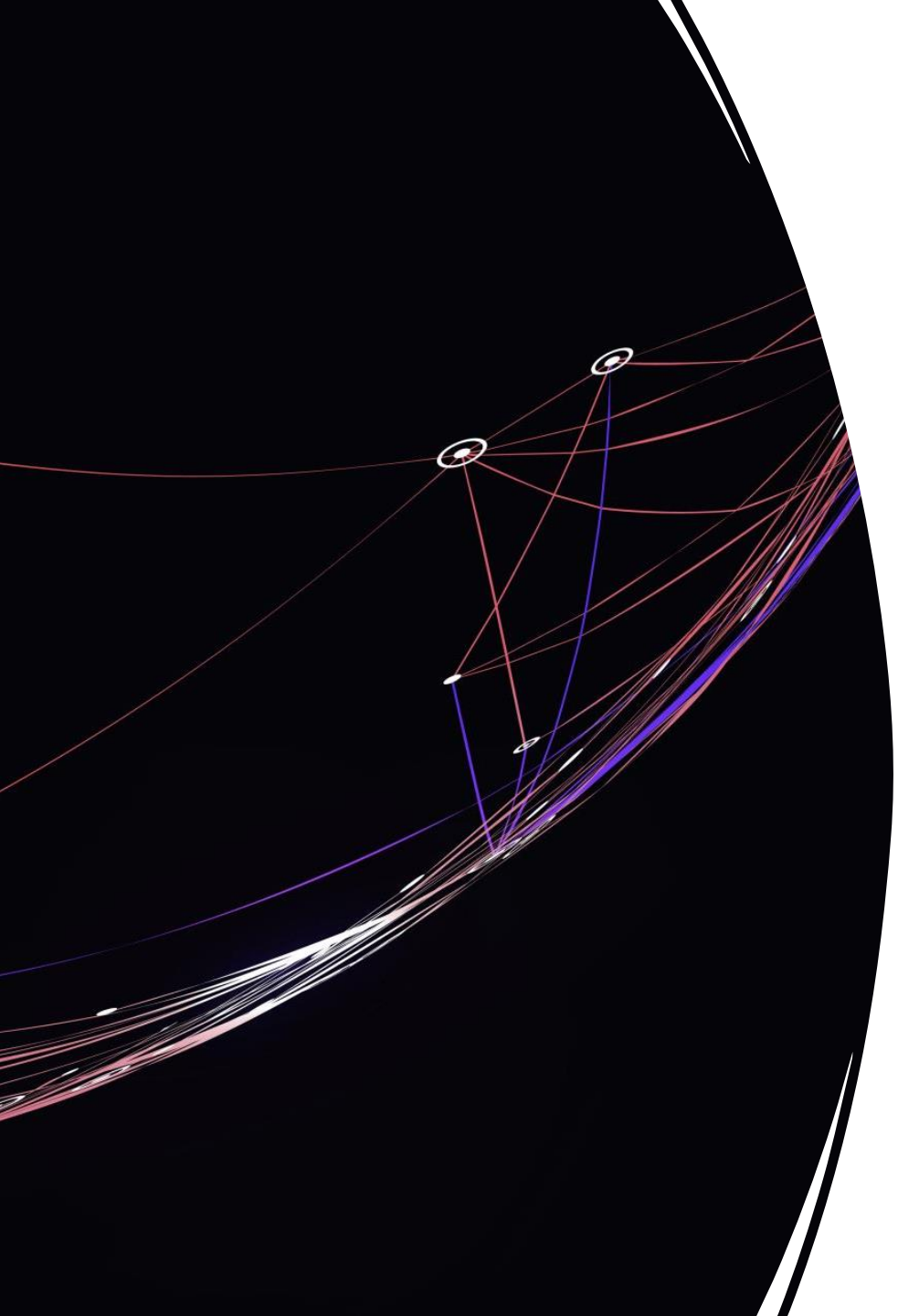
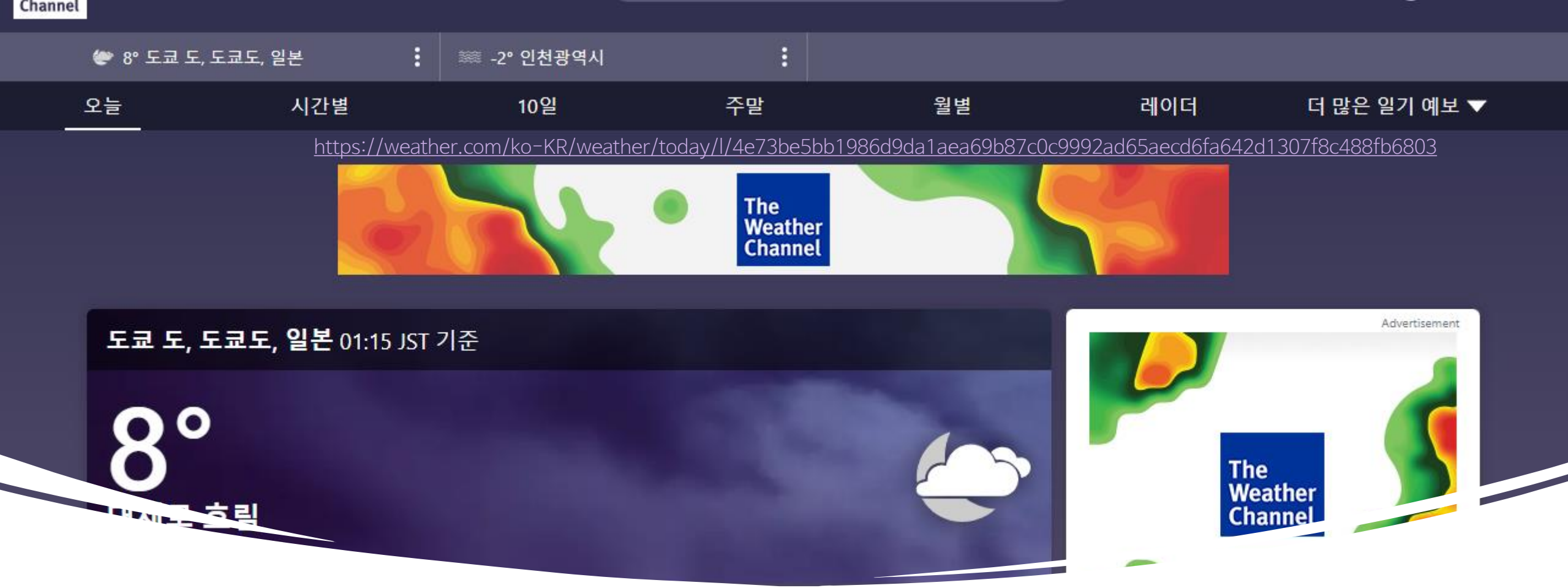


인트아이 C++ 심화 스터디

FINAL SUMMARY

인하대학교 인트아이



7회차 퀴즈 정답

```
const char *const host = "weather.com";
const char *const port = "443";
const char *const target = "/ko-KR/weather/today/l/4e73be5bb1986d9da1aea69b87c0c9992ad65aecd6fa642d1307f8c488fb6803";
```

스터디 일정

스터디 일정		매주 월/목 오후 3시-5시
1	1월 16일(월)	스터디 소개 Cppcheck, ClangFormat 설치 및 사용법 C++ Class
	1월 19일(목)	C++ RAII / Smart Pointer와 Guard 패턴
2	1월 23일(월)	C++ Functor, Lambda, Closure
	1월 26일(목)	C++ 동시성 제어 (Thread, Mutex, Semaphore)
3	1월 30일(월)	C++ Template
	2월 2일(목)	C++ SFINAE, Concepts 기초
4	2월 6일(월)	Vcpkg 설치 및 사용법 Boost 라이브러리 기초 (Asio, Beast)
	2월 9일(목)	스터디 마무리

1회차 스터디

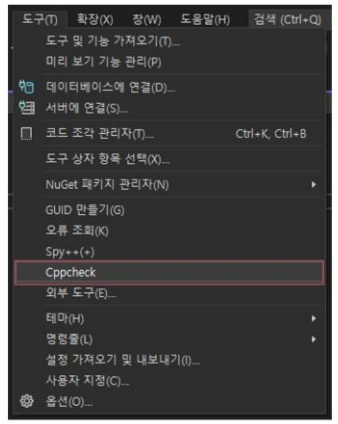
스터디 일정		매주 월/목 오후 3시-5시
1	1월 16일(월)	스터디 소개 Cppcheck, ClangFormat 설치 및 사용법 C++ Class
2		
3		
4		

① CppCheck C++ 코드에서 발생가능한 문제를 알려주는 린터(Linter)

② ClangFormat C++ 코드를 보기 좋게 정렬해주는 포맷터(Formatter)

③ C++의 클래스 상속 기능은 강력하지만, 남용하면 안 됨

Cppcheck




```
int main() {
    int *i = new int;
    delete i;
    return 0;
}
```

```
출력 보기 선택(S): Cppcheck
ss.cpp(4): error: Memory is allocated but not initialized: i
ss.cpp(2): style: Variable 'i' is allocated memory that is never used.
```

파일이 없다고 나오면 Ctrl+5로 코드 실행 한 번하고 돌리기

ClangFormat

```
#include <algorithm>
#include <iostream>
#include <vector>
using namespace std;

int A[1000];
int psum[1000];

int main()
    {
    int n, q;
    cin>>n>>q;
    for (int i = 1; i <= n; i++)
    {
        cin >> A[i];
    }
    sort(A, A + n);

    for (int i = 1; i <= n; i++)
    {
        psum[i]=psum[i -1]+A[i];
    }

    for (int i = 0; i<q; i++)
    {
        int L, R;
        cin >> L >> R;
        cout << psum[R] - psum[L - 1] << "\n";
    }
}
```

Ctrl+K Ctrl+D

→

```
#include <algorithm>
#include <iostream>
#include <vector>
using namespace std;

int A[1000];
int psum[1000];

int main() {
    int n, q;
    cin >> n >> q;
    for (int i = 1; i <= n; i++) {
        cin >> A[i];
    }
    sort(A, A + n);

    for (int i = 1; i <= n; i++) {
        psum[i] = psum[i - 1] + A[i];
    }

    for (int i = 0; i < q; i++) {
        int L, R;
        cin >> L >> R;
        cout << psum[R] - psum[L - 1] << "\n";
    }
}
```

2회차 스터디

RAII을 통해 Smart Pointer와 Scope Guard를 구현

스터디 일정		매주 월/목 오후 3시-5시
1	1월 16일(월)	스터디 소개 Cppcheck, ClangFormat 설치 및 사용법 C++ Class
	1월 19일(목)	C++ RAII / Smart Pointer와 Guard 패턴
	2월 9일(목)	스터디 마무리

RAII

Resource Acquisition Is Initialization

(자원의 획득은 초기화)

(=생성자)

대우 명제

소멸은 자원의 반납

(=소멸자)

Class의 생성자에서 new를 하고, 소멸자에서 delete를 한다.

Smart Pointer

- 프로그래머가 직접 delete하지 않아도, 사용이 끝나면 자동으로 delete되는 포인터
- RAII를 이용해 구현, 소멸자에 delete하는 코드를 넣어 줌
- C와 달리, C++ 코드는 스마트 포인터를 이용해 포인터가 최대한 안 보이게 숨겨야 함.
  - ✓ 프로그래머가 직접 관리하는 포인터가 많아질수록 프로그래머가 실수할 확률이 증가
  - ✓ 동적할당이 필요하면 RAII로 한 번 감싸서 사용
  - ✓ 가능한 한 스마트 포인터에 의존해 포인터를 관리

Scope Guard

- 일반적인 경우에서의 RAII의 활용
- 생성자와 소멸자를 이용해서 특정 기능이 자동으로 동작하도록 구현

```cpp
#pragma once
#include <iostream>

class CoutGuard {
public:
    CoutGuard() {
        std::cout << "{\n";
    }

    ~CoutGuard() {
        std::cout << "}\n";
    }
};
```

```cpp
#include <iostream>
#include "guard.h"

void test_guard() {
    CoutGuard _;
    std::cout << "after guard\n";
}
```

```
{
after guard
}
```

함수의 시작과 끝에 자동으로 중괄호를 출력함

3회차 스터디

Function Pointer는 인라이닝이 안되서 느림 → Functor로 인라이닝 가능하게 만듦 → Functor는 읽기 어려워서 Lambda가 나옴 → 외부의 변수를 Lambda 안에서 사용하기 위해 Closure 개념 도입

스터디 일정		매주 월/목 오후 3시
1	1월 16일(월)	스터디 소개 Cppcheck, ClangF C++ Class
	1월 19일(목)	C++ RAII / Smart P
2	1월 23일(월)	C++ Functor, Lambda, Closure

Functor [펑터]

=함수 객체(=함자) : 함수인 척하는 객체

객체이기 때문에 struct나 class 문법을 이용

함수처럼 동작해야 하므로 () 연산자를 오버로딩

함수 객체 클래스 자체를 타입으로 활용 가능 → C++ 템플릿과 결합하여 사용 가능

포인터 형태가 아니기 때문에 Function Pointer보다 inline 될 가능성이 높음

Lambda [람다]

=람다 함수(=익명 함수)

C++11에서 새로 추가된 문법 (functor 대체)

struct나 class를 선언할 필요가 없음

함수 포인터와 다르게, inline이 가능

```cpp
#include <algorithm>
#include <iostream>

int main() {
    int C[10] = { 5, 8, 2, 4, 7, 0, 6, 3, 9, 1 };

    // 내림차순: Lambda
    std::sort(C, C + 10, [](int a, int b) {
        return a > b;
    });
    for (int i : C) {
        std::cout << i << ' ';
    }
    std::cout << '\n';

    return 0;
}
```

Closure

람다 함수 내부에서 외부의 값에 접근할 수 있게 하는 방법

외부의 변수를 Capture하여 람다 함수에 풀어 줌

```cpp
int main() {
    int n = 5;
    int m = 5;  외부의 n을 Copy 방식으로 Capture
    auto addn = [n](int a, int b) {
        return a + b + n;
    };
    int j = addn(1, 2);
    return 0;
}
```

[n]	n을 복사하여 Capture
[n, m]	n과 m을 복사하여 Capture
[&n]	n을 참조 방식으로 Capture
[=]	외부의 모든 변수를 복사하여 Capture
[&]	외부의 모든 변수를 참조 방식으로 Capture

4회차 스터디

멀티코어 CPU를 전부 사용하기 위해서 작업을 병렬화해야 하는데, 작업의 단위를 스레드라 부름
다수의 스레드가 경쟁적으로 공유자원을 사용하는 것을 막기 위해 Mutex 사용,
Semaphore는 공유자원이 2개 이상일 때 사용

스터디 일정		매주 월/목 오
1	1월 16일(월)	스터디 소개 Cppcheck, C C++ Class
	1월 19일(목)	C++ RAII / S
2	1월 23일(월)	C++ Functor
	1월 26일(목)	C++ 동시성 제어 (Thread, Mutex, Semaphore)

Thread [스레드]

프로세스 = 메모리 안에서 실행 중인 프로그램

스레드 = 프로세스 안에서 코드를 동작시키는 단위

하나의 프로그램이 하나의 스레드만 사용하면 싱글 스레드, 여러 개의 스레드를 사용하면 멀티스레드

스레드는 전역 변수를 이용해 서로 값을 공유할 수 있음 (프로세스와의 차이점)

일반적인 C++ 코드는 하나의 스레드만 이용 → CPU 코어가 여러 개여도 하나 밖에 사용을 못함, 나머지 코어는 놀고 있음

프로그래머 직접 여러 개의 스레드를 사용할 수 있게 코드를 바꿔줘야 함

Mutex [뮤텍스]

상호 배제(mutual exclusion)의 약자

여러 개의 스레드가 공유자원(cout 등)을 경쟁적으로 사용할 때, 특정 스레드만 공유자원을 독점해서 사용할 수 있도록 하는 기능

```
#include <iostream>
#include <mutex>
#include <thread>

std::mutex mutex; 전역 변수로 mutex 선언

// Function Pointer
void f1() {
    for (int i = 0; i < 5; i += 1) {
        mutex.lock(); t1 스레드만 사용할 수 있게 mutex를 잠금, 나머지 스레드는 unlock 될 때까지 기다림
        std::cout << "f1();\n";
        mutex.unlock(); cout 사용이 끝나면 unlock 호출
    }
}
```

Semaphore [세마포어]

Mutex는 가질 수 있는 상태가 2가지(locked, unlocked)

Semaphore는 가질 수 있는 상태가 N가지인 Mutex (바꿔 말하면 Mutex는 Binary Semaphore)

5회차 스터디

Template을 이용한 정적 다형성의 구현

CRTP를 이용하면 virtual 없이 부모 클래스의 메소드를 재사용 가능(일반적인 상속보다 더 빠름)

스터디 일정		매주 월/목 오후 3시-
1	1월 16일(월)	스터디 소개 Cppcheck, ClangF C++ Class
	1월 19일(목)	C++ RAII / Smart P
2	1월 23일(월)	C++ Functor, Lam
	1월 26일(목)	C++ 동시성 제어 (T
3	1월 30일(월)	C++ Template
	2월 2일(목)	C++ SFINAE, Conc
4	2월 6일(월)	Vcpkg 설치 및 사용 Boost 라이브러리
	2월 9일(목)	스터디 마무리

Template [템플릿]

C 언어에는 없는 C++ 만의 특별한 문법

다양한 타입에 대한 코드를 여러 개 만들지 않아도 됨

컴파일 시간에 타입에 맞는 코드를 생성(코드를 찍어내는 틀) → 다형성 구현 가능

상속을 통해 구현한 다형성은 런타임(프로그램 실행 시간)에 필요한 코드를 불러오기 때문에 느림

템플릿을 통해 구현한 다형성은 컴파일 시간에 필요한 코드가 완성된 상태이기 때문에 빠름

CRTP

Curiously Recurring Template Pattern

: 기묘한 재귀 템플릿 패턴?

스스로를 인자로 받는 템플릿을 상속하는 클래스

virtual을 사용하지 않고 클래스의 메소드를 재사용하는 기법

curious ★★

1. 형용사 궁금한, 호기심이 많은 (=inquisitive)
2. 형용사 별난, 특이한, 기이한

recur

동사 되풀이 되다, 다시 일어나다

```
template<class T>
class Base {
};

class Derived : public Base<Derived> {
};
```

6회차 스터디

SFINAE는 C++ 템플릿의 핵심...이지만, 너무 어렵기 때문에 맛만 보고 넘어감

Concept는 타입에 제약조건 추가 (SFINAE 가독성 개선)

SFINAE와 Concept 모두 정적 다형성을 구현하기 위해 나옴 (템플릿에서 조건문 역할)

스터디 일정		매주 월/목 오후 3시-5시
3	2월 2일(목)	C++ SFINAE, Concepts 기초
4	2월 6일(월)	Vcpkg 설치 및 사용법 Boost 라이브러리 기초 (Asio, Beast)
	2월 9일(목)	스터디 마무리

SFINAE [스피내]

Substitution Failure Is Not An Error(치환 실패는 오류가 아님)

컴파일러가 잘못된 일부 템플릿 코드를 무시(오류x)할 수 있도록 기능

컴파일러가 상황에 따라 유효한 템플릿을 선택 → 템플릿에서의 조건문 구현

템플릿 특수화보다 더 복합적인 조건에 대해서 대응가능

(ex. 그냥 typename T가 아니라, T가 print 메소드를 가진 클래스 타입일 때만 선택되게)

CRTP와 마찬가지로, 정적 다형성 구현에 이용

Constraints & Concepts

C++20에서 추가, 타입에 제약조건을 걸 수 있는 새로운 문법

```
template<typename T>
struct Point {
    std::string name;
    T x;
    T y;
};

template<typename T>
    requires std::is_integral_v<T>  T가 is_integral_v를 만족해야 함
void print_point(const Point<T> &point) { // `T`가 정수형이면 이 함수를 오버로딩
    std::cout << point.name << '<' << typeid(T).name() << '>' << " = integral point (" << point.x << ", "
<< point.y << ")" << std::endl;
}

template<typename T>
void print_point(const Point<T> &point) { // `T`가 정수형이 아니면 이 함수를 오버로딩
     std::cout << point.name << '<' << typeid(T).name() << '>' << " = non-integral point (" << point.x << ", "
<< point.y << ")" << std::endl;
}
```

```
p0<int> = integral point (1, 2)
p1<__int64> = integral point (3, 4)
p2<float> = non-integral point (0.1, 0.2)
p3<double> = non-integral point (0.3, 0.4)
```

7회차 스터디

Vcpkg를 이용해 Boost 라이브러리 설치

Boost Beast를 이용해 weather.com 사이트 정보 가져오기 실습

스터디 일정		매주 월/목 오후 3
1	1월 16일(월)	스터디 소개 Cppcheck, Clan
4	2월 6일(월)	Vcpkg 설치 및 사용법 Boost 라이브러리 기초 (Asio, Beast)
	2월 9일(목)	스터디 마무리

Vcpkg

- 마이크로소프트에서 만든 C/C++ 라이브러리 관리자
- 명령어 한 번으로 Visual Studio에서 복잡한 외부 라이브러리 설정 과정을 생략할 수 있음
- Vcpkg는 Visual C++ Package Manager의 약자
- Vcpkg를 설치하기 위해서는 Git이 필요함(소스코드 다운받는 용도)

Boost

- 편의기능을 제공하는 함수들이 포함된 C++ 라이브러리
- Boost 라이브러리 개발자의 대부분이 C++ 표준 위원회 소속 → Boost의 기능이 C++ 신버전의 표준으로 들어오기도 함 (ex. 람다 문법)
- C++ 신버전에서만 제공하는 기능을 이전버전에서도 사용할 수 있게 함 (ex. std::thread는 C++11부터 지원하기 때문에, C++03에서 Thread가 필요하면 boost::thread를 이용)
- C++ 표준에는 없는 기능을 제공해주기도 함 (ex. 문자열 split)
- C++로 프로젝트를 진행한다면 거의 필수인 라이브러리

Boost Asio

- Asio = Asynchronous I/O (비동기 입출력)
- 비동기 = 함수의 결과가 바로 나오지 않음 (ex. readFile 함수로 파일을 읽을 때 파일 내용을 반환할 때까지 기다리지 않음)
- 함수의 반환 값을 기다리지 않기 때문에, Thread와 함께 사용하면 동시에 여러 작업을 처리할 수 있음
- 주로 실행시간이 오래 걸리는 파일, 네트워크(인터넷) 입출력에 이용
- Asio 안에는 비동기 관련 기능만 들어있는 것이 아니라, 입출력 전체에 관한 기능이 전부 들어있음(ex. TCP, UDP, IO context 등)

Boost Beast

- Boost에서 제공하는 HTTP 라이브러리
- HTTP를 이용해 인터넷에서 데이터를 가져오는 작업은 오래 걸리므로, 일반적으로는 Asio를 이용해 비동기로 진행
- 스터디에서 비동기까지 다루면 난이도가 너무 올라가기 때문에, 동기 방식으로 진행

수고하셨습니다