C++ 빠른 참조

클래스, 템플릿, STL, 스마트 포인터, 모던 C++ 핵심

기본

Hello World

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

컴파일 & 실행

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

변수 & 상수

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

네임스페이스

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

클래스

클래스 정의

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

상속

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

접근 지정자

`public`	어디서든 접근 가능
`protected`	클래스 및 파생 클래스에서 접근 가능
`private`	클래스 내에서만 접근 가능
`friend`	특정 함수 또는 클래스에 접근 권한 부여

특수 멤버

`Constructor`	`MyClass(args)` — 객체 초기화
`Destructor`	`~MyClass()` — 리소스 해제
`Copy ctor`	`MyClass(const MyClass&)`
`Move ctor`	`MyClass(MyClass&&)` — 소유권 이전
`Copy assign`	`operator=(const MyClass&)`
`Move assign`	`operator=(MyClass&&)`

템플릿

함수 템플릿

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

클래스 템플릿

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

컨셉 (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 컨테이너

시퀀스 컨테이너

`vector<T>`	동적 배열, 빠른 임의 접근
`deque<T>`	양방향 큐
`list<T>`	이중 연결 리스트
`array<T,N>`	고정 크기 배열 (컴파일 타임 크기)
`forward_list<T>`	단방향 연결 리스트

연관 컨테이너

`map<K,V>`	정렬된 키-값 쌍 (레드-블랙 트리)
`set<T>`	정렬된 고유 요소
`unordered_map<K,V>`	해시 맵, 평균 O(1) 조회
`unordered_set<T>`	해시 셋, 평균 O(1) 조회
`multimap<K,V>`	정렬됨, 중복 키 허용

Vector 연산

```
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
```

이터레이터 & 알고리즘

이터레이터 사용

```
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
```

주요 알고리즘

`sort(begin, end)`	요소를 오름차순으로 정렬
`find(begin, end, val)`	값의 첫 번째 발생 위치 찾기
`count(begin, end, val)`	값의 발생 횟수 세기
`transform(b, e, out, fn)`	각 요소에 함수 적용
`accumulate(b, e, init)`	요소 축약 (기본: 합계)
`reverse(begin, end)`	요소 순서 뒤집기
`unique(begin, end)`	연속 중복 제거

범위 (C++20)

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

스마트 포인터

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
```

비교

`unique_ptr<T>`	단독 소유권, 오버헤드 없음
`shared_ptr<T>`	참조 카운팅을 통한 공유 소유권
`weak_ptr<T>`	`shared_ptr`의 비소유 관찰자
`make_unique<T>()`	`unique_ptr` 생성 권장 방법
`make_shared<T>()`	`shared_ptr` 생성 권장 방법

람다

람다 문법

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

캡처 모드

`[x]`	`x`를 값으로 캡처 (복사)
`[&x]`	`x`를 참조로 캡처
`[=]`	사용된 모든 변수를 값으로 캡처
`[&]`	사용된 모든 변수를 참조로 캡처
`[=, &x]`	모두 값으로, `x`는 참조로
`[this]`	둘러싼 객체 포인터 캡처

STL과 람다

```
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; });
```

문자열 & 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();
```

문자열 변환

`std::to_string(42)`	숫자를 문자열로
`std::stoi(s)`	문자열을 `int`로
`std::stod(s)`	문자열을 `double`로
`std::stol(s)`	문자열을 `long`으로

I/O 스트림

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

파일 I/O

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

에러 처리

예외

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

표준 예외

`std::exception`	모든 표준 예외의 기반 클래스
`std::runtime_error`	메시지가 있는 런타임 오류
`std::logic_error`	논리 오류 (전제조건 위반)
`std::out_of_range`	인덱스 또는 이터레이터 범위 초과
`std::invalid_argument`	잘못된 함수 인수
`std::bad_alloc`	메모리 할당 실패

noexcept

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

모던 C++ (17/20)

구조화 바인딩 (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);
```

주요 모던 기능

`auto`	변수 및 반환 타입 추론
`constexpr`	컴파일 타임 평가
`if constexpr`	컴파일 타임 조건문 (C++17)
`std::span<T>`	연속 데이터에 대한 비소유 뷰 (C++20)
`std::format()`	타입 안전 포맷팅 (C++20)
`co_await`	코루틴 지원 (C++20)