C++ THAM KHẢO NHANH

Class, template, STL, smart pointer, C++ hiện đại

Cơ Bản

Hello World

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

Biên Dịch & Chạy

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

Biến & Hằng

```
int x = 42;
auto y = 3.14;              // suy luận kiểu
const int MAX = 100;
constexpr int SIZE = 256;  // hằng lúc biên dịch
```

Namespaces

```
namespace math {
    double pi = 3.14159;
}
using namespace std;        // dùng cẩn thận
using std::cout;            // ưu tiên chọn lọc
```

Classes

Định Nghĩa Class

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

Kế Thừa

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

Access Specifiers

`public`	Truy cập từ bất kỳ đâu
`protected`	Truy cập trong class và class dẫn xuất
`private`	Chỉ truy cập trong class
`friend`	Cấp quyền cho hàm hoặc class cụ thể

Member Đặc Biệt

`Constructor`	`MyClass(args)` — khởi tạo object
`Destructor`	`~MyClass()` — dọn dẹp tài nguyên
`Copy ctor`	`MyClass(const MyClass&)`
`Move ctor`	`MyClass(MyClass&&)` — chuyển quyền sở hữu
`Copy assign`	`operator=(const MyClass&)`
`Move assign`	`operator=(MyClass&&)`

Templates

Function Template

```
template <typename T>
T max_val(T a, T b) {
    return (a > b) ? a : b;
}
auto result = max_val(3, 7);  // suy luận thành 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 Containers

Sequence Containers

`vector<T>`	Mảng động, truy cập ngẫu nhiên nhanh
`deque<T>`	Hàng đợi hai đầu
`list<T>`	Danh sách liên kết đôi
`array<T,N>`	Mảng kích thước cố định (xác định lúc biên dịch)
`forward_list<T>`	Danh sách liên kết đơn

Associative Containers

`map<K,V>`	Cặp key-value có thứ tự (cây đỏ-đen)
`set<T>`	Phần tử duy nhất có thứ tự
`unordered_map<K,V>`	Hash map, tra cứu O(1) trung bình
`unordered_set<T>`	Hash set, tra cứu O(1) trung bình
`multimap<K,V>`	Có thứ tự, cho phép key trùng lặp

Thao Tác Vector

```
std::vector<int> v = {1, 2, 3};
v.push_back(4);
v.emplace_back(5);          // construct tại chỗ
v.size();  v.empty();
v[0];  v.at(0);             // at() kiểm tra biên
```

Iterators & Algorithms

Sử Dụng 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
```

Các Algorithm Thường Dùng

`sort(begin, end)`	Sắp xếp tăng dần
`find(begin, end, val)`	Tìm lần xuất hiện đầu tiên
`count(begin, end, val)`	Đếm số lần xuất hiện
`transform(b, e, out, fn)`	Áp dụng hàm cho mỗi phần tử
`accumulate(b, e, init)`	Tổng hợp phần tử (tổng mặc định)
`reverse(begin, end)`	Đảo ngược thứ tự phần tử
`unique(begin, end)`	Xóa phần tử liên tiếp trùng nhau

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 Pointers

unique_ptr

```
auto p = std::make_unique<int>(42);
std::cout << *p << std::endl;
// tự xóa khi ra khỏi scope
// không thể sao chép, chỉ được move
```

shared_ptr

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

So Sánh

`unique_ptr<T>`	Sở hữu độc quyền, không overhead
`shared_ptr<T>`	Sở hữu chia sẻ qua đếm tham chiếu
`weak_ptr<T>`	Quan sát không sở hữu của `shared_ptr`
`make_unique<T>()`	Cách ưu tiên tạo `unique_ptr`
`make_shared<T>()`	Cách ưu tiên tạo `shared_ptr`

Lambdas

Cú Pháp Lambda

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

Chế Độ Capture

`[x]`	Capture `x` theo giá trị (sao chép)
`[&x]`	Capture `x` theo tham chiếu
`[=]`	Capture tất cả biến dùng theo giá trị
`[&]`	Capture tất cả biến dùng theo tham chiếu
`[=, &x]`	Tất cả theo giá trị, `x` theo tham chiếu
`[this]`	Capture con trỏ object bao quanh

Lambda với STL

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

Chuỗi & I/O

std::string

```
std::string s = "hello";
s += " world";              // nối chuỗi
s.substr(0, 5);             // "hello"
s.find("world");            // 6 (vị trí)
s.length();  s.empty();
```

Chuyển Đổi Chuỗi

`std::to_string(42)`	Số thành chuỗi
`std::stoi(s)`	Chuỗi thành `int`
`std::stod(s)`	Chuỗi thành `double`
`std::stol(s)`	Chuỗi thành `long`

I/O Streams

```
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)) { }
```

Xử Lý Lỗi

Exceptions

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

Exceptions Chuẩn

`std::exception`	Base class cho tất cả exception chuẩn
`std::runtime_error`	Lỗi runtime với message
`std::logic_error`	Lỗi logic (vi phạm tiền điều kiện)
`std::out_of_range`	Index hoặc iterator ngoài phạm vi
`std::invalid_argument`	Tham số hàm không hợp lệ
`std::bad_alloc`	Thất bại cấp phát bộ nhớ

noexcept

```
void safe_func() noexcept {
    // đảm bảo không throw
}
bool can_throw = noexcept(safe_func());  // true
```

C++ Hiện Đại (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);
```

Tính Năng Hiện Đại Quan Trọng

`auto`	Suy luận kiểu cho biến và kiểu trả về
`constexpr`	Đánh giá lúc biên dịch
`if constexpr`	Điều kiện lúc biên dịch (C++17)
`std::span<T>`	View không sở hữu dữ liệu liên tiếp (C++20)
`std::format()`	Định dạng kiểu-an toàn (C++20)
`co_await`	Hỗ trợ coroutine (C++20)