REFERENSI CEPAT C++

Class, template, STL, smart pointer, fitur C++ modern

Dasar

Hello World

```
#include <iostream>
int main() {
    std::cout << "Hello, World!" << std::endl;
    return 0;
}
```

Kompilasi & Jalankan

```
g++ -std=c++20 -Wall -o app main.cpp
./app
clang++ -std=c++20 -o app main.cpp
```

Variabel & Konstanta

```
int x = 42;
auto y = 3.14;              // type deduction
const int MAX = 100;
constexpr int SIZE = 256;  // compile-time constant
```

Namespace

```
namespace math {
    double pi = 3.14159;
}
using namespace std;        // use sparingly
using std::cout;            // prefer selective
```

Class

Definisi Class

```
class Rectangle {
    double w_, h_;
public:
    Rectangle(double w, double h) : w_(w), h_(h) {}
    double area() const { return w_ * h_; } };
```

Inheritance

```
class Shape {
public:
    virtual double area() const = 0;   // pure virtual
    virtual ~Shape() = default;  };
// class Circle : public Shape { ... };
```

Access Specifier

`public`	Dapat diakses dari mana saja
`protected`	Dapat diakses di class dan class turunan
`private`	Hanya dapat diakses dalam class
`friend`	Berikan akses ke fungsi atau class tertentu

Member Khusus

`Constructor`	`MyClass(args)` — inisialisasi objek
`Destructor`	`~MyClass()` — bersihkan sumber daya
`Copy ctor`	`MyClass(const MyClass&)`
`Move ctor`	`MyClass(MyClass&&)` — transfer kepemilikan
`Copy assign`	`operator=(const MyClass&)`
`Move assign`	`operator=(MyClass&&)`

Template

Function Template

```
template <typename T>
T max_val(T a, T b) {
    return (a > b) ? a : b;
}
auto result = max_val(3, 7);  // deduced as int
```

Class Template

```
template <typename T>
class Stack {
    std::vector<T> data_;
public:
    void push(const T& v) { data_.push_back(v); }  };
```

Concepts (C++20)

```
template <typename T>
concept Numeric = std::integral<T> || std::floating_point<T>;
template <Numeric T>
T add(T a, T b) { return a + b; }
```

STL Container

Sequence Container

`vector<T>`	Array dinamis, akses acak cepat
`deque<T>`	Double-ended queue
`list<T>`	Linked list dua arah
`array<T,N>`	Array ukuran tetap (ukuran compile-time)
`forward_list<T>`	Linked list satu arah

Associative Container

`map<K,V>`	Pasangan key-value terurut (red-black tree)
`set<T>`	Elemen unik terurut
`unordered_map<K,V>`	Hash map, lookup rata-rata O(1)
`unordered_set<T>`	Hash set, lookup rata-rata O(1)
`multimap<K,V>`	Terurut, membolehkan key duplikat

Operasi Vector

```
std::vector<int> v = {1, 2, 3};
v.push_back(4);
v.emplace_back(5);          // construct in place
v.size();  v.empty();
v[0];  v.at(0);             // at() has bounds check
```

Iterator & Algoritma

Penggunaan Iterator

```
std::vector<int> v = {3, 1, 4, 1, 5};
for (auto it = v.begin(); it != v.end(); ++it) {
    std::cout << *it << " ";
}
for (const auto& val : v) { }  // range-based for
```

Algoritma Umum

`sort(begin, end)`	Urutkan elemen secara ascending
`find(begin, end, val)`	Temukan kemunculan pertama nilai
`count(begin, end, val)`	Hitung kemunculan nilai
`transform(b, e, out, fn)`	Terapkan fungsi ke setiap elemen
`accumulate(b, e, init)`	Reduksi elemen (default: penjumlahan)
`reverse(begin, end)`	Balik urutan elemen
`unique(begin, end)`	Hapus duplikat berurutan

Ranges (C++20)

```
namespace rv = std::views;
auto evens = v | rv::filter([](int n){ return n % 2 == 0; })
               | rv::transform([](int n){ return n * n; });
```

Smart Pointer

unique_ptr

```
auto p = std::make_unique<int>(42);
std::cout << *p << std::endl;
// auto-deleted when out of scope
// cannot be copied, only moved
```

shared_ptr

```
auto sp = std::make_shared<std::string>("hello");
auto sp2 = sp;              // reference count: 2
std::cout << sp.use_count(); // 2
```

Perbandingan

`unique_ptr<T>`	Kepemilikan eksklusif, zero overhead
`shared_ptr<T>`	Kepemilikan bersama melalui reference counting
`weak_ptr<T>`	Observer non-owning dari `shared_ptr`
`make_unique<T>()`	Cara yang disukai untuk membuat `unique_ptr`
`make_shared<T>()`	Cara yang disukai untuk membuat `shared_ptr`

Lambda

Sintaks Lambda

```
auto add = [](int a, int b) { return a + b; };
int sum = add(3, 4);          // 7
```

Mode Capture

`[x]`	Capture `x` by value (salin)
`[&x]`	Capture `x` by reference
`[=]`	Capture semua variabel yang digunakan by value
`[&]`	Capture semua variabel yang digunakan by reference
`[=, &x]`	Semua by value, `x` by reference
`[this]`	Capture pointer objek yang melingkupi

Lambda dengan STL

```
std::vector<int> v = {5, 2, 8, 1};
std::sort(v.begin(), v.end(),
    [](int a, int b) { return a > b; });  // descending
auto it = std::find_if(v.begin(), v.end(),
    [](int n) { return n > 3; });
```

String & I/O

std::string

```
std::string s = "hello";
s += " world";              // concatenation
s.substr(0, 5);             // "hello"
s.find("world");            // 6 (position)
s.length();  s.empty();
```

Konversi String

`std::to_string(42)`	Angka ke string
`std::stoi(s)`	String ke `int`
`std::stod(s)`	String ke `double`
`std::stol(s)`	String ke `long`

I/O Stream

```
std::cout << "output" << std::endl;
std::cin >> variable;
std::getline(std::cin, line);
```

File I/O

```
std::ofstream out("file.txt");
out << "hello" << std::endl;
std::ifstream in("file.txt");
std::string line;
while (std::getline(in, line)) { }
```

Penanganan Error

Exception

```
try {
    throw std::runtime_error("something failed");
} catch (const std::exception& e) {
    std::cerr << e.what() << std::endl;
} catch (...) { /* unknown error */ }
```

Exception Standar

`std::exception`	Base class semua exception standar
`std::runtime_error`	Error runtime dengan pesan
`std::logic_error`	Error logika (pelanggaran pre-condition)
`std::out_of_range`	Indeks atau iterator di luar batas
`std::invalid_argument`	Argumen fungsi tidak valid
`std::bad_alloc`	Kegagalan alokasi memori

noexcept

```
void safe_func() noexcept {
    // guaranteed not to throw
}
bool can_throw = noexcept(safe_func());  // true
```

C++ Modern (17/20)

Structured Bindings (C++17)

```
std::map<std::string, int> m = {{"a", 1}, {"b", 2}};
for (auto& [key, value] : m) {
    std::cout << key << ": " << value << "\n";
}
```

std::optional (C++17)

```
std::optional<int> find(int id) {
    if (id > 0) return id * 10;
    return std::nullopt;
}
auto val = find(3);             // has_value() == true
```

std::variant & std::any (C++17)

```
std::variant<int, std::string> v = "hello";
std::cout << std::get<std::string>(v);
std::any a = 42;
int n = std::any_cast<int>(a);
```

Fitur Modern Utama

`auto`	Deduksi tipe untuk variabel dan tipe return
`constexpr`	Evaluasi compile-time
`if constexpr`	Kondisional compile-time (C++17)
`std::span<T>`	View non-owning atas data berurutan (C++20)
`std::format()`	Pemformatan type-safe (C++20)
`co_await`	Dukungan coroutine (C++20)