* Prerequisite

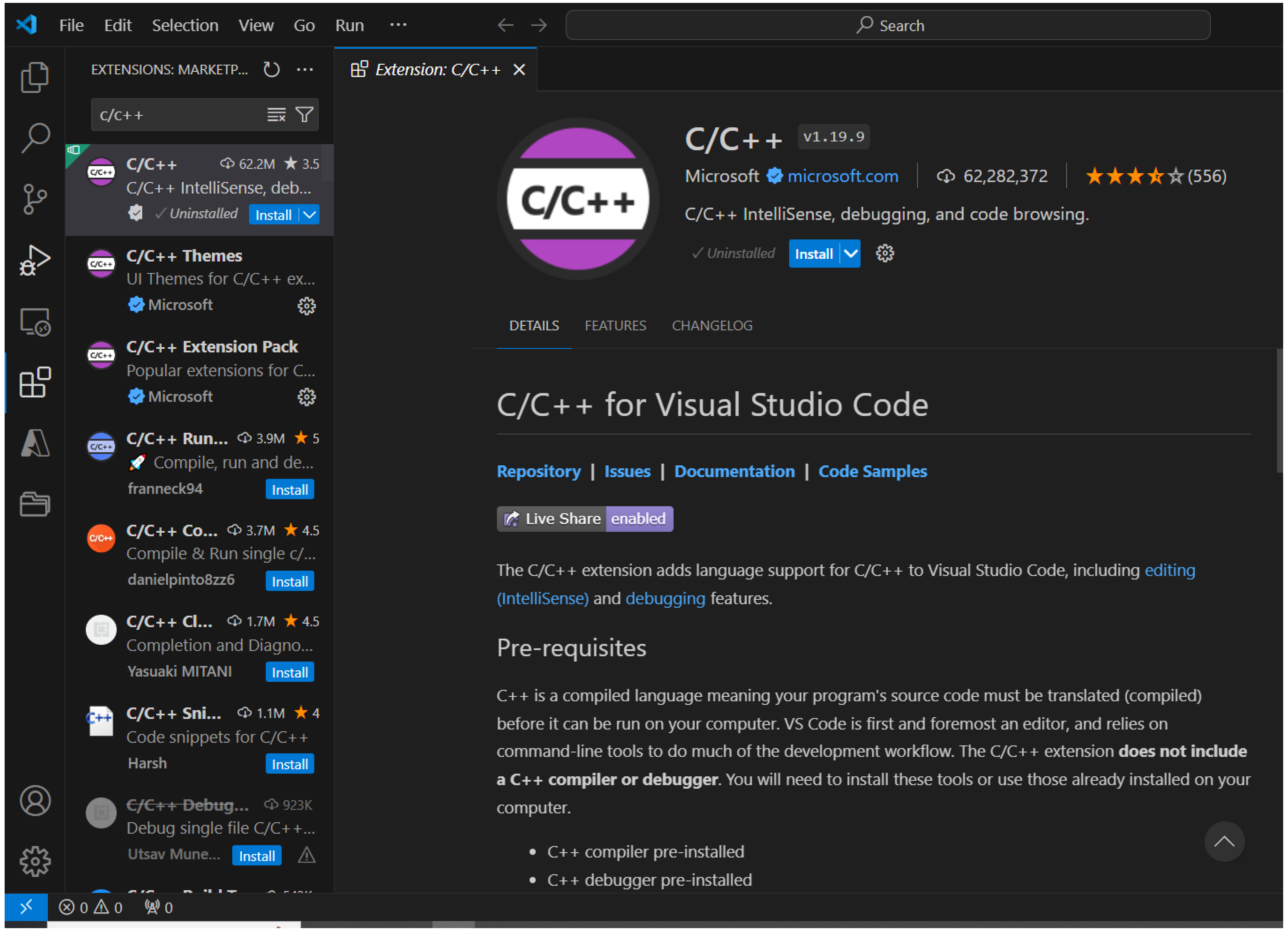
Debugging in Visual Studio Code

Jongwook Han

Overview

- Extension
  - C/C++
  - For Mac(M1/M2)
- Configuration
  - c_cpp_properties.json
  - tasks.json
  - launch.json
- Exercise
  - 1. Checking local variables
  - 2. Multiple .cpp files

Extension: C/C++



C/C++ 62.2M ★ 3.5
C/C++ IntelliSense, deb...
✓ Uninstalled Install

C/C++ Themes
UI Themes for C/C++ ex...
Microsoft

C/C++ Extension Pack
Popular extensions for C...
Microsoft

C/C++ Run... 3.9M ★ 5
🚀 Compile, run and de...
franneck94 Install

C/C++ Co... 3.7M ★ 4.5
Compile & Run single c/...
danielpinto8zz6 Install

C/C++ Cl... 1.7M ★ 4.5
Completion and Diagno...
Yasuaki MITANI Install

C/C++ Sni... 1.1M ★ 4
Code snippets for C/C++
Harsh Install

~~C/C++ Debug...~~ 923K
Debug single file C/C++...
Utsav Mune... Install

C/C++ v1.19.9
Microsoft microsoft.com | 62,282,372 | ★★★★☆ (556)
C/C++ IntelliSense, debugging, and code browsing.
✓ Uninstalled Install

C/C++ for Visual Studio Code

Repository | **Issues** | **Documentation** | **Code Samples**

Live Share enabled

The C/C++ extension adds language support for C/C++ to Visual Studio Code, including editing (IntelliSense) and debugging features.

Pre-requisites

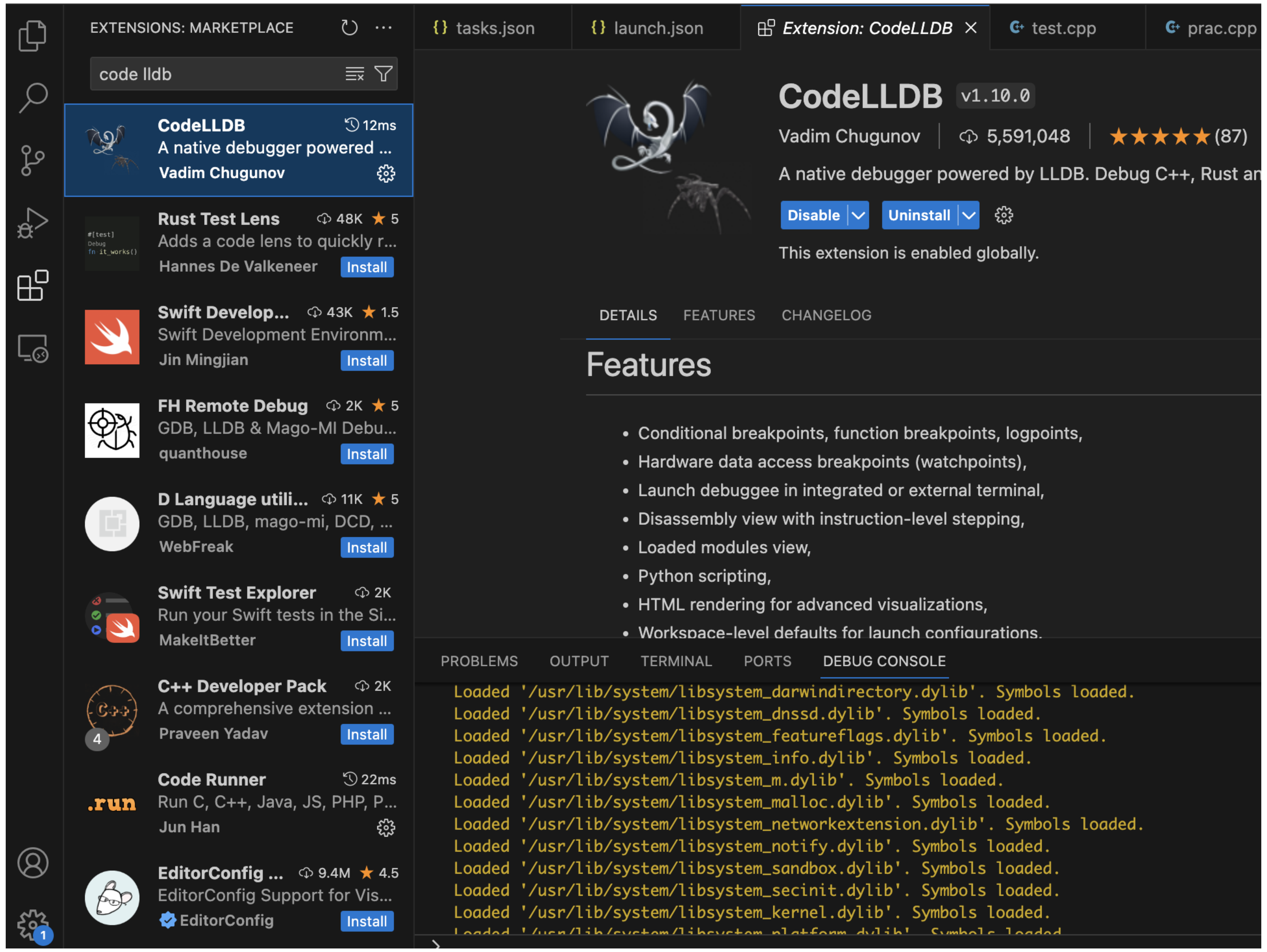
C++ is a compiled language meaning your program's source code must be translated (compiled) before it can be run on your computer. VS Code is first and foremost an editor, and relies on command-line tools to do much of the development workflow. The C/C++ extension **does not include a C++ compiler or debugger**. You will need to install these tools or use those already installed on your computer.

- C++ compiler pre-installed
- C++ debugger pre-installed

⊗ 0 ⚠ 0 0

- Download the C/C++ extension in VSCode

For Mac(M1/M2) users



CodeLLDB 12ms
A native debugger powered ...
Vadim Chugunov

Rust Test Lens 48K ★ 5
Adds a code lens to quickly r...
Hannes De Valkeneer Install

Swift Develop... 43K ★ 1.5
Swift Development Environm...
Jin Mingjian Install

FH Remote Debug 2K ★ 5
GDB, LLDB & Mago-MI Debu...
quanthouse Install

D Language utili... 11K ★ 5
GDB, LLDB, mago-mi, DCD, ...
WebFreak Install

Swift Test Explorer 2K
Run your Swift tests in the Si...
MakeItBetter Install

C++ Developer Pack 2K
A comprehensive extension ...
Praveen Yadav Install

Code Runner 22ms
Run C, C++, Java, JS, PHP, P...
Jun Han

EditorConfig ... 9.4M ★ 4.5
EditorConfig Support for Vis...
EditorConfig Install

CodeLLDB v1.10.0
Vadim Chugunov | 5,591,048 | ★★★★★ (87)
A native debugger powered by LLDB. Debug C++, Rust an

This extension is enabled globally.

Features

- Conditional breakpoints, function breakpoints, logpoints,
- Hardware data access breakpoints (watchpoints),
- Launch debuggee in integrated or external terminal,
- Disassembly view with instruction-level stepping,
- Loaded modules view,
- Python scripting,
- HTML rendering for advanced visualizations,
- Workspace-level defaults for launch configurations.

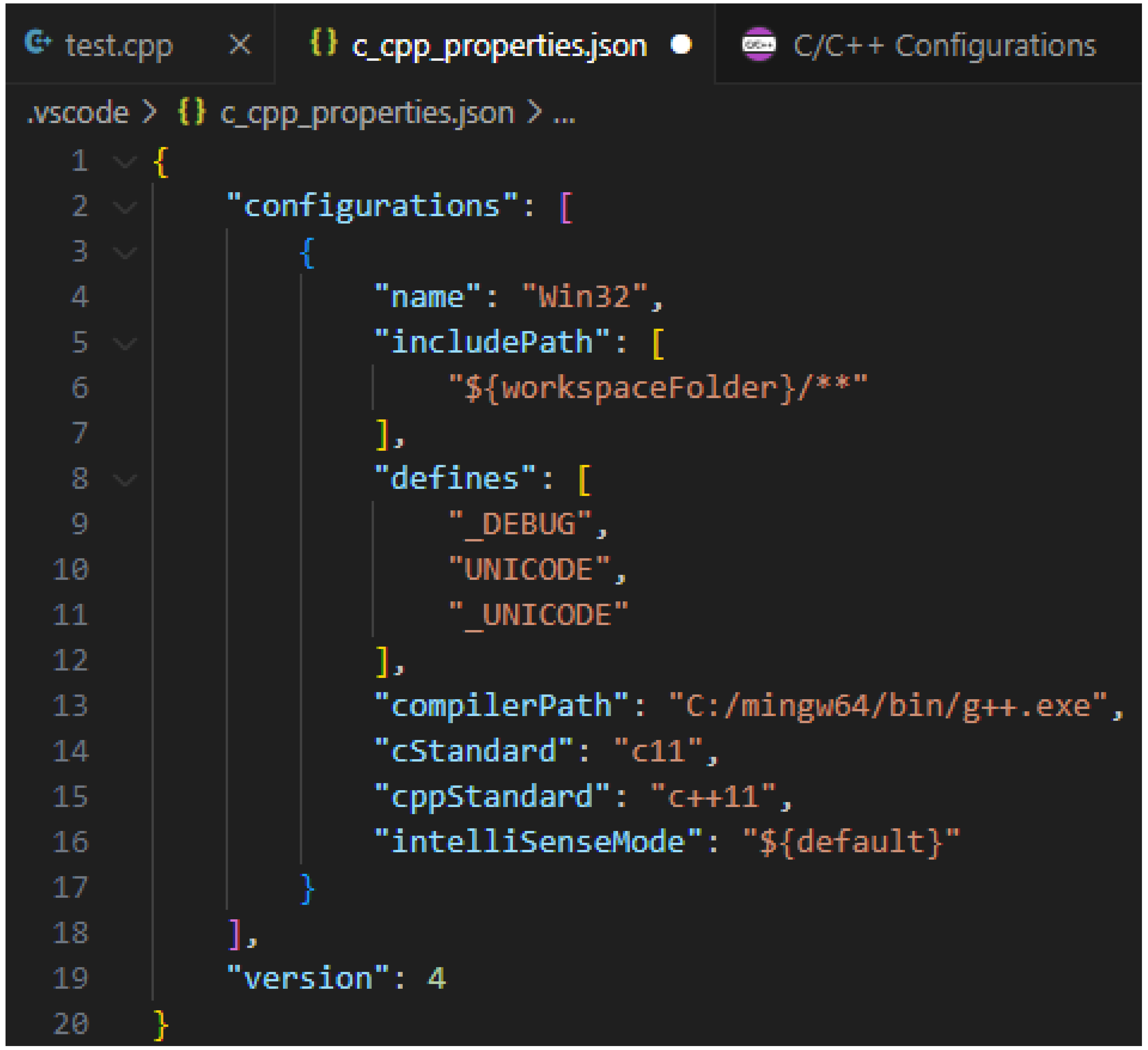
```
Loaded '/usr/lib/system/libsystem_darwindirectory.dylib'. Symbols loaded.
Loaded '/usr/lib/system/libsystem_dnssd.dylib'. Symbols loaded.
Loaded '/usr/lib/system/libsystem_featureflags.dylib'. Symbols loaded.
Loaded '/usr/lib/system/libsystem_info.dylib'. Symbols loaded.
Loaded '/usr/lib/system/libsystem_m.dylib'. Symbols loaded.
Loaded '/usr/lib/system/libsystem_malloc.dylib'. Symbols loaded.
Loaded '/usr/lib/system/libsystem_networkextension.dylib'. Symbols loaded.
Loaded '/usr/lib/system/libsystem_notify.dylib'. Symbols loaded.
Loaded '/usr/lib/system/libsystem_sandbox.dylib'. Symbols loaded.
Loaded '/usr/lib/system/libsystem_secinit.dylib'. Symbols loaded.
Loaded '/usr/lib/system/libsystem_kernel.dylib'. Symbols loaded.
Loaded '/usr/lib/system/libsystem_platform.dylib'. Symbols loaded.
```

- Download the CodeLLDB extension
- Reason:
- The GDB(GNU Debugger) is not currently supported in M1/M2 Mac
- Instead use the LLDB(Low-Level Debugger)

Configuration overview

- We need to make the following three files in ./vscode folder:
- 1. c_cpp_properties.json
- 2. tasks.json
- 3. launch.json

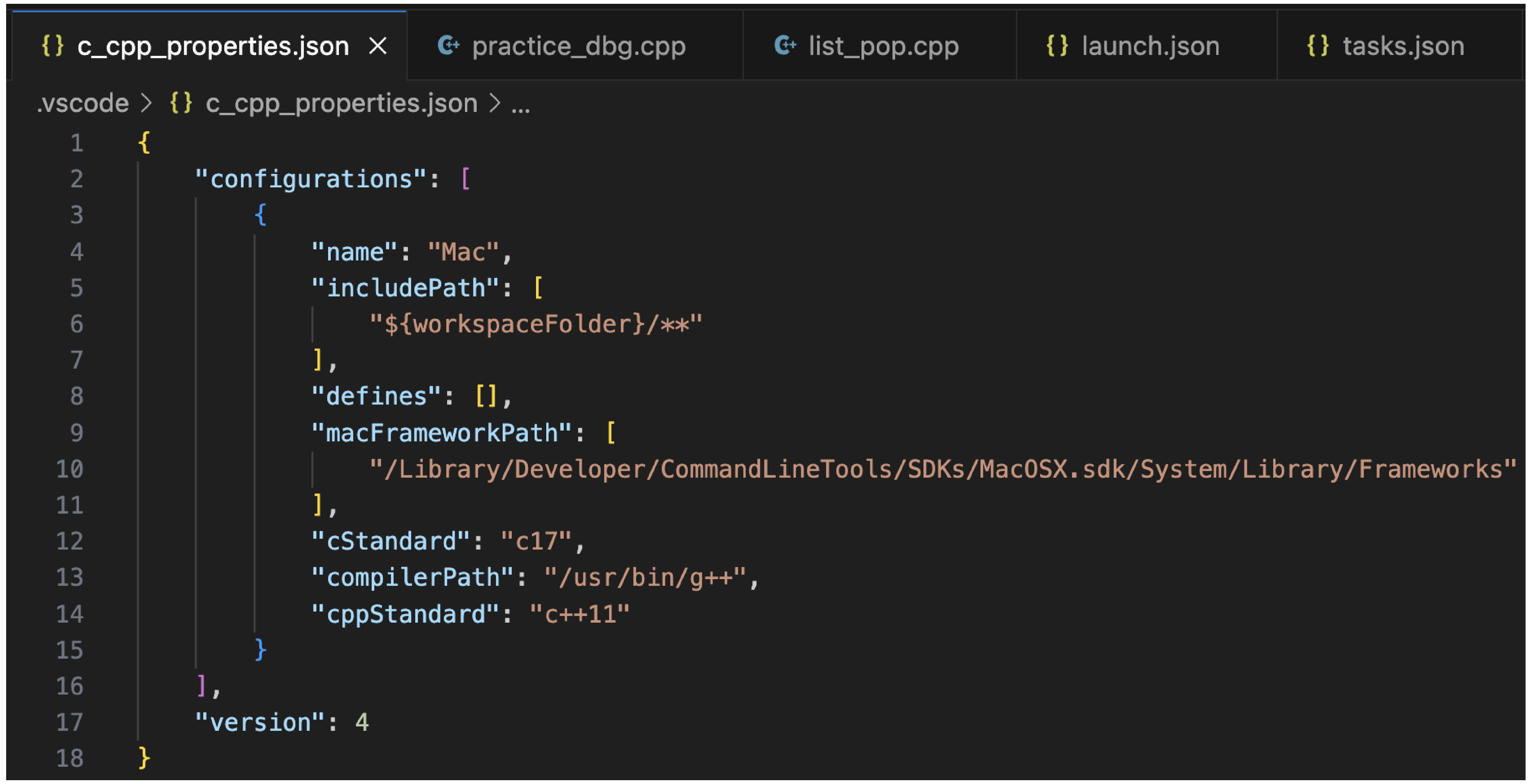
c_cpp_properties.json



```json
{
    "configurations": [
        {
            "name": "Win32",
            "includePath": [
                "${workspaceFolder}/**"
            ],
            "defines": [
                "_DEBUG",
                "UNICODE",
                "_UNICODE"
            ],
            "compilerPath": "C:/mingw64/bin/g++.exe",
            "cStandard": "c11",
            "cppStandard": "c++11",
            "intelliSenseMode": "${default}"
        }
    ],
    "version": 4
}
```

- The left figure is an example of the json file in Windows.
- includePath: Specify the path to the header files that you will use.
- compilerPath: Path to the compiler. In this session we use g++.
- cppStandard: The version of the C++ language standard to use for IntelliSense. In this session we use c++11.
- intelliSenseMode: The IntelliSense mode to use that maps to the computer architecture. Set to default.

c_cpp_properties.json

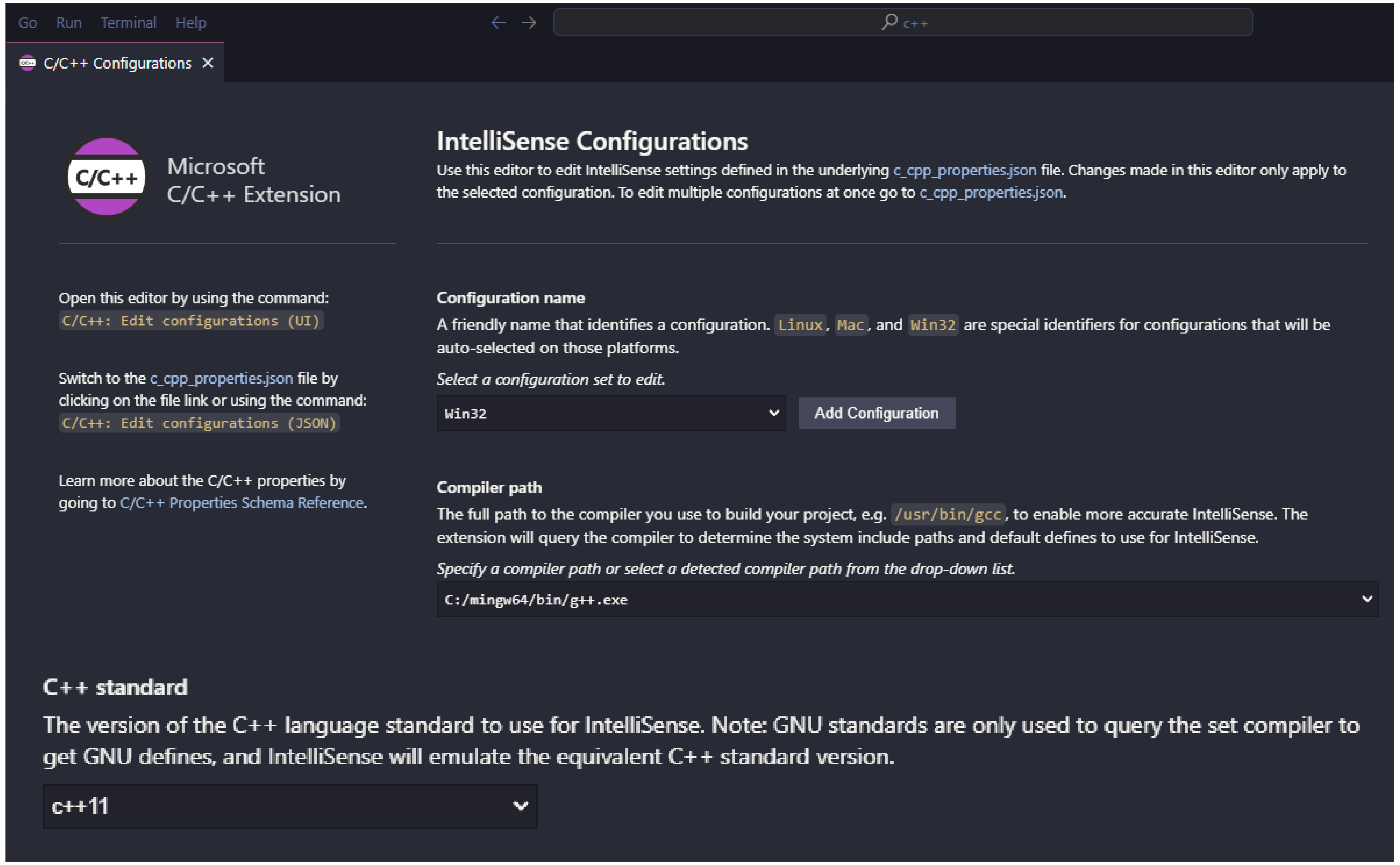


```json
{
    "configurations": [
        {
            "name": "Mac",
            "includePath": [
                "${workspaceFolder}/**"
            ],
            "defines": [],
            "macFrameworkPath": [
                "/Library/Developer/CommandLineTools/SDKs/MacOSX.sdk/System/Library/Frameworks"
            ],
            "cStandard": "c17",
            "compilerPath": "/usr/bin/g++",
            "cppStandard": "c++11"
        }
    ],
    "version": 4
}
```

- The left figure is an example of the json file in Mac.
- For more information about c_cpp_properties.json see link

How to make c_cpp_properties.json

- Windows: press F1 / ctrl + shift + p -> select C/C++: Edit configurations(UI)
- Mac: press command + shift + p -> select C/C++: Edit configurations(UI)



IntelliSense Configurations

Use this editor to edit IntelliSense settings defined in the underlying c_cpp_properties.json file. Changes made in this editor only apply to the selected configuration. To edit multiple configurations at once go to c_cpp_properties.json.

Open this editor by using the command: `C/C++: Edit configurations (UI)`

Switch to the c_cpp_properties.json file by clicking on the file link or using the command: `C/C++: Edit configurations (JSON)`

Learn more about the C/C++ properties by going to C/C++ Properties Schema Reference.

Configuration name

A friendly name that identifies a configuration. `Linux`, `Mac`, and `Win32` are special identifiers for configurations that will be auto-selected on those platforms.

Select a configuration set to edit.

Win32 — Add Configuration

Compiler path

The full path to the compiler you use to build your project, e.g. `/usr/bin/gcc`, to enable more accurate IntelliSense. The extension will query the compiler to determine the system include paths and default defines to use for IntelliSense.

Specify a compiler path or select a detected compiler path from the drop-down list.

C:/mingw64/bin/g++.exe

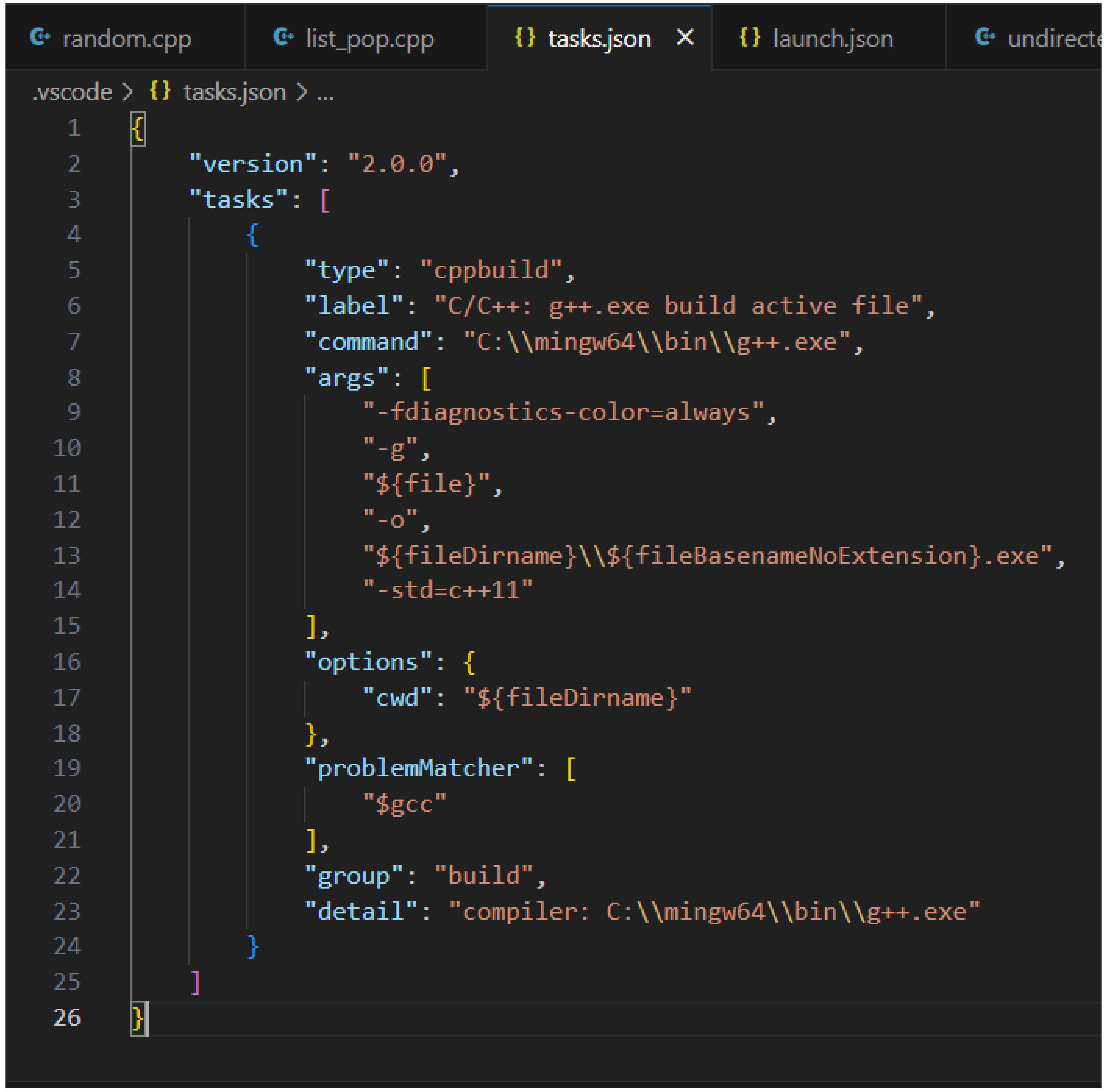
C++ standard

The version of the C++ language standard to use for IntelliSense. Note: GNU standards are only used to query the set compiler to get GNU defines, and IntelliSense will emulate the equivalent C++ standard version.

c++11

- The configuration name will be automatically selected matching the computer’s OS.
- <u>Compiler path</u>
- Windows: path/to/ur/g++ file. If you followed the C 프로그래밍 환경설정.pdf file in etl, it should be C:/migw64/bin/g++.exe
- Mac: /usr/bin/g++
- C++ standard: c++11

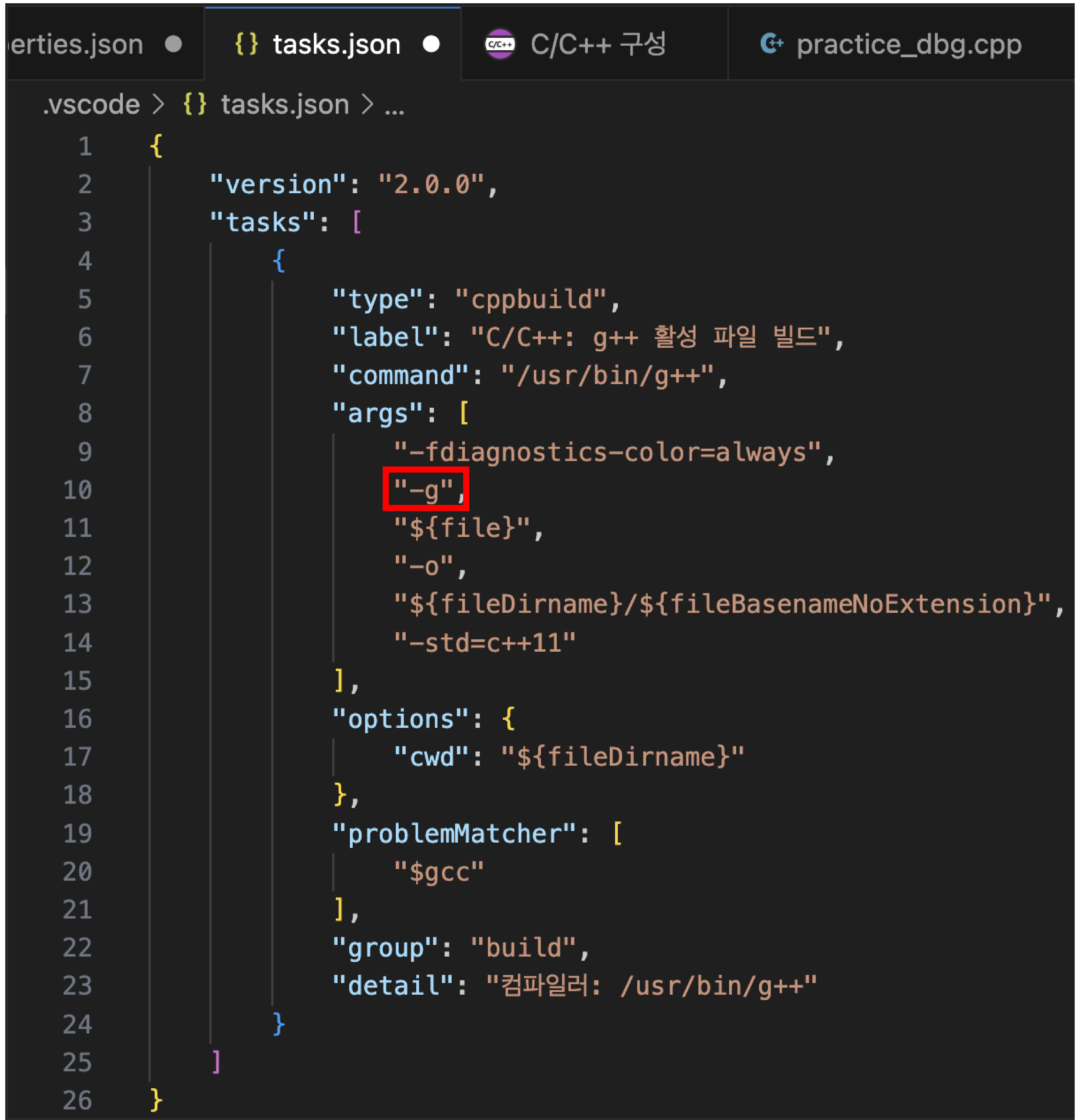
tasks.json



```
{
    "version": "2.0.0",
    "tasks": [
        {
            "type": "cppbuild",
            "label": "C/C++: g++.exe build active file",
            "command": "C:\\mingw64\\bin\\g++.exe",
            "args": [
                "-fdiagnostics-color=always",
                "-g",
                "${file}",
                "-o",
                "${fileDirname}\\${fileBasenameNoExtension}.exe",
                "-std=c++11"
            ],
            "options": {
                "cwd": "${fileDirname}"
            },
            "problemMatcher": [
                "$gcc"
            ],
            "group": "build",
            "detail": "compiler: C:\\mingw64\\bin\\g++.exe"
        }
    ]
}
```

- The left figure is an example of the json file in Windows.
- The command to execute, arguments, working directory is defined in this file.
- For example, we normally type
  g++ ./test.cpp –o ./test –std=c++11
  in our terminal
- The "command" part is g++
- "${file} " : ./test.cpp
- "${fileDirname}\\${fileBasenameNoExtension}.exe" : ./test

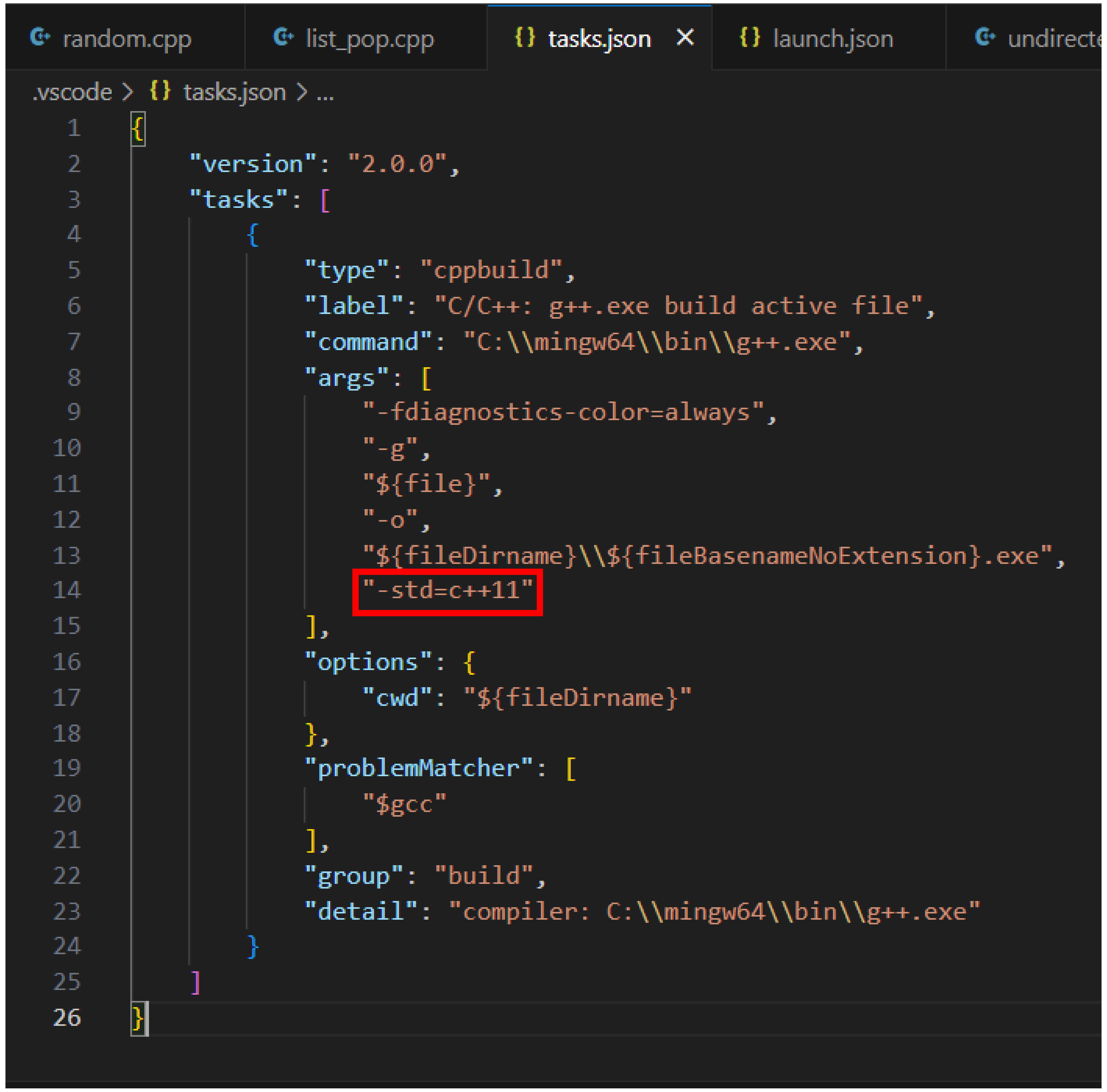
tasks.json



```
{
    "version": "2.0.0",
    "tasks": [
        {
            "type": "cppbuild",
            "label": "C/C++: g++ 활성 파일 빌드",
            "command": "/usr/bin/g++",
            "args": [
                "-fdiagnostics-color=always",
                "-g",
                "${file}",
                "-o",
                "${fileDirname}/${fileBasenameNoExtension}",
                "-std=c++11"
            ],
            "options": {
                "cwd": "${fileDirname}"
            },
            "problemMatcher": [
                "$gcc"
            ],
            "group": "build",
            "detail": "컴파일러: /usr/bin/g++"
        }
    ]
}
```

- The left figure is an example of the json file in Mac
- label: The task's label used in the user interface
- The –g flag tells the compiler to generate debugging information
- For more information about tasks.json see link

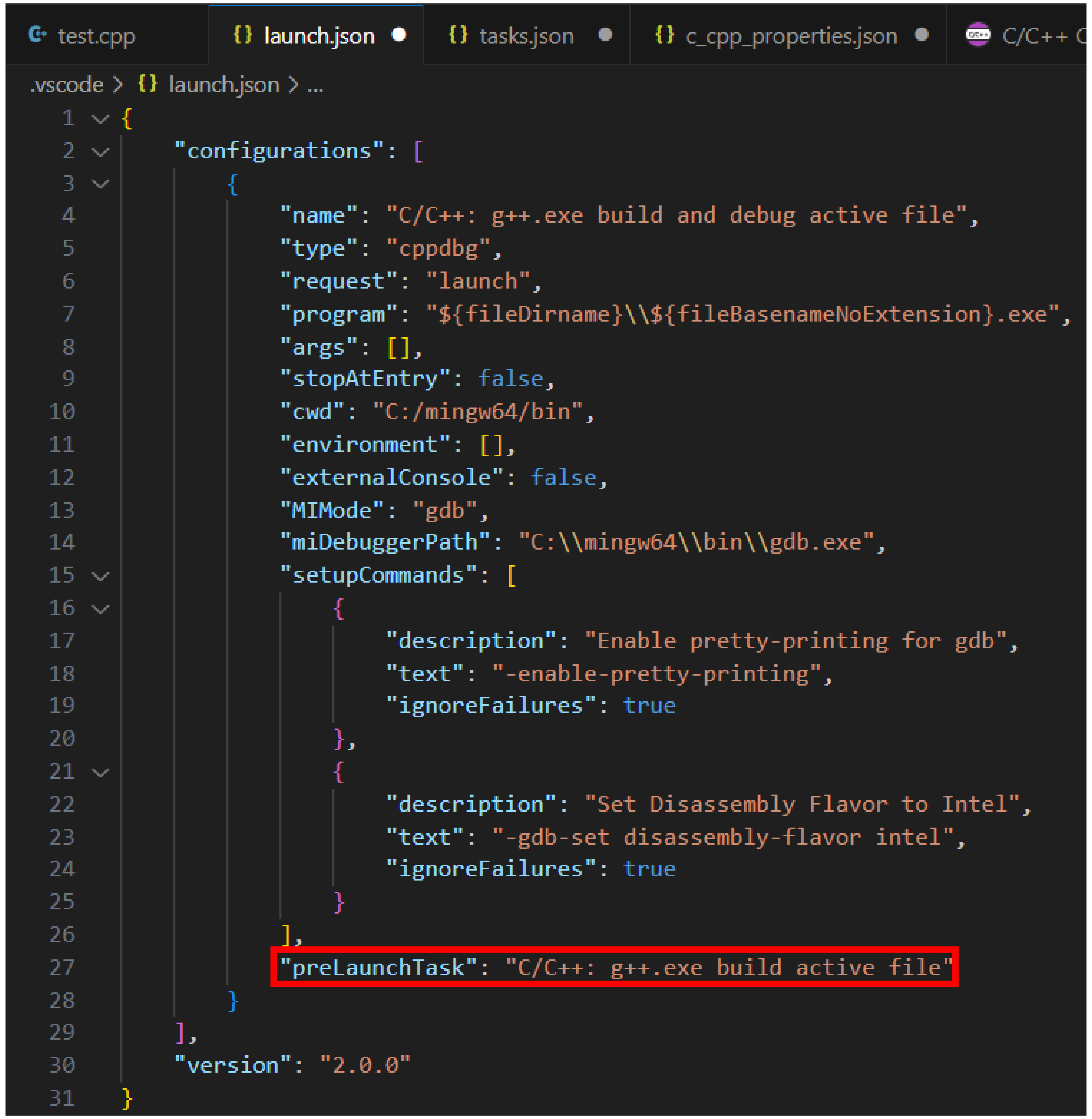
How to make tasks.json



```json
{
    "version": "2.0.0",
    "tasks": [
        {
            "type": "cppbuild",
            "label": "C/C++: g++.exe build active file",
            "command": "C:\\mingw64\\bin\\g++.exe",
            "args": [
                "-fdiagnostics-color=always",
                "-g",
                "${file}",
                "-o",
                "${fileDirname}\\${fileBasenameNoExtension}.exe",
                "-std=c++11"
            ],
            "options": {
                "cwd": "${fileDirname}"
            },
            "problemMatcher": [
                "$gcc"
            ],
            "group": "build",
            "detail": "compiler: C:\\mingw64\\bin\\g++.exe"
        }
    ]
}
```

- Open a .cpp file. If you don't have one make one.
- Press F1 / command + shift + p -> Tasks: Configure Task -> C/C++: g++
- Need to add the "-std=c++11" line in args so that we build our file in c++11 standard
- "cwd" is the path to the current working directory

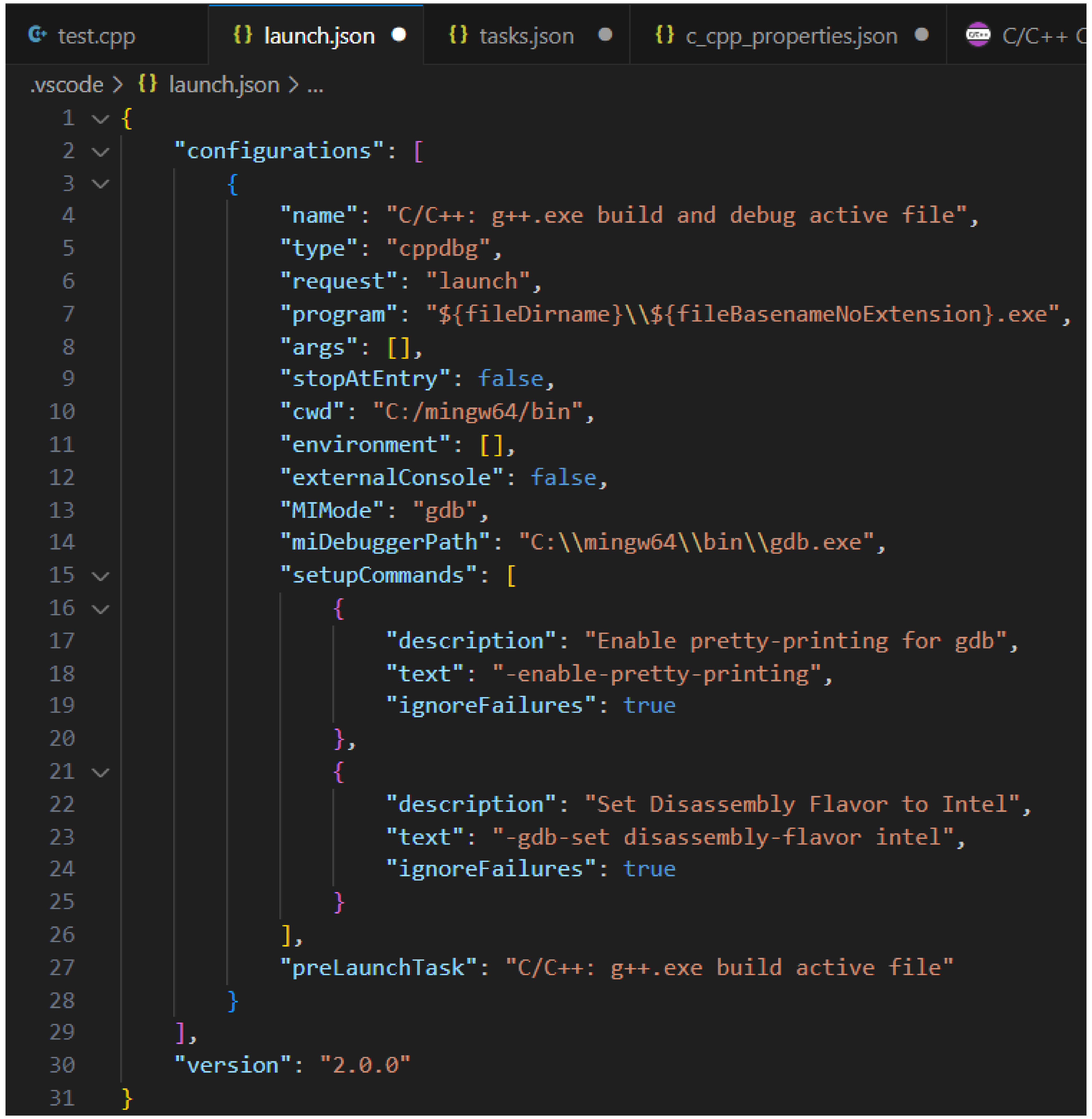
launch.json



```json
{
    "configurations": [
        {
            "name": "C/C++: g++.exe build and debug active file",
            "type": "cppdbg",
            "request": "launch",
            "program": "${fileDirname}\\${fileBasenameNoExtension}.exe",
            "args": [],
            "stopAtEntry": false,
            "cwd": "C:/mingw64/bin",
            "environment": [],
            "externalConsole": false,
            "MIMode": "gdb",
            "miDebuggerPath": "C:\\mingw64\\bin\\gdb.exe",
            "setupCommands": [
                {
                    "description": "Enable pretty-printing for gdb",
                    "text": "-enable-pretty-printing",
                    "ignoreFailures": true
                },
                {
                    "description": "Set Disassembly Flavor to Intel",
                    "text": "-gdb-set disassembly-flavor intel",
                    "ignoreFailures": true
                }
            ],
            "preLaunchTask": "C/C++: g++.exe build active file"
        }
    ],
    "version": "2.0.0"
}
```

- launch.json has the configuration about the debugger
- preLaunchTask: to launch a task before the start of a debug session, set this attribute to the label of a task specified in tasks.json
- Note that the name of preLaunchTask should match the label of tasks.json
- For more info see link

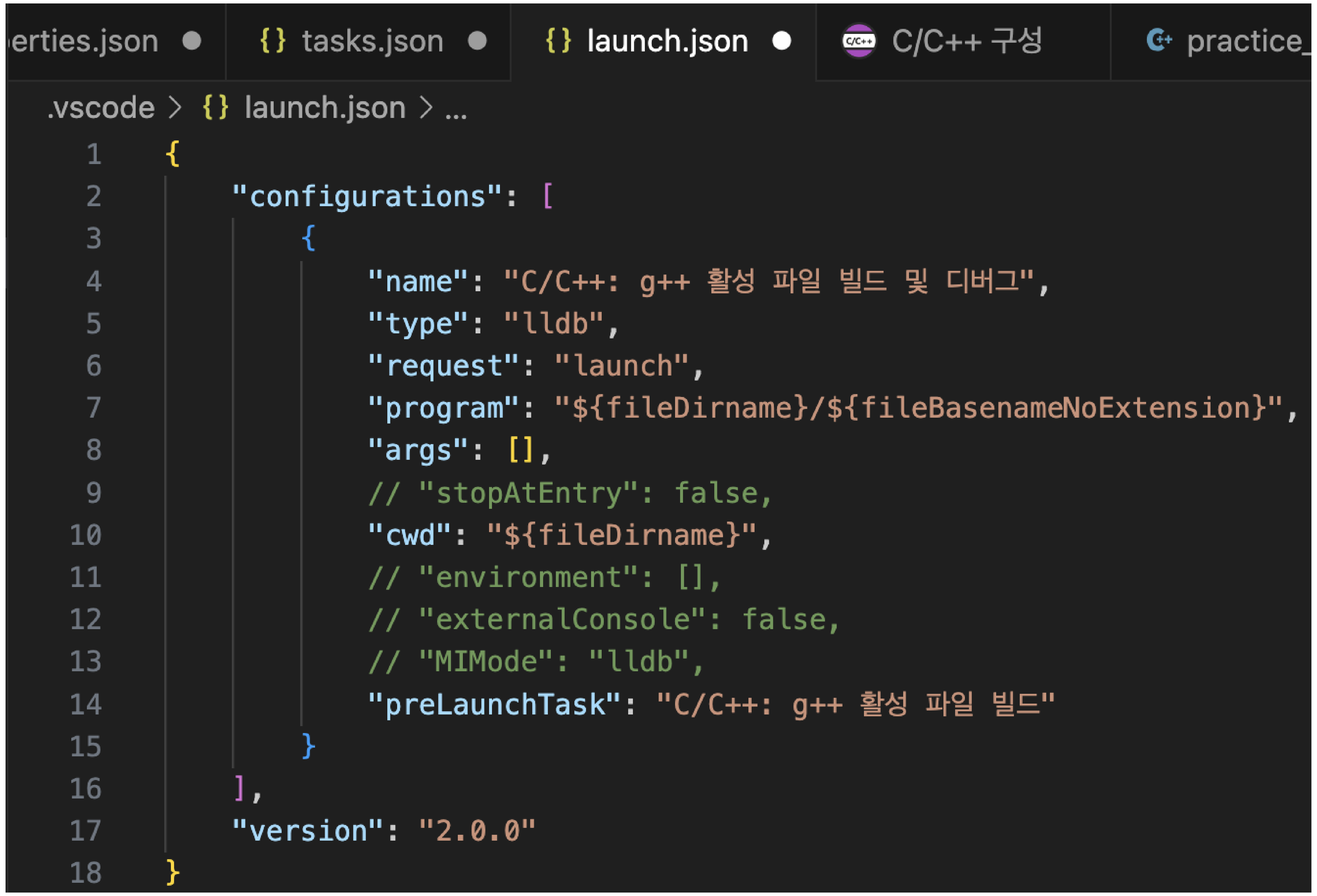
How to make launch.json



```
{
    "configurations": [
        {
            "name": "C/C++: g++.exe build and debug active file",
            "type": "cppdbg",
            "request": "launch",
            "program": "${fileDirname}\\${fileBasenameNoExtension}.exe",
            "args": [],
            "stopAtEntry": false,
            "cwd": "C:/mingw64/bin",
            "environment": [],
            "externalConsole": false,
            "MIMode": "gdb",
            "miDebuggerPath": "C:\\mingw64\\bin\\gdb.exe",
            "setupCommands": [
                {
                    "description": "Enable pretty-printing for gdb",
                    "text": "-enable-pretty-printing",
                    "ignoreFailures": true
                },
                {
                    "description": "Set Disassembly Flavor to Intel",
                    "text": "-gdb-set disassembly-flavor intel",
                    "ignoreFailures": true
                }
            ],
            "preLaunchTask": "C/C++: g++.exe build active file"
        }
    ],
    "version": "2.0.0"
}
```

- Open a .cpp file.
- Press F1 / command + shift + p -> C/C++: Add Debug Configuration-> C/C++: g++

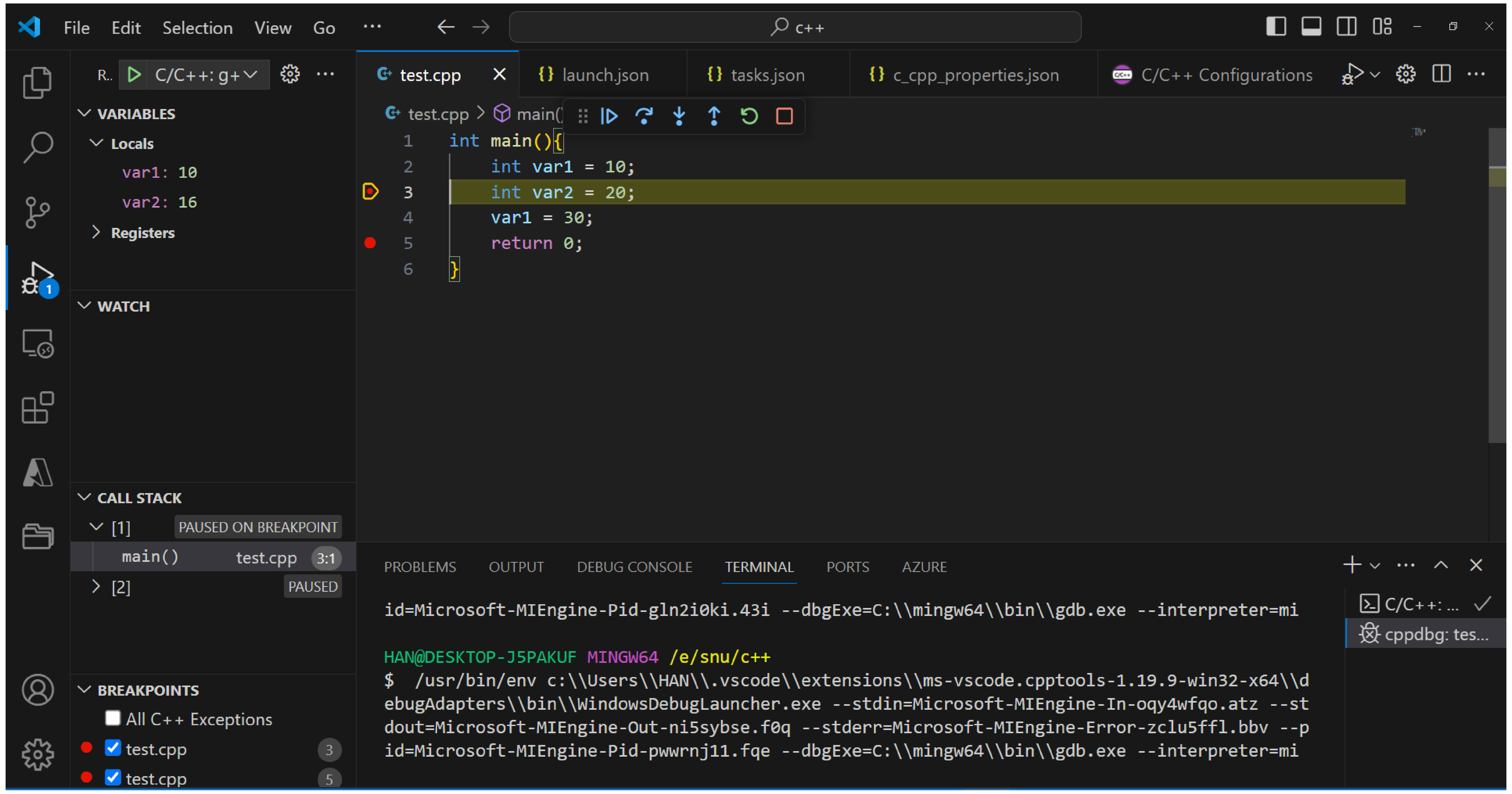
If you use M1/M2



```
{
    "configurations": [
        {
            "name": "C/C++: g++ 활성 파일 빌드 및 디버그",
            "type": "lldb",
            "request": "launch",
            "program": "${fileDirname}/${fileBasenameNoExtension}",
            "args": [],
            // "stopAtEntry": false,
            "cwd": "${fileDirname}",
            // "environment": [],
            // "externalConsole": false,
            // "MIMode": "lldb",
            "preLaunchTask": "C/C++: g++ 활성 파일 빌드"
        }
    ],
    "version": "2.0.0"
}
```

- Change type from cppdbg to lldb
- Reason: we are using the LLDB
- Comment the parts that have yellow squiggles(these configurations are not needed in LLDB)

Exercise 1



```
var1: 10
var2: 16
```

```
[1]  main()  test.cpp 3:1
[2]
```

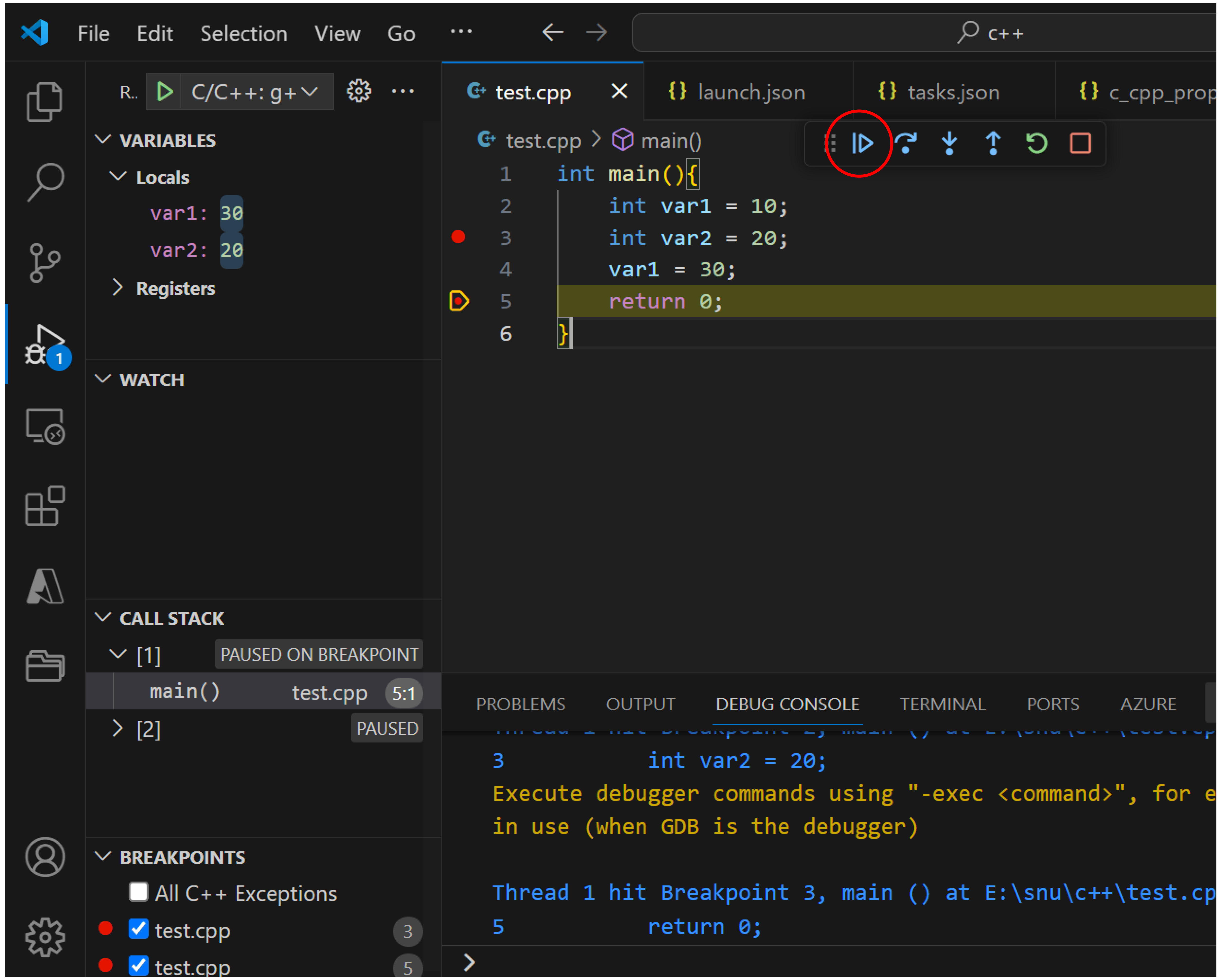
```
test.cpp 3
test.cpp 5
```

```
1  int main(){
2      int var1 = 10;
3      int var2 = 20;
4      var1 = 30;
5      return 0;
6  }
```

```
id=Microsoft-MIEngine-Pid-gln2i0ki.43i --dbgExe=C:\\mingw64\\bin\\gdb.exe --interpreter=mi

HAN@DESKTOP-J5PAKUF MINGW64 /e/snu/c++
$  /usr/bin/env c:\\Users\\HAN\\.vscode\\extensions\\ms-vscode.cpptools-1.19.9-win32-x64\\d
ebugAdapters\\bin\\WindowsDebugLauncher.exe --stdin=Microsoft-MIEngine-In-oqy4wfqo.atz --st
dout=Microsoft-MIEngine-Out-ni5sybse.f0q --stderr=Microsoft-MIEngine-Error-zclu5ffl.bbv --p
id=Microsoft-MIEngine-Pid-pwwrnj11.fqe --dbgExe=C:\\mingw64\\bin\\gdb.exe --interpreter=mi
```

- Now let's use debugger in a .cpp file
- Press a red dot on the left side of the code line(3,5 for this example). It is a breakpoint where the debugger stops.
- Then press F5 or debug button(bug icon)
- As you can see on the figure, we can see the value of local variables. Note that in code 3, var2 is not yet initialized but has a value of 16. This is because currently there is garbage value in that memory.

Exercise 1



```
var1: 30
var2: 20
```

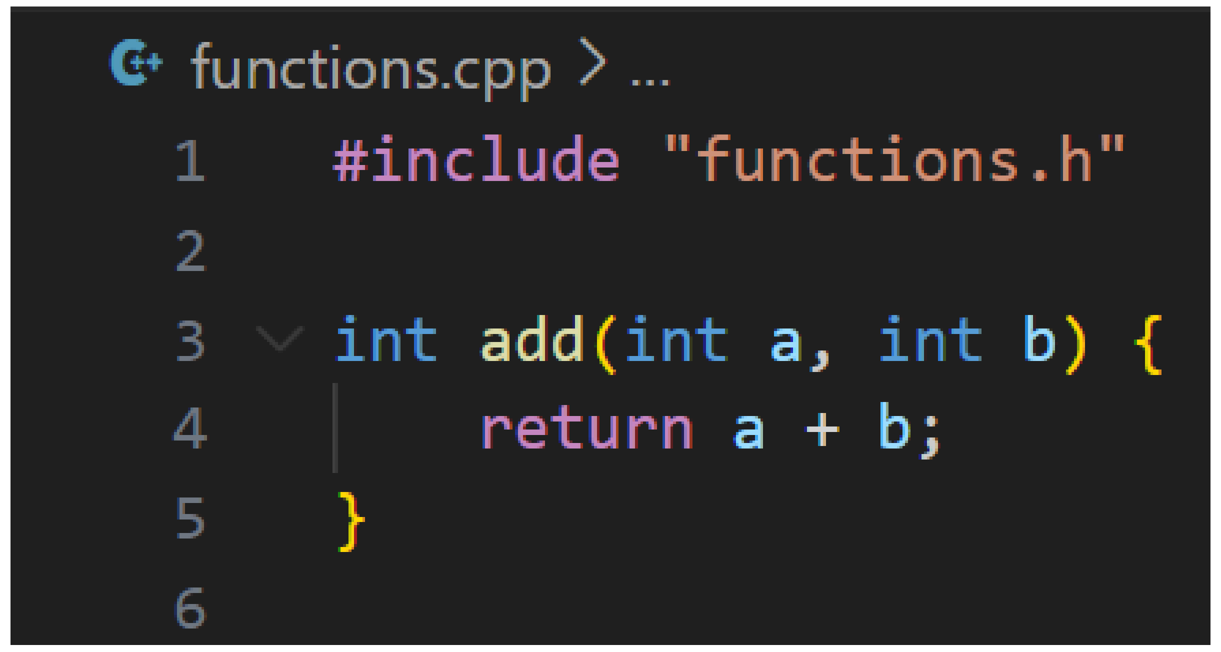
```
int main(){
    int var1 = 10;
    int var2 = 20;
    var1 = 30;
    return 0;
}
```

```
3           int var2 = 20;
Execute debugger commands using "-exec <command>", for e
in use (when GDB is the debugger)

Thread 1 hit Breakpoint 3, main () at E:\snu\c++\test.cp
5           return 0;
```

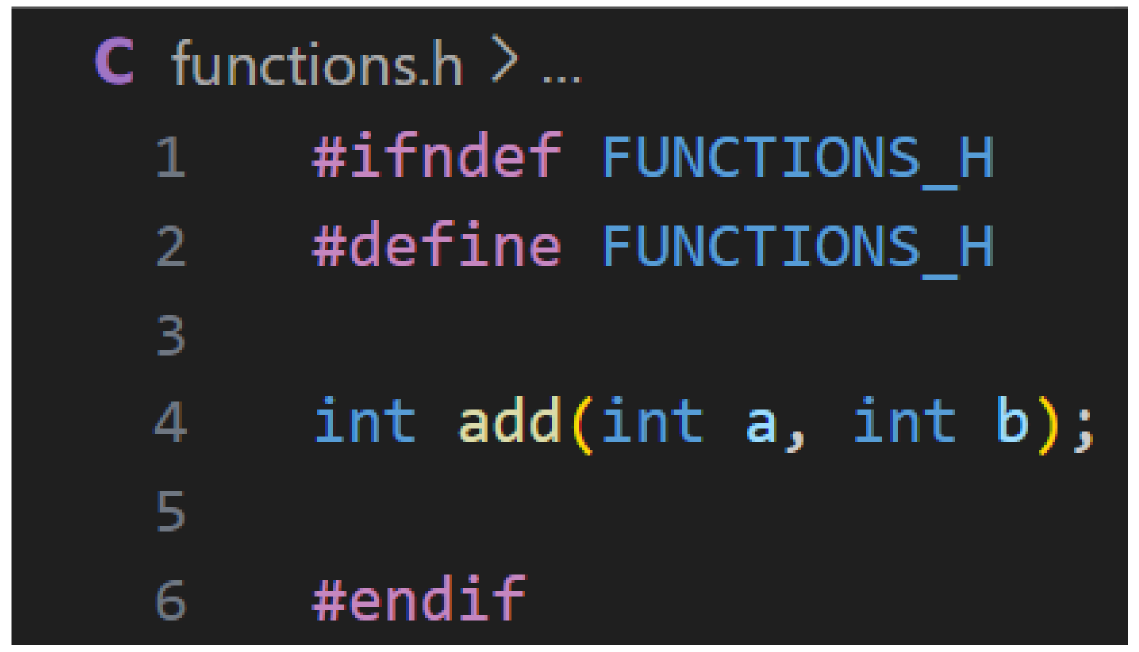
- Now let’s move to the next breakpoint
- Press the continue button or F5. Debugger will stop at the next breakpoint
- As you can see now var1 is updated to 30.
- Always note that if the debugger is stopped at line 5, the code of line 5 is not executed yet.

Exercise 2: Multiple .cpp files

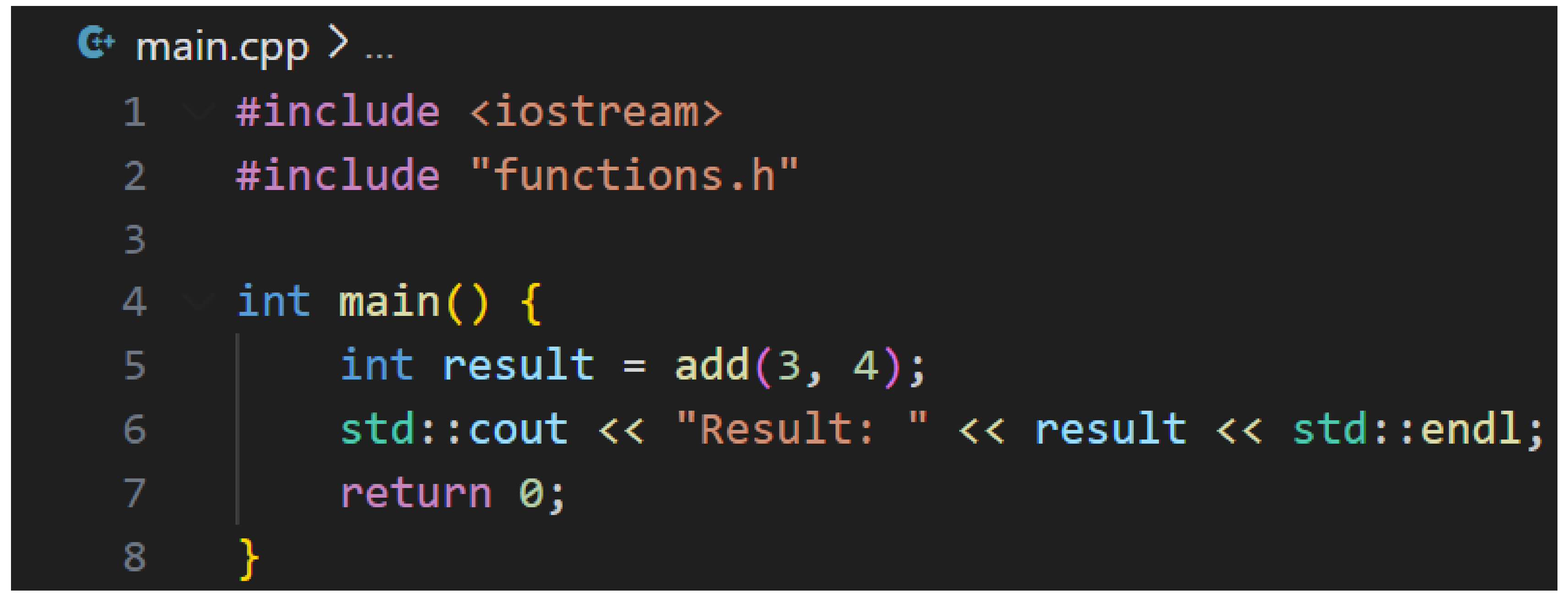
- Let’s say we have functions.cpp , functions.h , main.cpp as follows:



```cpp
#include "functions.h"

int add(int a, int b) {
    return a + b;
}

```



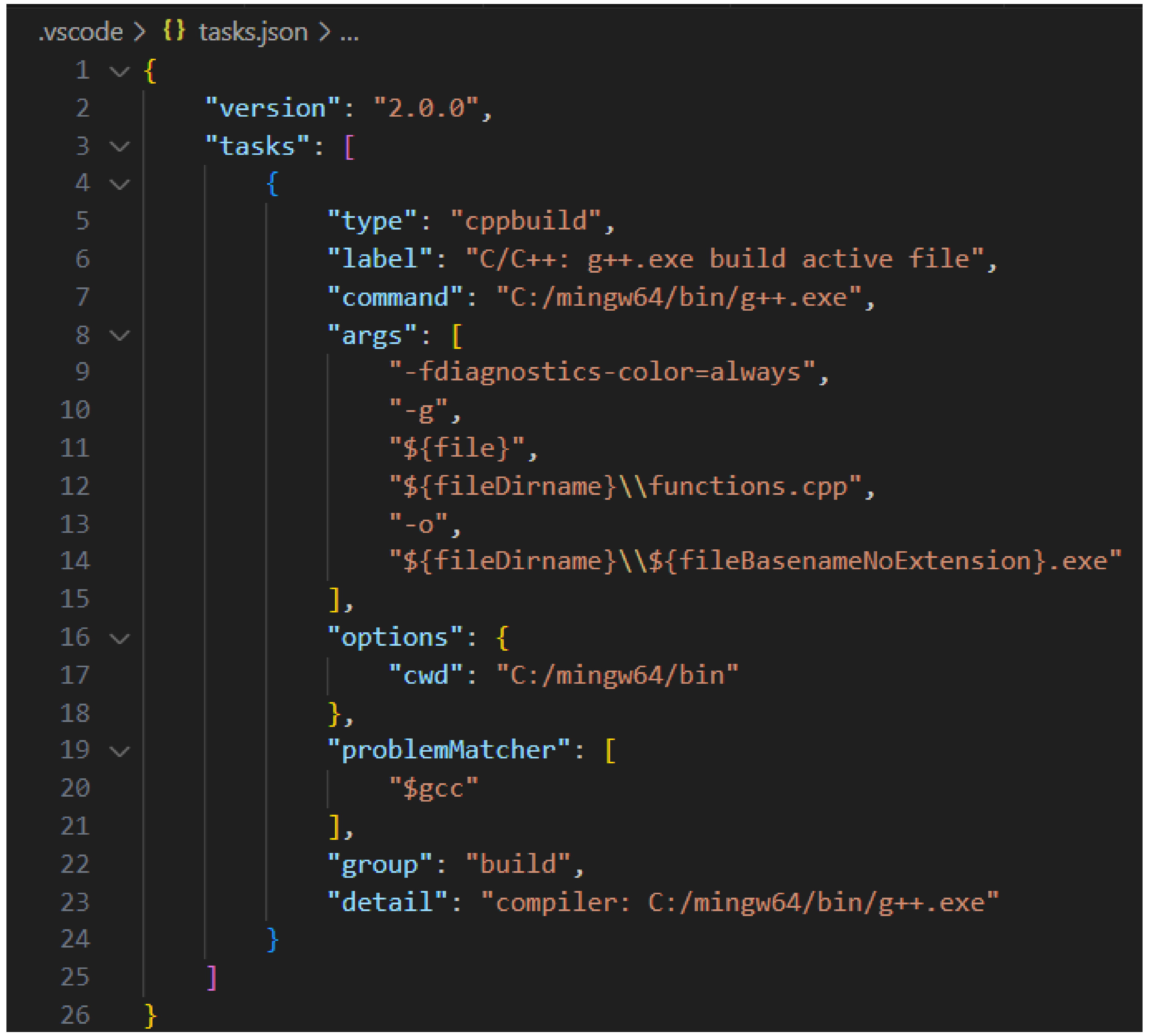
```cpp
#ifndef FUNCTIONS_H
#define FUNCTIONS_H

int add(int a, int b);

#endif
```



```cpp
1   #include <iostream>
2   #include "functions.h"
3
4   int main() {
5       int result = add(3, 4);
6       std::cout << "Result: " << result << std::endl;
7       return 0;
8   }
```

- I want to make a breakpoint on the 6th line of main.cpp to see the value of result. How can I do it?
  - Hint: edit tasks.json

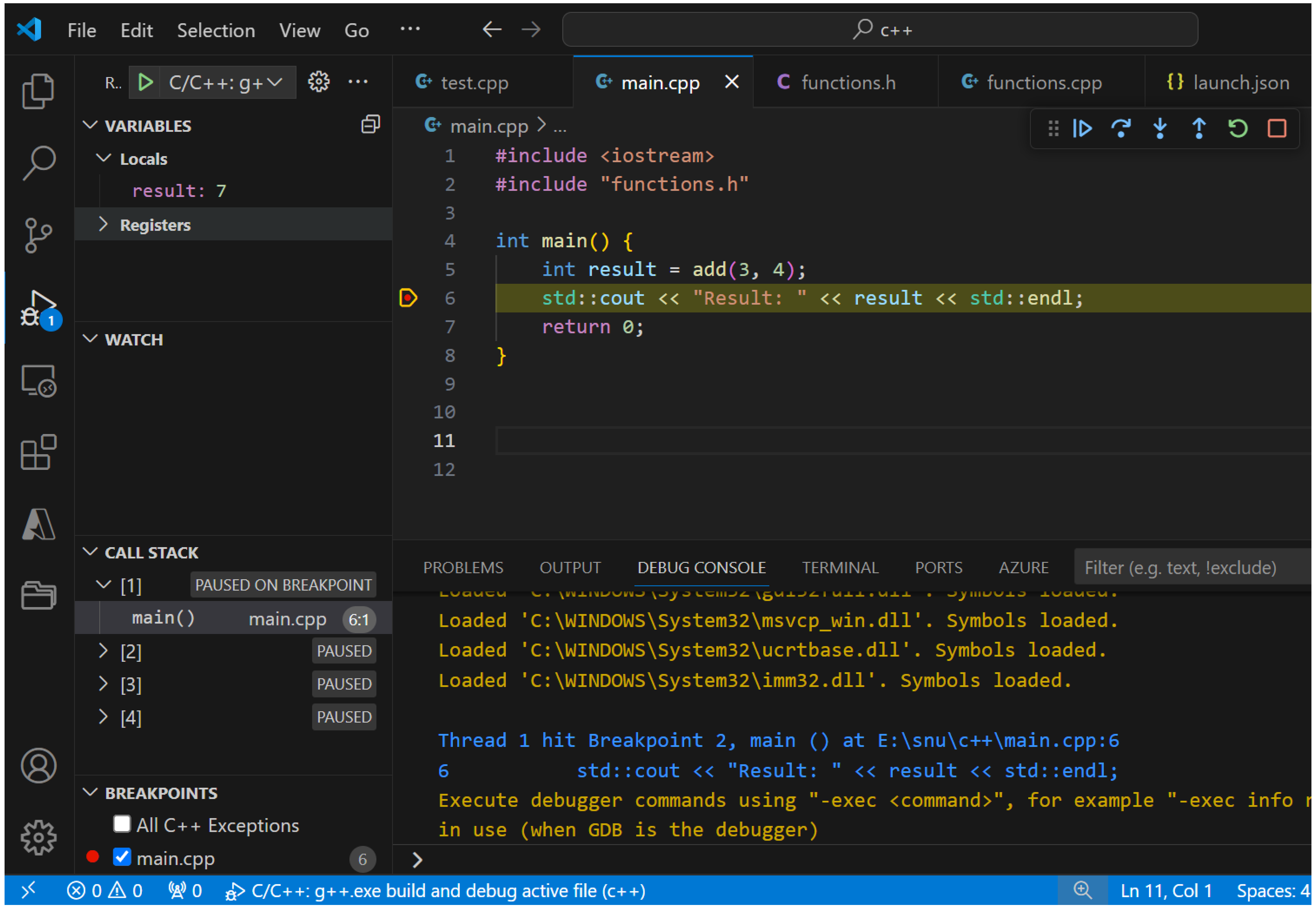
Exercise 2: Multiple .cpp files



```
{
    "version": "2.0.0",
    "tasks": [
        {
            "type": "cppbuild",
            "label": "C/C++: g++.exe build active file",
            "command": "C:/mingw64/bin/g++.exe",
            "args": [
                "-fdiagnostics-color=always",
                "-g",
                "${file}",
                "${fileDirname}\\functions.cpp",
                "-o",
                "${fileDirname}\\${fileBasenameNoExtension}.exe"
            ],
            "options": {
                "cwd": "C:/mingw64/bin"
            },
            "problemMatcher": [
                "$gcc"
            ],
            "group": "build",
            "detail": "compiler: C:/mingw64/bin/g++.exe"
        }
    ]
}
```

- This is an example of editing tasks.json in windows so that main.cpp and functions.cpp are compiled simultaneously
- Assumed that I press the debug button while viewing main.cpp
- Why does this matter?
  - Hint: "${file}"
- There are also other ways to edit the tasks.json file

Exercise 2: Multiple .cpp files



```
result: 7
```

```
[1]  PAUSED ON BREAKPOINT
main()   main.cpp  6:1
[2]  PAUSED
[3]  PAUSED
[4]  PAUSED
```

```
main.cpp  6
```

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

int main() {
    int result = add(3, 4);
    std::cout << "Result: " << result << std::endl;
    return 0;
}
```

```
Loaded 'C:\WINDOWS\System32\msvcp_win.dll'. Symbols loaded.
Loaded 'C:\WINDOWS\System32\ucrtbase.dll'. Symbols loaded.
Loaded 'C:\WINDOWS\System32\imm32.dll'. Symbols loaded.

Thread 1 hit Breakpoint 2, main () at E:\snu\c++\main.cpp:6
6           std::cout << "Result: " << result << std::endl;
Execute debugger commands using "-exec <command>", for example "-exec info r
in use (when GDB is the debugger)
```

```
C/C++: g++.exe build and debug active file (c++)
```

- Type result in debug console
- Check the output!

If you have any problems

1. Delete cpp_properties.json, tasks.json, launch.json and do it over again
2. Search the error you are getting in google
3. Post on the Q&A 게시판(please attach detailed info about the error, computer OS)

Note that most of the errors are due to configuration!

Think about the compilation process in C++!

Thank you :)

- Now you know how to use a debugger in VSCode ☺