsi, satu nama.

`printLine('=', 30)` hanya cocok dengan yang ketiga; `printLine("PROFITINA LAUNDRY")` hanya dengan yang pertama.

Linker tetap memerlukan nama unik, jadi kompilator mendekorasi setiap nama dengan tipe parameternya (name mangling): `printLine(char, int)` *menjadi kira-kira* `_Z9printLineci`*. Buku: Subbab 13.7.*

Argumen Bawaan dan Harga Ekspres

`long long charge(int grams, long long ratePerKg, bool express = false)`: `charge(3450, 7000)` berarti `charge(3450, 7000, false)`.

Argumen bawaan harus berada paling kanan, karena argumen dicocokkan dari kiri.

$$P_{\text{express}} = P + \lfloor \frac{P}{2} \rfloor \tag{13.2}$$

Persamaan (13.2): untuk cucian 3450 g, $P = 24{,}150$ menurut Persamaan (13.1), jadi $P_{\text{express}} = 24{,}150 + 12{,}075 = 36{,}225$.

Python juga punya argumen bawaan, tetapi tidak punya overloading: satu nama, satu fungsi; `overload_default.py` *memakai tiga nama. Buku: Subbab 13.7.*

`new` dan `delete`

Kode 13.4 Dari `new_delete.cpp`: `new[]` dengan `delete[]`.

```
    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
```

`new` mengetahui tipenya: tanpa `sizeof`, tanpa *cast*. `new` berpasangan dengan `delete`, `new[]` dengan `delete[]`.

Dalam praktik `std::vector` mengerjakan tugas yang sama tanpa keduanya.

Kode 13.4: `nullptr` adalah pointer nol bertipe milik C++. Python membebaskan memori sendiri. Buku: Subbab 13.8.

Alat yang Sama di Python

`str` sejak dulu berperilaku seperti `std::string`, tetapi *immutable*: `full.upper()` mengembalikan string baru; *slicing* `full[space + 1:]` berperan sebagai `substr`, dan `find` mengembalikan -1 alih-alih `npos`.

`list` adalah `std::vector` milik Python: `append`, `len`, `sorted`, dan `IndexError` untuk indeks yang salah.

Tanpa referensi — nama diikat ke objek; fungsi yang harus memberi dua hasil mengembalikan *tuple*.

Tanpa *overloading* — satu nama, satu fungsi; argumen bawaan bekerja seperti di C++.

Kemudahannya sama; hanya ejaannya yang berbeda. Itulah sebabnya kembaran Python sama ringkasnya dengan program C++. Buku: Subbab 13.4–13.8.

Kasir v1.0

Membaca pesanan sehari dari `input/orders.txt`: `customer;service;grams;express`, `#` untuk komentar.

Tiga layanan: `wash` Rp 7.000/kg, `washiron` Rp 10.000/kg, `iron` Rp 5.000/kg.

Memeriksa setiap baris, mencetak satu baris per pesanan yang diterima, menjelaskan setiap baris yang ditolak, dan menutup dengan total dan ringkasan per layanan.

Memakai semuanya: `std::string`, `std::vector`, referensi, *overload*, argumen bawaan, `std::ifstream`, dan `<iomanip>` untuk kolom.

`cashier.cpp` dan `cashier.py` mencetak teks yang sama. Buku: Subbab 13.9.

Mengujinya: Dua Bahasa Saling Memeriksa

Setiap program punya keluaran yang wajib dicetaknya di `expected/`; daftar program, masukan, dan argumennya adalah berkas teks kecil di `tests/`.

`tests/run_tests.py` mengompilasi setiap program C++ dengan `-Wall -Wextra -Werror`, sehingga satu peringatan pun dihitung gagal.

Ia menjalankan program C++ dan kembaran Python-nya dengan masukan yang sama dan membandingkan keduanya dengan teks yang diharapkan, karakter demi karakter.

Dengan `--sanitize`, *sanitizer* alamat dan perilaku tak terdefinisi menghentikan program pada akses memori salah atau luapan yang pertama.

```
references                C++ ok   Python ok
cashier                   C++ ok   Python ok
cashier_bad               C++ ok   Python ok
all programs agree
```

Perbedaan pembulatan atau urutan baris di salah satu bahasa akan menggagalkan perbandingan. Sesudah setiap perubahan, jalankan ujinya lagi. Buku: Subbab 13.11.

Menjalankan Kasir v1.0

```
 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
Revenue         : Rp 188950
```

Periksa satu baris dengan tangan: 5.200 g cuci-setrika milik Pak Budi dengan Rp 10.000 memberi $52{,}000$ *menurut Persamaan (13.1), dan ekspres menambah setengahnya:* $78{,}000$*. Buku: Subbab 13.10.*

Menolak Baris Buruk dengan Sopan

```
 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
```

Dari `input/orders_bad.txt`: setiap baris buruk ditolak beserta nomornya dan alasan dalam kata-kata biasa, sedangkan baris baik di sekitarnya tetap diterima. Program yang dapat dipercaya tidak menebak, tidak jatuh, dan mengatakan apa yang dilakukannya. Buku: Subbab 13.12.

Memakainya secara Nyata

Daftar harga — penjahit per potong, fotokopi per halaman dengan harga lebih rendah di atas 100, cuci mobil per golongan kendaraan.

Pemeriksaan — tanyakan kepada petugas di meja kesalahan apa yang benar-benar terjadi; satu pemeriksaan, satu pesan jelas untuk masing-masing.

Ringkasan — tanyakan kepada pemilik angka mana yang ia lihat setiap sore.

Uang dalam rupiah bulat, dibulatkan sekali di tempat yang tercatat; Bab 15 menjadikannya kelas `Rupiah`.

Pemeriksaan yang tidak pernah diperlukan tidak merugikan apa-apa; pemeriksaan yang hilang dapat merusak pembukuan satu hari. Buku: Subbab 13.13.

Kesalahan yang Sering Terjadi

Lupa `-std=c++17`; mencampur `>>` dan `getline`; membandingkan string C dengan `==`.

Mengindeks *vector* melewati ujungnya (`v[i]` tidak memeriksa); **mengembalikan referensi ke variabel lokal** (*dangling reference*).

`new[]` dibebaskan dengan `delete`; uang dalam `double`; *overload* yang ambigu.

Kembalikan lewat nilai, bukan referensi ke variabel lokal; C++ memindahkan nilainya keluar dengan murah. Buku: Subbab 13.14.

Ringkasan: Lima Hal yang Perlu Diingat

1. C++ mengompilasi hampir setiap program C; `g++ -std=c++17 -Wall -Wextra` di setiap sistem (Tabel 13.1).

2. `cout <<` dan `cin >>` tidak memerlukan *format string*; `getline` membaca satu baris; `std::string` dan `std::vector` tumbuh sendiri.

3. Referensi adalah nama lain bagi variabel (Gambar 13.1); `const&` mengirim objek besar tanpa menyalin.

4. *Overload* berbagi nama dan berbeda parameter; argumen bawaan mengisi parameter paling kanan; keduanya saat kompilasi.

5. `new`/`delete` harus berpasangan; kontainer biasanya membuatnya tidak perlu; uang tetap bilangan bulat (Persamaan (13.1), Persamaan (13.2)).

Python memberi kemudahan yang sama dengan caranya sendiri, dan kasir v1.0 mencetak sama persis dalam keduanya. Buku: Subbab 13.15.

Pertemuan Berikutnya

Bab 14 — Kelas dan Abstraksi Data. `Order` yang tidak mungkin salah: data `private`, konstruktor, invarian, destruktor, dan sekilas pewarisan.

Kasir menjadi kelas `Shop` yang hanya dapat dipakai bagian lain program lewat fungsi-fungsi publiknya.

Sebelum itu: Soal Latihan 13.1–13.6, terutama menerima berat dalam kilogram dengan koma desimal.

Pertemuan berikutnya dibuka dengan cucian gratis yang terberikan tanpa sengaja, dan kelas yang mencegahnya. Buku: Subbab 13.16–13.17.

Kuliah 14 — Kelas dan Abstraksi Data

Mata kuliah: Pemrograman Dasar (EF234101)

Buku: Irfan Subakti, *Pemrograman Dasar dengan C, C++, dan Python*, Bab 14

Dosen: Irfan Subakti — yifana@gmail.com

Hari ini sebuah pesanan belajar menjaga aturannya sendiri: data private, konstruktor yang menolak nilai buruk, siklus hidup yang tidak dapat dilanggar, destruktor yang membersihkan sendiri, sekilas pewarisan, dan kelas `Shop` untuk seluruh hari. Buku: Bab 14, Subbab 14.1–14.13.

Cakupan Pertemuan Ini

1. **Abstraksi data dan invarian** — antarmuka lawan implementasi; `public`, `protected`, `private`.
2. **Kelas `Order`** — *header* dan implementasi, *getter*, `const`, anggota statis, konstruktor yang melempar eksepsi.
3. **Konstruktor dan destruktor** — lahir sah, dibersihkan otomatis: RAII.
4. **Sekilas pewarisan** — kelas dasar `Service`, fungsi virtual, polimorfisme.
5. **`Shop`** — kelas yang memiliki banyak pesanan, dan satu hari di meja kasir.

Kelas-kelasnya ada dalam C++ dan Python dan berperilaku sama. Buku: Bab 14.

Posisi Kita: Cucian Gratis Tanpa Sengaja

Suatu Sabtu laci kurang Rp 24.150. Seorang pembantu “memperbaiki” pesanan dengan pilihan menu yang langsung mengubah berat, mengetik 0, dan harganya menjadi nol.

Di kasir C, pesanan adalah `struct`: fungsi mana pun dapat mengisi medan mana pun dengan nilai apa pun, −500 gram, tarif 700, nama kosong.

Aturan pesanan yang masuk akal hanya ada di kepala pemrogram dan di beberapa pernyataan `if` pada fungsi masukan.

Gagasan hari ini: letakkan aturan itu di dalam pesanan. Ia memeriksa nilainya saat dibuat, menolak ada bila nilainya salah, dan menyembunyikan datanya di balik fungsi-fungsi yang menjaga aturan. Buku: Subbab 14.1.

Abstraksi Data: Penanak Nasi

Anda memasukkan beras dan air, menekan satu tombol, dan menunggu lampu berpindah dari “masak” ke “hangat”.

Anda tidak menyetel elemen pemanasnya; ia tertutup di dalam. Anda mendapat **antarmuka** yang sederhana dan hasil yang terjamin.

Kelas melakukan hal yang sama untuk data: **antarmukanya** adalah fungsi yang boleh dipanggil pihak lain; **implementasinya**, data dan kode di dalamnya, disembunyikan.

Tipe yang didefinisikan oleh apa yang dapat dilakukan dengannya, bukan oleh cara penyimpanannya, adalah *tipe data abstrak*.

Kelas adalah batu bata dasar C++, Java, C#, Python, dan hampir setiap bahasa yang akan Anda pakai sesudah buku ini. Buku: Subbab 14.2.1.

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.

$$1 \le g \le 50\,000 \tag{14.1}$$

Untuk sebuah pesanan: nama tidak kosong, layanan yang dikenal, berat dalam gram seperti Persamaan (14.1), dan status yang hanya bergerak Received → Washing → Ready → Picked up.

Bila datanya private dan setiap fungsi anggota menjaga invarian, invarian itu berlaku selamanya: diperiksa di satu tempat, benar di semua tempat, termasuk kode yang baru ditulis bertahun-tahun kemudian. Buku: Subbab 14.2.2.

Dari `struct` ke `class`

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

Tabel 14.1: di C++, `struct` dan `class` hanya berbeda pada bawaannya, *public* untuk `struct`, *private* untuk `class`.

Kebiasaannya: `struct` untuk kumpulan data polos tanpa aturan; `class` untuk tipe yang punya invarian. Buku: Subbab 14.2.3.

Masalahnya, Diukur

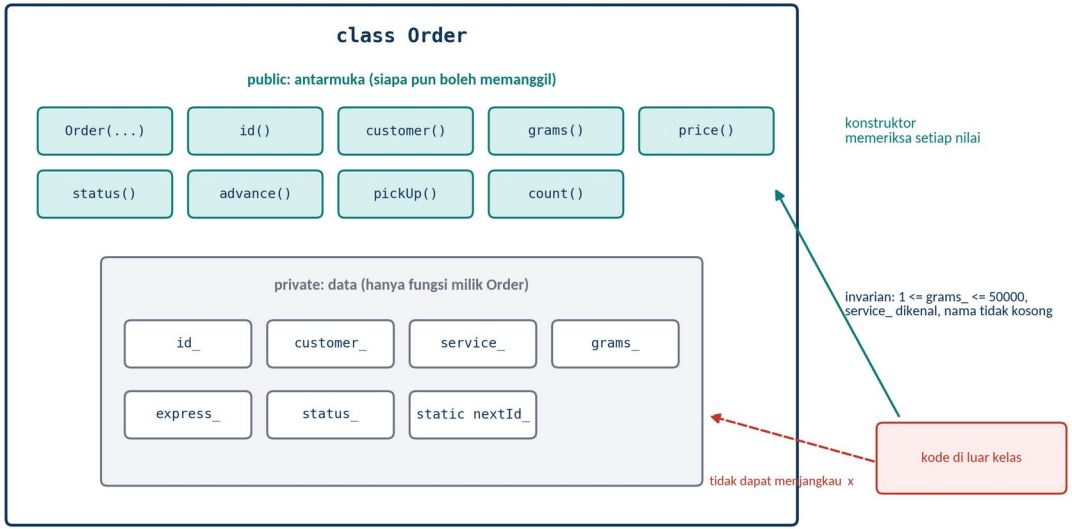
```
Bu Rina, 3450 g at Rp 7000/kg: Rp 24150
Bu Rina, 0 g at Rp 7000/kg: Rp 0
Bu Rina, 3450 g at Rp 700/kg: Rp 2415
, 3450 g at Rp 700/kg: Rp 2415
```

`struct_problem.cpp` membuat satu pesanan yang benar, lalu tiga penugasan: cucian gratis, harga sepuluh kali terlalu murah, pesanan tanpa pemilik.

Kompilator tidak dapat menolong, karena tidak ada apa pun di program yang menyatakan nilai-nilai itu salah. *Dataclass* Python sama terbukanya.

Sebelum memperbaiki masalah, lihat dulu masalahnya. Buku: Subbab 14.3.

Rancangan Kelas `Order`



Gambar 14.1 Kelas `Order`: antarmuka *public* mengelilingi data *private*; kode dari luar hanya dapat memanggil fungsi *public*, dan konstruktor memeriksa setiap nilai.

Gambar 14.1: konstruktor, fungsi-fungsi pembaca, dua fungsi yang mengubah dengan cara yang diizinkan (`advance`, `pickUp`), dan sebuah penghitung; datanya private. Buku: Subbab 14.4.

Antarmukanya: `order.h`

Kode 14.1 Dari `order.h`: antarmuka kelas `Order` (sebagian *getter* tidak ditampilkan).

```
enum class Status { Received, Washing, Ready, PickedUp };

class Order {
public:
    Order(const std::string& customer, const std::string& service, int grams,
          bool express = false);

    int id() const { return id_; }
    int grams() const { return grams_; }
    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);
};
```

Kode 14.1 berada di dalam include guard, seperti di Bab 11. Nama `enum class` harus ditulis `Status::Ready` dan tidak diam-diam berubah menjadi bilangan bulat. Buku: Subbab 14.4.1.

Membaca Antarmukanya

Konstruktor bernama sama dengan kelasnya dan tanpa tipe kembalian; ia berjalan setiap kali `Order` dibuat dan menegakkan invarian.

Getter seperti `id()` memberi akses baca; ditulis di dalam kelas, ia bersifat *inline* dan tidak memakan biaya saat program berjalan.

`const` sesudah parameter berjanji *fungsi ini tidak mengubah objek*; hanya fungsi `const` yang boleh dipanggil pada `const Order&`.

`price()` dihitung, tidak pernah disimpan, sehingga tidak mungkin berbeda dengan beratnya. Anggota **`static`** milik kelas: satu `nextId_` untuk semua pesanan.

Garis bawah di belakang (`grams_`) menandai anggota data, sehingga di dalam fungsi anggota jelas nama mana yang anggota. Buku: Subbab 14.4.1.

Konstruktornya: `order.cpp`

Kode 14.2 Dari `order.cpp`: konstruktor, dengan *member initializer list* dan tiga pemeriksaannya.

```
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
}
```

Kode 14.2: member initializer list sesudah titik dua mengisi setiap anggota secara langsung; badannya memeriksa invarian dan melempar eksepsi bila ada nilai yang salah. `int Order::nextId_ = 1;` *mendefinisikan anggota statis di tepat satu berkas. Buku: Subbab 14.4.2.*

Nomor Tanpa Celah

`id_ = nextId_++;` diletakkan *sesudah* pemeriksaan, sehingga pesanan yang ditolak tidak memakai nomor; pesanan sah bernomor 1, 2, 3, …

$$\text{count} = \text{nextId} - 1 \tag{14.2}$$

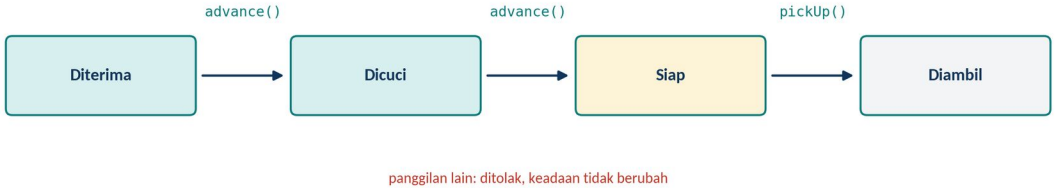
Persamaan (14.2) persis yang dikembalikan `count()`.

`price()` menerapkan Persamaan (14.3) dan menambah setengahnya untuk ekspres, seperti di Bab 13.

$$P = \left\lfloor \frac{g \cdot r + 500}{1000} \right\rfloor \tag{14.3}$$

Aturan harga dari Bab 13 berpindah ke dalam kelas, tempat tidak ada pemanggil yang dapat menghitungnya dengan cara lain. Buku: Subbab 14.4.2.

Siklus Hidup Sebuah Pesanan



Gambar 14.2 Siklus hidup sebuah pesanan: `advance()` memajukannya dua kali, `pickUp()` menyerahkannya, dan setiap panggilan lain ditolak tanpa mengubah keadaannya.

Gambar 14.2: `advance` dan `pickUp` adalah satu-satunya fungsi yang mengubah pesanan sesudah dibuat. Keduanya mengembalikan `false` untuk apa pun yang tidak diizinkan diagram, tanpa mengubah keadaan. Buku: Subbab 14.4.2.

Memakai Kelasnya

```
#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 yang tidak sah tidak pernah ada. `a.grams_ = 0;` tidak dapat dikompilasi: `grams_` bersifat *private*.

`order_main.cpp`, dikompilasi dengan `g++ -std=c++17 -O2 -Wall -Wextra cpp/order.cpp cpp/order_main.cpp -o order_main`: setiap `.cpp` dari program disebutkan. Kecelakaan di Subbab 14.1 kini bukan sekadar jarang, melainkan mustahil. Buku: Subbab 14.4.3.

Contoh Terurai 1: Pesanan yang Ditolak, Langkah demi Langkah

CONTOH. `main` menjalankan `Order bad("Mbak Sari", "wash", -500);` di dalam blok `try`.

1. Memori untuk `bad` dipesan; konstruktor dipanggil, `express = false` dari argumen bawaan.

2. *Initializer list* berjalan: `grams_` menjadi −500. Objek melanggar invariannya, tetapi belum ada yang dapat melihatnya dari luar.

3. Nama tidak kosong; `wash` bertarif 7.000; `grams_ < 1` bernilai **benar**.

4. `throw` membuat `std::invalid_argument` dan langsung keluar; `id_ = nextId_++;` tidak pernah berjalan, jadi `nextId_` tetap 3.

5. Objek itu dianggap tidak pernah ada; kedua string-nya dihancurkan dengan benar. **6.** `catch` terdekat mencetak `refused: weight must be 1..50000 g, not -500`.

Di C++ tidak ada objek setengah jadi yang dapat disentuh program: konstruktor selesai dan invarian berlaku, atau objeknya tidak ada. Python memberi jaminan yang sama bila `__init__` melempar eksepsi. Buku: Subbab 14.4.6.

Kelas yang Sama di Python

Konstruktornya adalah `__init__`; `self` ditulis terang-terangan di tempat C++ memakai pointer tersembunyi `this`.

Tidak ada kata kunci `private`: garis bawah di depan (`_grams`) menandai nama sebagai *private berdasarkan kesepakatan*.

Property yang hanya dapat dibaca memberi antarmuka yang sama dengan `order.h`; `raise ValueError(...)` menolak nilai buruk.

`order_main.py` mencetak persis apa yang dicetak `order_main.cpp`.

Mengapa perlu getter? Antarmuka seharusnya menjanjikan perilaku, bukan penyimpanan: besok `price()` mungkin membaca daftar harga, dan pemanggilnya tidak akan menyadarinya. Buku: Subbab 14.4.4–14.4.5.

Destruktor dan RAII

Destruktor, `~ShiftLog()`, berjalan saat objek mati: untuk variabel lokal, di kurung kurawal penutup bloknya, lewat jalan keluar mana pun.

RAII, *resource acquisition is initialization*: ambil sumber daya di konstruktor, kembalikan di destruktor, maka ia tidak mungkin terlupa.

Objek dalam satu lingkup mati dengan urutan *terbalik* dari pembuatannya, seperti piring yang diambil dari tumpukan.

```
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
```

`raii_demo.cpp`: catatan Dewi ditutup di kurung kurawal bloknya; catatan Eko ditutup terakhir, sesudah "shop closes". Python tidak menjanjikan kapan `__del__` berjalan; alatnya untuk pembersihan terjamin adalah `with`, yang `__exit__`-nya berjalan di akhir blok. Buku: Subbab 14.5.

Sekilas Pewarisan

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.

`Service` adalah kelas dasarnya: nama yang `protected`, serta `price` dan `unit` yang *pure virtual*, sehingga ia *abstrak*.

`PerKilo`, `PerPiece`, dan `Express`, yang *membungkus* layanan lain dan menambah setengahnya.

Konstruktor tidak diwariskan: setiap konstruktor turunan memanggil konstruktor dasarnya, `: Service(name), rate_(ratePerKg)`. *Kelas dasar diberi* `virtual ~Service() = default;`. *Buku: Subbab 14.6.*

Contoh Terurai 2: Satu Panggilan, Beberapa Perilaku

```
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
```

`menu` adalah `std::vector<const Service*>`; perulangan memanggil `menu[i]->price(amounts[i])` tanpa mengetahui kelasnya.

wash: $(3450 \cdot 7000 + 500)/1000 = 24{,}150$. **wash express**: $24{,}150 + 12{,}075 = 36{,}225$.

bed cover, 2 potong: $2 \cdot 25{,}000 = 50{,}000$; ekspres: $75{,}000$.

Inilah polimorfisme. Saat berjalan setiap objek membawa pointer ke virtual table kelasnya, yang berisi alamat versi fungsi virtual milik kelas itu. Di Python setiap method berperilaku seperti fungsi virtual. Buku: Subbab 14.6.

`Shop`: Kelas yang Memiliki Pesanan

Pewarisan berarti “adalah”; `Shop` **memiliki** `std::vector<Order>`.

Operasinya: menerima pesanan baru, mencari menurut nomor, memajukan, menyerahkan, mendaftar, melaporkan uang.

Setiap operasi mengembalikan satu baris teks; `Shop` tidak memutuskan *di mana* teks dicetak.

`Shop` tidak pernah memeriksa berat sendiri: ia mengandalkan konstruktor `Order`.

`shop.h`, `shop.cpp`, dan `shop_main.cpp`, serta `shop.py` dengan `shop_main.py`. Buku: Subbab 14.7.

Satu Hari di Meja Kasir

```
> new Mbak Sari;washiron;-500;no
refused: weight must be 1..50000 g, not -500
> pickup 2
order #2 is not ready (received)
> advance 2
order #2 cannot advance (picked up)
> advance 9
order #9 does not exist
> pickup two
not a number in 'pickup two'
> report
picked up 2 of 3, paid Rp 39150, still in the shop Rp 52000
```

Baris-baris pilihan dari `./shop < input/day.txt`: setiap kesalahan ditolak beserta alasannya, dan laporannya cocok. Program penguji membandingkan C++ dan Python pada hari ini dan pada demo-demonya. Buku: Subbab 14.8–14.9.

Membuatnya Lebih Baik, Memakainya secara Nyata

Uang sebagai tipe dan **mencetak dengan** `<<` — Bab 15. **Menyimpan hari** — simpan di destruktor, muat di konstruktor. **Mencari lebih cepat** — `std::map`.

Tiga pertanyaan: *Apa yang membuat satu item sah?* (konstruktor) *Dalam urutan apa keadaannya boleh berubah?* (satu fungsi per panah) *Apa yang ingin diketahui pemilik di akhir hari?* (`report`).

Warung katering Mbak Wulan: seorang pelanggan, 1 sampai 200 porsi, pengantaran antara 06:00 dan 20:00; Dipesan, Dimasak, Dalam perjalanan, Diantar.

Untuk klinik, program antrean hanya menjaga urutan antrean; memutuskan siapa yang mendesak adalah tugas perawat dan dokter. Buku: Subbab 14.10–14.11.

Kesalahan yang Sering Terjadi

Data *public* “untuk sementara”; **lupa `const` pada fungsi pembaca**; **memeriksa invarian di luar kelas**.

Menugaskan di badan konstruktor alih-alih di *initializer list*; inisialisasi sesuai urutan deklarasi.

Anggota statis didefinisikan dua kali atau tidak sama sekali; **hanya satu berkas yang dikompilasi** (“undefined reference”).

Kelas dasar tanpa destruktor virtual; **mengandalkan `__del__` di Python**.

Buku: Subbab 14.12.

Ringkasan: Lima Hal yang Perlu Diingat

1. Abstraksi data memisahkan antarmuka dari implementasi (Gambar 14.1); penentu akses menegakkannya (Tabel 14.1).

2. Data *private* ditambah fungsi anggota yang menjaga aturan membuat invarian berlaku selamanya (Persamaan (14.1)).

3. Konstruktor menegakkan invarian dan melempar eksepsi untuk menolak; tidak pernah ada objek yang tidak sah; destruktor memungkinkan RAII.

4. Anggota `const` berjanji tidak mengubah; anggota statis milik kelas (Persamaan (14.2)); diagram keadaan menjadi satu fungsi per panah (Gambar 14.2).

5. Pewarisan berarti “adalah”, komposisi berarti “memiliki”; fungsi virtual memberi satu panggilan beberapa perilaku.

Di Python: `__init__`, garis bawah berdasarkan kesepakatan, property, `with`, dan setiap method virtual. Buku: Subbab 14.13.

Pertemuan Berikutnya

Bab 15 — Overloading Operator dan Studi Kasus Utuh. Kelas `Rupiah` dengan `+`, `*`, `==`, `<`, dan `<<`; pecahan dengan algoritma Euclid.

Lalu program Profitina Laundry yang lengkap: pelanggan, layanan, pesanan, diskon, laporan harian.

Sebelum itu: Soal Latihan 14.1–14.6, terutama menyimpan pesanan toko di destruktornya.

Lain kali struk-struk akan cocok dengan sendirinya karena rancangannya. Buku: Subbab 14.14–14.15.

Kuliah 15 — Overloading Operator dan Studi Kasus Utuh

Mata kuliah: Pemrograman Dasar (EF234101)

Buku: Irfan Subakti, *Pemrograman Dasar dengan C, C++, dan Python*, Bab 15

Dosen: Irfan Subakti — yifana@gmail.com

Hari ini uang menjadi kelas dengan operatornya sendiri, pecahan bertemu algoritma Euclid, dan Bab 13, 14, dan 15 disatukan dalam program Profitina Laundry yang lengkap. Buku: Bab 15, Subbab 15.1–15.11.

Cakupan Pertemuan Ini

1. **Operator adalah fungsi** — cara kompilator membaca `wash + iron`, dan aturan-aturan *overloading*.
2. **Kelas `Rupiah`** — penugasan majemuk sebagai anggota, aritmetika dan perbandingan sebagai non-anggota, satu aturan pembulatan, mencetak dengan `<<`.
3. **Kelas `Fraction`** — bentuk paling sederhana dengan FPB Euclid, penjumlahan dengan KPK, konstruktor pengubah.
4. **Studi kasus utuh** — tiga berkas data, struk, laporan harian terurut di layar dan di berkas.
5. **Memeriksanya dengan tangan** — struk yang paling rumit, dan total-total laporan.

Setiap program dalam C++ dan Python, dengan keluaran yang sama. Buku: Bab 15.

Posisi Kita: Struk yang Cocok

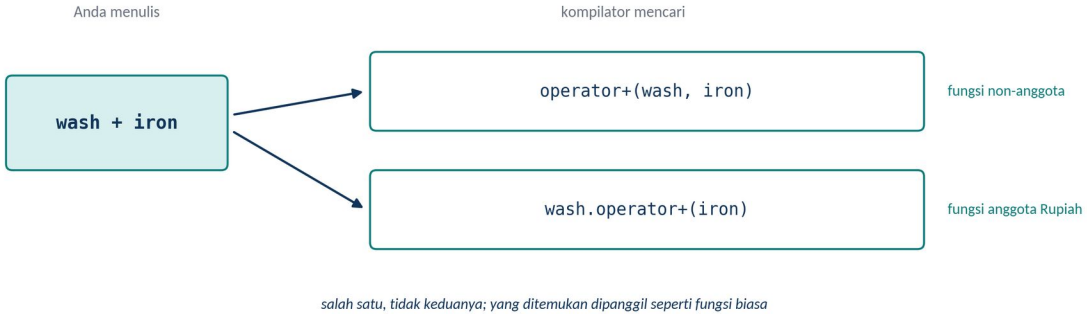
Setiap bulan struk-struk Bu Rina selisih beberapa ratus rupiah dari total program.

Dua pembantunya menghitung struk yang sama dengan tangan, dan masing-masing membulatkan diskon anggota 10% dengan cara berbeda.

Masalahnya bukan aritmetika. *Uang* diperlakukan sebagai bilangan biasa, dan setiap potong kode membuat keputusan kecilnya sendiri: cara membulatkan, cara mencetak, boleh tidaknya berat dijumlahkan dengan harga.

Obatnya adalah gagasan Bab 14 yang diterapkan pada bilangan: jadikan uang sebuah kelas yang menjumlah, menghitung persen dengan satu aturan, membandingkan, dan mencetak dirinya, lalu biarkan overloading operator menjaga kode tetap mudah dibaca. Buku: Subbab 15.1.

Operator Adalah Fungsi



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.

Gambar 15.1: untuk dua `int` kompilator menghasilkan penjumlahan mesin; untuk objek kelas ia mencari fungsi bernama `operator+`. `std::cout << x` juga merupakan panggilan `operator<<(std::cout, x)`. Buku: Subbab 15.2.1.

Operator Overloading

> **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).

Batas-batasnya — tidak ada operator baru (tidak ada `**`); presedensi, banyak operan, dan asosiativitas tidak pernah berubah; `1 + 1` selalu 2; `.`, `::`, `?:`, dan `sizeof` tidak dapat di-*overload*.

Aturan tak tertulis — operator harus melakukan apa yang diharapkan pembaca. `+` pada uang menjumlahkan uang; bila ragu, tulis fungsi bernama.

`operator[]` juga operator yang di-overload: itulah sebabnya `std::map` dapat membuat entri yang belum ada sedangkan `std::vector` tidak memeriksa indeksnya. Buku: Subbab 15.2.2.

Bentuk Mana untuk Operator Mana

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__`

Tabel 15.1: operator yang mengubah operan kirinya menjadi anggota; operator simetris menjadi non-anggota yang dibangun di atasnya; `<<` harus non-anggota karena operan kirinya adalah stream. Buku: Subbab 15.2.2.

`Rupiah`: Anggota-anggotanya

Kode 15.1 Dari `rupiah.h`: kelas `Rupiah` (`operator-=` tidak ditampilkan).

```
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 { 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;
};
```

Kode 15.1: satu anggota data, rupiah bulat dalam `long long`. Konstruktornya `explicit`, sehingga bilangan polos tidak pernah diam-diam menjadi uang. `+=` mengembalikan `*this` lewat referensi, sehingga `till += wash;` berperilaku seperti `+=` bawaan. Buku: Subbab 15.3.1.

Satu Aturan Pembulatan untuk Semua

$$\text{percent}(a, p) = \left\lfloor \frac{a \cdot p + 50}{100} \right\rfloor \tag{15.1}$$

Persamaan (15.1) membulatkan p % dari jumlah tak negatif *a setengah menjauhi nol*; jumlah negatif dibulatkan sebagai nilai mutlaknya lalu tandanya dikembalikan.

Menambahkan 50 sebelum membagi 100 adalah trik Bab 13, menambahkan 500 sebelum membagi 1000.

10% dari Rp 24.150: $\lfloor (241{,}500 + 50)/100 \rfloor = 2{,}415$.

Kedua pembantu di Subbab 15.1 tidak dapat lagi berbeda pendapat: hanya ada satu tempat persen dibulatkan. Buku: Subbab 15.3.1.

`Rupiah`: Non-anggotanya

Kode 15.2 Dari `rupiah.h`: aritmetika, perbandingan, dan keluaran sebagai non-anggota.

```
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; }
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.value() < b.value();
}
inline std::ostream& operator<<(std::ostream& os, const Rupiah& r) {
    return os << toText(r);                    // honors setw, left and right
}
```

Kode 15.2: `+` menyalin operan kirinya dan memakai `+=`. Dua versi `*` membuat `rate * amount` maupun `3 * iron` berjalan; tidak ada `Rupiah * Rupiah`. Empat perbandingan lainnya ditulis dari `==` dan `<`, sehingga tidak mungkin bertentangan. `toText` mencetak `Rp 1.234.567.890`. Buku: Subbab 15.3.1.

Kelas Itu Bekerja

```
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
[     Rp 24.150]
[Rp 24.150     ]
```

Dari `rupiah_demo.cpp`: setiap baris dibaca sebagai aritmetika biasa dan di baliknya adalah panggilan fungsi. `setw(14)` meratakan jumlah uang ke kanan atau ke kiri seperti bilangan. `Rupiah x = 5000;` dan `wash * wash` tidak dapat dikompilasi. Buku: Subbab 15.3.2.

Contoh Terurai 1: `base + express - discount`

`base` Rp 27.500, `express` Rp 13.750, `discount` Rp 4.125; `+` dan `-` berjalan dari kiri ke kanan: `(base + express) - discount`.

1. `operator+` non-anggota: `a` adalah *salinan* `base`, `b` nama lain bagi `express`.

2. `a += b`: $27{,}500 + 13{,}750 = 41{,}250$; `Rupiah` sementara dikembalikan; `base` tidak tersentuh.

3. `operator-` menyalin objek sementara itu dan memanggil `-=`: $41{,}250 - 4{,}125 = 37{,}125$.

4. Objek-objek sementara mati di akhir pernyataan; pengoptimal membuatnya semurah bilangan bulat biasa.

Keamanan tanpa biaya saat berjalan: ekspresi itu tidak mungkin tertulis dengan berat atau hitungan secara tidak sengaja. Buku: Subbab 15.3.3.

Kelas yang Sama di Python

Metode khusus berperan sebagai operator: `__add__`, `__sub__`, `__mul__`, dan `__rmul__` untuk `3 * iron`. `__eq__` dan `__lt__`, dengan `functools.total_ordering` yang membangkitkan perbandingan lainnya. `__str__` memberi teksnya; `__format__` memungkinkan *f-string* meratakan jumlah uang seperti bilangan. `rupiah_demo.py` mencetak persis apa yang dicetak `rupiah_demo.cpp`.

Kelas Python yang cermat mengembalikan `NotImplemented` bila tidak dapat menangani operan lainnya, agar Python dapat mencoba metode cerminan milik operan itu. Buku: Subbab 15.3.4.

Pecahan dan Algoritma Euclid

$$\gcd(a,b)=\gcd(b,a \bmod b),\gcd(a,0)=a \tag{15.2}$$

Pecahan disimpan dalam **bentuk paling sederhana**: $\frac{6}{8}$ disimpan sebagai $\frac{3}{4}$, dengan membagi oleh faktor persekutuan terbesar (FPB), yang dicari dengan Persamaan (15.2) (Euclid, sekitar 300 SM).

Terurai: $\gcd(270,192)=\gcd(192,78)=\gcd(78,36)=\gcd(36,6)=\gcd(6,0)=6$.

Alasannya: setiap pembagi a dan b juga membagi $a \bmod b$, dan sebaliknya; banyak langkahnya hanya tumbuh menurut banyak *digit*.

Deterjen dan pelembut yang diukur dengan sendok takar adalah pecahan alami: setengah sendok ditambah sepertiga. Buku: Subbab 15.4.

KPK dan Menjumlahkan Pecahan

$$\operatorname{lcm}(a,b) = \frac{a}{\gcd(a,b)} \cdot b \tag{15.3}$$

Persamaan (15.3): bagi *lebih dulu*; $a \cdot b$ dapat meluap walaupun KPK-nya sendiri muat.

$$\frac{a}{b} + \frac{c}{d} = \frac{a \cdot \frac{m}{b} + c \cdot \frac{m}{d}}{m}, m = \operatorname{lcm}(b,d) \tag{15.4}$$

Persamaan (15.4): untuk $\frac{1}{2} + \frac{1}{3}$, $m = 6$, dan jumlahnya $\frac{1 \cdot 3 + 1 \cdot 2}{6} = \frac{5}{6}$.

`fraction.cpp` *mewujudkan* `+`, `-`, `*`, `/`, `==`, `<`, *dan* `<<` *di atas rumus-rumus ini. Buku: Subbab 15.4.*

`Fraction` Bekerja

```
gcd(270, 192) = 6, lcm = 8640
1/2 + 1/3   = 5/6
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
```

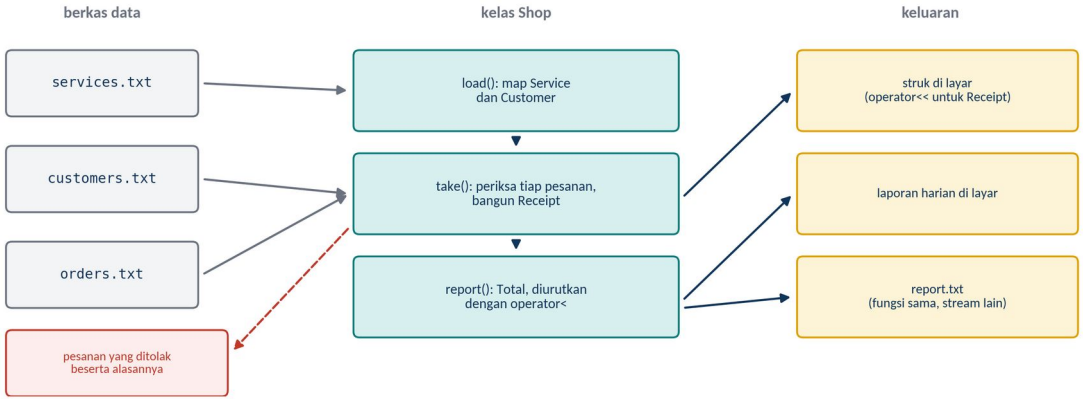
Invarian di konstruktor — penyebut nol melempar eksepsi; tanda penyebut negatif dipindahkan ke pembilang; setiap pecahan disederhanakan.

Itu membuat `==` sepele — pecahan sama dalam bentuk paling sederhana punya pembilang dan penyebut yang sama.

Sengaja tidak `explicit` — `Fraction(long long num = 0, long long den = 1)` mengubah `2` menjadi $\frac{2}{1}$, sehingga `2 + half` berjalan.

Baris-baris pilihan. `Rupiah` menolak konversi; `Fraction` menyambutnya. Putuskan untuk setiap konstruktor satu argumen: apakah setiap nilai tipe parameternya sungguh nilai tipe Anda? Buku: Subbab 15.4.

Studi Kasus Utuh



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`.

Gambar 15.2: rancangan program yang dapat dipakai Bu Rina setiap akhir hari. Buku: Subbab 15.5.

Datanya

`data/services.txt` — kode, nama, satuan (`kg` atau `pcs`), tarif: `W` Wash & fold 7.000/kg, `WI` Wash & iron 10.000/kg, `I` Iron only 5.000/kg, `BC` Bed cover 25.000/potong, `SH` Shoes 35.000/potong.

`data/customers.txt` — id, nama, anggota; anggota mendapat diskon 10%.

`data/orders.txt` — nomor, pelanggan, layanan, jumlah (gram atau potong), ekspres.

Dua baris sengaja salah: pesanan 1006 menyebut pelanggan `C05` yang tidak ada; pesanan 1009 menyebut layanan `DC`.

Daftar harga tidak lagi berada di dalam program: penjahit atau cuci mobil mengubah berkas, bukan kode. Buku: Subbab 15.5.1.

Aturannya

Harga dasar — tarif kali jumlah, dengan pembulatan Bab 13 untuk layanan per kilogram.

Biaya tambahan ekspres — 50% dari harga dasar menurut Persamaan (15.1), untuk pesanan ekspres.

Diskon anggota — 10% dari harga dasar ditambah biaya ekspres, untuk anggota.

$$\text{total} = \text{base} + \text{express} - \text{discount} \tag{15.5}$$

Karena keempatnya objek `Rupiah`, Persamaan (15.5) muncul di program persis sebagai `base + express - discount`. Buku: Subbab 15.5.2.

Programnya

`std::map` dari kode ke `Service` dan dari id ke `Customer`: `find` mengembalikan `end()` untuk kunci yang tidak dikenal.

`take` memeriksa setiap baris pesanan, membangun `Receipt` atau mencatat penolakan beserta alasannya.

`operator<<` mencetak struk; `operator<` pada `Total` mengurutkan laporan, terbesar lebih dulu.

Satu fungsi laporan menerima `std::ostream&`, sehingga kode yang sama menulis ke layar dan ke `report.txt`.

`laundry.cpp` meng-include `rupiah.h` tanpa perubahan. `laundry.py` memakai `@dataclass`, `__str__`, `__lt__`, `dict` dengan `setdefault`, dan fungsi laporan yang menulis ke objek apa pun yang punya metode `write`. Buku: Subbab 15.5.3.

Contoh Terurai 2: Dua Pesanan Melewati `take`

Pesanan 1006 — lima medan, jumlahnya benar. `customers_.find("C05")` melewati C01 sampai C04, tidak menemukan apa-apa, mengembalikan `end()`: ditolak, “unknown customer C05”.

Pesanan 1002 — `1002;C02;WI;5200;yes`. Pak Budi, bukan anggota; cuci-setrika 10.000 per kg.

Dasar: $(5200 \times 10000 + 500)/1000 = 52{,}000$. Ekspres: $(52{,}000 \times 50 + 50)/100 = 26{,}000$. Tanpa diskon.

Total menurut Persamaan (15.5): $52{,}000 + 26{,}000 - 0 = 78{,}000$.

Pesanan yang ditolak tidak pernah menjadi struk; yang baik membawa pointer ke pelanggan dan layanannya. Buku: Subbab 15.5.4.

Struk yang Diperiksa dengan Tangan

```
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
```

Dasar $(2750 \times 10000 + 500)/1000 = 27{,}500$; tambahan $(27{,}500 \times 50 + 50)/100 = 13{,}750$.

Diskon: 10% dari $41{,}250$ adalah $(412{,}500 + 50)/100 = 4{,}125$. Total $37{,}125$.

Struk paling rumit hari itu, dengan ketiga aturan sekaligus. Buku: Subbab 15.6.

Laporan Harian

```
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
Member discounts given          Rp 12.540
GRAND TOTAL                    Rp 272.060
```

Periksa: cuci-setrika $78{,}000+37{,}125=115{,}125$*, yang terbesar, sehingga paling atas. Keempat diskon anggota* $2{,}415+5{,}000+1{,}000+4{,}125=12{,}540$*. C++ dan Python mencetak hal yang sama, dan keduanya menulis* `report.txt`*. Buku: Subbab 15.6–15.7.*

Membuatnya Lebih Baik, Memakainya secara Nyata

Urutan kedua sebagai fungsi pembanding untuk `std::sort`; **pesanan yang ditolak** ke `refused.txt` lewat *stream* kedua.

Kelas `Grams` agar `Rupiah + Grams` tidak dapat dikompilasi; **beberapa hari** dengan kolom tanggal dan *map* dari tanggal ke `Shop`.

Secara nyata — penjahit memasukkan “hem trousers;pcs;15000” ke `services.txt`; cuci mobil, golongan kendaraan; katering, porsi.

Tiga kebiasaan: uang dalam tipe tersendiri; aturan usaha di satu tempat yang terlihat; setiap baris masukan diperiksa dan ditolak beserta alasannya. Buku: Subbab 15.8–15.9.

Kesalahan yang Sering Terjadi

Mengembalikan salinan dari `+=`; `operator+` sebagai anggota (lalu `3 * iron` gagal).

`==` dan `!=` didefinisikan terpisah; `<` yang tidak ketat (`<=` merusak `std::sort`).

Lupa `explicit`; luapan pada `a * b / gcd`; operator dengan makna yang mengejutkan.

Python: kembalikan `NotImplemented` alih-alih menebak bila tipe operannya tidak dikenal.

Buku: Subbab 15.10.

Ringkasan: Lima Hal yang Perlu Diingat

1. `a + b` pada objek adalah panggilan `operator+` (Gambar 15.1); presedensi dan banyak operan tidak pernah berubah.
2. `+=` adalah anggota yang mengembalikan `*this`; `+`, `==`, `<` adalah non-anggota yang dibangun di atasnya; `<<` mengembalikan *stream* (Tabel 15.1).
3. `Rupiah` membulatkan persen dengan satu cara (Persamaan (15.1)) dan menolak `Rupiah * Rupiah` serta konversi diam-diam.
4. `Fraction` tetap dalam bentuk paling sederhana dengan FPB Euclid (Persamaan (15.2)) dan menjumlah dengan KPK (Persamaan (15.3), Persamaan (15.4)).
5. Studi kasusnya membaca berkas, memeriksa setiap pesanan, dan menghitung Persamaan (15.5) secara harfiah (Gambar 15.2).

Python melakukan hal yang sama dengan metode khusus dan `functools.total_ordering`. Buku: Subbab 15.11.

Pertemuan Berikutnya

Bab 16 — Evaluasi dan Pembahasan IV. Lima soal diselesaikan langkah demi langkah: bidak catur, *overloading* `add`, anagram, perkalian matriks, jam pengambilan.

Masing-masing dengan contoh tangan, rancangan, C++ dan Python, serta penyelesaian keliru yang menggoda.

Sebelum itu: Soal Latihan 15.1–15.6.

Pertemuan terakhir memakai seluruh buku. Buku: Subbab 15.12–15.13.

Kuliah 16 — Evaluasi dan Pembahasan IV: Kelas, Overloading, dan Seluruh Buku

Mata kuliah: Pemrograman Dasar (EF234101)

Buku: Irfan Subakti, *Pemrograman Dasar dengan C, C++, dan Python*, Bab 16

Dosen: Irfan Subakti — yifana@gmail.com

Hari ini kita menelusuri lima soal utuh yang memakai C++ dari Bab 13–15 dan fondasi seluruh buku, masing-masing dengan contoh tangan, rancangan, program, dan penyelesaian keliru yang menggoda yang dibongkar. Buku: Bab 16, Subbab 16.1–16.11.

Cakupan Pertemuan Ini

Metodenya — tujuh langkah, dari menyatakan ulang soal sampai mengubah satu hal dan meramalkan hasilnya.

16.1 Bidak catur — melengkapi hierarki kelas dengan meniru; *exclusive or* untuk benteng.

16.2 *Overloading* `add` — dua sampai lima argumen, dan mengapa argumen bawaan di samping *overload* gagal.

16.3 Anagram — setiap jendela dalam teks, dan batas perulangan yang menghilangkan jendela terakhir.

16.4 Perkalian matriks dan **16.5 Waktu pengambilan** — pemeriksaan ukuran; kelas `Time` dengan invarian.

Setiap penyelesaian dalam C++ dan Python, keluarannya sama. Buku: Bab 16.

Posisi Kita: Kotak Perkakasnya

Bab 1–12 — tipe, perulangan, fungsi, larik, pointer, string, struktur, senarai, berkas, dan kebiasaan menghitung dengan tangan lebih dulu.

Bab 13–15 — string, *vector*, referensi, dan *overloading* C++; kelas dengan invarian; operator.

16.1 memerlukan pewarisan, 16.2 *overloading*, 16.3 string dan batas perulangan yang tepat, 16.4 data dua dimensi, 16.5 kelas dengan operator.

Buku: Subbab 16.1.

Tujuh Langkah

1. Nyatakan ulang — masukan, keluaran, setiap keputusan yang dibiarkan terbuka. **2. Contoh dengan tangan** — contoh dari soal, dan kasus tepi buatan sendiri.

3. Rancang di kertas — data, *pseudocode*, biaya. **4. Tulis programnya** — lengkap, tanpa peringatan.

5. Jalankan contoh tangan — sama karakter demi karakter. **6. Coba patahkan** — penyelesaian keliru yang menggoda.

7. Ubah satu hal dan ramalkan — kotak “periksa diri”, dijawab *sebelum* menjalankan apa pun.

Metode pemecahan masalah Pólya yang disesuaikan untuk program. Buku: Subbab 16.2.

Soal 16.1: Bidak Catur

> **SOAL 16.1.** Kelas dasar `Piece` menyimpan nama bidak, pihaknya, dan posisinya serta mencetak posisi itu; kelas `Pawn` (pion) yang lengkap sudah diberikan. 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.

Keputusan kita: koordinat $0 \le x, y \le 7$; `faction_` bernilai $+1$ untuk putih, -1 untuk hitam; setiap bidak diuji sendirian di papan; gerakan yang ditolak mencetak `Invalid movement!!` dan bidak tetap di tempatnya.

Kreativitas diletakkan pada syarat-syaratnya, bukan pada strukturnya: kelas-kelas baru meniru `Pawn` persis. Buku: Subbab 16.3.1.

Lima Aturan sebagai Lima Syarat

Dengan $d_x = x' - x$ dan $d_y = y' - y$: **Pion** $d_x = 0$ dan $d_y = \text{faction}$. **Gajah** $|d_x| = |d_y|$ dan $d_x \neq 0$.

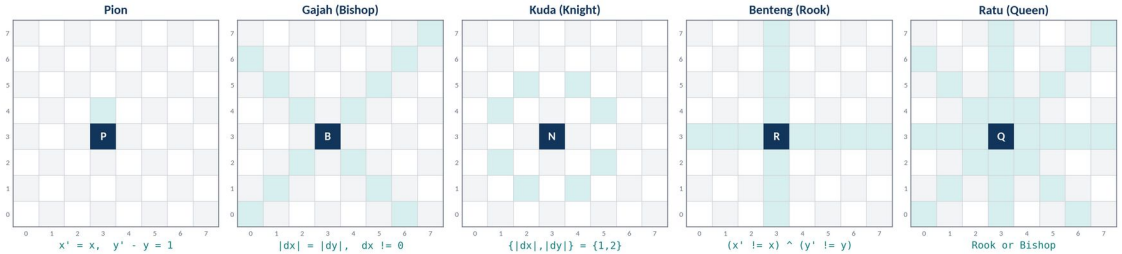
Kuda $\{|d_x|, |d_y|\} = \{1, 2\}$. **Ratu** seperti benteng atau seperti gajah.

Benteng — *tepat satu* dari x dan y berubah: *exclusive or*.

$$\text{rook}(x, y \to x', y') = (x' \neq x) \oplus (y' \neq y) \tag{16.1}$$

Persamaan (16.1): di C++ dan Python, `^` *yang dikenakan pada dua nilai* `bool` *persis sama dengan* $\oplus$*. Buku: Subbab 16.3.2.*

Petak-petak dari (3, 3)



Gambar 16.1 Petak-petak yang dapat dicapai setiap bidak dari (3, 3) di papan kosong, beserta syarat yang menentukannya.

Gambar 16.1: periksa beberapa petak terhadap syaratnya dengan pensil sebelum menulis kode. Buku: Subbab 16.3.2.

Satu Gerakan untuk Semua Bidak

Pseudokode 16.1 Gerakan yang dipakai bersama oleh 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 berada di `Piece` dan memanggil `moveCondition` virtual yang ditimpa setiap kelas turunan (Bab 14). Setiap kelas punya konstruktor sendiri, karena konstruktor tidak diwariskan. Buku: Subbab 16.3.3.

Contoh Terurai 1: Gajah dan Benteng

Gajah dari $(0,0)$ ke $(3,3)$: $|d_x|=|d_y|=3$, $d_x \neq 0$: berpindah.

$(3,3) \to (3,5)$: $0 \neq 2$: tidak sah, ia tetap. $(3,3) \to (5,1)$: $2=2$: berpindah.

Benteng dari $(0,0)$ ke $(0,6)$: hanya y berubah: berpindah. $(0,6) \to (3,3)$: keduanya berubah, $\text{true} \oplus \text{true}$ bernilai salah: tidak sah.

$(0,6) \to (7,6)$: berpindah. $(7,6) \to (8,6)$: di luar papan: tidak sah.

`chess.cpp` *mencetak persis putusan-putusan ini, dan* `chess.py` *juga. Buku: Subbab 16.3.2 dan 16.3.4.*

Penyelesaian Keliru yang Menggoda: `||` untuk Benteng

Kode 16.1 Dari `wrong_rook.cpp`: syarat benteng dengan `||` dan syarat gajah tanpa $d_x \neq 0$.

```
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
}
```

```
rook   (0,6) -> (3,3): 1
rook   (0,6) -> (5,1): 1
bishop (3,3) -> (3,3): 1
```

Kode 16.1: "sepanjang baris atau kolom" bukan `||`; pada diagonal kedua koordinat berubah dan `||` menerimanya. Gajah yang keliru menerima diam di tempat. Periksa diri: syarat raja adalah `max(|dx|, |dy|) == 1`. Buku: Subbab 16.3.6.

Soal 16.2: *Overloading* `add`

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.

$$\text{add}(a_1,\ldots,a_k) = \text{add}(\text{add}(a_1,\ldots,a_{k-1}),a_k) \tag{16.2}$$

Persamaan (16.2): setiap *overload* dibangun di atas yang di bawahnya. Kantong-kantong cucian seberat 3.450, 1.200, 5.000, 2.750, dan 4.100 g; jumlahnya 4.650, 9.650, 12.400, dan 16.500.

Overload resolution memilih menurut banyaknya argumen, sehingga keempat fungsi dapat hidup berdampingan (Subbab 13.7). Buku: Subbab 16.4.1–16.4.3.

Penyelesaiannya, dan Lebih Jauh

Kode 16.2 Dari `add_overload.cpp`: `add` untuk dua sampai lima bilangan bulat, untuk dua `double`, dan *variadic template*.

```cpp
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
}
```

Kode 16.2: overload `double` punya banyak argumen yang sama tetapi tipenya lain; fold expression memperlihatkan batas overloading. Python cukup satu fungsi, `def add(*values)`. Buku: Subbab 16.4.4–16.4.5.

Penyelesaian Keliru yang Menggoda: Argumen Bawaan di Samping *Overload*

Kode 16.3 Dari `wrong_add.cpp`: `add` berargumen bawaan di samping yang sudah diberikan; pemanggilan dengan dua argumen menjadi ambigu.

```
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;
}
```

`add(1, 2, 3)` baik-baik saja: hanya yang kedua cocok. `add(1, 2)`: keduanya cocok persis, dan `g++` berhenti dengan **call of overloaded ‘add(int, int)’ is ambiguous**, menyebut kedua kandidat.

Kode 16.3: sebagai pengganti fungsi yang diberikan, versi berargumen bawaan itu rapi; di sampingnya, ia galat. Buku: Subbab 16.4.6.

Soal 16.3: Mencari Anagram

> **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.

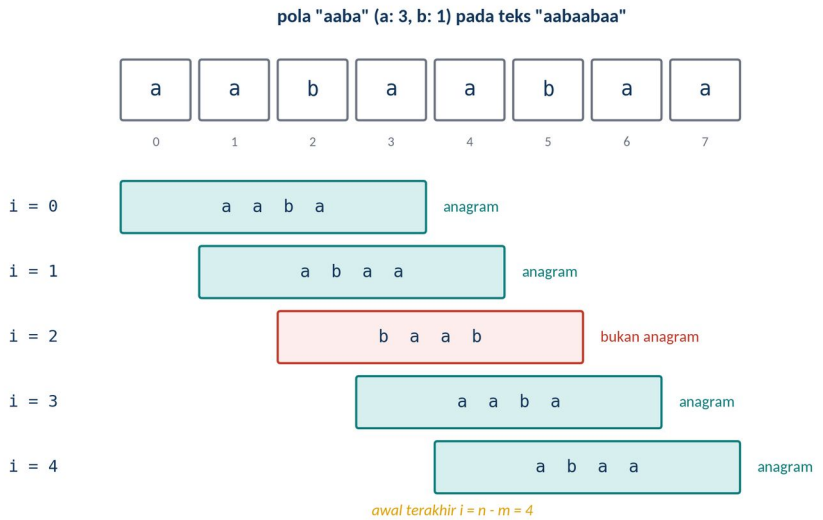
Anagram memakai huruf yang sama dengan banyak yang sama: *kasur* dan *rusak*.

$$W = n - m + 1 \tag{16.3}$$

Persamaan (16.3): teks sepanjang n punya W jendela sepanjang m, yang terakhir dimulai di $n - m$, asalkan $m \le n$.

Contoh buatan kita: `wash` dalam `washwash`, tempat setiap jendela adalah anagram, dan `aab` dalam `ab`, yang tidak boleh menghasilkan apa-apa dan tidak boleh membuat program jatuh. Buku: Subbab 16.5.1–16.5.2.

Contoh Terurai 2: Kelima Jendela



Gambar 16.2 Kelima jendela sepanjang 4 dalam `aabaabaa`: empat di antaranya anagram dari `aaba`; jendela terakhir dimulai di $n - m = 4$.

Gambar 16.2: $n = 8$, $m = 4$, $W = 5$. `aaba` punya tiga `a` dan satu `b`; begitu pula jendela di 0, 1, 3, dan 4; jendela di 2, `baab`, punya dua `b`. Jawaban soal terkonfirmasi. Buku: Subbab 16.5.2.

Mencari Anagram, di Atas Kertas

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
```

Pseudokode 16.2: sampai $O\big((n-m+1)\,m^2\big)$ *dengan* `is_permutation`*. Hitungan bergeser menyimpan 256 penghitung untuk jendela saat ini dan memperbarui dua di antaranya per langkah,* $O(n+m)$*; program menjalankan kedua cara dan memeriksa bahwa hasilnya sama. Buku: Subbab 16.5.3.*

Penyelesaian Keliru yang Menggoda: Kurang Satu Jendela

```
[0, 1, 3]
[0, 1, 2, 3]
```

`range(n - m)` di Python, atau `i < n - m` di C++, berhenti di $n - m - 1$: temuan di indeks 4 hilang pada `aabaabaa` dan pada `washwash`.

Tidak ada peringatan apa pun; hanya contoh dari soal itu sendiri, yang menyebut indeks 4, yang membongkarnya.

> **PERINGATAN:** Di C++, `text.size()` bertipe `std::size_t` yang *unsigned*. Bila pola lebih panjang daripada teks, `n - m` tidak menjadi -1; ia berputar balik menjadi sekitar 1.8×10^{19}, dan perulangan membaca jauh melewati ujung string. Jaga $m > n$ sebelum mengurangkan.

Dari `wrong_anagram.py`. Periksa diri: `aa` dalam `aaaa` punya $W = 3$ jendela, tiga temuan. Buku: Subbab 16.5.6.

Soal 16.4: Perkalian Matriks

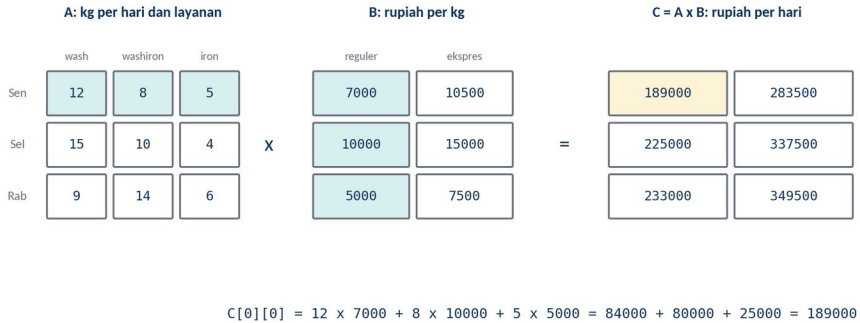
SOAL 16.4. Bacalah dua matriks A dan B. Bila banyaknya kolom A sama dengan banyaknya baris B, hitung dan cetak $C = A \times B$; bila tidak, cetak pesan bahwa kedua matriks tidak dapat dikalikan. Sebuah program parsial yang disediakan belum memberi hasil yang benar; perbaikilah.

$$C_{ij} = \sum_{k=0}^{m-1} A_{ik} B_{kj}, 0 \le i < n, 0 \le j < p \tag{16.4}$$

$$\text{cols}(A) = \text{rows}(B) \tag{16.5}$$

Persamaan (16.4) untuk setiap elemen; Persamaan (16.5) adalah pemeriksaannya: $(n \times m) \times (m \times p) = (n \times p)$*, kedua bilangan dalam sama dan hilang seperti kartu domino. Buku: Subbab 16.6.1–16.6.2.*

Kilogram Harian Dikali Harga



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.

Gambar 16.3: A berisi kilogram per hari (baris) dan layanan (kolom); B harga per kilogram, biasa dan ekspres. Baris Senin adalah $(12, 8, 5)$: $C_{00} = 12 \cdot 7{,}000 + 8 \cdot 10{,}000 + 5 \cdot 5{,}000 = 189{,}000$. Buku: Subbab 16.6.2.

Perulangan Bersarang Tiga

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
```

Pseudokode 16.3: $O(nmp)$. *Matriks adalah* `std::vector<std::vector<long long>>`; `a[i][k]` *adalah baris* i, *kolom* k. *Carilah pemeriksaan yang hilang, jumlah yang tidak mulai dari 0, atau indeks yang tertukar. Dengan tangan:* $\begin{pmatrix}1&2&3\\4&5&6\end{pmatrix}\times\begin{pmatrix}7&8\\9&10\\11&12\end{pmatrix}=\begin{pmatrix}58&64\\139&154\end{pmatrix}$.

Buku: Subbab 16.6.3.

Penyelesaian Keliru yang Menggoda: `b[j][k]`

```
A x I = [2 3] [5 7]
A x B = [8 18] [19 43]
```

Indeks yang tertukar mengalikan dengan *transpos* B. Matriks identitas adalah transposnya sendiri, sehingga uji identitas **lolos**.

Contoh tangan yang tak simetris membongkarnya: $\begin{pmatrix} 2 & 3 \\ 5 & 7 \end{pmatrix} \times \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}$ seharusnya diawali $2+9=11$, bukan 8.

Dari `wrong_matrix.cpp`*. Periksa diri: baris Rabu* $(9, 14, 6)$ *memberi* $C_{20}=233,000$ *dan* $C_{21}=349,500$*;* 3×3 *kali* 2×3 *mencetak "cannot multiply". Buku: Subbab 16.6.6.*

Soal 16.5: Kapan Bisa Diambil?

> **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` dengan `+`, `-`, `<`, `==`, dan pencetakan dalam bentuk `Mon 09:15`; untuk setiap pesanan cetak waktu penitipan, selesai, pengambilan, dan lama menunggu, lalu daftar pesanan menurut urutan pengambilan.

Keputusan kita: hari 0 adalah Senin; cucian yang selesai tepat pukul 20.00 masih dapat diambil pukul 20.00; jam di luar 0–23 atau menit di luar 0–59 ditolak.

Buku: Subbab 16.7.1.

Invarian Kelas `Time`

$$\text{day}' = \text{day} + \left\lfloor \frac{\text{minute} + \Delta}{1440} \right\rfloor, \quad \text{minute}' = (\text{minute} + \Delta) \bmod 1440 \tag{16.6}$$

Simpan nomor hari dan menit sesudah tengah malam, $0 \le \text{minute} < 1440$; Persamaan (16.6) memulihkannya sesudah setiap penjumlahan.

Selisih: $1440 \cdot (\text{day}_a - \text{day}_b) + (\text{minute}_a - \text{minute}_b)$; perbandingan adalah selisih itu terhadap nol.

Pak Budi, cuci setrika ekspres, Senin 14.40: $720/2 = 360$; $880 + 360 = 1240$ adalah 20.40, sesudah tutup: Selasa 08.00, menunggu 17 jam 20 menit.

`+=` *adalah anggota yang menerapkan Persamaan (16.6);* `+` *non-anggota yang dibangun di atasnya;* `std::stable_sort` *mengurutkan pesanan menurut waktu pengambilan dan menjaga urutan asal untuk waktu yang sama. Buku: Subbab 16.7.2–16.7.3.*

Menghitung Waktu Pengambilan

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
```

```
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)
1005 Pak Hendra in Sun 18:00  ready Mon 06:00  pick up Mon 08:00  (14 h 0 min)
Pick-up order: 1004 1001 1002 1003 1005
Time(0, 24, 0) refused: hour must be 0..23
```

Pseudokode 16.4 dan baris-baris pilihan dari `pickup_time.cpp`: pesanan Minggu milik Pak Hendra melintas ke hari 7, yaitu Senin, menurut Persamaan (16.6), lalu menunggu toko buka. Buku: Subbab 16.7.3–16.7.4.

Penyelesaian Keliru yang Menggoda: Tanpa Simpanan

```
14:400 19:210
b ready before closing? True
```

Jam dan menit disimpan terpisah, menit dijumlahkan tanpa simpanan: cucian Pak Budi “selesai pukul 14:400”.

“19:210” dianggap sebelum tutup, karena perbandingan melihat jamnya lebih dulu.

Perbaikannya bukan perbandingan yang lebih pintar, melainkan representasi dengan invarian.

Dari `wrong_time.py`. Periksa diri: setrika ekspres Jumat pukul 18.30 selesai pukul 20.00 dan langsung diambil; pukul 18.31 ia menunggu sampai Sabtu 08.00, 13 jam 29 menit. Buku: Subbab 16.7.6.

Kelima Soal Berdampingan

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	$O(n\,m\,p)$	`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

Tabel 16.1, kolom terakhir: setiap jebakan ditangkap oleh contoh tertentu yang dipilih pada langkah 2. Contoh yang Anda pilih sebelum menulis kode menentukan kesalahan mana yang akan Anda temukan. Buku: Subbab 16.8.

Kesalahan yang Sering Terjadi

Aturan diterjemahkan dari kalimat yang longgar (“atau” bukan `||`); **mengubah kode yang diberikan alih-alih menirunya**.

Argumen bawaan dicampur dengan *overload*; **selisih satu dalam perulangan jendela**, dan pengurangan *unsigned*.

Menguji hanya dengan data simetris; **jumlah yang tidak diinisialisasi 0**.

Representasi yang memungkinkan keadaan tidak sah; ***stream*** **yang ditinggalkan dengan** `fill('0')` **atau** `std::left` **masih terpasang**.

Buku: Subbab 16.10.

Ringkasan: Lima Hal yang Perlu Diingat

1. Tujuh langkah: nyatakan ulang, contoh dengan tangan, rancang, tulis kode, jalankan contohnya, coba patahkan, ubah satu hal dan ramalkan.

2. Tiru hierarki yang diberikan; tulis setiap aturan sebagai syarat yang tepat, *exclusive or* untuk benteng (Persamaan (16.1)).

3. Bangun *overload* di atas satu sama lain (Persamaan (16.2)); jangan letakkan argumen bawaan di samping *overload* yang menerima pemanggilan yang sama.

4. Hitung jendela dengan tepat (Persamaan (16.3)); periksa ukuran sebelum mengalikan (Persamaan (16.5)); uji dengan data tak simetris.

5. Pilih representasi dengan invarian (Persamaan (16.6)), maka setiap operasi menjadi satu baris.

Buku: Subbab 16.11.

Sesudah Mata Kuliah Ini

Soal Latihan 16.1–16.6 — `King`, `Board` yang menghalangi lintasan, `add` untuk `Rupiah`, anagram tanpa membedakan huruf besar-kecil, `transpose`, jam buka hari Minggu yang sebenarnya.

Kasir tumbuh dari `printf` pertama menjadi program lengkap dengan kelas, berkas, dan laporan harian, dalam tiga bahasa.

Mata kuliah berikutnya: Struktur Data — senarai, pohon, dan antrean dari Bab 10 dan 12, dipelajari mendalam beserta biayanya.

Metodenya bertahan lebih lama daripada bahasa mana pun: nyatakan ulang, kerjakan contoh dengan tangan, rancang sebelum menulis kode, dan simpan contoh-contoh itu sebagai uji. Buku: Subbab 16.12–16.13.