C++ 快速参考

类、模板、STL、智能指针、现代 C++ 要点

基础

Hello World

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

编译与运行

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

变量与常量

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

命名空间

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

类

类定义

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

继承

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

访问说明符

`public`	任意位置可访问
`protected`	类内及派生类可访问
`private`	仅类内可访问
`friend`	授权特定函数或类访问

特殊成员函数

`构造函数`	`MyClass(args)` — 初始化对象
`析构函数`	`~MyClass()` — 释放资源
`拷贝构造`	`MyClass(const MyClass&)`
`移动构造`	`MyClass(MyClass&&)` — 转移所有权
`拷贝赋值`	`operator=(const MyClass&)`
`移动赋值`	`operator=(MyClass&&)`

模板

函数模板

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

类模板

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

Concepts（C++20）

```cpp
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 容器

序列容器

`vector<T>`	动态数组，随机访问快
`deque<T>`	双端队列
`list<T>`	双向链表
`array<T,N>`	固定大小数组（编译期大小）
`forward_list<T>`	单向链表

关联容器

`map<K,V>`	有序键值对（红黑树）
`set<T>`	有序唯一元素
`unordered_map<K,V>`	哈希表，平均 O(1) 查找
`unordered_set<T>`	哈希集合，平均 O(1) 查找
`multimap<K,V>`	有序，允许重复键

vector 操作

```cpp
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
```

迭代器与算法

迭代器用法

```cpp
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
```

常用算法

`sort(begin, end)`	升序排序
`find(begin, end, val)`	查找第一个匹配值
`count(begin, end, val)`	统计出现次数
`transform(b, e, out, fn)`	对每个元素应用函数
`accumulate(b, e, init)`	归约（默认求和）
`reverse(begin, end)`	反转元素顺序
`unique(begin, end)`	移除连续重复元素

Ranges（C++20）

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

智能指针

unique_ptr

```cpp
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

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

对比

`unique_ptr<T>`	独占所有权，零开销
`shared_ptr<T>`	引用计数共享所有权
`weak_ptr<T>`	`shared_ptr` 的非拥有观察者
`make_unique<T>()`	创建 `unique_ptr` 的推荐方式
`make_shared<T>()`	创建 `shared_ptr` 的推荐方式

Lambda

Lambda 语法

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

捕获模式

`[x]`	按值捕获 `x`
`[&x]`	按引用捕获 `x`
`[=]`	按值捕获所有用到的变量
`[&]`	按引用捕获所有用到的变量
`[=, &x]`	其余按值，`x` 按引用
`[this]`	捕获外层对象指针

Lambda 与 STL 结合

```cpp
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; });
```

字符串与 I/O

std::string

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

字符串转换

`std::to_string(42)`	数字转字符串
`std::stoi(s)`	字符串转 `int`
`std::stod(s)`	字符串转 `double`
`std::stol(s)`	字符串转 `long`

I/O 流

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

文件 I/O

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

异常处理

异常

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

标准异常

`std::exception`	所有标准异常的基类
`std::runtime_error`	带消息的运行时错误
`std::logic_error`	逻辑错误（前置条件违反）
`std::out_of_range`	索引或迭代器越界
`std::invalid_argument`	非法函数参数
`std::bad_alloc`	内存分配失败

noexcept

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

现代 C++（17/20）

结构化绑定（C++17）

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

std::optional（C++17）

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

std::variant 与 std::any（C++17）

```cpp
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);
```

关键现代特性

`auto`	变量和返回值类型推导
`constexpr`	编译期求值
`if constexpr`	编译期条件（C++17）
`std::span<T>`	连续数据的非拥有视图（C++20）
`std::format()`	类型安全格式化（C++20）
`co_await`	协程支持（C++20）