C++ クイックリファレンス

クラス、テンプレート、STL、スマートポインタ、モダン C++ 必須事項

基礎

Hello World

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

コンパイル & 実行

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

変数 & 定数

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

名前空間

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

クラス

クラスの定義

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

継承

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

アクセス指定子

`public`	どこからでもアクセス可能
`protected`	クラスおよび派生クラスからアクセス可能
`private`	クラス内からのみアクセス可能
`friend`	特定の関数またはクラスにアクセスを許可

特殊メンバー関数

`コンストラクタ`	`MyClass(args)` — オブジェクトの初期化
`デストラクタ`	`~MyClass()` — リソースのクリーンアップ
`コピーコンストラクタ`	`MyClass(const MyClass&)`
`ムーブコンストラクタ`	`MyClass(MyClass&&)` — 所有権の転送
`コピー代入`	`operator=(const MyClass&)`
`ムーブ代入`	`operator=(MyClass&&)`

テンプレート

関数テンプレート

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

クラステンプレート

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

コンセプト（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 コンテナ

シーケンスコンテナ

`vector<T>`	動的配列、高速なランダムアクセス
`deque<T>`	両端キュー
`list<T>`	双方向リンクリスト
`array<T,N>`	固定サイズ配列（コンパイル時サイズ）
`forward_list<T>`	単方向リンクリスト

連想コンテナ

`map<K,V>`	順序付きキーバリューペア（赤黒木）
`set<T>`	順序付き一意要素
`unordered_map<K,V>`	ハッシュマップ、平均 O(1) ルックアップ
`unordered_set<T>`	ハッシュセット、平均 O(1) ルックアップ
`multimap<K,V>`	順序付き、重複キーを許可

vector の操作

```
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
```

イテレータ & アルゴリズム

イテレータの使い方

```
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
```

よく使うアルゴリズム

`sort(begin, end)`	要素を昇順にソート
`find(begin, end, val)`	値の最初の出現を検索
`count(begin, end, val)`	値の出現回数をカウント
`transform(b, e, out, fn)`	各要素に関数を適用
`accumulate(b, e, init)`	要素を集約（デフォルトは合計）
`reverse(begin, end)`	要素の順序を逆にする
`unique(begin, end)`	連続する重複を削除

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; });
```

スマートポインタ

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
```

比較

`unique_ptr<T>`	排他的所有権、オーバーヘッドなし
`shared_ptr<T>`	参照カウントによる共有所有権
`weak_ptr<T>`	`shared_ptr` の非所有オブザーバー
`make_unique<T>()`	`unique_ptr` の推奨作成方法
`make_shared<T>()`	`shared_ptr` の推奨作成方法

ラムダ

ラムダの構文

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

キャプチャモード

`[x]`	`x` を値でキャプチャ（コピー）
`[&x]`	`x` を参照でキャプチャ
`[=]`	使用中の変数を全て値でキャプチャ
`[&]`	使用中の変数を全て参照でキャプチャ
`[=, &x]`	全て値でキャプチャ、`x` のみ参照
`[this]`	外側のオブジェクトポインタをキャプチャ

STL とのラムダ

```
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; });
```

文字列 & 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();
```

文字列変換

`std::to_string(42)`	数値を文字列に変換
`std::stoi(s)`	文字列を `int` に変換
`std::stod(s)`	文字列を `double` に変換
`std::stol(s)`	文字列を `long` に変換

I/O ストリーム

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

ファイル I/O

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

エラーハンドリング

例外

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

標準例外

`std::exception`	全標準例外の基底クラス
`std::runtime_error`	メッセージ付きランタイムエラー
`std::logic_error`	ロジックエラー（事前条件違反）
`std::out_of_range`	インデックスまたはイテレータの範囲外
`std::invalid_argument`	無効な関数引数
`std::bad_alloc`	メモリ割り当て失敗

noexcept

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

モダン C++（17/20）

構造化バインディング（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);
```

主要なモダン機能

`auto`	変数と戻り値型の型推論
`constexpr`	コンパイル時評価
`if constexpr`	コンパイル時条件分岐（C++17）
`std::span<T>`	連続データへの非所有ビュー（C++20）
`std::format()`	型安全な文字列フォーマット（C++20）
`co_await`	コルーチンサポート（C++20）