**Kurikulum C++: Dari Pemula sampai Pro**

Selamat datang! Dokumen ini menyajikan materi C++ dari nol hingga mahir, dengan contoh kode, penjelasan mendalam, latihan, dan mini‑project. Ikuti berurutan; setiap modul berdiri sendiri tapi saling terkait.

**Peta Jalan (Roadmap Singkat)**

1. **Dasar Bahasa**: sintaks, tipe data, operator, kontrol alur, fungsi.
2. **Struktur Data Dasar**: array, std::vector, std::string.
3. **Pointer & Reference**, memori dinamis, RAII.
4. **Pemrograman Berorientasi Objek (OOP)**: class, ctor/dtor, copy/move, pewarisan, polimorfisme.
5. **Template & Generics**: function/class template, SFINAE, concepts (C++20).
6. **STL (Standard Library)**: container, iterator, algoritma, lambda, smart pointer.
7. **Error Handling**: exception, std::expected (C++23, konsep), alternatif berbasis hasil.
8. **I/O & File**: stream, format, serialisasi sederhana.
9. **Konkruensi**: thread, async, future, mutex, atomic.
10. **Fitur Modern C++**: range, structured bindings, optional/variant, coroutines (gambaran).
11. **Praktik Rekayasa Perangkat Lunak**: CMake, layout proyek, testing, debugging, profiling.
12. **Mini‑Projects & Capstone**.

Catatan: Materi menargetkan C++17/20. Jika kompiler Anda belum mendukung penuh C++20, gunakan C++17 untuk sebagian besar contoh.

**0. Setup Lingkungan**

**Pilihan Kompiler (semua gratis)**

* **Windows**:
  + *MSVC* (Visual Studio/Build Tools),
  + *MSYS2 MinGW* (g++),
  + *LLVM* (clang++).
* **macOS**: *clang++* via Xcode Command Line Tools.
* **Linux**: g++ atau clang++ dari package manager.

**Cek versi**

g++ --version

clang++ --version

**Kompilasi cepat dari terminal**

# C++17

g++ -std=c++17 hello.cpp -o hello

./hello

# C++20

g++ -std=c++20 main.cpp -O2 -Wall -Wextra -pedantic -o app

**Struktur Proyek Minimal**

project/

├─ src/

│ └─ main.cpp

├─ include/

│ └─ app.hpp

└─ CMakeLists.txt (opsional)

**1. Dasar Bahasa C++**

**1.1 Program Pertama**

#include <iostream>

int main() {

std::cout << "Halo, C++!\n";

return 0; // 0 = sukses

}

**Penjelasan**: #include menarik deklarasi std::cout; main titik masuk program.

**1.2 Tipe Data & Variabel**

#include <cstdint>

int a = 42; // int

double pi = 3.14159; // floating point

char c = 'A'; // karakter

bool ok = true; // boolean

std::int64\_t big = 9'000'000'000; // integral eksplisit

**Konstanta & auto**

const double g = 9.8;

auto x = 3.5; // infer: double

auto y = 42; // infer: int

**1.3 Operator & Ekspresi**

Aritmatika, relasional (==, !=, <, >), logika (&&, ||, !), assignment (+=, -=), ternary (cond ? a : b).

**1.4 Kontrol Alur**

if (x > 10) { /\*...\*/ }

else if (x == 10) { /\*...\*/ }

else { /\*...\*/ }

for (int i = 0; i < 5; ++i) { /\*...\*/ }

int i = 0;

while (i < 5) { /\*...\*/ ++i; }

do { /\*...\*/ } while(false);

switch (c) {

case 'A': /\*...\*/ break;

default: /\*...\*/

}

**1.5 Fungsi**

#include <string>

int tambah(int a, int b) { return a + b; }

std::string greet(const std::string& name) {

return "Hi, " + name;

}

**Parameter**: by value, by reference (&), by const reference (const &).

**1.6 Namespace & Header**

// file: mathx.hpp

#pragma once

int kali(int, int);

// file: mathx.cpp

#include "mathx.hpp"

int kali(int a, int b) { return a \* b; }

**Latihan 1**: Buat fungsi max3(a,b,c) yang mengembalikan nilai terbesar tanpa std::max.

**2. Struktur Data Dasar**

**2.1 Array & std::vector**

#include <vector>

#include <iostream>

int main(){

std::vector<int> v = {1,2,3};

v.push\_back(4);

for (int x : v) std::cout << x << ' ';

}

**Kenapa vector?** Ukuran dinamis, aman, mendukung iterator & algoritma STL.

**2.2 std::string**

#include <string>

#include <iostream>

int main(){

std::string s = "C++";

s += " Modern";

std::cout << s << "\n";

}

**2.3 std::array & std::pair/std::tuple**

#include <array>

#include <tuple>

std::array<int,3> a{1,2,3};

auto [x,y,z] = std::tuple{10,20,30}; // structured binding (C++17)

**Latihan 2**: Hitung frekuensi huruf pada string (abaikan spasi & case-insensitive) menggunakan std::unordered\_map<char,int>.

**3. Pointer, Reference, dan Memori**

**3.1 Reference (&) vs Pointer (\*)**

* **Reference**: alias wajib terikat ke objek; tidak bisa null.
* **Pointer**: menyimpan alamat; bisa nullptr.

int n = 10;

int& ref = n; // reference

int\* p = &n; // pointer

\*p = 20; // ubah n melalui pointer

**3.2 Memori Dinamis & RAII**

Hindari new/delete manual; gunakan smart pointer.

#include <memory>

struct Node { int value; };

auto sp = std::make\_shared<Node>(Node{42});

auto up = std::make\_unique<Node>(Node{7});

**3.3 Rule of 0/3/5/0+**

* **Rule of 0**: Lebih baik biarkan compiler generate special member functions.
* **Rule of 3/5**: Jika Anda mendefinisikan dtor/copy/move, definisikan yang relevan lainnya.

**Latihan 3**: Implementasi stack sederhana menggunakan std::vector dan std::unique\_ptr untuk buffer opsional.

**4. OOP (Object-Oriented Programming)**

**4.1 Class & Konstruktor**

#include <string>

#include <iostream>

class Account {

std::string owner\_;

double balance\_{}; // inisialisasi default

public:

Account(std::string owner, double balance)

: owner\_(std::move(owner)), balance\_(balance) {}

void deposit(double amount) { balance\_ += amount; }

bool withdraw(double amount) {

if (amount > balance\_) return false;

balance\_ -= amount; return true;

}

double balance() const { return balance\_; }

const std::string& owner() const { return owner\_; }

};

int main(){

Account a{"Lutfi", 100.0};

a.deposit(50);

std::cout << a.owner() << ": " << a.balance() << "\n";

}

**4.2 Pewarisan & Polimorfisme**

#include <iostream>

struct Shape {

virtual ~Shape() = default; // virtual destructor

virtual double area() const = 0; // pure virtual

};

struct Rect : Shape {

double w, h;

Rect(double w, double h) : w(w), h(h) {}

double area() const override { return w\*h; }

};

struct Circle : Shape {

double r;

explicit Circle(double r): r(r) {}

double area() const override { return 3.14159\*r\*r; }

};

int main(){

Shape\* s1 = new Rect(3,4);

Shape\* s2 = new Circle(2);

std::cout << s1->area() << ", " << s2->area() << "\n";

delete s1; delete s2; // Prefer smart pointers dalam praktik

}

**4.3 Overloading & Overriding, final, override**

Gunakan override untuk keamanan, final untuk mencegah pewarisan lebih lanjut.

**Latihan 4**: Buat hierarki Employee → Salaried, Hourly dengan metode pay() polimorfik.

**5. Template & Generics**

**5.1 Fungsi Template**

template <typename T>

T add(T a, T b) { return a + b; }

**5.2 Class Template & Partial Specialization (gambaran)**

template <typename T>

class Box { T value\_; public: explicit Box(T v): value\_(std::move(v)) {} };

**5.3 Concepts (C++20)**

#include <concepts>

template <std::integral I>

I gcd(I a, I b) {

while (b != 0) { auto t = a % b; a = b; b = t; }

return a;

}

**Latihan 5**: Tulis minmax(T a, T b) yang mengembalikan std::pair<T,T> dengan *requires clause* untuk std::totally\_ordered.

**6. STL: Container, Iterator, Algoritma, Lambda**

**6.1 Container Inti**

* **Sequence**: vector, deque, list (jarang dipakai kecuali perlu splice)
* **Associative**: set, map, multiset, multimap
* **Unordered** (hash): unordered\_set, unordered\_map

**6.2 Algoritma & Iterator**

#include <algorithm>

#include <vector>

#include <iostream>

int main(){

std::vector<int> v{9,3,7,1,5};

std::sort(v.begin(), v.end());

if (std::binary\_search(v.begin(), v.end(), 7)) std::cout << "Ada 7\n";

}

**6.3 Lambda & std::function**

#include <algorithm>

#include <vector>

std::vector<int> v{1,2,3,4,5};

auto odd = [](int x){ return x % 2; };

v.erase(std::remove\_if(v.begin(), v.end(), odd), v.end());

**6.4 Smart Pointer**

std::unique\_ptr, std::shared\_ptr, std::weak\_ptr – pilih paling ketat (unique\_ptr) bila memungkinkan.

**Latihan 6**: Sorting daftar siswa berdasarkan skor desc lalu nama asc menggunakan std::sort dengan comparator lambda.

**7. Penanganan Error**

**7.1 Exception**

#include <stdexcept>

double safe\_div(double a, double b){

if (b == 0.0) throw std::runtime\_error("divide by zero");

return a/b;

}

Tangkap selektif, hindari catch(...) kecuali di boundary.

**7.2 Alternatif Tanpa Exception**

* Nilai kembali std::optional<T> jika *mungkin tidak ada nilai*.
* Gunakan hasil terstruktur: struct { T value; std::string error; }.
* C++23 mengenalkan std::expected<T,E> (konsep); banyak implementasi pihak ketiga di C++20.

**Latihan 7**: Parsel integer dari string menjadi std::optional<int>.

**8. I/O & File**

#include <fstream>

#include <string>

int main(){

std::ofstream out{"data.txt"};

out << "Halo" << '\n';

std::ifstream in{"data.txt"};

std::string line;

while (std::getline(in, line)) {/\* proses \*/}

}

**Formatting Modern**

* std::format (C++20/23) atau fmt library: std::format("Hello {}", name).

**Latihan 8**: Simpan dan baca kembali daftar transaksi (CSV sederhana).

**9. Konkruensi & Paralelisme**

**9.1 Thread Dasar**

#include <thread>

#include <iostream>

void work(){ std::cout << "hi\n"; }

int main(){

std::thread t{work};

t.join(); // tunggu selesai

}

**9.2 Sinkronisasi**

std::mutex, std::lock\_guard, std::unique\_lock, std::condition\_variable.

#include <mutex>

#include <condition\_variable>

std::mutex m;

std::condition\_variable cv;

int data; bool ready=false;

void producer(){

{

std::lock\_guard<std::mutex> lk(m);

data = 42; ready = true;

}

cv.notify\_one();

}

void consumer(){

std::unique\_lock<std::mutex> lk(m);

cv.wait(lk, []{return ready;});

// gunakan data

}

**9.3 async/future & Atomics**

#include <future>

auto fut = std::async(std::launch::async, []{ return 40+2; });

int result = fut.get();

std::atomic<int> untuk operasi thread‑safe pada primitive.

**Latihan 9**: Hitung jumlah kata file besar dengan *worker threads* (bagi per potongan).

**10. Fitur Modern (C++17/20/23 Gambaran)**

* **Structured bindings**: auto [a,b] = pair;
* **if/switch dengan init**: if (auto it = map.find(k); it!=map.end()) {...}
* **std::optional, std::variant, std::any**.
* **Range (C++20)**: std::ranges::views::filter/transform.
* **Concepts**: batasan tipe generik.
* **constexpr lebih kuat**: komputasi saat kompilasi.
* **Coroutines (C++20)**: dasar async/generator (butuh runtime/lib pendukung).

**Contoh Range**

#include <ranges>

#include <vector>

#include <iostream>

int main(){

std::vector<int> v{1,2,3,4,5};

auto squared\_even = v | std::views::filter([](int x){return x%2==0;})

| std::views::transform([](int x){return x\*x;});

for (int x : squared\_even) std::cout << x << ' ';

}

**11. Rekayasa Perangkat Lunak**

**11.1 CMake Dasar**

cmake\_minimum\_required(VERSION 3.20)

project(App LANGUAGES CXX)

set(CMAKE\_CXX\_STANDARD 20)

add\_executable(app src/main.cpp)

**11.2 Layout Proyek**

project/

├─ CMakeLists.txt

├─ include/ (header publik)

├─ src/ (implementasi)

├─ tests/ (unit test)

└─ third\_party/

**11.3 Testing (contoh Catch2 minimal)**

#define CATCH\_CONFIG\_MAIN

#include <catch2/catch.hpp>

TEST\_CASE("tambah"){ REQUIRE(1+2==3); }

**11.4 Debugging & Tools**

* **Compiler warnings**: -Wall -Wextra -Wpedantic.
* **Sanitizers**: -fsanitize=address,undefined.
* **Profiling**: perf (Linux), Instruments (macOS), VTune (Intel), chrono untuk micro-benchmark sederhana.

**12. Pola Desain & Praktik Terbaik**

* **RAII**: bungkus resource di class yang mengelola ctor/dtor.
* **PIMPL**: sembunyikan detail implementasi.
* **Non‑copyable**: = delete untuk ctor copy bila perlu.
* **Prefer komposisi daripada pewarisan**.
* **Zero‑cost abstractions**: gunakan template & inline untuk tanpa overhead.
* **Jangan premature optimization**; ukur dulu.

**13. Mini‑Projects**

**13.1 CLI Bank Sederhana (OOP + File I/O)**

**Fitur**: buat akun, setor, tarik, simpan ke file CSV.

// bank.hpp

#pragma once

#include <string>

#include <unordered\_map>

#include <optional>

struct Account { std::string owner; double balance{}; };

class Bank {

std::unordered\_map<std::string, Account> db; // key: owner

public:

bool create(const std::string& owner, double initial);

std::optional<double> deposit(const std::string& owner, double amt);

std::optional<double> withdraw(const std::string& owner, double amt);

std::optional<Account> get(const std::string& owner) const;

bool load(const std::string& path);

bool save(const std::string& path) const;

};

// bank.cpp

#include "bank.hpp"

#include <fstream>

#include <sstream>

bool Bank::create(const std::string& owner, double initial){

return db.emplace(owner, Account{owner, initial}).second;

}

std::optional<double> Bank::deposit(const std::string& owner, double amt){

auto it = db.find(owner); if (it==db.end()) return std::nullopt;

it->second.balance += amt; return it->second.balance;

}

std::optional<double> Bank::withdraw(const std::string& owner, double amt){

auto it = db.find(owner); if (it==db.end() || it->second.balance < amt) return std::nullopt;

it->second.balance -= amt; return it->second.balance;

}

std::optional<Account> Bank::get(const std::string& owner) const{

if (auto it=db.find(owner); it!=db.end()) return it->second; else return std::nullopt;

}

bool Bank::load(const std::string& path){

std::ifstream in(path); if(!in) return false; db.clear();

std::string line; while(std::getline(in,line)){

std::istringstream ss(line); std::string owner; double bal;

if(std::getline(ss, owner, ',') && (ss>>bal)) db[owner] = Account{owner,bal};

}

return true;

}

bool Bank::save(const std::string& path) const{

std::ofstream out(path); if(!out) return false;

for(auto& [\_, acc] : db) out << acc.owner << "," << acc.balance << "\n";

return true;

}

// main.cpp

#include <iostream>

#include "bank.hpp"

int main(){

Bank bank; bank.load("data.csv");

int menu; std::string name; double amt;

do{

std::cout << "1.Buat 2.Setor 3.Tarik 4.Cek 5.Simpan 0.Keluar\n";

std::cin >> menu;

switch(menu){

case 1: std::cout << "Nama & saldo awal: "; std::cin >> name >> amt;

std::cout << (bank.create(name,amt)?"OK\n":"Gagal\n"); break;

case 2: std::cout << "Nama & jumlah: "; std::cin >> name >> amt;

if(auto b=bank.deposit(name,amt)) std::cout<<"Saldo:"<<\*b<<"\n"; else std::cout<<"Akun?\n"; break;

case 3: std::cout << "Nama & jumlah: "; std::cin >> name >> amt;

if(auto b=bank.withdraw(name,amt)) std::cout<<"Saldo:"<<\*b<<"\n"; else std::cout<<"Gagal\n"; break;

case 4: std::cout << "Nama: "; std::cin >> name;

if(auto a=bank.get(name)) std::cout<<a->owner<<":"<<a->balance<<"\n"; else std::cout<<"Akun?\n"; break;

case 5: std::cout << (bank.save("data.csv")?"Tersimpan\n":"Gagal simpan\n"); break;

}

} while(menu!=0);

}

**13.2 Analyzer Teks (STL + Algoritma)**

Hitung 10 kata paling sering dari file.

#include <fstream>

#include <string>

#include <unordered\_map>

#include <vector>

#include <algorithm>

#include <iostream>

int main(){

std::ifstream in{"book.txt"};

std::unordered\_map<std::string,int> freq; std::string w;

while (in >> w) ++freq[w];

std::vector<std::pair<std::string,int>> v(freq.begin(), freq.end());

std::partial\_sort(v.begin(), v.begin()+std::min<size\_t>(10, v.size()), v.end(),

[](auto& a, auto& b){ return a.second > b.second; });

for (size\_t i=0;i<std::min<size\_t>(10,v.size());++i)

std::cout << v[i].first << ": " << v[i].second << '\n';

}

**13.3 Multithreaded Image Blur (Konsep)**

Pisah gambar PGM menjadi beberapa *stripe* dan proses paralel dengan std::thread (kode lengkap di luar ruang ini; fokus pada partisi dan join thread).

**14. Capstone: Task Manager CLI (End‑to‑End)**

**Tujuan**: Mempraktikkan OOP, STL, file I/O, exception, parsing argumen, dan unit test.

**Fitur**:

* Tambah tugas: judul, prioritas (1–5), due date (YYYY‑MM‑DD).
* List & sort (by due/priority).
* Selesai/hapus tugas.
* Simpan/Load JSON sederhana (format baris per baris).

**Struktur**

capstone/

├─ include/task.hpp

├─ src/task.cpp

├─ src/main.cpp

└─ tests/

**task.hpp**

#pragma once

#include <string>

#include <optional>

#include <vector>

#include <chrono>

struct Task{

std::string title;

int priority{}; // 1..5

std::chrono::sys\_days due{}; // C++20 chrono calendar

bool done{};

};

struct Repo{

bool load(const std::string& path);

bool save(const std::string& path) const;

void add(Task t);

std::vector<Task> list(bool onlyOpen=false) const;

bool finish(const std::string& title);

private:

std::vector<Task> data;

};

Task parse\_line(const std::string& line); // JSON-like minimal

std::string to\_line(const Task& t);

std::chrono::sys\_days parse\_date(const std::string& iso);

**Garis besar implementasi**

* parse\_date: gunakan std::chrono::year\_month\_day.
* parse\_line/to\_line: format {"title":"...","priority":3,"due":"2025-12-31","done":false}.
* main.cpp: baca argumen (add, list, finish, save).

Latihan: Lengkapi implementasi dan tambahkan unit test untuk parse\_date & Repo::finish.

**15. Praktik & Anti‑Pola Umum**

* **Jangan** pakai using namespace std; di header.
* **Selalu** inisialisasi variabel.
* **Gunakan** const/constexpr bila memungkinkan.
* **Tangani** resource via RAII; hindari new/delete langsung.
* **Pisahkan** interface (header) dan implementasi (cpp).
* **Aktifkan** warning tinggi; perbaiki semua warning.

**16. Ringkasan Perintah Kompilasi Berguna**

# Warning & standar

-g -O0 -Wall -Wextra -Wpedantic -std=c++20

# Sanitizers (clang/gcc)

-fsanitize=address,undefined -fno-omit-frame-pointer

# Definisi makro praprosesor

-DNDEBUG -DMY\_FLAG=1

**17. Latihan Tambahan (dengan petunjuk)**

1. **FizzBuzz Variant**: Untuk n, cetak atura