

# Introduction

프로그래밍 입문(2)

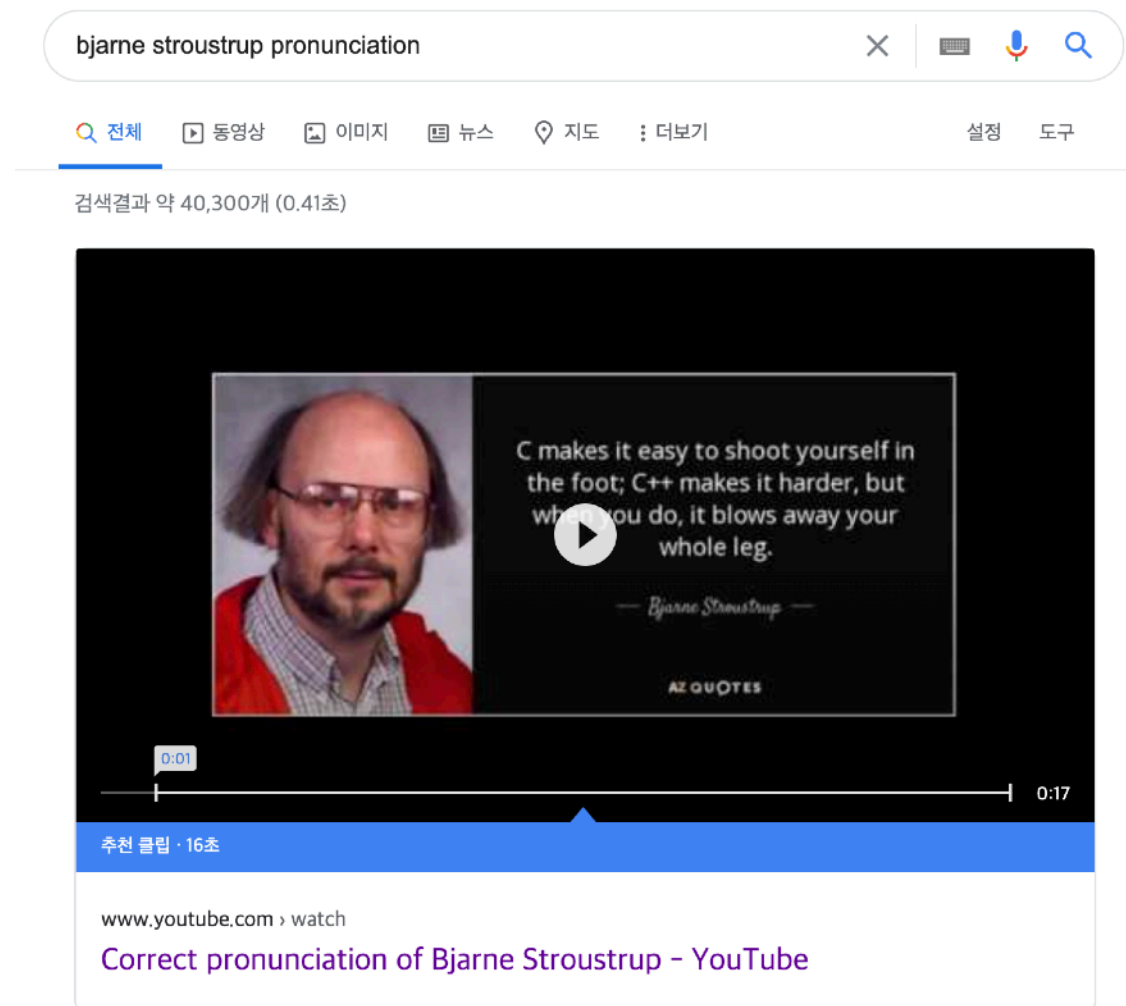
# Topics

- C++ 소개
- C++ 실습 준비

# C++ Introduction

# What is C++?

- Bjarne Stroustrup(비야네 스트룹 스트룹?)이 개발하여 1983년 발표.
- 처음은 C with Classes라는 이름에서 차후 C++로 이름이 변경됨.
- C를 기초로 삼아 객체지향 프로그래밍(OOP), 일반화(Generic)에 대한 지원이 추가되었음.



# Which Areas using C++?

- 성능이 중요시 되거나, 하드웨어에 접근할 필요가 있는 경우 자주 사용됨.
- System Programming
- Game / Graphics
- Embedded Software

# Why do we learn C++?

- C++ 언어 자체를 활용하기 위해서.
- OOP와 Generic 개념에 대해 이해하고 이를 활용하기 위해서.
- 매도 먼저 맞는게 낫기 때문에...?

# What's gonna be the topics?

- 기본적인 C++ 문법
- C++에서의 OOP: 객체와 클래스
- C++에서의 Generic: 템플릿
- 표준 템플릿 라이브러리 (Standard Template Library)

# 기본적인 C++ 문법

- 대부분 C에서 배웠던 것을 C++에서 복습한다고 생각하면 됩니다.
- 데이터 타입
- 문자열(String)
- 배열(Array)
- 제어문(Control Flow)
- 함수(Function)
- 포인터(Pointer)



# Object Oriented Programming (OOP)

- 객체(데이터)를 중심으로 하는 프로그래밍 패러다임.
- 모든 것을 객체로 나타내어 객체 내부에 데이터를 저장하고 이의 동작을 구현한 함수를 작성하여 프로그램을 만드는 방식.
  - e.g.) Screen 객체: width, height, pixels, print(), delete()
- 상속, 캡슐화, 다형성 등의 주요 개념들을 학습.

# Why OOP?

- 과거의 프로그램은 주로 수치계산을 위한 것으로, 명령어를 나열하고 이를 순차적으로 수행하여 원하는 결과를 얻는 것.
- 알고리즘이 중심이 되어 어떻게 명령어를 효율적으로 나열할 것인가가 프로그래밍의 중심이 됨 → 100만 개의 숫자를 작은 순으로 정렬.
- 하지만 점점 더 프로그램으로 많은 일을 처리하게 되면서 이런 방식은 더 이상 적합하지 않게 되었음.
- 더욱 복잡한 요구사항을 만족시키기 위해 다양한 개념을 프로그램에서 표현할 수 있는 객체 지향 프로그래밍이 발전하게 됨.

# Data Abstraction

- 다루는 데이터를 추상화하는 것이 복잡한 프로그램을 만드는데 도움이 됨.
- e.g.) 학생의 정보를 관리하는 프로그램
  - 학생의 정보에는 여러가지가 있을 수 있음 - 이름, 학번, 수강하는 과목, 성적, etc.
  - 이 모든 정보를 따로 관리(각각의 변수)하면 프로그램이 복잡해짐.
  - 학생(Student)라는 객체의 단위로 다루면 일이 훨씬 단순해짐.

# Generic Programming

- 특정 알고리즘을 구현한 프로그램이 그 알고리즘이 동작하는 데이터의 타입에 의존하지 않도록 일반화(generic)하여 프로그래밍 하는 방법.
- e.g.) 정렬 함수 `sort()`
  - 정수(integer)나 문자열(string) 여럿을 입력으로 받아 오름차순으로 정렬해주는 함수
  - 정렬 자체에는 동일한 알고리즘을 사용: bubble sort, quick sort, etc.
  - 일반화 없이는 정수 정렬 함수 `sort_int()`와 문자열 정렬 함수 `sort_string()`을 따로 작성해야 함.

# How does it look like?

- Hello World Program
- C와 상당히 유사합니다.
- 진입지점은 `main()` 함수.
- 화면 출력을 위한 `cout`.

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

int main() {
    cout << "Hello World!\n";
    return 0;
}
```

# Compare to C

- 전처리문과 화면 출력을 위한 명령이 다른 부분을 제외하면 코드가 거의 동일합니다.
- 이미 C문법에 익숙하다면, 비교를 통해 빠르게 C++문법을 익힐 수 있습니다.
- C++이 C를 기반으로 개발되었으므로 실제 유효한 C코드는 유효한 C++코드일 확률이 매우 높습니다.

## C++

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

int main() {
    cout << "Hello World!\n";
    return 0;
}
```

## C

```
#include<stdio.h>

int main() {
    printf("Hello World!\n");
    return 0;
}
```

# From Code To Execution

- C++도 C와 마찬가지로 작성된 코드를 실행하기 위하여 컴퓨터가 이해할 수 있는 형태로 변환해 주는 작업이 필요.
- 이 작업을 보통 컴파일(compile) 또는 빌드(build)라고 부름.
- C++코드 자체는 단순한 텍스트 파일이고, 컴파일러를 설치하여 이를 실행가능한 프로그램으로 바꾸는 것.
- 일반적인 흐름은 코드작성(.cpp파일) → 컴파일(.exe파일) → 프로그램 실행.

# C++ 실습

- 실습 내용은 대부분 강의시간에 설명한 내용이나 예제 등을 직접 코드를 작성하여 복습해 보는 것입니다.
- 실습은 간단한 프로그램을 실행하여 나오는 결과를 확인하는 방식으로 진행됩니다.
- 제공되는 실습 자료에 나온대로 따라하고, 결과를 확인합니다.
- 실제 실습을 수행했는지 확인하기 위해 나온 결과를 확인받아야 할 수 있습니다.



# C++ 실습 준비

- 개발환경 구축을 위하여 다음의 프로그램 설치가 필요합니다.
  - Compiler - MinGW or g++
  - IDE - VSCode
- 이미 자신이 익숙한 개발환경이 있다면 굳이 새로 환경을 구축할 필요는 없습니다.

# 개발환경 구축의 필요성

- 성공적인 소프트웨어 개발을 위한 필수적인 단계
- 프로그래밍(programming)은 단순히 코드 작성을 의미하는 코딩(coding) 외에도 프로그램의 설계(design), 검증(test), 버그수정(debugging) 등 다양한 작업을 포함.
- 또한 복잡하고 규모가 큰 프로그램의 경우 여러 명의 개발자가 동시에 작업을 해야하고, 지속적으로 변경되는 프로그램을 관리해야 함.
- 이런 작업들을 보다 효율적이고 능률적으로 할 수 있는 환경을 구축하는 것은 좋은 프로그래머가 되기 위한 필수조건.

# 개발환경 구축이란?

- 보통은 자신이 만들고자 하는 프로그램의 코드를 작성하고, 이를 실행하여 결과를 검증하는 작업을 할 수 있는 환경을 구축하는 것을 의미.
- 우리도 실습을 위해 이런 개발환경을 구축하는 것을 목표로 합니다.
  - 컴파일러(compiler) 또는 인터프리터(interpreter) 설치
  - 통합 개발 환경(IDE) 프로그램 설치
- 이 외에도 버전 관리(version control), 이슈 추적 시스템(issue tracking system) 등 추가적인 환경이 필요할 수 있지만 이 강의의 범주를 벗어남.

# Compiler

- 컴파일러(Compiler): 자신이 작성한 C++ 코드를 컴퓨터가 이해할 수 있도록 변환해주는 역할을 함.
- OS Dependent: 각 운영체제(Operating System, OS)별로 다른 컴파일러가 필요.
  - Windows: MinGW
  - macOS: g++ → Xcode 설치하면 자동으로 따라옴.
  - Linux: g++ → 보통 설치되어 있으나 없는 경우 g++ 패키지 설치.
    - CentOS: `sudo yum install gcc-c++`
    - Ubuntu: `sudo apt install g++`

# Integrated Development Environment

- IDE: 통합 개발환경으로 소스코드 작성에서부터 테스트, 빌드, 디버깅까지 지원하는 프로그램. (e.g., VSCode, Eclipse, IntelliJ, PyCharm, etc.)
- 주요 기능들
  - Syntax Highlight
  - Auto Completion
  - Build
  - Program Execution
  - Debugging Support

# Syntax Highlight

- 다른 색상으로 문법적으로 다른 위치의 단어들을 표시해 주는 기능.
- 코드의 가독성이 훨씬 좋아져 생산성을 높여줌.
- 문법적인 오류가 있는 경우 밑줄로 오류를 표시해주는 기능이 제공되는 경우도 있음.

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

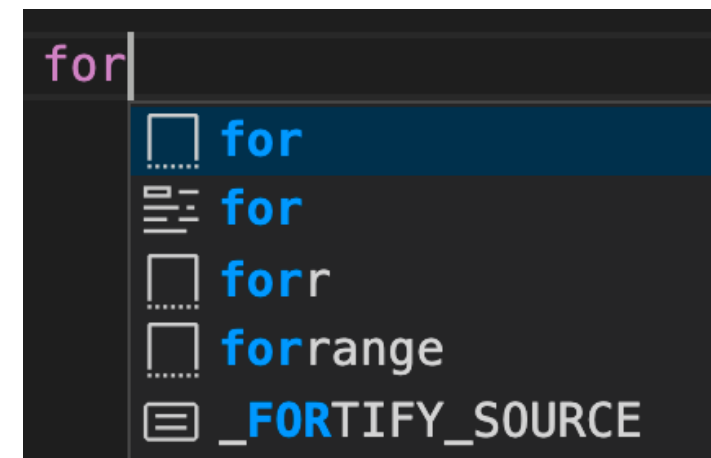
int main() {
    cout << "Hello World!\n";
    return 0;
}
```

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

int main() {
    cout << "Hello World!\n";
    return 0;
}
```

# Auto Completion

- 코드의 일부만 입력해주면 자동으로 나머지를 완성해 주는 기능.
- 개발자의 생산성 향상에 막대한 영향을 끼치는 기능임.
- 이를 효율적으로 하기위한 다양한 연구들이 다수 존재함.



```
for (size_t i = 0; i < count; i++)  
{  
    /* code */  
}
```

# Build

- 자동으로 필요한 파일들을 컴파일하여 실행가능한 프로그램을 만들어주는 기능.
- 대형 프로그램에서 다수의 파일이 존재하는 경우 컴파일 작업 자체가 매우 복잡하게 됨.
- IDE에서는 명령어 하나로 변경된 파일들을 다시 컴파일하여 수정 사항이 반영된 실행 프로그램을 생성할 수 있음.



# Program Execution

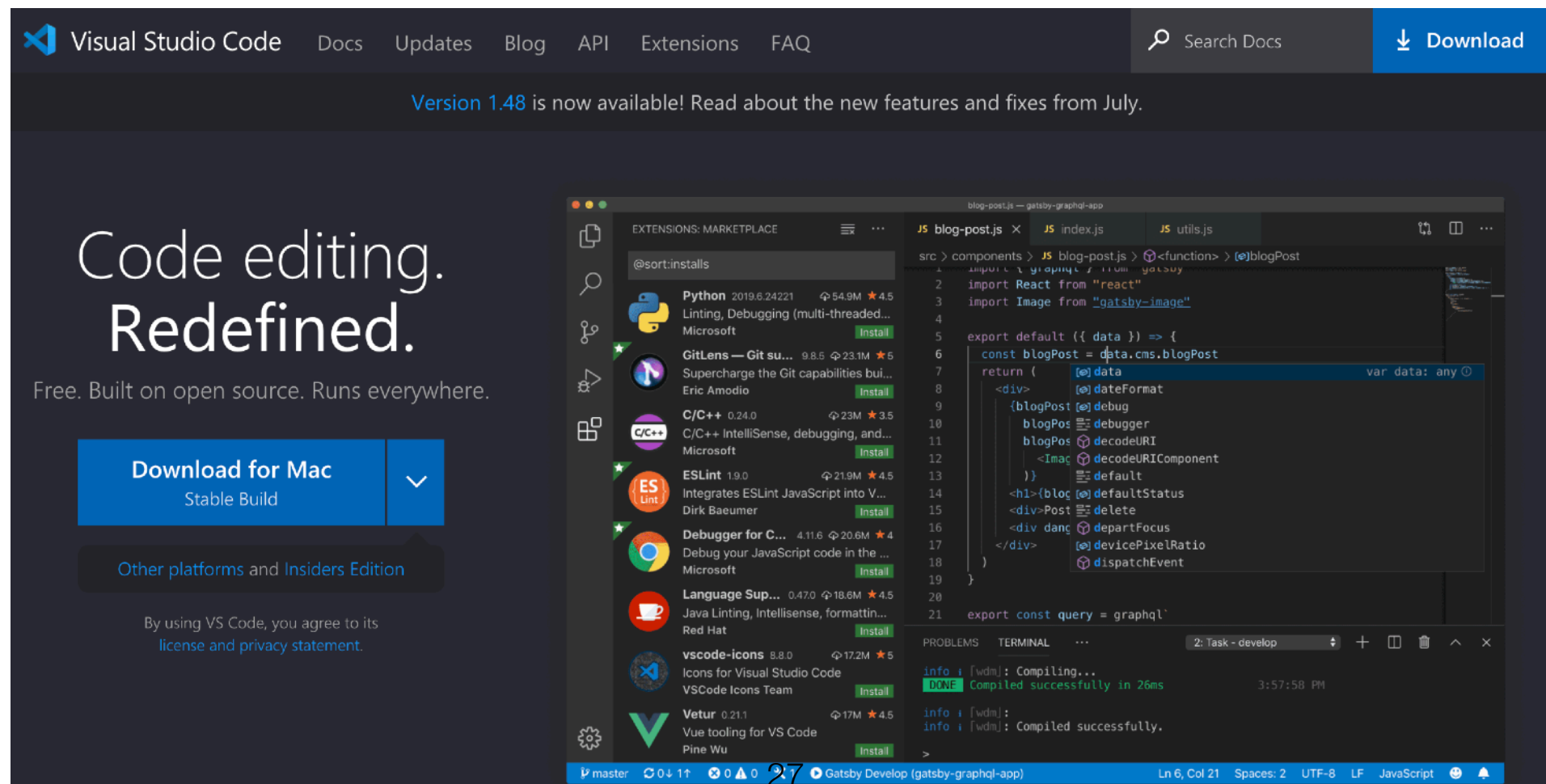
- IDE 내부에서 바로 프로그램을 실행하여 결과를 확인할 수 있음.
- 코드를 변경한 후 빌드 + 실행을 통해 바로 변경사항이 반영된 프로그램을 확인하며 개발을 진행하는 것이 가능.
- 프로그램에 입력이 필요하거나 특정한 실행 환경이 필요한 경우 한 번의 설정을 통해 반복적으로 실행할 수 있다는 장점이 있음.

# Debugging Support

- 언어별로 차이는 있으나, 일반적으로 코드를 순차적으로 수행하며 실행상태를 확인하는 것이 가능.
- 예를 들어, 코드의 15번째 줄에 Break Point를 설정하고 디버깅 모드로 실행하면 프로그램이 그 곳에 도달한 시점에서 일시 멈춤.
- 그 상태에서 각각의 변수들이 어떤 값을 가지고 있는지 확인하여 현재 프로그램의 상태가 의도한 대로인지 확인 가능.
- 또한 멈춘 프로그램을 다시 정해진 만큼씩 실행하며 상태가 어떻게 변해가는지 관찰하는 기능이 지원되기도 함.

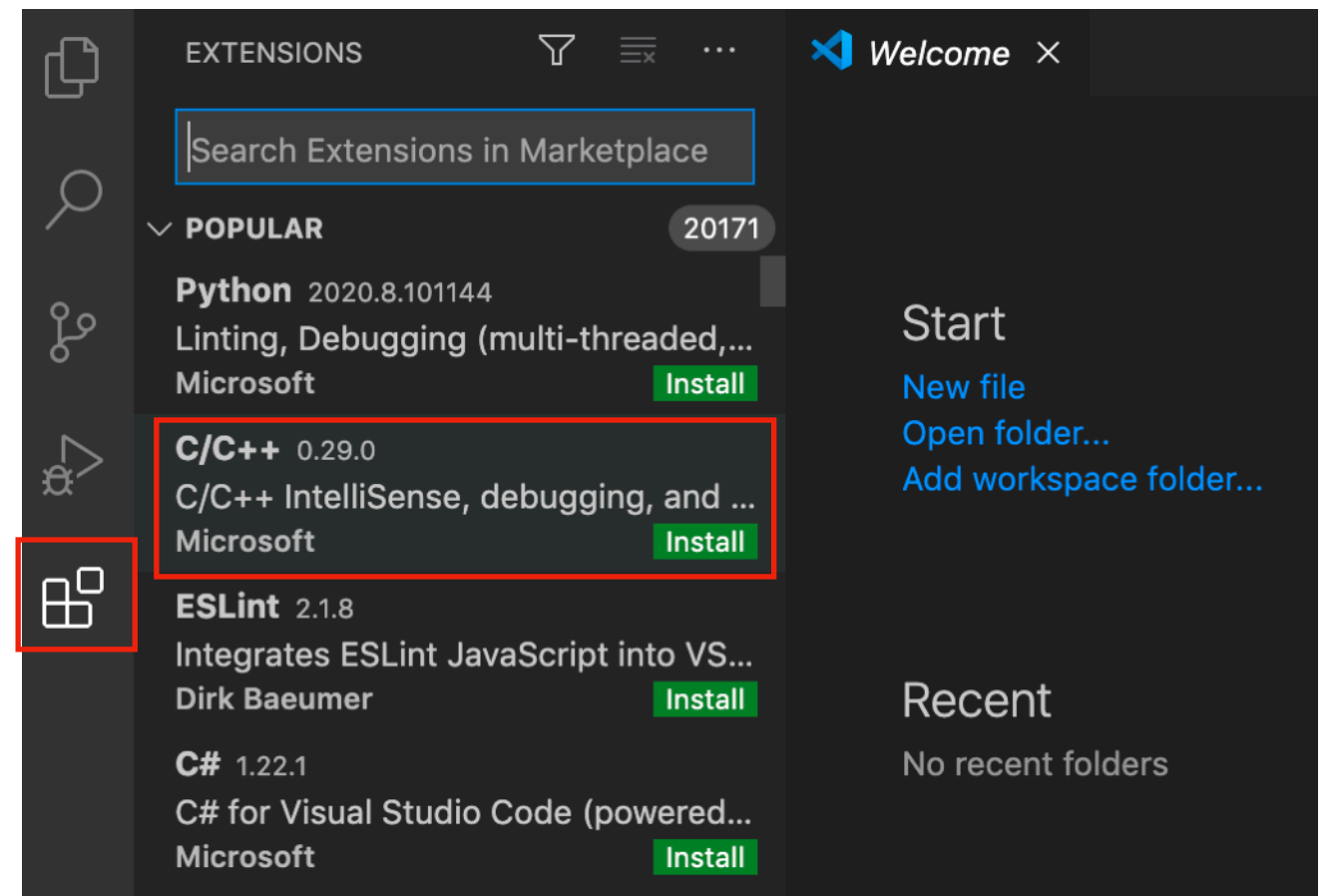
# VSCode

- Visual Studio Code: Microsoft에서 개발한 무료 IDE.
- Windows, Mac, Linux를 모두 지원
- Extension으로 다양한 언어를 지원



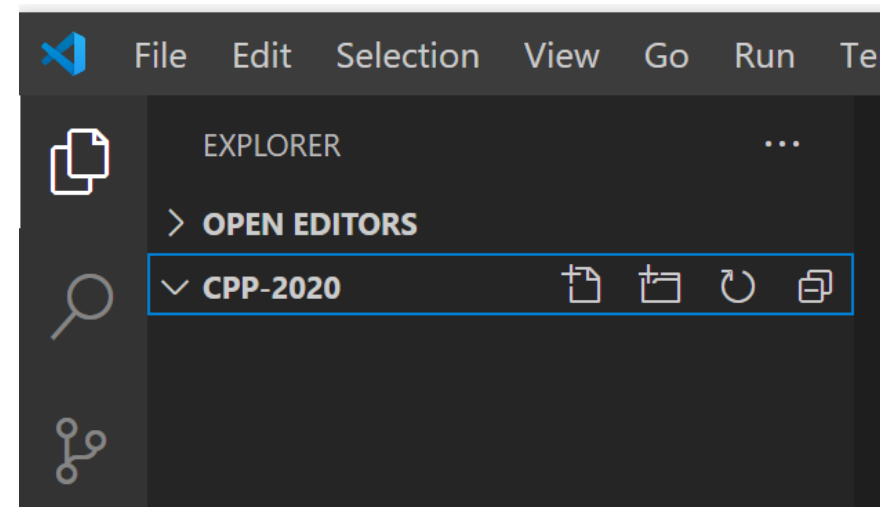
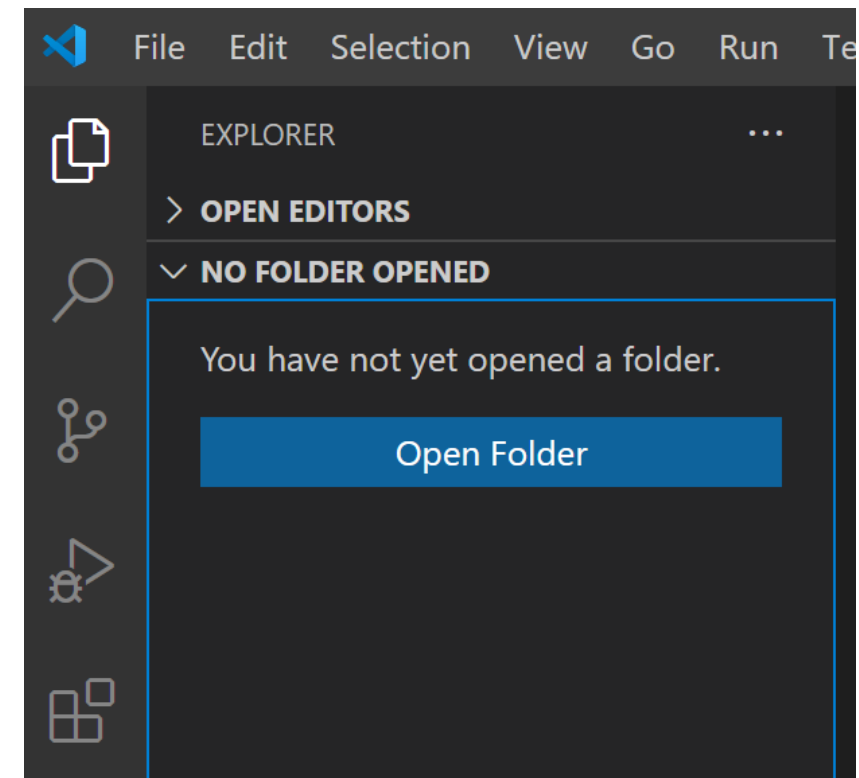
# Extensions

- 다양한 언어에 대한 지원이 확장 프로그램의 형태로 지원됨.
- 새로운 언어 개발환경을 구축하기 위해 보통 다음과 같은 과정을 거칩니다.
  1. 원하는 언어의 컴파일러나 인터프리터 등을 설치.
  2. Extension을 설치하고 1에서 설치한 것에 맞춰 설정.



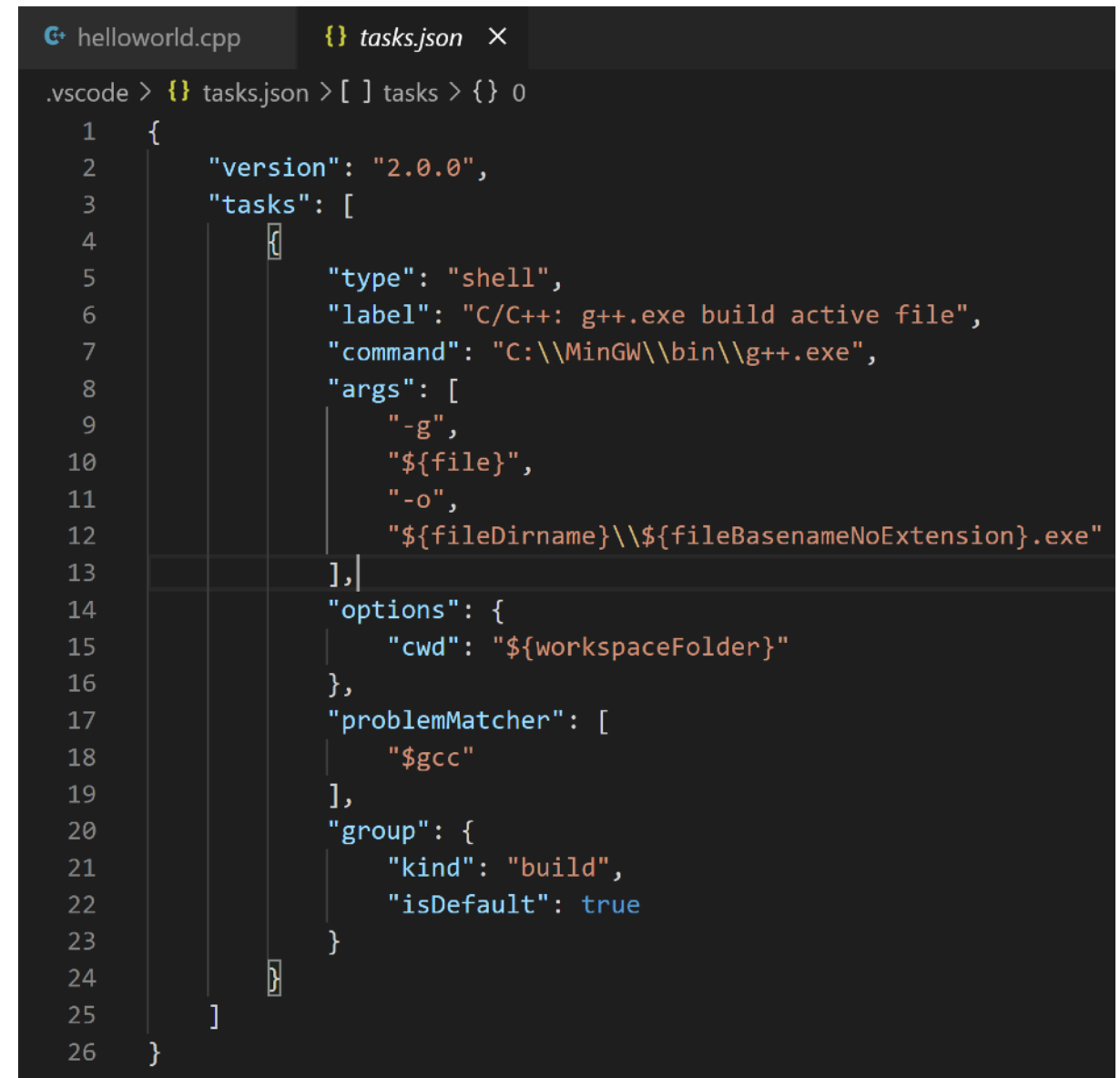
# Workspace

- 개발을 위한 작업공간.
- 여러분의 프로그램에 관련된 파일들이 모여있는 집합소.
- Workspace별로 환경설정을 다르게 하거나, workspace 내에서 여러 프로그램들이 하나의 설정을 공유하는 것도 가능.
- VSCode에서는 File > Open으로 폴더를 생성하고 여는 것으로 workspace 생성 및 열기가 가능함.



# 빌드 및 실행 설정

- 앞으로 작성할 코드를 빌드(컴파일)하고 실행하기 위하여 추가적인 설정이 필요함.
- VSCode에서는 이 때 실행되어야 할 Task를 tasks.json 파일로 저장함.
- 오른쪽은 Windows에서 VSCode가 기본 제공하는 tasks.json의 예시임.
- 구글에서 'VSCode C++ 개발환경 구축'을 검색하면 다양한 설정을 확인할 수 있음.



```
.vscode > {} tasks.json > [ ] tasks > {} 0
1  {
2      "version": "2.0.0",
3      "tasks": [
4          {
5              "type": "shell",
6              "label": "C/C++: g++.exe build active file",
7              "command": "C:\\MinGW\\bin\\g++.exe",
8              "args": [
9                  "-g",
10                 "${file}",
11                 "-o",
12                 "${fileDirname}\\${fileBasenameNoExtension}.exe"
13             ],
14             "options": {
15                 "cwd": "${workspaceFolder}"
16             },
17             "problemMatcher": [
18                 "$gcc"
19             ],
20             "group": {
21                 "kind": "build",
22                 "isDefault": true
23             }
24         }
25     ]
26 }
```

# Hello World

- 첫 프로그램의 대명사 Hello World를 확인해 봅시다.
- helloworld.cpp 파일에 간단한 메시지를 출력하는 코드를 작성하였습니다.
- 개발환경 구축이 완료되면 VSCode에서 이 코드를 쉽게 빌드하고 실행할 수 있습니다.

```
helloworld.cpp X
practice1 > helloworld.cpp > main()
1  #include<iostream>
2  using namespace std;
3
4  int main() {
5      cout << "Hello World!!\n";
6      return 0;
7  }
```

# Hello World 빌드

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

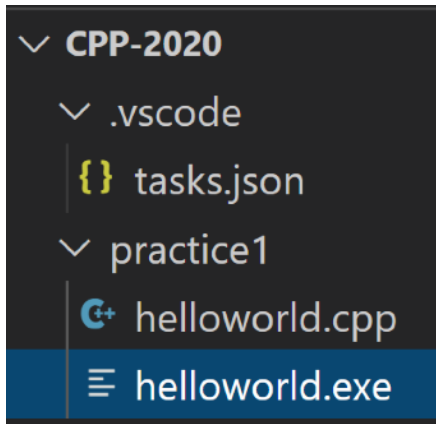
```
> Executing task: C:\MinGW\bin\g++.exe -g c:\cpp-2020\practice1\helloworld.cpp -  
o c:\cpp-2020\practice1\helloworld.exe <
```

Terminal will be reused by tasks, press any key to close it.

- helloworld.cpp를 선택하고, 빌드를 실행.
- 위와 같이 컴파일러가 실행되는 화면이 밑의 TERMINAL에 표시됩니다.
- 보통은 단축키나 간단한 메뉴선택으로 빌드를 실행할 수 있습니다.



# Hello World 실행



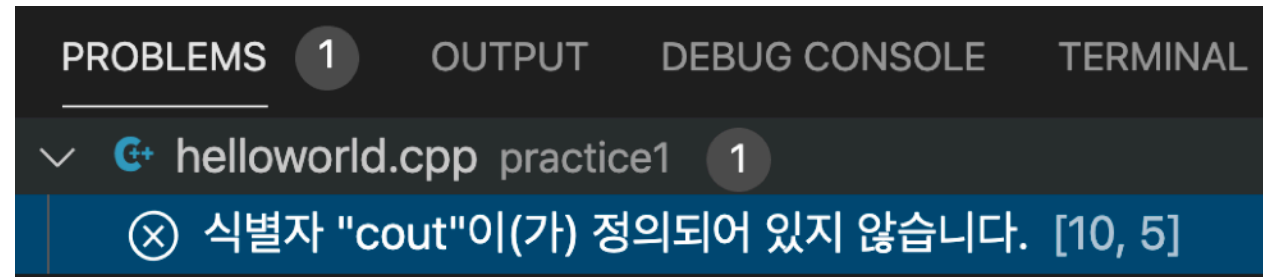
```
> Executing task: C:\Windows\system32\cmd.exe /C c:\cpp-2020\practice1\helloworld.exe <

Hello World!!

Terminal will be reused by tasks, press any key to close it.
```

- 빌드가 성공적이었다면 helloworld.exe 파일이 생성됩니다.
- helloworld.exe를 VSCode에서 실행해 볼 수 있습니다.
- 실행을 위한 단축키를 누르거나 메뉴를 선택하면 위와 같이 실행되어 TERMINAL에 메시지가 출력되는 것을 볼 수 있습니다.

# 에러 확인방법



- 코드 작성 중, 뭔가 잘못된 부분이 있다면 VSCode 하단의 PROBLEMS탭에 문제와 위치가 표시됩니다.
- 또는 빌드를 실행하고 나면 TERMINAL 탭에 컴파일 에러 메시지가 표시됩니다.

```
/Users/jindae/vscode-workspace/cpp-2020/practice1/helloworld.cpp:10:5:
error:
    use of undeclared identifier 'cout'; did you mean 'std::cout'?
    cout << "Hello World!!\n";
    ^~~~
    std::cout
/Library/Developer/CommandLineTools/usr/include/c++/v1/iostream:54:33:
note:
    'std::cout' declared here
extern _LIBCPP_FUNC_VIS ostream cout;
1 error generated.
```

# Summary

- C++에 대한 소개
- Object Oriented Programming
- Generic Programming
- C++ 실습 준비
- 통합 개발 환경(IDE)