

# 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

<b>public</b>	어디서든 접근 가능
<b>protected</b>	클래스 및 파생 클래스에서 접근 가능
<b>private</b>	클래스 내에서만 접근 가능
<b>friend</b>	특정 함수 또는 클래스에 접근 권한 부여

### 특수 멤버

<b>Constructor</b>	<b>MyClass(args)</b> — 객체 초기화
<b>Destructor</b>	<b>~MyClass()</b> — 리소스 해제
<b>Copy ctor</b>	<b>MyClass(const MyClass&amp;)</b>
<b>Move ctor</b>	<b>MyClass(MyClass&amp;&amp;)</b> — 소유권 이전
<b>Copy assign</b>	<b>operator=(const MyClass&amp;)</b>
<b>Move assign</b>	<b>operator=(MyClass&amp;&amp;)</b>

## 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

<b>vector&lt;T&gt;</b>	동적 배열, 빠른 임의 접근
<b>deque&lt;T&gt;</b>	양방향 큐
<b>list&lt;T&gt;</b>	이중 연결 리스트
<b>array&lt;T,N&gt;</b>	고정 크기 배열 (컴파일 타임 크기)
<b>forward_list&lt;T&gt;</b>	단방향 연결 리스트

### 연관 컨테이너

<b>map&lt;K,V&gt;</b>	정렬된 키-값 쌍 (레드-블랙 트리)
<b>set&lt;T&gt;</b>	정렬된 고유 요소
<b>unordered_map&lt;K,V&gt;</b>	해시 맵, 평균 O(1) 조회
<b>unordered_set&lt;T&gt;</b>	해시 셋, 평균 O(1) 조회
<b>multimap&lt;K,V&gt;</b>	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

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

<b>unique_ptr&lt;T&gt;</b>	단독 소유권, 오버헤드 없음
<b>shared_ptr&lt;T&gt;</b>	참조 카운팅을 통한 공유 소유권
<b>weak_ptr&lt;T&gt;</b>	<b>shared_ptr</b> 의 비소유 관찰자
<b>make_unique&lt;T&gt;()</b>	<b>unique_ptr</b> 생성 권장 방법
<b>make_shared&lt;T&gt;()</b>	<b>shared_ptr</b> 생성 권장 방법

## 람다

### 람다 문법

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

### 캡처 모드

<b>[x]</b>	<b>x</b> 를 값으로 캡처 (복사)
<b> [&amp;x]</b>	<b>x</b> 를 참조로 캡처
<b>[=]</b>	사용된 모든 변수를 값으로 캡처
<b>[&amp;]</b>	사용된 모든 변수를 참조로 캡처
<b>[=, &amp;x]</b>	모두 값으로, <b>x</b> 는 참조로
<b>[this]</b>	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

<b>std::to_string(42)</b>	숫자를 문자열로
<b>std::stoi(s)</b>	문자열을 <b>int</b> 로
<b>std::stod(s)</b>	문자열을 <b>double</b> 로
<b>std::stol(s)</b>	문자열을 <b>long</b> 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

<b>std::exception</b>	모든 표준 예외의 기반 클래스
<b>std::runtime_error</b>	메시지가 있는 런타임 오류
<b>std::logic_error</b>	논리 오류 (전제조건 위반)
<b>std::out_of_range</b>	인덱스 또는 이터레이터 범위 초과
<b>std::invalid_argument</b>	잘못된 함수 인수
<b>std::bad_alloc</b>	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

<b>auto</b>	변수 및 반환 타입 추론
<b>constexpr</b>	컴파일 타임 평가
<b>if constexpr</b>	컴파일 타임 조건문 (C++17)
<b>std::span&lt;T&gt;</b>	연속 데이터에 대한 비소유 뷰 (C++20)
<b>std::format()</b>	타입 안전 포매팅 (C++20)
<b>co_await</b>	코루틴 지원 (C++20)