Baik, aku akan buatkan **materi dasar C++ lengkap dengan contoh kode dan penjelasan**. Materi ini cocok untuk kamu yang baru pertama kali belajar sampai paham konsep dasar pemrograman dengan C++.

**📘 Materi Dasar C++**

**1. Apa itu C++?**

C++ adalah bahasa pemrograman yang bersifat **compiled, statically typed, dan multi-paradigm** (mendukung prosedural, OOP, dan generic programming).  
C++ banyak digunakan untuk:

* Aplikasi desktop
* Game development
* Sistem operasi
* Embedded system

**2. Struktur Program Dasar C++**

Contoh program "Hello World":

#include <iostream> // pustaka untuk input-output

using namespace std; // supaya bisa pakai cout tanpa std::

int main() {

cout << "Hello, World!" << endl; // menampilkan teks ke layar

return 0; // mengakhiri program

}

📌 **Penjelasan:**

* #include <iostream> → library untuk input-output.
* using namespace std; → agar tidak menulis std::cout.
* int main() → fungsi utama, program mulai dari sini.
* cout → digunakan untuk mencetak ke layar.
* endl → membuat baris baru.

**3. Variabel dan Tipe Data**

**Contoh:**

#include <iostream>

using namespace std;

int main() {

int umur = 20; // bilangan bulat

float tinggi = 170.5; // bilangan desimal

char grade = 'A'; // karakter

string nama = "Lutfi"; // teks

bool isStudent = true; // boolean

cout << "Nama: " << nama << endl;

cout << "Umur: " << umur << " tahun" << endl;

cout << "Tinggi: " << tinggi << " cm" << endl;

cout << "Grade: " << grade << endl;

cout << "Mahasiswa? " << isStudent << endl;

return 0;

}

**4. Input dari User**

#include <iostream>

using namespace std;

int main() {

string nama;

int umur;

cout << "Masukkan nama: ";

cin >> nama; // input teks tanpa spasi

cout << "Masukkan umur: ";

cin >> umur; // input angka

cout << "Halo, " << nama << "! Umurmu " << umur << " tahun." << endl;

return 0;

}

📌 Gunakan getline(cin, nama); jika ingin input teks dengan spasi.

**5. Operator Dasar**

#include <iostream>

using namespace std;

int main() {

int a = 10, b = 3;

cout << "a + b = " << a + b << endl;

cout << "a - b = " << a - b << endl;

cout << "a \* b = " << a \* b << endl;

cout << "a / b = " << a / b << endl; // pembagian bulat

cout << "a % b = " << a % b << endl; // sisa bagi

return 0;

}

**6. Percabangan (if-else & switch)**

#include <iostream>

using namespace std;

int main() {

int nilai;

cout << "Masukkan nilai: ";

cin >> nilai;

if (nilai >= 90) {

cout << "Grade A" << endl;

} else if (nilai >= 75) {

cout << "Grade B" << endl;

} else {

cout << "Grade C" << endl;

}

return 0;

}

📌 Alternatif dengan switch:

int pilihan;

cout << "1. Makan\n2. Tidur\n3. Belajar\nPilih: ";

cin >> pilihan;

switch (pilihan) {

case 1: cout << "Kamu memilih Makan"; break;

case 2: cout << "Kamu memilih Tidur"; break;

case 3: cout << "Kamu memilih Belajar"; break;

default: cout << "Pilihan tidak ada";

}

**7. Perulangan (Loop)**

#include <iostream>

using namespace std;

int main() {

// for

for (int i = 1; i <= 5; i++) {

cout << "Perulangan ke-" << i << endl;

}

// while

int j = 1;

while (j <= 5) {

cout << "While ke-" << j << endl;

j++;

}

// do-while

int k = 1;

do {

cout << "Do-While ke-" << k << endl;

k++;

} while (k <= 5);

return 0;

}

**8. Array (Kumpulan Data)**

#include <iostream>

using namespace std;

int main() {

int angka[5] = {10, 20, 30, 40, 50};

for (int i = 0; i < 5; i++) {

cout << "Index ke-" << i << ": " << angka[i] << endl;

}

return 0;

}

**9. Fungsi**

#include <iostream>

using namespace std;

int tambah(int a, int b) {

return a + b;

}

int main() {

int hasil = tambah(5, 3);

cout << "Hasil penjumlahan: " << hasil << endl;

return 0;

}

**10. Dasar OOP (Class & Object)**

#include <iostream>

using namespace std;

class Mahasiswa {

public:

string nama;

int umur;

void perkenalan() {

cout << "Halo, nama saya " << nama << ", umur " << umur << " tahun." << endl;

}

};

int main() {

Mahasiswa m1;

m1.nama = "Lutfi";

m1.umur = 20;

m1.perkenalan();

return 0;

}

**📌 Rangkuman**

1. **Struktur dasar C++** → #include, main(), cout, cin.
2. **Variabel & tipe data** → int, float, char, string, bool.
3. **Operator** → aritmatika, perbandingan, logika.
4. **Kontrol alur** → if-else, switch, loop.
5. **Array** → menyimpan kumpulan data.
6. **Fungsi** → modularisasi kode.
7. **OOP dasar** → class, object, method.

Mau aku bikinkan juga **latihan soal C++ dasar + kunci jawaban** biar sekalian latihan?

**Anda bilang:**

Butakan aku materi menengahnya juga

**ChatGPT bilang:**

Oke 👍, setelah kamu menguasai **materi dasar C++**, sekarang kita masuk ke **materi menengah (intermediate)**. Di level ini, kamu akan belajar hal-hal yang lebih kompleks seperti pointer, reference, string lanjutan, struct, class lebih dalam, inheritance, template, hingga file handling.

**📘 Materi Menengah C++**

**1. Pointer**

Pointer adalah variabel yang menyimpan alamat memori variabel lain.

#include <iostream>

using namespace std;

int main() {

int x = 10;

int \*ptr = &x; // pointer menyimpan alamat variabel x

cout << "Nilai x: " << x << endl;

cout << "Alamat x: " << &x << endl;

cout << "Nilai pointer ptr: " << ptr << endl;

cout << "Nilai yang ditunjuk ptr: " << \*ptr << endl;

\*ptr = 20; // mengubah nilai x lewat pointer

cout << "Nilai baru x: " << x << endl;

return 0;

}

📌 Penting untuk memahami pointer karena dipakai pada array, dynamic memory, dan OOP.

**2. Reference**

Reference adalah alias dari variabel. Lebih aman daripada pointer.

#include <iostream>

using namespace std;

int main() {

int a = 5;

int &ref = a; // ref menjadi alias dari a

cout << "a = " << a << ", ref = " << ref << endl;

ref = 10; // mengubah ref juga mengubah a

cout << "a = " << a << ", ref = " << ref << endl;

return 0;

}

**3. Fungsi dengan Pointer & Reference**

#include <iostream>

using namespace std;

void tambahSatu(int &n) { // pass by reference

n++;

}

int main() {

int nilai = 10;

tambahSatu(nilai);

cout << "Nilai setelah fungsi: " << nilai << endl;

return 0;

}

**4. Struct (Tipe Data Buatan)**

#include <iostream>

using namespace std;

struct Mahasiswa {

string nama;

int umur;

float ipk;

};

int main() {

Mahasiswa m1 = {"Lutfi", 20, 3.8};

cout << "Nama: " << m1.nama << endl;

cout << "Umur: " << m1.umur << endl;

cout << "IPK: " << m1.ipk << endl;

return 0;

}

**5. Class & Constructor**

#include <iostream>

using namespace std;

class Mahasiswa {

public:

string nama;

int umur;

// constructor

Mahasiswa(string n, int u) {

nama = n;

umur = u;

}

void perkenalan() {

cout << "Halo, saya " << nama << ", umur " << umur << " tahun." << endl;

}

};

int main() {

Mahasiswa m1("Lutfi", 20);

m1.perkenalan();

return 0;

}

**6. Inheritance (Pewarisan)**

#include <iostream>

using namespace std;

class Orang {

public:

string nama;

int umur;

void identitas() {

cout << "Nama: " << nama << ", Umur: " << umur << endl;

}

};

class Mahasiswa : public Orang {

public:

string jurusan;

void belajar() {

cout << nama << " sedang belajar di jurusan " << jurusan << endl;

}

};

int main() {

Mahasiswa m1;

m1.nama = "Lutfi";

m1.umur = 20;

m1.jurusan = "Informatika";

m1.identitas();

m1.belajar();

return 0;

}

**7. Overloading (Fungsi dengan Nama Sama)**

#include <iostream>

using namespace std;

class Kalkulator {

public:

int tambah(int a, int b) {

return a + b;

}

double tambah(double a, double b) {

return a + b;

}

};

int main() {

Kalkulator k;

cout << "Tambah int: " << k.tambah(5, 3) << endl;

cout << "Tambah double: " << k.tambah(2.5, 3.7) << endl;

return 0;

}

**8. Template (Generic Programming)**

#include <iostream>

using namespace std;

template <typename T>

T maxValue(T a, T b) {

return (a > b) ? a : b;

}

int main() {

cout << "Max dari 3 dan 7: " << maxValue(3, 7) << endl;

cout << "Max dari 4.5 dan 2.3: " << maxValue(4.5, 2.3) << endl;

return 0;

}

**9. File Handling**

#include <iostream>

#include <fstream>

using namespace std;

int main() {

ofstream outFile("data.txt"); // buat file

outFile << "Halo, ini ditulis ke file." << endl;

outFile.close();

string isi;

ifstream inFile("data.txt");

while (getline(inFile, isi)) {

cout << isi << endl;

}

inFile.close();

return 0;

}

**10. Dynamic Memory (new & delete)**

#include <iostream>

using namespace std;

int main() {

int \*ptr = new int; // alokasi memory

\*ptr = 10;

cout << "Nilai: " << \*ptr << endl;

delete ptr; // hapus memory

return 0;

}

**📌 Rangkuman Materi Menengah**

1. **Pointer & Reference** → akses langsung ke memori.
2. **Struct** → tipe data buatan.
3. **Class dengan Constructor & Destructor**.
4. **Inheritance** → pewarisan sifat antar class.
5. **Overloading & Polymorphism**.
6. **Template** → fungsi & class generic.
7. **File Handling** → baca/tulis file.
8. **Dynamic Memory** → new & delete.

Kalau di **materi dasar** kita fokus ke pondasi (variabel, input-output, loop), di **materi menengah** ini sudah ke arah **OOP, pointer, dan manajemen memori**.

👉 Mau aku lanjutkan ke **materi lanjutan (advance C++)** sekalian biar lengkap sampai pro?

**Anda bilang:**

lanjutkan ke materi lanjutan (advance C++) sekalian biar lengkap sampai pro

**ChatGPT bilang:**

Mantap 🚀, kalau sudah paham **dasar** dan **menengah**, sekarang kita masuk ke **materi lanjutan (Advance C++)**. Di level ini, kita bahas konsep yang dipakai di dunia nyata saat membuat aplikasi besar: OOP lanjutan, STL, memory management, multithreading, hingga best practice modern C++ (C++11 ke atas).

**📘 Materi Lanjutan (Advance) C++**

**1. Polymorphism (Virtual Function & Override)**

Polymorphism memungkinkan method yang sama bekerja berbeda di class turunan.

#include <iostream>

using namespace std;

class Hewan {

public:

virtual void suara() { // virtual untuk polymorphism

cout << "Hewan bersuara" << endl;

}

};

class Kucing : public Hewan {

public:

void suara() override {

cout << "Meong!" << endl;

}

};

class Anjing : public Hewan {

public:

void suara() override {

cout << "Guk guk!" << endl;

}

};

int main() {

Hewan\* h1 = new Kucing();

Hewan\* h2 = new Anjing();

h1->suara(); // Meong

h2->suara(); // Guk guk

delete h1;

delete h2;

return 0;

}

**2. Abstract Class & Pure Virtual Function**

Abstract class hanya bisa diturunkan, tidak bisa dibuat objek langsung.

#include <iostream>

using namespace std;

class Shape {

public:

virtual void draw() = 0; // pure virtual

};

class Circle : public Shape {

public:

void draw() override {

cout << "Menggambar lingkaran" << endl;

}

};

int main() {

Shape\* s = new Circle();

s->draw();

delete s;

return 0;

}

**3. Operator Overloading**

Kita bisa mendefinisikan cara kerja operator untuk class buatan sendiri.

#include <iostream>

using namespace std;

class Vector {

public:

int x, y;

Vector(int a, int b) : x(a), y(b) {}

// overload operator +

Vector operator+(const Vector& v) {

return Vector(x + v.x, y + v.y);

}

};

int main() {

Vector v1(2, 3), v2(4, 5);

Vector v3 = v1 + v2; // pakai operator overload

cout << "Hasil: (" << v3.x << ", " << v3.y << ")" << endl;

return 0;

}

**4. Exception Handling**

Menangani error dengan try, catch, throw.

#include <iostream>

using namespace std;

int bagi(int a, int b) {

if (b == 0) throw runtime\_error("Error: Pembagian dengan nol!");

return a / b;

}

int main() {

try {

cout << "Hasil: " << bagi(10, 0) << endl;

} catch (exception &e) {

cout << e.what() << endl;

}

return 0;

}

**5. Smart Pointer (Modern C++)**

Mengelola memory otomatis tanpa delete.

#include <iostream>

#include <memory> // smart pointer

using namespace std;

class Data {

public:

Data() { cout << "Constructor dipanggil\n"; }

~Data() { cout << "Destructor dipanggil\n"; }

};

int main() {

unique\_ptr<Data> d1 = make\_unique<Data>(); // otomatis dihapus

shared\_ptr<Data> d2 = make\_shared<Data>();

shared\_ptr<Data> d3 = d2; // reference count bertambah

cout << "Jumlah shared\_ptr: " << d2.use\_count() << endl;

return 0;

}

**6. Standard Template Library (STL)**

STL menyediakan **container, algoritma, dan iterator** siap pakai.

**Contoh: vector, map, set**

#include <iostream>

#include <vector>

#include <map>

#include <set>

using namespace std;

int main() {

// vector

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

v.push\_back(4);

for (int x : v) cout << x << " "; // 1 2 3 4

cout << endl;

// map (key-value)

map<string, int> umur;

umur["Lutfi"] = 20;

umur["Budi"] = 22;

cout << "Umur Lutfi: " << umur["Lutfi"] << endl;

// set (unik)

set<int> s = {1, 2, 2, 3};

for (int x : s) cout << x << " "; // 1 2 3

}

**7. Lambda Function (Anonymous Function)**

Fungsi singkat tanpa nama.

#include <iostream>

#include <algorithm>

#include <vector>

using namespace std;

int main() {

vector<int> v = {5, 2, 8, 1};

sort(v.begin(), v.end(), [](int a, int b) {

return a < b; // ascending

});

for (int x : v) cout << x << " "; // 1 2 5 8

return 0;

}

**8. Multithreading (C++11)**

Menjalankan beberapa task sekaligus.

#include <iostream>

#include <thread>

using namespace std;

void cetak(string pesan) {

for (int i = 0; i < 5; i++) {

cout << pesan << " " << i << endl;

}

}

int main() {

thread t1(cetak, "Thread 1");

thread t2(cetak, "Thread 2");

t1.join();

t2.join();

return 0;

}

**9. Namespace Custom**

Untuk menghindari konflik nama.

#include <iostream>

using namespace std;

namespace matematika {

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

}

int main() {

cout << "Hasil: " << matematika::tambah(3, 4) << endl;

return 0;

}

**10. Design Pattern Dasar (Singleton)**

Contoh design pattern yang sering dipakai.

#include <iostream>

using namespace std;

class Singleton {

private:

static Singleton\* instance;

Singleton() {} // private constructor

public:

static Singleton\* getInstance() {

if (instance == nullptr) {

instance = new Singleton();

}

return instance;

}

void tampil() {

cout << "Ini Singleton Pattern" << endl;

}

};

Singleton\* Singleton::instance = nullptr;

int main() {

Singleton\* s1 = Singleton::getInstance();

Singleton\* s2 = Singleton::getInstance();

s1->tampil();

cout << (s1 == s2 ? "Objek sama" : "Objek berbeda") << endl;

return 0;

}

**📌 Rangkuman Advance C++**

1. **Polymorphism & Abstract Class** → dasar OOP tingkat lanjut.
2. **Operator Overloading** → bikin operator custom.
3. **Exception Handling** → cara aman tangani error.
4. **Smart Pointer** → memory management modern.
5. **STL** → vector, map, set, algorithm.
6. **Lambda Function** → fungsi singkat untuk callback/algoritma.
7. **Multithreading** → eksekusi paralel.
8. **Namespace** → menghindari konflik nama.
9. **Design Pattern** → pola coding yang reusable.

OOP (Object Oriented Programming) atau Pemrograman Berorientasi Objek adalah paradigma pemrograman yang mengorganisasi kode di sekitar objek, yaitu data dan fungsi yang saling terkait. C++ adalah bahasa berorientasi objek yang mengimplementasikan konsep ini melalui Kelas dan Objek, yang memungkinkan pengembangan perangkat lunak modular, dapat digunakan kembali, dan lebih mudah dikelola.

Konsep Dasar OOP

OOP berpusat pada beberapa konsep utama:

Kelas (Class):

Cetak biru atau templat untuk membuat objek. Kelas mendefinisikan atribut (data) dan metode (fungsi) yang dimiliki oleh objek.

Objek (Object):

Instans dari sebuah kelas. Objek adalah representasi nyata dari kelas dan memiliki data serta kemampuan sesuai definisinya.

Enkapsulasi:

Menggabungkan data dan metode yang beroperasi pada data tersebut ke dalam satu unit, yaitu kelas. Ini menyembunyikan detail implementasi dan hanya memperlihatkan fungsionalitas yang perlu.

Abstraksi:

Menyederhanakan data dengan hanya menampilkan detail penting dan menyembunyikan informasi yang tidak perlu.

Inheritance (Pewarisan):

Memungkinkan sebuah kelas baru (kelas anak) mewarisi sifat-sifat dari kelas yang sudah ada (kelas induk). Ini mendukung penggunaan kembali kode dan membangun hubungan hierarkis antar kelas.

Polimorfisme:

Memungkinkan objek dari kelas yang berbeda untuk merespons panggilan metode yang sama dengan cara yang berbeda-beda.

Contoh OOP di C++

Berikut adalah contoh sederhana membuat kelas Mobil dan objeknya di C++:

C++

#include <iostream> // untuk input/output

// Mendefinisikan kelas Mobil

class Mobil {

public: // Aksesibilitas public untuk atribut dan metode

std::string merek; // Atribut (data)

std::string model; // Atribut (data)

// Metode (fungsi) untuk menampilkan informasi mobil

void tampilkanInfo() {

std::cout << "Merek: " << merek << ", Model: " << model << std::endl;

}

};

int main() {

// Membuat objek 'mobil1' dari kelas Mobil

Mobil mobil1;

// Mengisi atribut objek mobil1

mobil1.merek = "Toyota";

mobil1.model = "Avanza";

// Memanggil metode tampilkanInfo() pada objek mobil1

mobil1.tampilkanInfo();

return 0;

}

Penjelasan Kode:

class Mobil { ... };: Ini adalah deklarasi kelas Mobil, yang menjadi templat untuk membuat objek mobil.

public:: Menunjukkan bahwa anggota merek, model, dan tampilkanInfo() dapat diakses dari luar kelas.

std::string merek; std::string model;: Atribut (variabel data) dari objek Mobil.

void tampilkanInfo() { ... }: Metode (fungsi) yang mengoperasikan data merek dan model untuk menampilkannya.

Mobil mobil1;: Ini adalah pembuatan objek bernama mobil1 dari kelas Mobil.

mobil1.merek = "Toyota"; mobil1.model = "Avanza";: Memberikan nilai pada atribut objek mobil1.

mobil1.tampilkanInfo();: Memanggil metode tampilkanInfo() untuk menampilkan informasi mobil.

Dengan OOP, Anda dapat membuat kode yang terstruktur dengan baik, mudah dikelola, dan dapat digunakan kembali di berbagai proyek.

|  |  |  |
| --- | --- | --- |
| **Operator** | **Fungsi** | **Contoh** |
| + | Penjumlahan | a + b |
| - | Pengurangan | a - b |
| \* | Perkalian | a \* b |
| / | Pembagian | a / b |
| % | Modulus (sisa bagi) | a % b |

**OOP**

**Object-Oriented Programming (Pemrograman Berorientasi Objek)** adalah paradigma pemrograman yang berfokus pada **objek**, bukan hanya fungsi/prosedur.  
Objek ini punya **atribut (data/variabel)** dan **perilaku (fungsi/metode)**.

**🔹 Konsep Dasar OOP**

1. **Class (Kelas)** → blueprint/cetak biru untuk membuat objek.
2. **Object (Objek)** → hasil nyata dari class (instance).
3. **Encapsulation (Enkapsulasi)** → menyembunyikan detail implementasi (pakai private, public, protected).
4. **Inheritance (Pewarisan)** → class bisa mewarisi properti/metode dari class lain.
5. **Polymorphism (Polimorfisme)** → kemampuan fungsi/metode punya bentuk berbeda (overloading, overriding).
6. **Abstraction (Abstraksi)** → hanya menampilkan hal penting, detail disembunyikan.

**🔹 Contoh Sederhana OOP di C++**

Mari kita buat contoh dengan **class Mobil**.

#include <iostream>

using namespace std;

// Class = blueprint

class Mobil {

private: // encapsulation: hanya bisa diakses dari dalam class

string merek;

int tahun;

public:

// Constructor → otomatis dipanggil saat objek dibuat

Mobil(string m, int t) {

merek = m;

tahun = t;

}

// Method (fungsi dalam class)

void infoMobil() {

cout << "Merek: " << merek << ", Tahun: " << tahun << endl;

}

};

// Inheritance: class SportCar mewarisi dari Mobil

class SportCar : public Mobil {

private:

int topSpeed;

public:

SportCar(string m, int t, int speed) : Mobil(m, t) {

topSpeed = speed;

}

// Overriding → menimpa method dari parent

void infoMobil() {

Mobil::infoMobil(); // panggil method parent

cout << "Top Speed: " << topSpeed << " km/h" << endl;

}

};

int main() {

// Membuat objek dari class Mobil

Mobil m1("Toyota", 2020);

m1.infoMobil();

// Membuat objek dari class SportCar (turunan Mobil)

SportCar s1("Ferrari", 2022, 350);

s1.infoMobil();

return 0;

}

**🔎 Penjelasan Kode**

1. class Mobil → mendefinisikan **blueprint** mobil.
   * merek & tahun → **atribut (data)**.
   * infoMobil() → **method (perilaku)**.
2. private → hanya bisa diakses di dalam class (contoh: merek, tahun).
3. public → bisa diakses dari luar (contoh: infoMobil()).
4. **Constructor** Mobil(string m, int t) → dipanggil otomatis saat objek dibuat.
5. class SportCar : public Mobil → pewarisan (inheritance), SportCar punya semua properti Mobil.
6. void infoMobil() di SportCar → overriding (polymorphism).

**📌 Output Program**

Merek: Toyota, Tahun: 2020

Merek: Ferrari, Tahun: 2022

Top Speed: 350 km/h

**Intinya**

* **Class** = cetakan
* **Object** = hasil nyata dari cetakan
* **Encapsulation** = sembunyikan data → akses pakai fungsi
* **Inheritance** = class bisa menurun ke class lain
* **Polymorphism** = satu nama fungsi bisa punya banyak bentuk
* **Abstraction** = fokus ke fitur penting, detail disembunyikan