



창의적소프트웨어프로그래밍

Lab 1-1

TA. 정지나

snowgina00@hanyang.ac.kr

Instructor

TA. 정지나

이메일 snowgina00@hanyang.ac.kr

주소 공업센터 별관 503-2 (Software Engineering Lab.)

Today's Contents

1. Environment Setup

- **VS Code Installation**
- **Compiler Setup**
- **VS Code Setup**

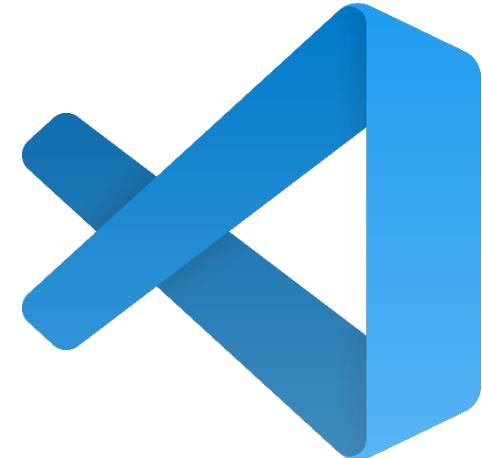
How to install VS Code

Windows & macOS



Practice environment

- C++
- Visual Studio Code (VS Code)
 - Code editor
 - a text editor program that is specifically optimized for writing and executing program codes
 - Supports Windows, macOS and Linux, ...
 - it can be used on almost any PC



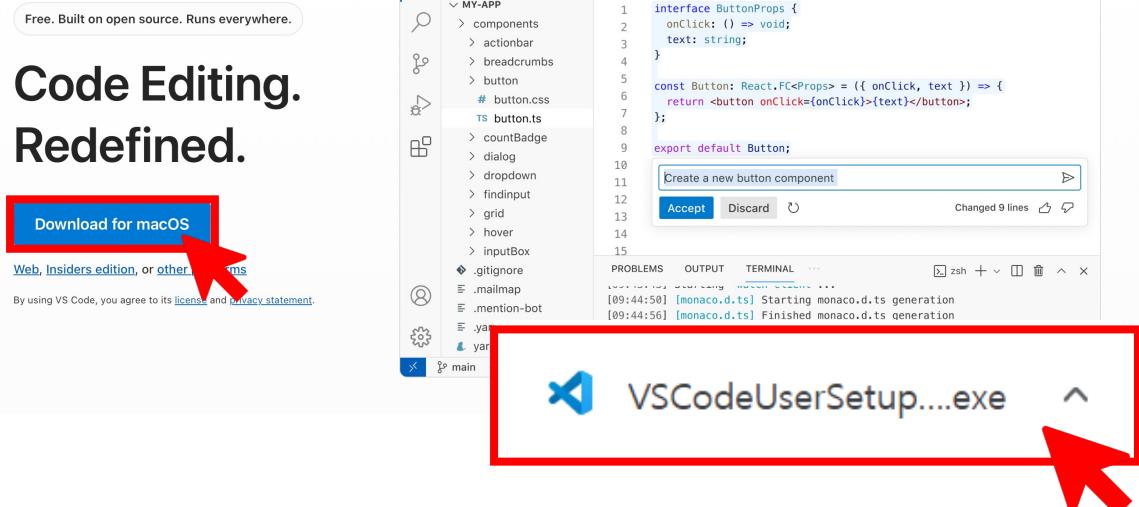
VS Code Installation (Windows)

1. Visit VSCode official website

- Link : <https://code.visualstudio.com/>

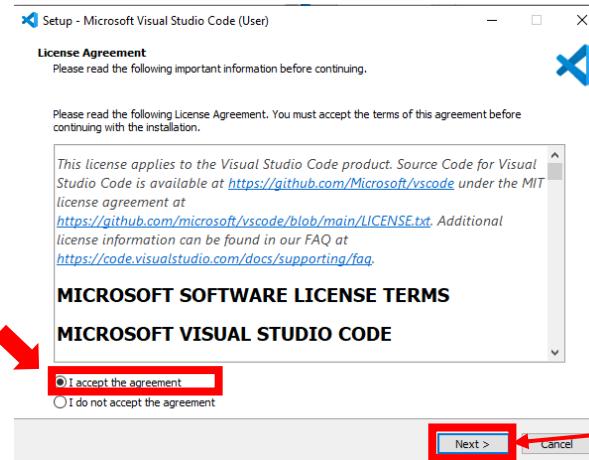


2. Download and open .exe file



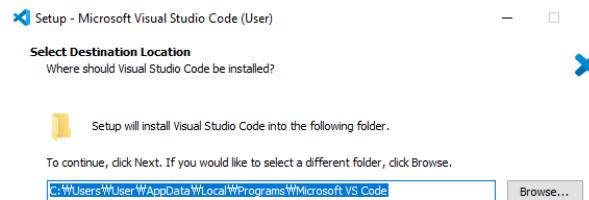
VS Code Installation (Windows)

3. Check “I accept the agreement”

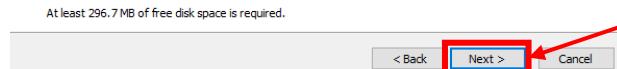


Click "Next" button

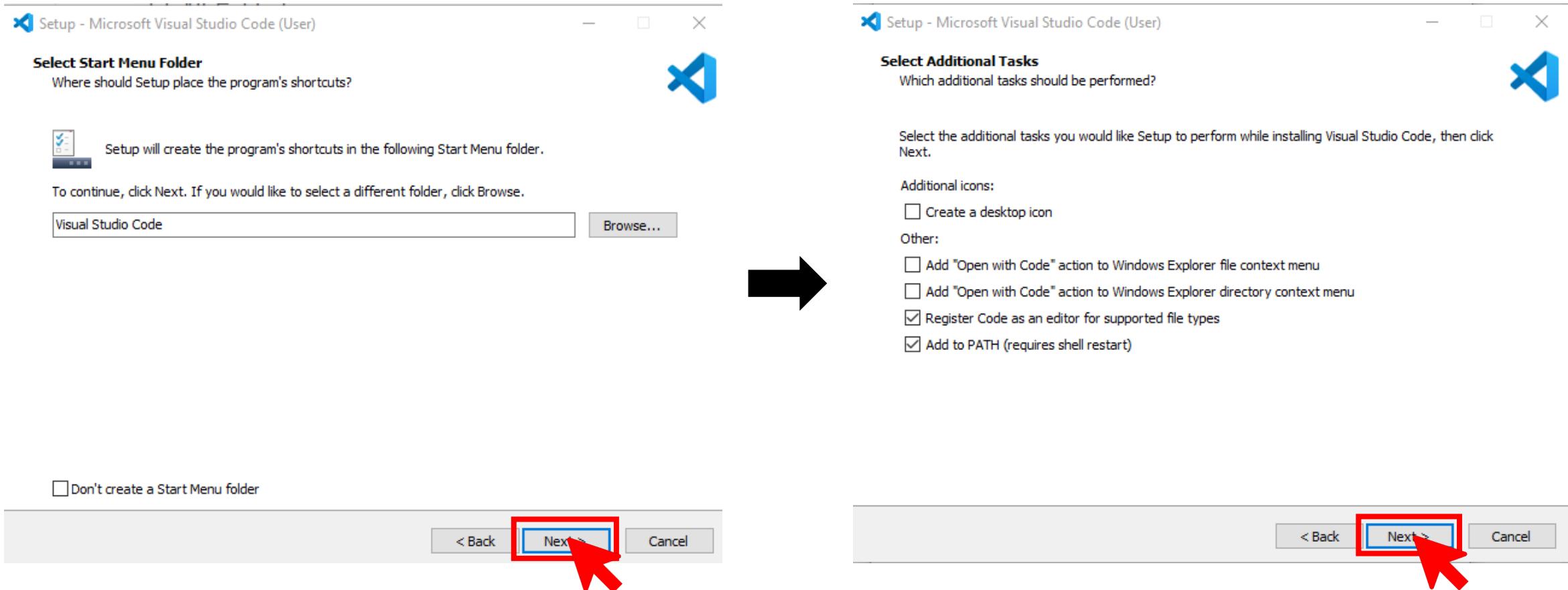
4. Check installation path



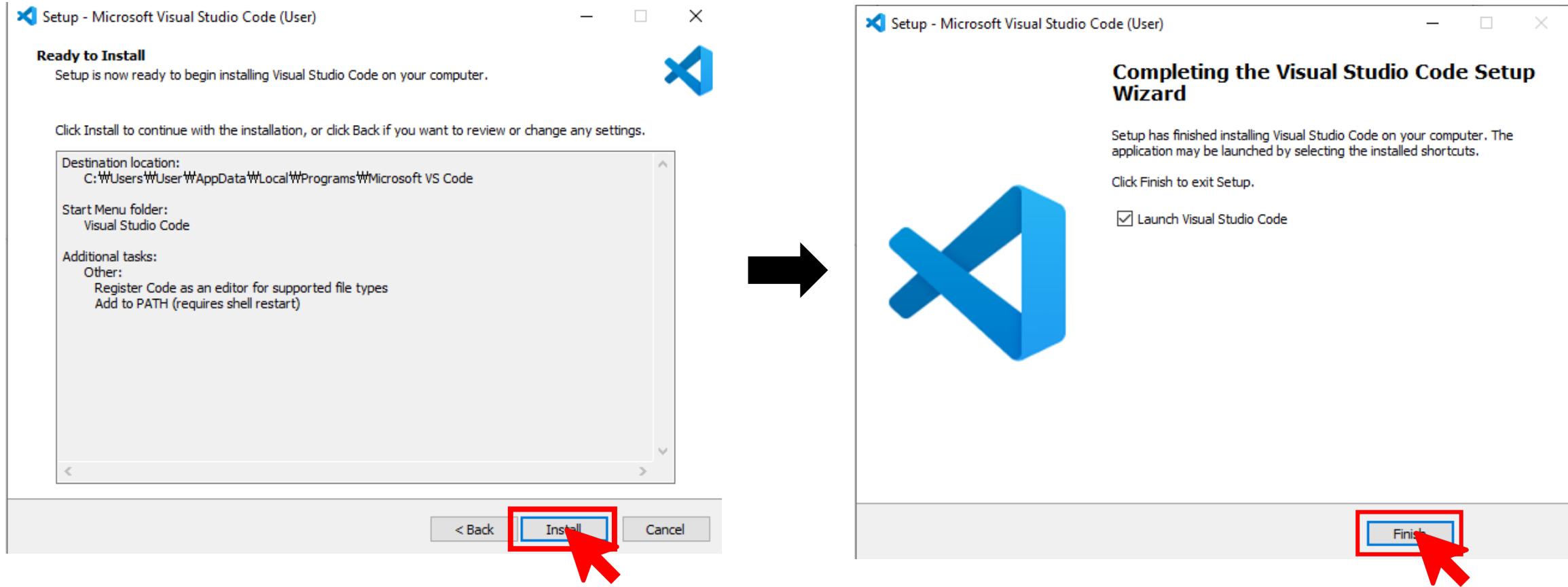
Click "Next" button



VS Code Installation (Windows)



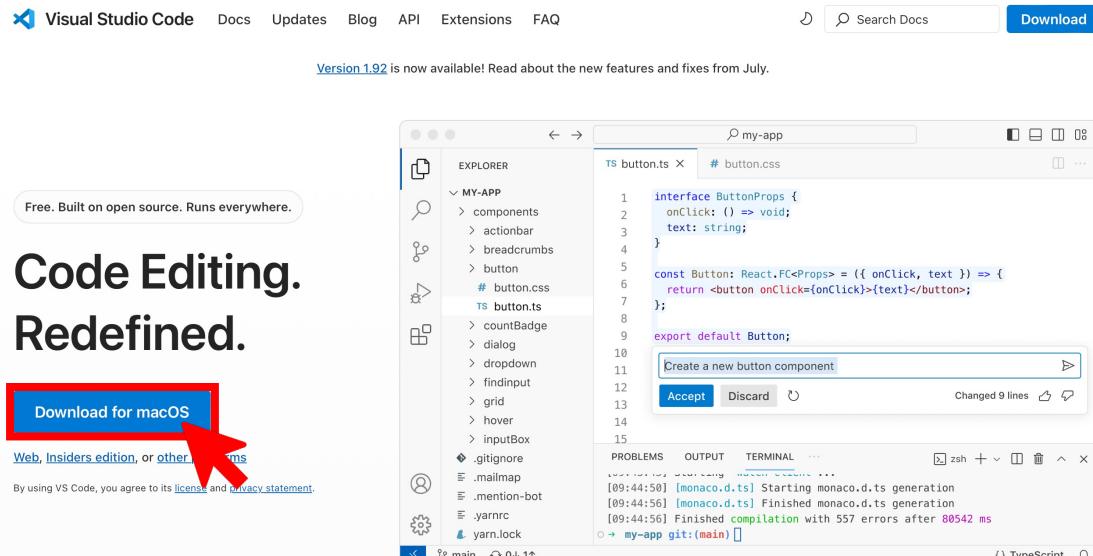
VS Code Installation (Windows)



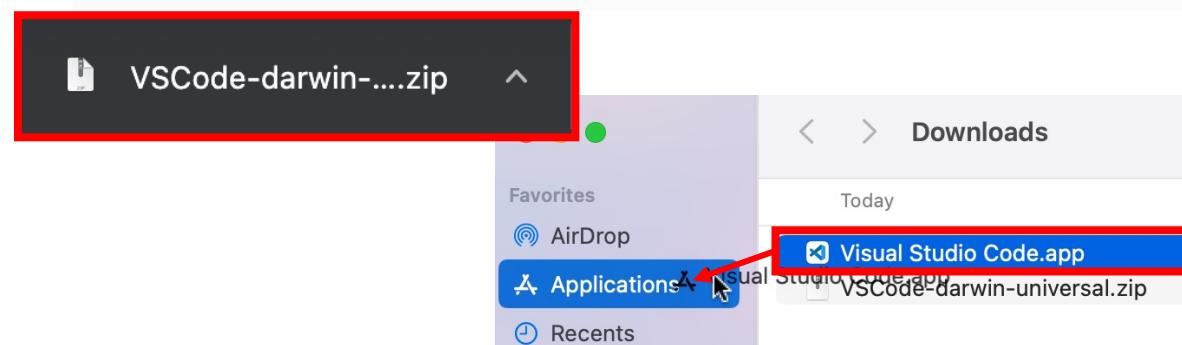
VS Code Installation (macOS)

1. Visit VSCode official website

- Link : <https://code.visualstudio.com/>



2. Unzip and move the VS Code to Applications folder



How to install C/C++ Compiler

Windows

What is MSYS2?

MSYS2

- Software distribution and a building platform for Windows.
- Provides a Unix-like environment, a command-line interface and a software
- Use a port of *pacman* for package management.



MSYS2 Installation

1. Visit MSYS2 official website

- Link : <https://www.msys2.org/>

The screenshot shows the MSYS2 official website at <https://www.msys2.org/>. The page has a dark header with the MSYS2 logo, search bar, and GitHub link. The main content area features the title "MSYS2" and a brief introduction. Below the introduction is a section titled "Installation" with steps 1-3. A red box highlights the download link for the installer. To the right, a red box highlights the "Recent download history" section, which lists "msys2-x86_64-20240113.exe" with a download progress bar and a hand cursor icon.

MSYS2
Software Distribution and Building Platform for Windows

MSYS2 is a collection of tools and libraries providing you with an easy-to-use environment for building, installing and running native Windows software.

It consists of a command line terminal called [mintty](#), bash, version control systems like git and subversion, tools like tar and awk and even build systems like autotools, all based on a modified version of [Cygwin](#). Despite some of these central parts being based on Cygwin, the main focus of MSYS2 is to provide a build environment for native Windows software and the Cygwin-using parts are kept at a minimum. MSYS2 provides up-to-date native builds for GCC, mingw-w64, CPython, CMake, Meson, OpenSSL, FFmpeg, Rust, Ruby, just to name a few.

To provide easy installation of packages and a way to keep them updated it features a package management system called [Pacman](#), which should be familiar to Arch Linux users. It brings many powerful features such as dependency resolution and simple complete system upgrades, as well as straight-forward and reproducible package building. Our package repository contains [more than 3200 pre-built packages](#) ready to install.

For more details see ['What is MSYS2'](#) which also compares MSYS2 to other software distributions and development environments like [Cygwin](#), [WSL](#), [Chocolatey](#), [Scoop](#), ... and ['Who Is Using MSYS2'](#) to see which projects are using MSYS2 and what for.

Installation

- Download the installer: [msys2-x86_64-20240727.exe](#)
- (Optional) For more information on the installer, command line options, or how to verify the checksum and signature of the installer, see the [installer guide](#).
- Run the installer. Installing MSYS2 requires
- Enter your desired [Installation Folder](#) (short symlinks, no subst or network drives, no FAT)

Recent download history

msys2-x86_64-20240113.exe
75.3 MB • Done

2. Download and open installation file

file

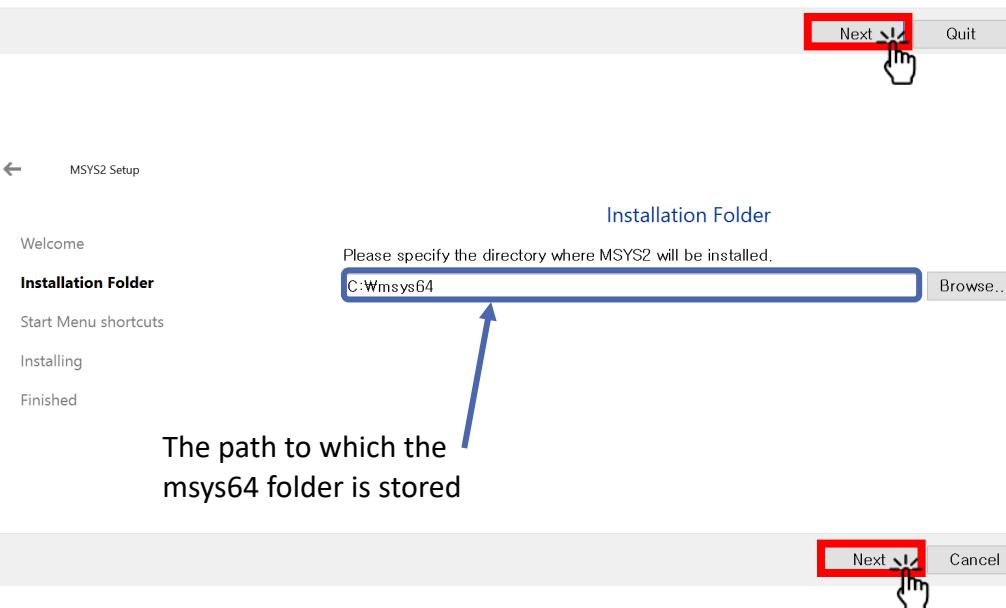
MSYS2 Installation

**3. Click “Next” button until
“Installing” step**



4. Click “Finish” button

- Run MSYS2 after the installation is completed.



MSYS2 Setup

1. Open MSYS2 Terminal

- Terminal : An interface on a computer system that allows users to enter commands and control the operation of the system



2. Update the package database and base packages

- Write a command “**pacman -Syu**” in the terminal window.
- pacman**(PACkage MANager)
 - Download, install, update, and uninstall **compiled binary** software packages on Arch Linux system.

- Syu** : pacman option
 - S : stand for sync
 - y : refresh(local cache)
 - u : system update



MSYS2 Setup

3. Write 'Y' and then press Enter

- If you see the message “To complete this update ... Confirm to proceed[Y/N]”, write ‘Y’.

```
user@DESKTOP-NFTOJAP UCRT64 ~
$ pacman -Syu
:: Synchronizing package databases...
clangarm64          450.3 KiB  260 KiB/s 00:02 [########################################] 100%
mingw32              244.4 KiB  151 KiB/s 00:02 [########################################] 100%
mingw64              486.5 KiB  363 KiB/s 00:01 [########################################] 100%
ucrt64               504.7 KiB  356 KiB/s 00:01 [########################################] 100%
clang32              234.1 KiB  127 KiB/s 00:02 [########################################] 100%
clang64               493.5 KiB  877 KiB/s 00:01 [########################################] 100%
msys                 488.0 KiB  429 KiB/s 00:01 [########################################] 100%
:: Starting core system upgrade...
warning: terminate other MSYS2 programs before proceeding
resolving dependencies...
looking for conflicting packages...

Packages (5) bash-5.2.026-1 mintty-1~3.7.1-1 msys2-runtime-3.4.10-6 pacman-6.0.2-13
          pacman-mirrors-20240210-1

Total Download Size:  11.40 MiB
Total Installed Size: 60.04 MiB
Net Upgrade Size:    0.06 MiB

:: Proceed with installation? [Y/n] |
```

```
:: Proceed with installation? [Y/n] Y
:: Retrieving packages...
mintty-1~3.7.1-1-x86_64          831.7 KiB  1375 KiB/s 00:01 [########################################] 100%
msys2-runtime-3.4.10-6-x86_64   1836.8 KiB  2.74 MiB/s 00:01 [########################################] 100%
pacman-mirrors-20240210-1-any   3.5 KiB   5.30 KiB/s 00:01 [########################################] 100%
pacman-6.0.2-13-x86_64         6.4 MiB   2.80 MiB/s 00:02 [########################################] 100%
bash-5.2.026-1-x86_64          2.4 MiB   712 KiB/s 00:03 [########################################] 100%
Total (5/5)                      11.4 MiB  3.25 MiB/s 00:04 [########################################] 100%
(5/5) checking keys in keyring
(5/5) checking package integrity
(5/5) loading package files
(5/5) checking for file conflicts
(5/5) checking available disk space
:: Processing package changes...
(1/5) upgrading bash
(2/5) upgrading mintty
(3/5) upgrading msys2-runtime
(4/5) upgrading pacman-mirrors
(5/5) upgrading pacman
:: To complete this update all MSYS2 processes including this terminal will be closed. Confirm to proceed [Y/n]
```

MSYS2 Setup

4. Run 'MSYS2 MSYS' again and then write 'pacman -Syu' again.

- Update the rest of the base packages



A screenshot of a terminal window titled 'M2 ~'. The window has standard window controls (minimize, maximize, close) at the top right. The terminal itself is black with white text. In the top left corner of the terminal area, there is a small purple icon with a white 'M' and a tilde symbol (~). The main text area shows a user's command prompt: 'user@DESKTOP-NFT0JAP MSYS ~' followed by the command '\$ pacman -Syu'. The entire command line is highlighted with a red rectangular box. The rest of the terminal window is blank black space.

MSYS2 Setup

5. Write 'Y' and then press Enter

```
user@DESKTOP-NFTOJAP MSYS ~
$ pacman -Syu
:: Synchronizing package databases...
clangarm64 is up to date
mingw32 is up to date
mingw64 is up to date
ucrt64 is up to date
clang32 is up to date
clang64 is up to date
msys is up to date
:: Starting core system upgrade...
there is nothing to do
:: Starting full system upgrade...
resolving dependencies...
looking for conflicting packages...

Packages (27) ca-certificates-20240203-1 curl-8.7.1-1 gnupg-2.4.5-1 libcurl-8.7.1-1
libexpat-2.6.2-1 libffi-3.4.6-1 libgnutls-3.8.3-1 libgpg-error-1.48-1
libidn2-2.3.7-1 libksba-1.6.6-1 liblzma-5.6.1-2 libnghttp2-1.60.0-1 libnpth-1.7-1
libopenssl-3.2.1-1 libpcre2_8-10.43-1 libpsl-0.21.5-2 libreadline-8.2.010-1
libsqlite-3.45.1-1 libxml2-2.12.6-1 libzstd-1.5.6-1 openssl-3.2.1-1
pacman-contrib-1.10.5-1 tzcode-2024a-1 wget-1.24.5-1 xz-5.6.1-2 zlib-1.3.1-1
zstd-1.5.6-1

Total Download Size: 16.69 MiB
Total Installed Size: 49.63 MiB
Net Upgrade Size: 0.44 MiB

:: Proceed with installation? [Y/n] |
```

MSYS2 Setup

6. Write the command “pacman -Su”

- If you see the message below, the basic installation and update have been completed.

```
user@DESKTOP-NFTOJAP MSYS ~
$ pacman -Su
:: Starting core system upgrade...
there is nothing to do
:: Starting full system upgrade...
there is nothing to do
```

MSYS2 Setup

7. Write the command to install clang package

- **pacman -S --needed base-devel mingw-w64-clang-x86_64-toolchain**

```
user@DESKTOP-NFTOJAP MSYS ~
$ pacman -S --needed base-devel mingw-w64-clang-x86_64-toolchain
:: There are 22 members in group mingw-w64-clang-x86_64-toolchain:
:: Repository clang64
 1) mingw-w64-clang-x86_64-clang  2) mingw-w64-clang-x86_64-clang-analyzer
 3) mingw-w64-clang-x86_64-clang-libs  4) mingw-w64-clang-x86_64-clang-tools-extra
 5) mingw-w64-clang-x86_64-compiler-rt  6) mingw-w64-clang-x86_64-crt-git
 7) mingw-w64-clang-x86_64-gcc-compat  8) mingw-w64-clang-x86_64-headers-git
 9) mingw-w64-clang-x86_64-libc++  10) mingw-w64-clang-x86_64-libmangle-git
 11) mingw-w64-clang-x86_64-libunwind  12) mingw-w64-clang-x86_64-libwinpthread-git
 13) mingw-w64-clang-x86_64-lld  14) mingw-w64-clang-x86_64-lldb  15) mingw-w64-clang-x86_64-llvm
 16) mingw-w64-clang-x86_64-llvmlibs  17) mingw-w64-clang-x86_64-make
 18) mingw-w64-clang-x86_64-openmp  19) mingw-w64-clang-x86_64-pkgconf
 20) mingw-w64-clang-x86_64-tools-git  21) mingw-w64-clang-x86_64-winthreads-git
 22) mingw-w64-clang-x86_64-winstorecompat-git

Enter a selection (default=all): |
```

- pacman optional : -S -needed
 - -S : install package
 - --needed : only install it if is not installed.
- base-devel
 - MSYS2 package group name
- toolchain
 - Basically clang, llvm, etc... are built in.

MSYS2 Setup

8. Press *Enter* for ① and write “Y” for ②.

- Download mingw-w64-x86-clang_64-toolchain package.

```
14) mingw-w64-clang-x86_64-gdb 15) mingw-w64-clang
16) mingw-w64-clang-x86_64-make 17) mingw-w64-clang
18) mingw-w64-clang-x86_64-pkgconf 19) mingw-w64-cl
20) mingw-w64-clang-x86_64-tools-git
21) mingw-w64-clang-x86_64-winpthreads-git
22) mingw-w64-clang-x86_64-winstorecompat-git
```

① Enter a selection (default=all):

```
mingw-w64-clang-x86_64-tools-git-11.0.0.r655.gdbf
mingw-w64-clang-x86_64-winpthreads-git-11.0.0.r655.gdbf
mingw-w64-clang-x86_64-winstorecompat-git-11.0.0.r655.gdbf

Total Download Size:    244.42 MiB
Total Installed Size:  1708.18 MiB
```

② :: Proceed with installation? [Y/n] |

MSYS2 Setup

9. Write the command to install gdb package

- **pacman -S mingw-w64-clang-x86_64-gdb**

- GNU debugger(gdb) package

```
user@DESKTOP-NFT0JAP MSYS ~
$ pacman -S mingw-w64-clang-x86_64-gdb
resolving dependencies...
looking for conflicting packages...
```

- **pacman -S mingw-w64-clang-x86_64-gdb-multiarch**

- GNU debugger(gdb) support all targets package

```
user@DESKTOP-NFT0JAP MSYS ~
$ pacman -S mingw-w64-clang-x86_64-gdb-multiarch
resolving dependencies...
looking for conflicting packages...
```

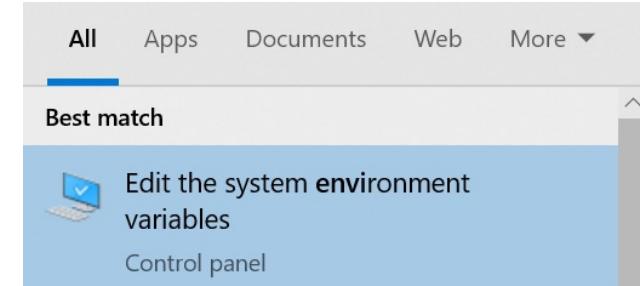
- Write 'Y' each in each installation process

```
:: Proceed with installation? [Y/n] Y
```

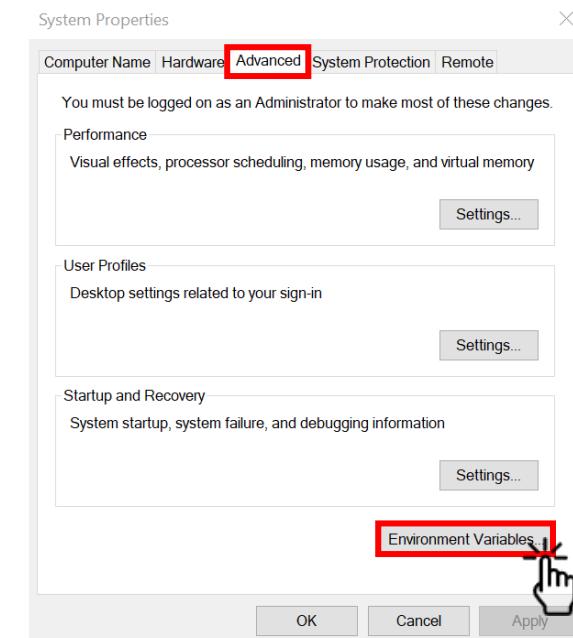
MSYS2 Setup

10. Edit environment variables

- Select “Edit the system environment variables” section in the control panel.



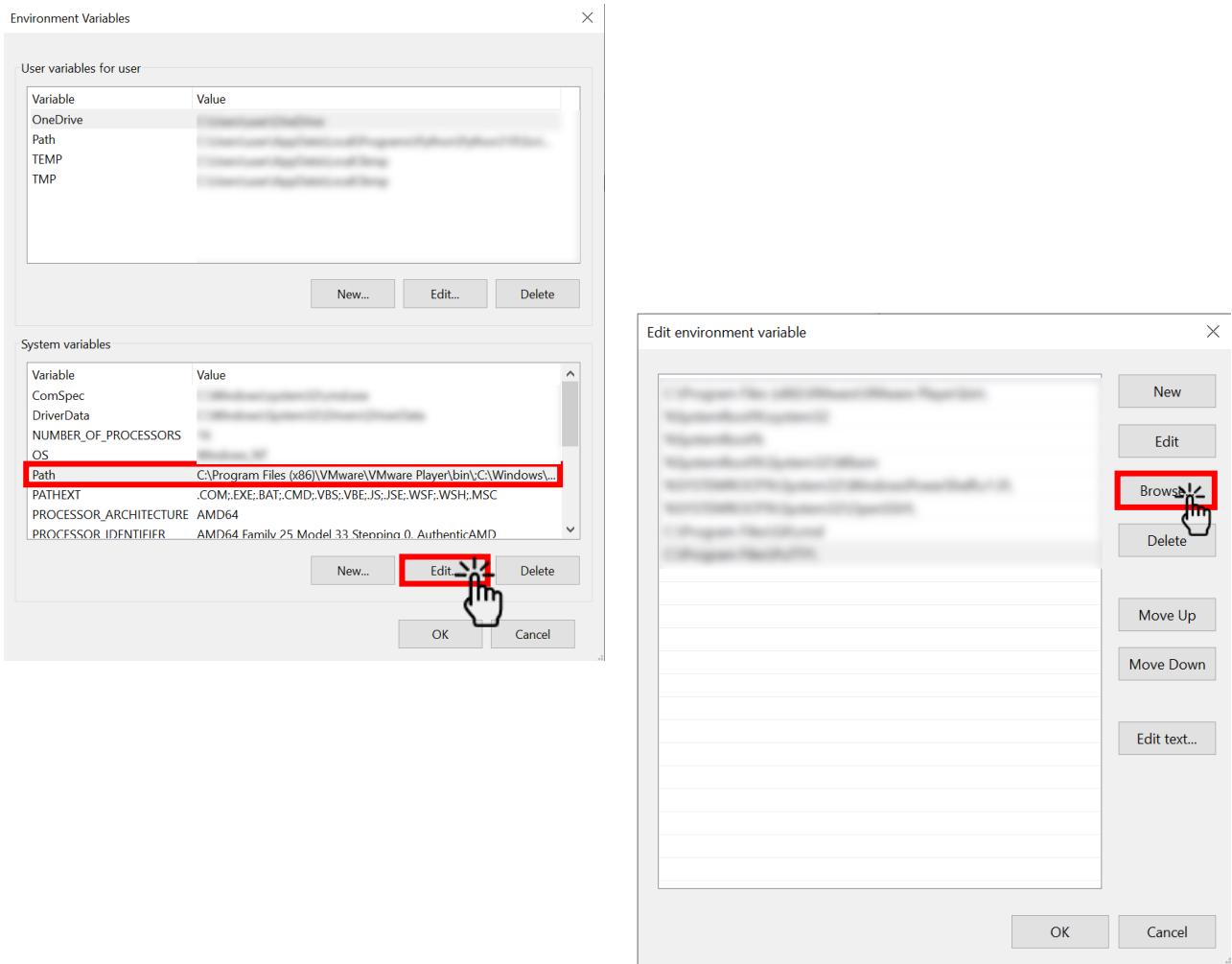
11. Click “Environment Variables...” button in “Advanced” tab.



MSYS2 Setup

12. Find and edit path of the system variable

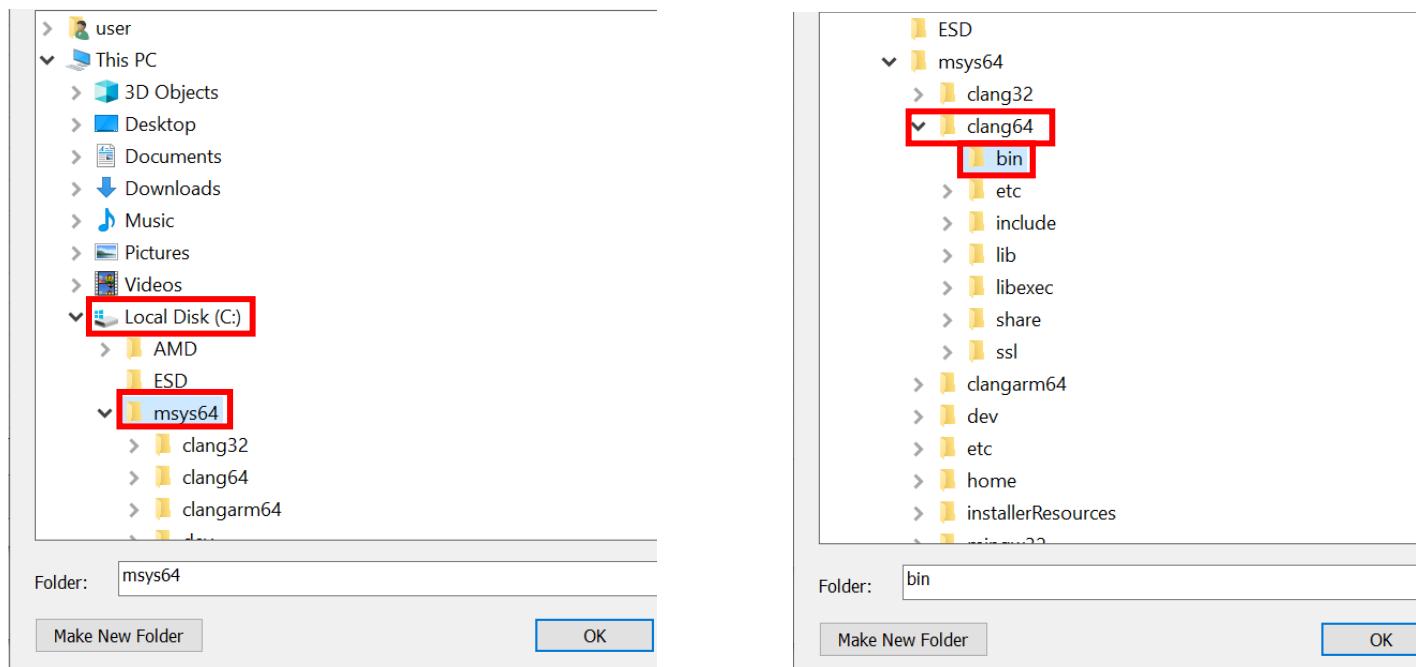
- Find a variable “Path” in the system variables.
- Click “Edit” button.



13. Click “Browse” button.

MSYS2 Setup

14. Select “C:\msys64\clang64\bin\” folder.



MSYS2 Setup

- **Check if your setups are valid**

- Open **VSCode** and press “*Ctrl + Shift + `*” to open new terminal window.
- Write “**clang -v**” in the terminal below.
- Your Clang version will be shown if your settings are valid.

```
PS C:\Users\user\Desktop\test> clang -v
clang version 18.1.2
```

```
Target: x86_64-w64-windows-gnu
```

```
Thread model: posix
```

```
InstalledDir: C:/msys64/clang64/bin
```

How to install C/C++ Compiler

macOS

What is Xcode?

Xcode

- Apple's IDE for software development.
- Includes various programming modules for source compilation.
- Includes the **Clang**/LLVM compilers and debugger.

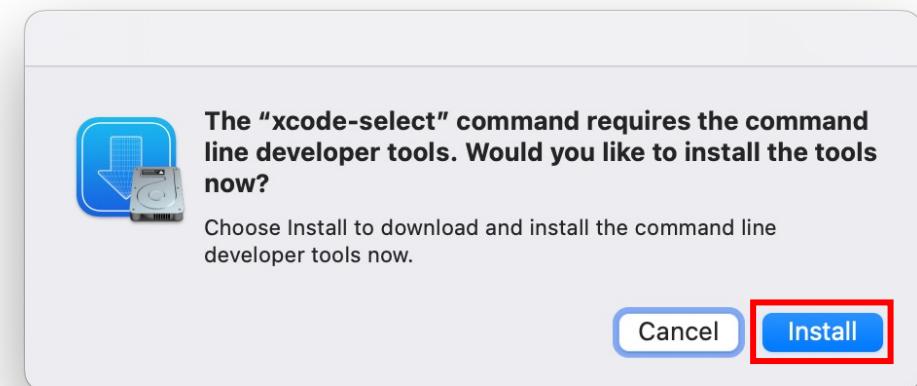


Xcode Installation & Setup

- **Install with Xcode Command Line Tools**

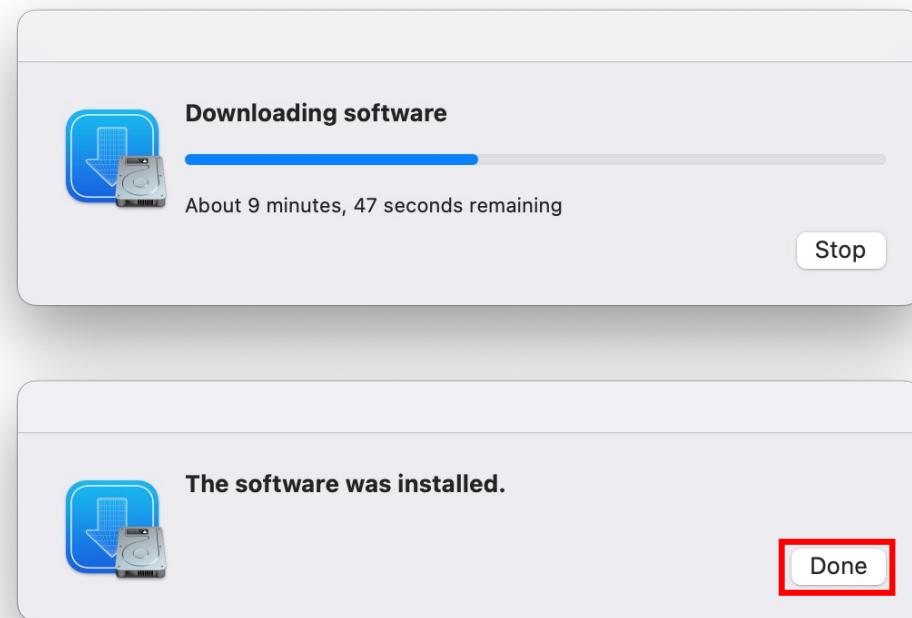
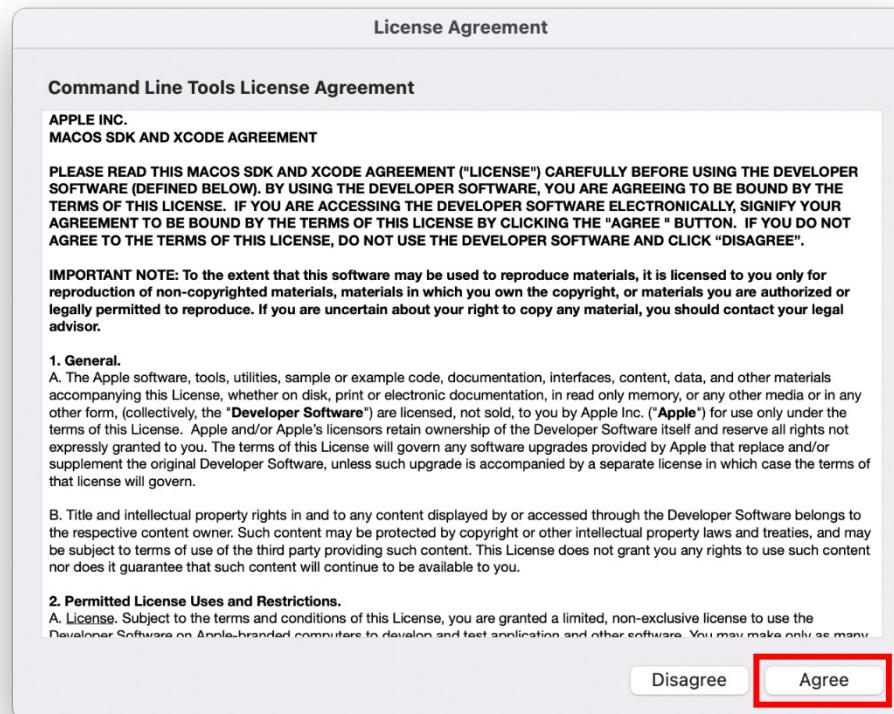
- Open the Terminal app and write “**xcode-select --install**”.
- Select install

```
> xcode-select --install
xcode-select: note: install requested for command line developer tools
● base
```



Xcode Installation & Setup

- **Install with Xcode Command Line Tools**
 - Click “Agree” button for the license agreement and wait for finishing installation.



Xcode Installation & Setup

- **Check if you have valid clang installed**
 - Enter “clang -v” to check the clang is available.
 - Outputs may look different depending on your mac’s CPU (Apple silicon / Intel).

```
› clang -v
Apple clang version 15.0.0 (clang-1500.3.9.4)
Target: arm64-apple-darwin23.3.0
Thread model: posix
InstalledDir: /Library/Developer/CommandLineTools/usr/bin
```

VS Code Setup

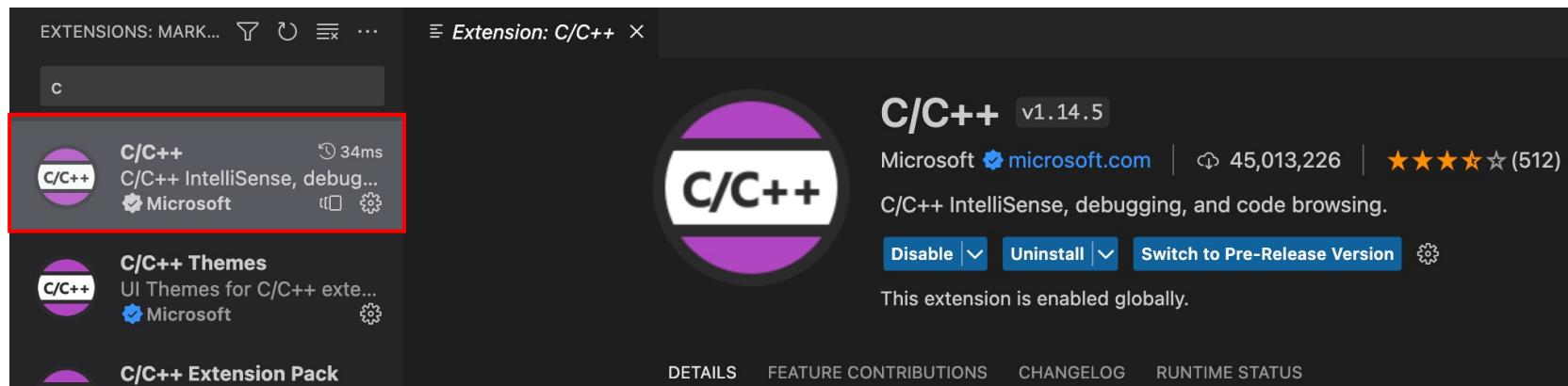
Windows & macOS



VS Code Setup

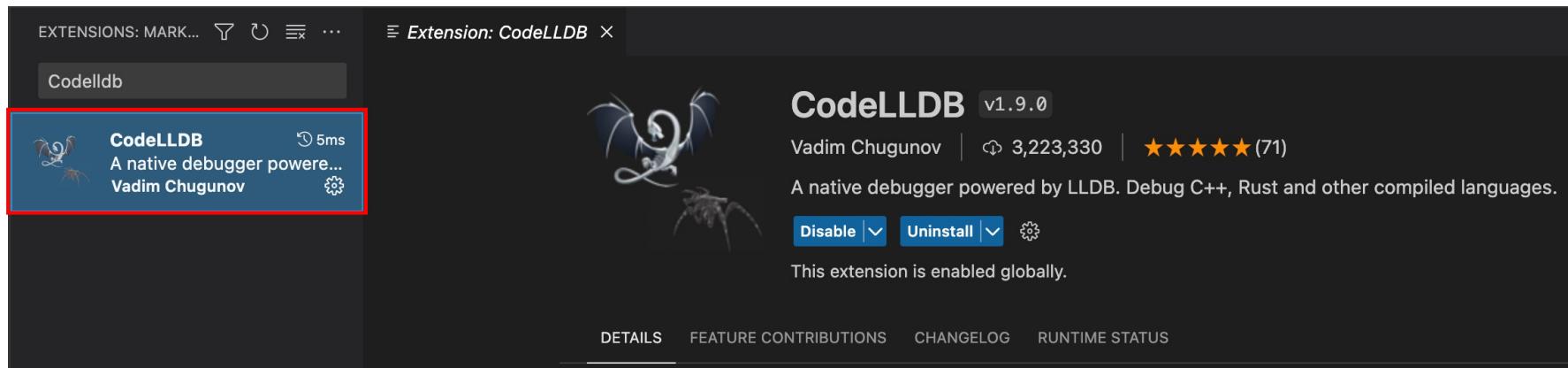
- **Install C/C++ extension**

- Tap extensions menu and search ‘C’ in the bar.
- Select the first “C/C++” extension and click “Install” button.



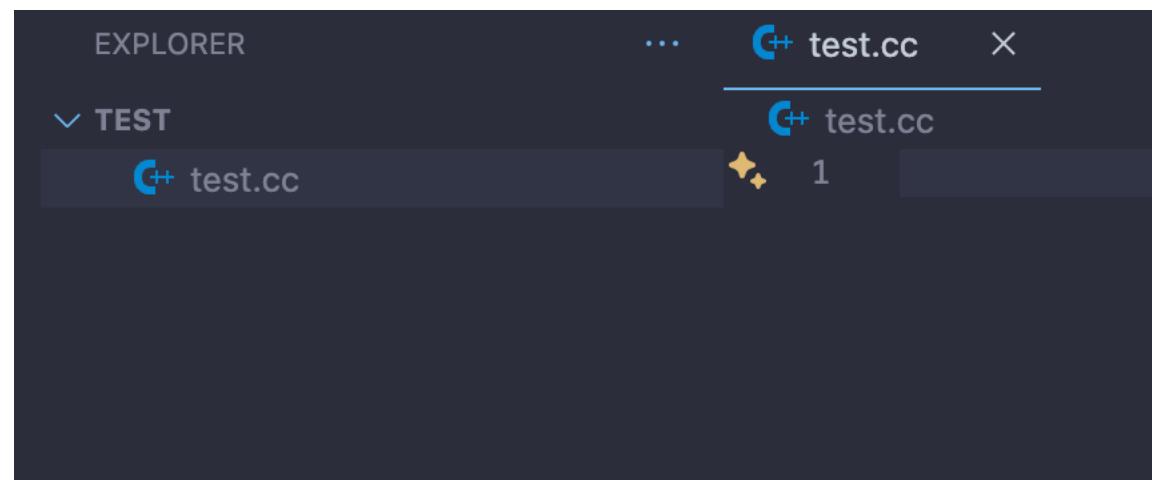
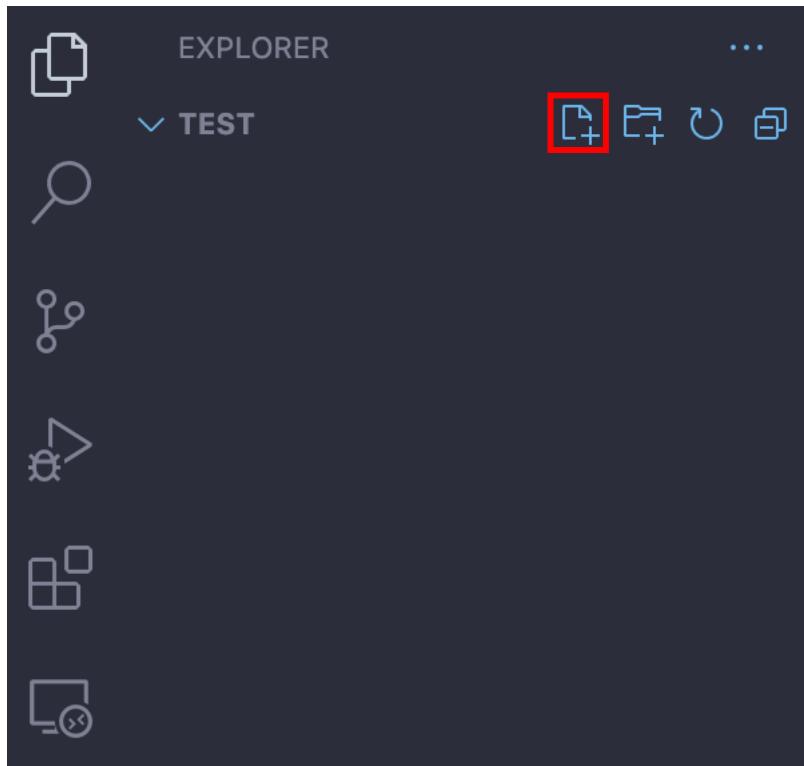
VS Code Setup

- **Install ‘CodeLLDB’ extension – Only MacOS**
 - Tap extensions menu and search ‘CodeLLDB’ in the bar.
 - Select the “CodeLLDB” extension and click “Install” button.



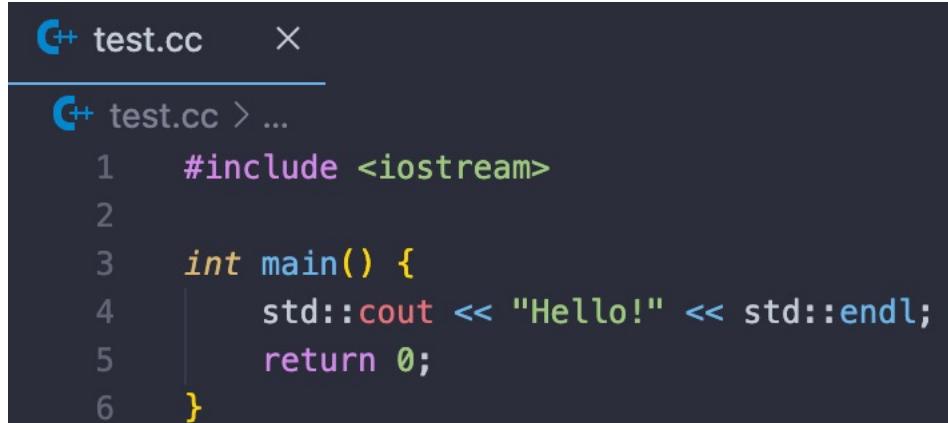
VS Code Setup

- Create '*test.cc*' file



VS Code Setup

- Write a simple C++ codes in the file



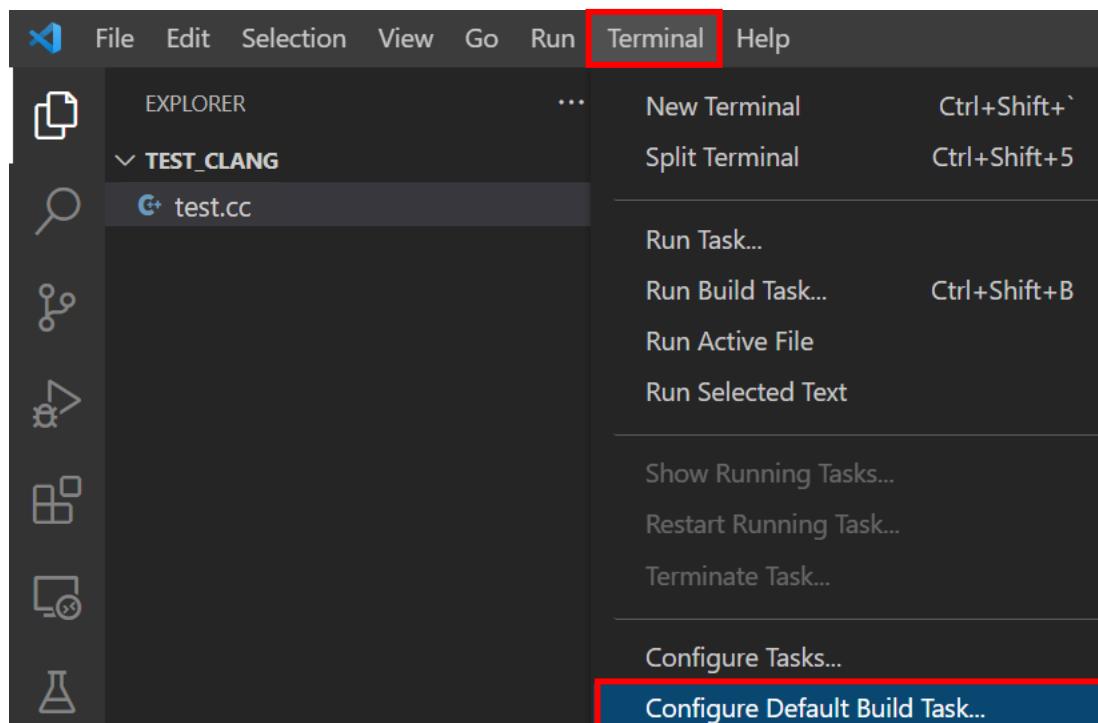
A screenshot of the VS Code interface showing a single file named "test.cc". The code is a simple C++ program that includes the iostream library, defines a main function, and prints "Hello!" to the console. The code is color-coded by the IDE.

```
C++ test.cc ×  
C++ test.cc > ...  
1 #include <iostream>  
2  
3 int main() {  
4     std::cout << "Hello!" << std::endl;  
5     return 0;  
6 }
```

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

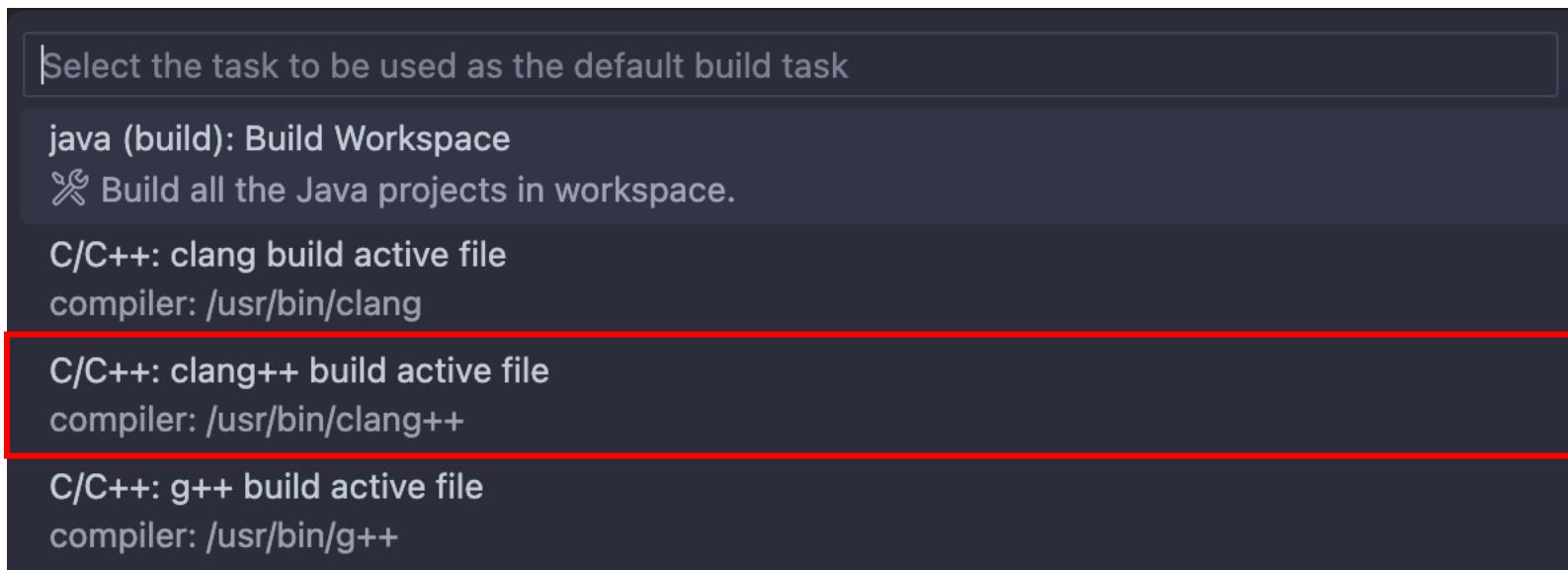
VS Code Setup

- Click 'Terminal > Configure Default Build Task'



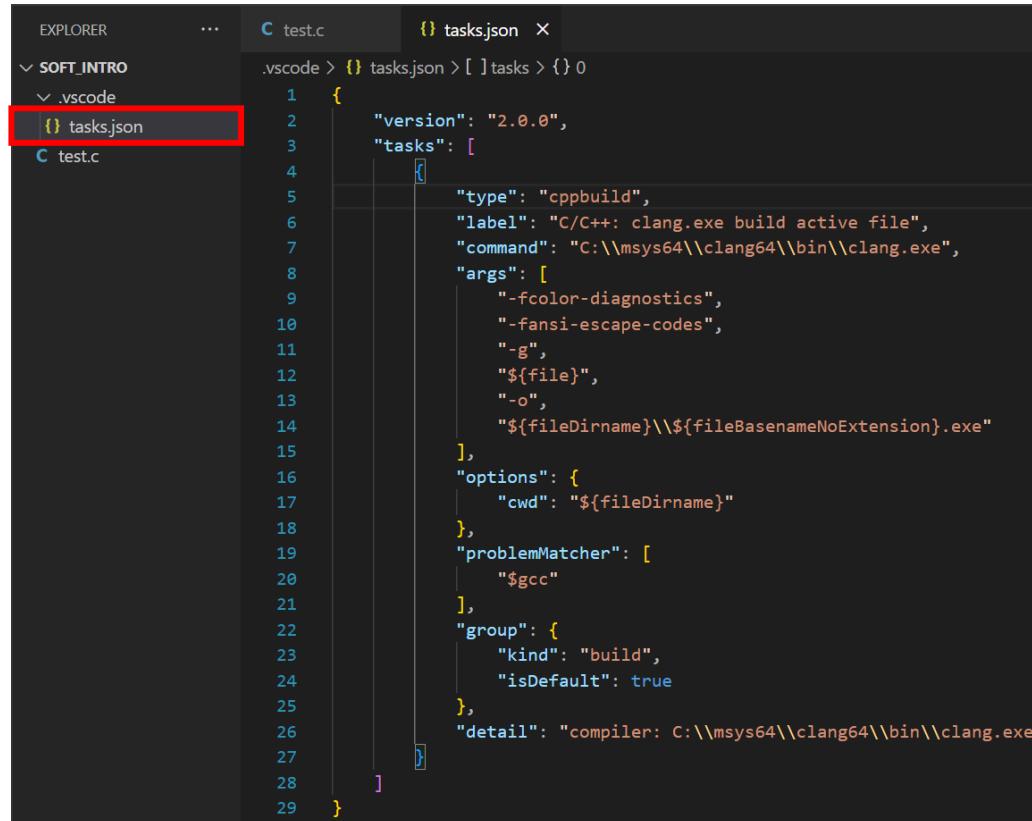
VS Code Setup – Build Source

- Select “C/C++: clang++ build active file”



VS Code Setup – Build Source (Windows)

- Create '*tasks.json*' file in '*.vscode*' folder



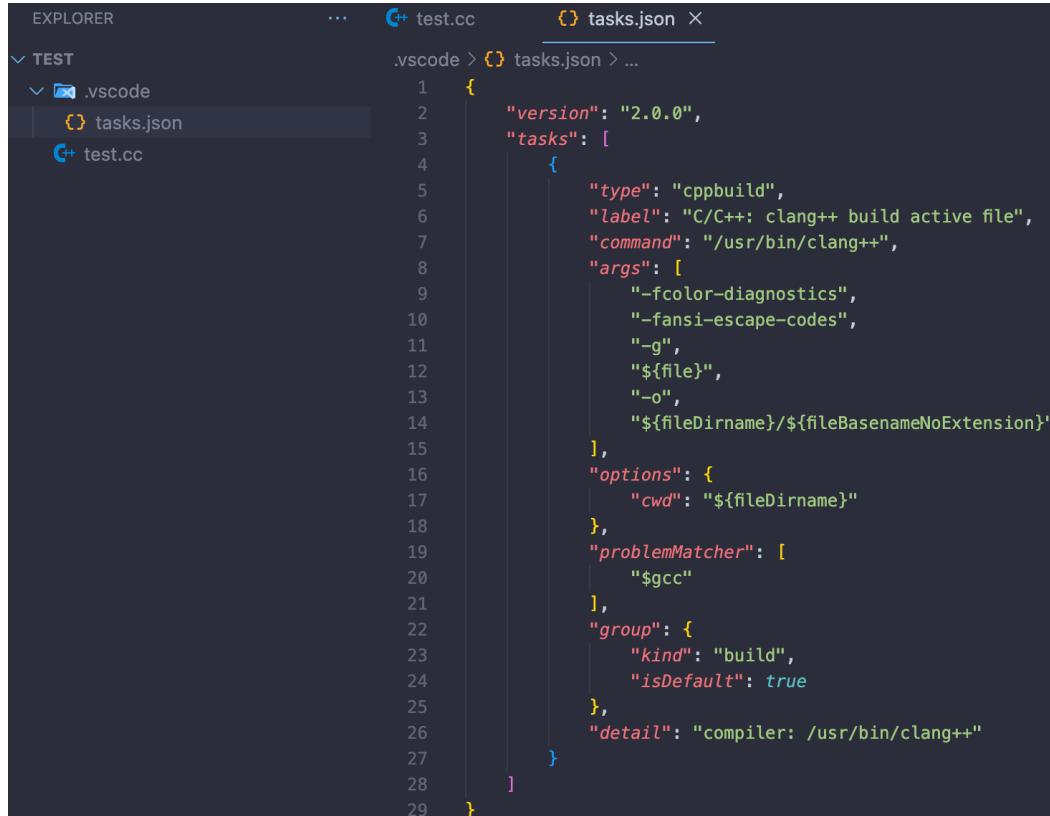
The screenshot shows the VS Code interface with the following details:

- Explorer View:** Shows a project structure with a folder named "SOFT_INTRO". Inside "SOFT_INTRO", there is a ".vscode" folder which contains a "tasks.json" file.
- Editor View:** The "tasks.json" file is open in the editor. A red box highlights the "tasks.json" file in the Explorer view, and another red box highlights the "tasks.json" file in the Editor view.
- Content of tasks.json:**

```
1  {
2    "version": "2.0.0",
3    "tasks": [
4      {
5        "type": "cppbuild",
6        "label": "C/C++: clang.exe build active file",
7        "command": "C:\\\\msys64\\\\clang64\\\\bin\\\\clang.exe",
8        "args": [
9          "-fcolor-diagnostics",
10         "-fansi-escape-codes",
11         "-g",
12         "${file}",
13         "-o",
14         "${fileDirname}\\\\${fileBasenameNoExtension}.exe"
15       ],
16       "options": {
17         "cwd": "${fileDirname}"
18       },
19       "problemMatcher": [
20         "$gcc"
21       ],
22       "group": {
23         "kind": "build",
24         "isDefault": true
25       },
26       "detail": "compiler: C:\\\\msys64\\\\clang64\\\\bin\\\\clang.exe"
27     }
28   }
```

VS Code Setup – Build Source (macOS)

- Create '*tasks.json*' file in '*.vscode*' folder



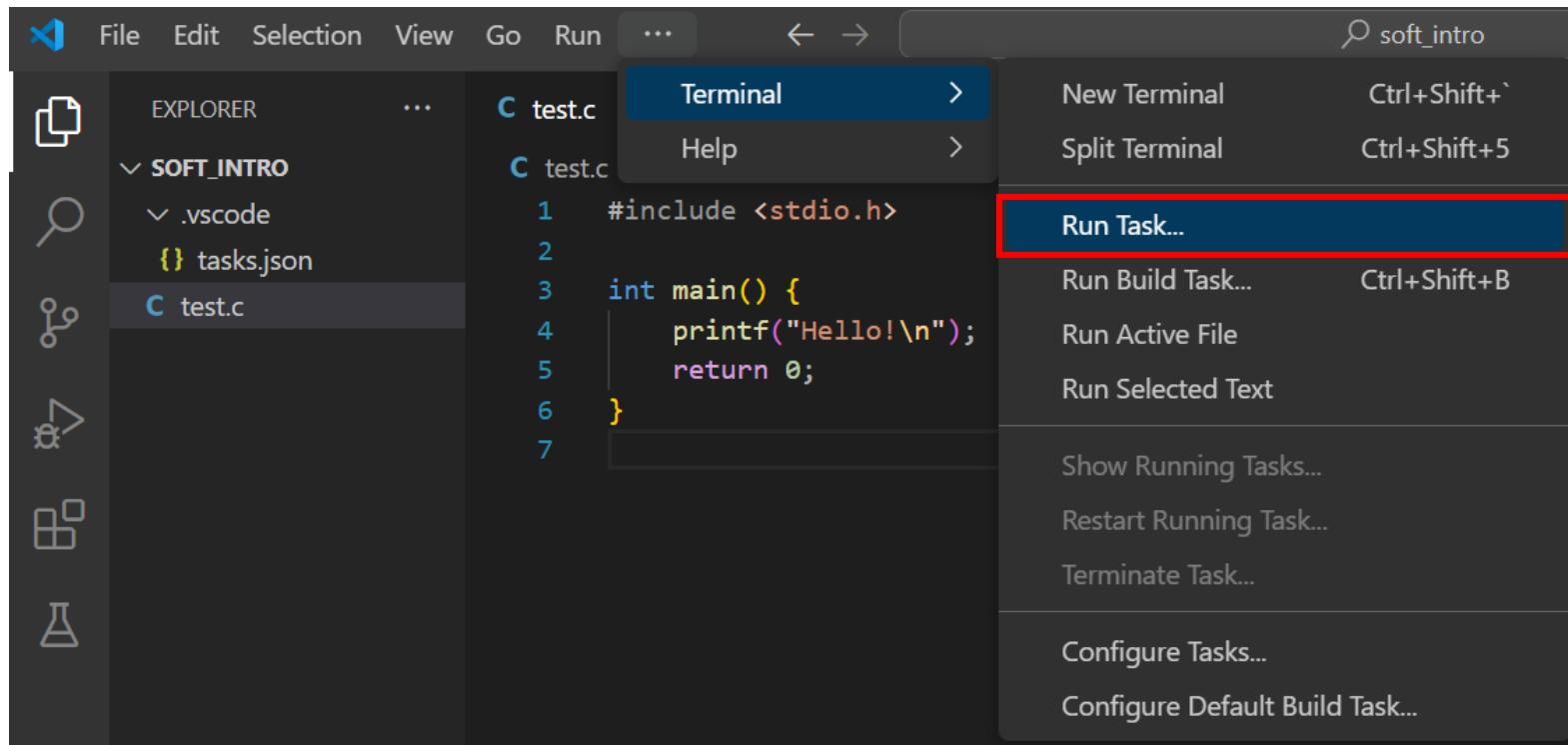
The screenshot shows the VS Code interface with the 'tasks.json' file open in the editor. The file is located in the '.vscode' folder of a project named 'TEST'. The code in the file defines a task for building C/C++ code using clang++.

```
EXPLORER      ...   C++ test.cc   tasks.json X
TEST
  .vscode
    tasks.json
    test.cc

1  {
2    "version": "2.0.0",
3    "tasks": [
4      {
5        "type": "cppbuild",
6        "label": "C/C++: clang++ build active file",
7        "command": "/usr/bin/clang++",
8        "args": [
9          "-fcolor-diagnostics",
10         "-fansi-escape-codes",
11         "-g",
12         "${file}",
13         "-o",
14         "${fileDirname}/${fileBasenameNoExtension}"
15       ],
16       "options": {
17         "cwd": "${fileDirname}"
18       },
19       "problemMatcher": [
20         "$gcc"
21       ],
22       "group": {
23         "kind": "build",
24         "isDefault": true
25       },
26       "detail": "compiler: /usr/bin/clang++"
27     }
28   }
29 }
```

VS Code Setup – Build Source

- Click Terminal > Run Task...



VS Code Setup – Build Source

- Check build result in the terminal window

Windows

```
* | Executing task: C/C++: clang.exe build active file

Starting build...
cmd /c chcp 65001>nul && C:\msys64\clang64\bin\clang.exe -fcolor-diagnostics -fansi-escape-codes -g C:\Users\user\Desktop\test\soft_intro\test.c -o C:\Users\user\Desktop\test\soft_intro\test.exe

Build finished successfully.
* | Terminal will be reused by tasks, press any key to close it.
```

macOS

```
* | Executing task: C/C++: clang++ build active file

Starting build...
/usr/bin/clang++ -std=gnu++14 -fcolor-diagnostics -fansi-escape-codes -g /Users/kevinbang/Desktop/test.cc -o /Users/kevinbang/Desktop/test

Build finished successfully.
* | Terminal will be reused by tasks, press any key to close it.
```

VS Code Setup – Execution & Debugging

- Open '*test.cc*' file and Click 'Add Debug configuration' icon



A screenshot of the Visual Studio Code interface. The left side shows a dark-themed code editor with a C++ file named "test.cc". The code contains a simple "Hello World" program:

```
C++ test.cc    X
C++ test.cc > ...
1 #include <iostream>
2
3 int main() {
4     std::cout << "Hello!" << std::endl;
5     return 0;
6 }
```

The right side of the interface features a dark-themed toolbar with several icons: a play button, a gear icon (which is highlighted with a red box), a pause button, and a three-dot menu. A small tooltip for the gear icon reads "Run Code Action on Selection".

VS Code Setup – Execution & Debugging

- **Select a configuration**
 - “**C/C++: clang++(.exe) build and debug active file”**

Select a debug configuration

C/C++: clang.exe build and debug active file preLaunchTask: C/C++: clang.exe build active file
Configured Task (compiler: C:\msys64\clang64\bin\clang.exe)

C/C++: gcc.exe build and debug active file preLaunchTask: C/C++: gcc.exe build active file
Detected Task (compiler: C:\msys64\clang64\bin\gcc.exe)

C/C++: cl.exe build and debug active file preLaunchTask: C/C++: cl.exe build active file
Detected Task (compiler: cl.exe)

(gdb) Launch

(Windows) Launch

Windows

Select a debug configuration

C/C++: clang++ build and debug active file preLaunchTask: C/C++: clang build active file
Configured Task (Task generated by Debugger.)

C/C++: clang++ build and debug active file preLaunchTask: C/C++: clang++ build active file
Detected Task (compiler: /usr/bin/clang++)

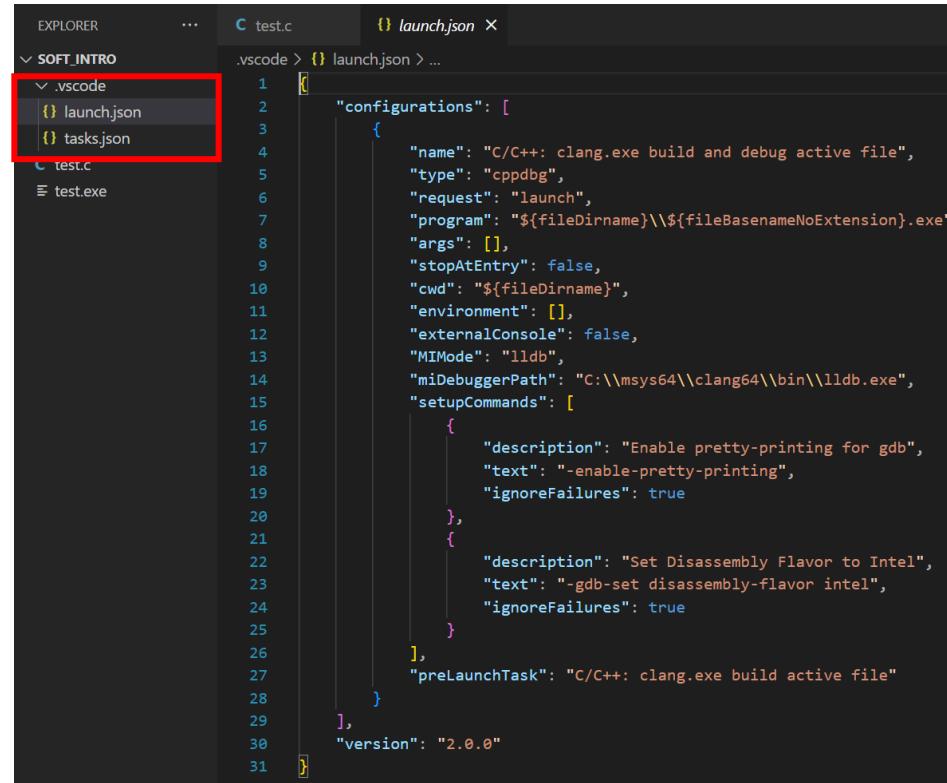
C/C++: g++ build and debug active file preLaunchTask: C/C++: g++ build active file
Detected Task (compiler: /usr/bin/g++)

(lldb) Launch

macOS

VS Code Setup - Execution & Debugging (Windows)

- Create '*launch.json*' file in '*.vscode*' folder



The screenshot shows the VS Code interface with the following details:

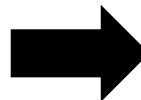
- EXPLORER View:** Shows a tree structure with the root folder ".SOFT_INTRO". Under ".vscode", there are two files: "launch.json" and "tasks.json". Both of these files are highlighted with a red rectangular box.
- Editor View:** The "test.c" file is open in the editor. Above it, the "launch.json" file is shown in the editor tab. The content of the "launch.json" file is displayed in the editor area, showing a JSON configuration for debugging C/C++ code using clang.exe and lldb.

```
1: {
2:   "configurations": [
3:     {
4:       "name": "C/C++: clang.exe build and debug active file",
5:       "type": "cppdbg",
6:       "request": "launch",
7:       "program": "${fileDirname}\\${fileBasenameNoExtension}.exe",
8:       "args": [],
9:       "stopAtEntry": false,
10:      "cwd": "${fileDirname}",
11:      "environment": [],
12:      "externalConsole": false,
13:      "MIMode": "lldb",
14:      "miDebuggerPath": "C:\\msys64\\clang64\\bin\\lldb.exe",
15:      "setupCommands": [
16:        {
17:          "description": "Enable pretty-printing for gdb",
18:          "text": "-enable-pretty-printing",
19:          "ignoreFailures": true
20:        },
21:        {
22:          "description": "Set Disassembly Flavor to Intel",
23:          "text": "-gdb-set disassembly-flavor intel",
24:          "ignoreFailures": true
25:        }
26:      ],
27:      "preLaunchTask": "C/C++: clang.exe build active file"
28:    }
29:  ],
30:  "version": "2.0.0"
31: }
```

VS Code Setup – Execution & Debugging (Windows)

- Open the '*launch.json*' file
 - replace the '*lldb*' with '*gdb*' in the '*launch.json*' file
 - If it is written as '*gdb*', there is no need to change it.

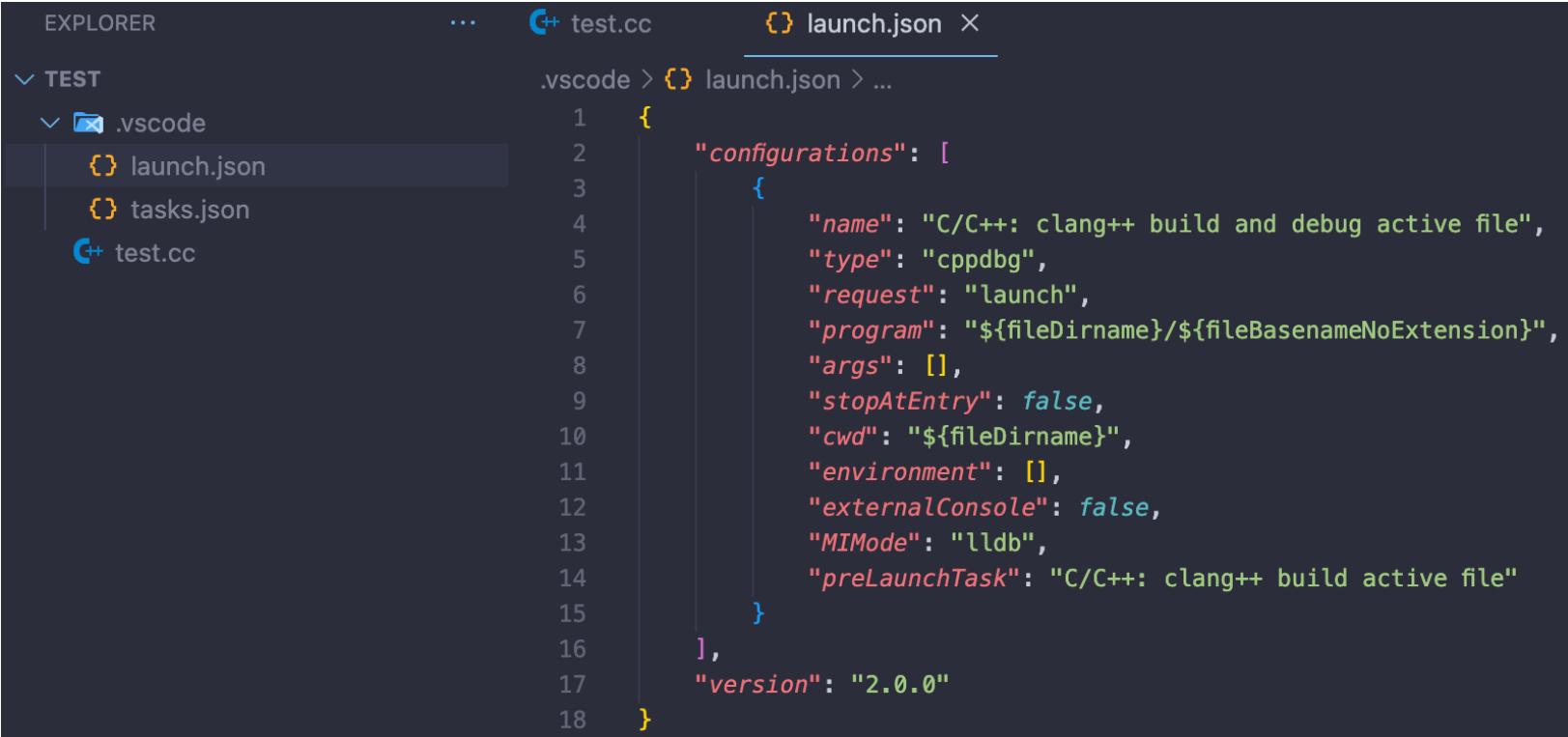
```
{  
  "configurations": [  
    {  
      "name": "C/C++: clang.exe build and debug active file",  
      "type": "cppdbg",  
      "request": "launch",  
      ...  
      "MIMode": "lldb",  
      "miDebuggerPath": "C:\\msys64\\clang64\\bin\\lldb.exe",  
      "setupCommands": [  
        ...  
      ],  
      "preLaunchTask": "C/C++: clang.exe build active file"  
    }  
  ],  
  "version": "2.0.0"  
}
```



```
{  
  "configurations": [  
    {  
      "name": "C/C++: clang.exe build and debug active file",  
      "type": "cppdbg",  
      "request": "launch",  
      ...  
      "MIMode": "gdb",  
      "miDebuggerPath": "C:\\msys64\\clang64\\bin\\gdb.exe",  
      "setupCommands": [  
        ...  
      ],  
      "preLaunchTask": "C/C++: clang.exe build active file"  
    }  
  ],  
  "version": "2.0.0"  
}
```

VS Code Setup – Execution & Debugging (macOS)

- Create '*launch.json*' file in '*.vscode*' folder



The screenshot shows the VS Code interface with the following details:

- EXPLORER** sidebar: TEST folder expanded, showing .vscode (selected), launch.json, tasks.json, and test.cc.
- Editor Area**: The file `launch.json` is open, showing its JSON configuration:

```
1  {
2      "configurations": [
3          {
4              "name": "C/C++: clang++ build and debug active file",
5              "type": "cppdbg",
6              "request": "launch",
7              "program": "${fileDirname}/${fileBasenameNoExtension}",
8              "args": [],
9              "stopAtEntry": false,
10             "cwd": "${fileDirname}",
11             "environment": [],
12             "externalConsole": false,
13             "MIMode": "lldb",
14             "preLaunchTask": "C/C++: clang++ build active file"
15         }
16     ],
17     "version": "2.0.0"
18 }
```

VS Code Setup - Execution & Debugging (macOS)

- Open the '*launch.json*' file
 - Replace the '*cppdbg*' with '*lldb*' in the '*launch.json*' file
 - Delete '*stopAtEntry*' , '*externalConsole*' , '*environment*' and '*MIMode*'

```
{  
  "configurations": [  
    {  
      "name": "C/C++: clang++ build and debug active file",  
      "type": "cppdbg",  
      "request": "launch",  
      "program": "${fileDirname}/${fileBasenameNoExtension}",  
      "args": [],  
      "stopAtEntry: false,  
      "cwd": "${fileDirname}",  
      "environment: [],  
      "externalConsole: false,  
      "MIMode: "lldb",  
      "preLaunchTask": "C/C++: clang++ build active file"  
    }  
  "version": "2.0.0"  
}
```

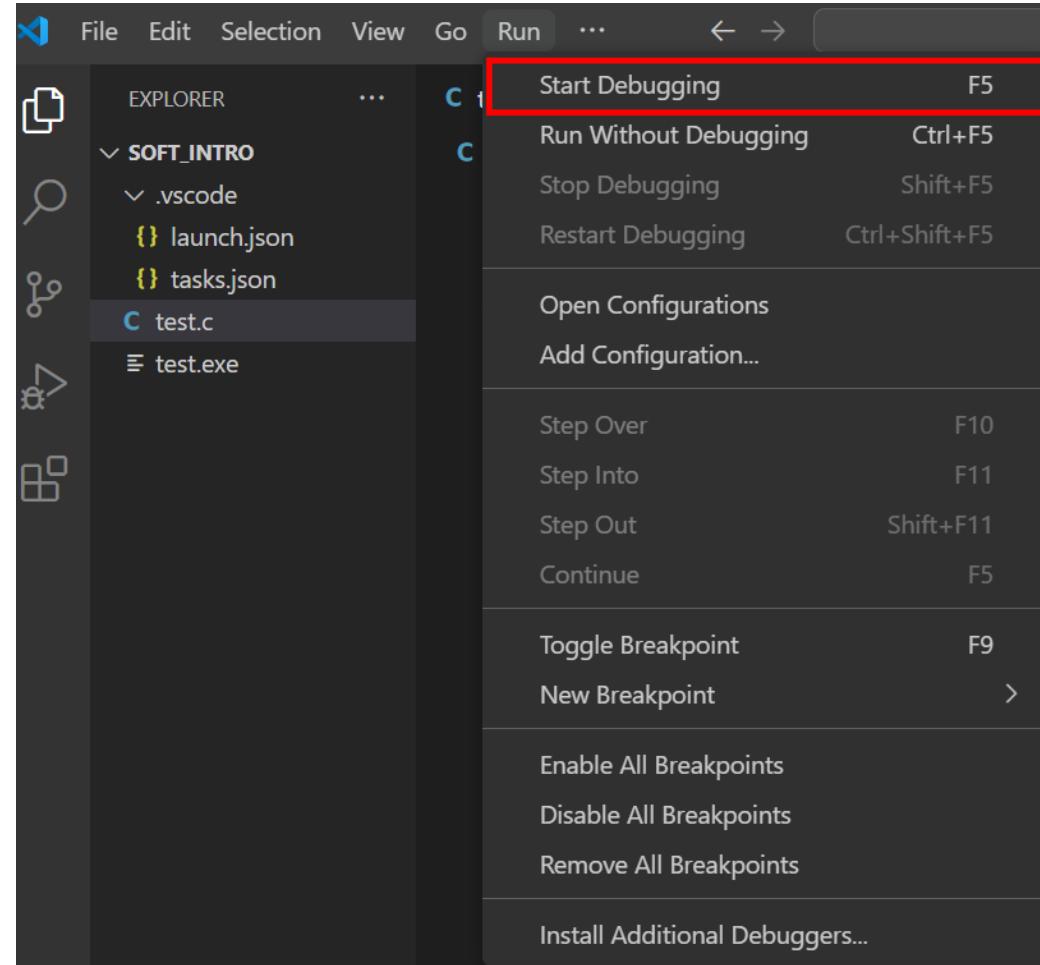


```
{  
  "configurations": [  
    {  
      "name": "C/C++: clang++ build and debug active file",  
      "type": "lldb",  

```

VS Code Setup – Execution & Debugging

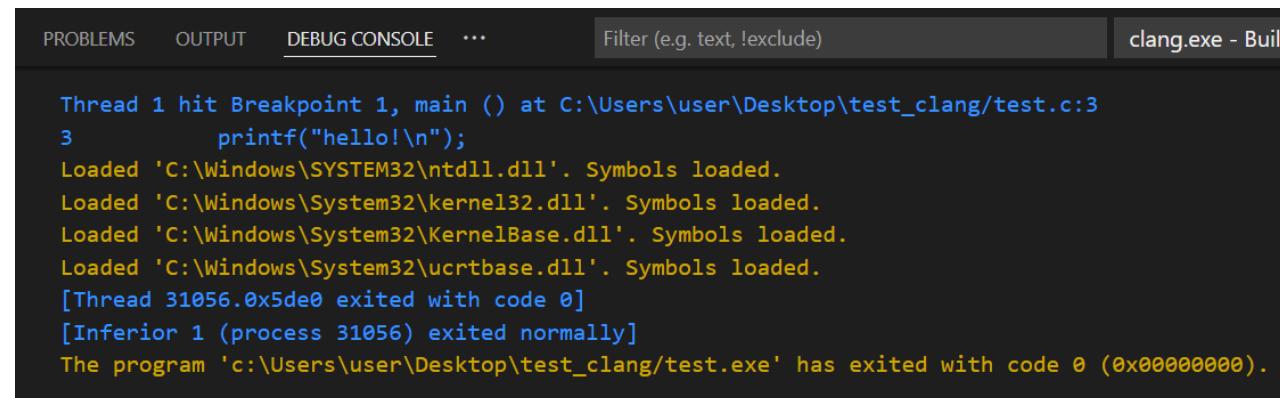
- Open *test.c* file and press '**F5**' or click '**Run > Start Debugging**'



VS Code Setup – Execution & Debugging (Windows)

- Complete debugging *test.cc* file

```
PS C:\Users\user\Desktop\test\soft_intro> & 'c:\Users\user\.vscode\extensions\ms-vscode.cpptools-1.19.9-win32-x64\debugAdapters\bin\WindowsDebugLauncher.exe' '--stdin=Microsoft-MIEngine-In-claz4lmx.5pp' '--stdout=Microsoft-MIEngine-Out-i3tte2cg.t53' '--stderr=Microsoft-MIEngine-Error-rvk4p2fj.syq' '--pid=Microsoft-MIEngine-Pid-jmgjq5kg.w24' '--dbgExe=C:\msys64\clang64\bin\gdb.exe' '--interpreter=mi'
Hello!
```



The screenshot shows the VS Code interface with the DEBUG CONSOLE tab selected. The console window displays the following text:

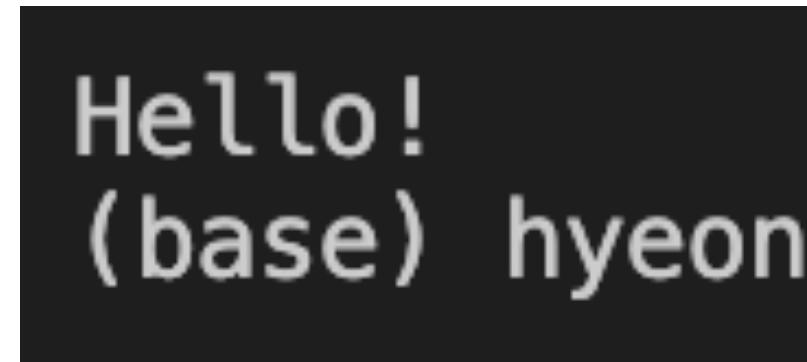
```
PROBLEMS OUTPUT DEBUG CONSOLE ...
Filter (e.g. text, exclude) clang.exe - Build

Thread 1 hit Breakpoint 1, main () at C:\Users\user\Desktop\test_clang/test.c:3
3         printf("hello!\n");
Loaded 'C:\Windows\SYSTEM32\ntdll.dll'. Symbols loaded.
Loaded 'C:\Windows\System32\kernel32.dll'. Symbols loaded.
Loaded 'C:\Windows\System32\KernelBase.dll'. Symbols loaded.
Loaded 'C:\Windows\System32\ucrtbase.dll'. Symbols loaded.
[Thread 31056.0x5de0 exited with code 0]
[Inferior 1 (process 31056) exited normally]
The program 'c:\Users\user\Desktop\test_clang/test.exe' has exited with code 0 (0x00000000).
```

VS Code Setup – Execution & Debugging (macOS)

- Complete debugging *test.cc* file

```
* Executing task: C/C++: clang++ build active file
Starting build...
/usr/bin/clang++ -std=gnu++14 -fcolor-diagnostics -fansi-escape-codes -g /Users/kevinbang/Desktop/test/test.cc -o /Users/kevinbang/Desktop/test/test
Build finished successfully.
* Terminal will be reused by tasks, press any key to close it.
```



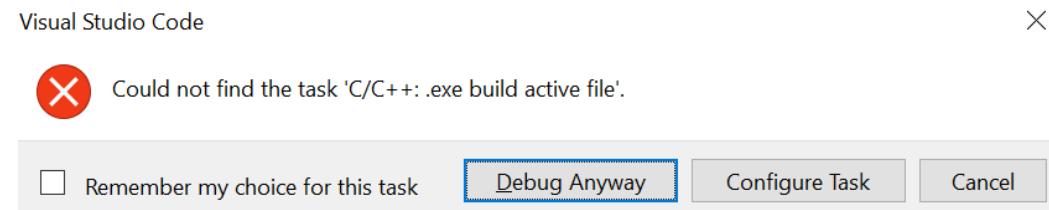
A terminal window showing the output of a C/C++ build using clang++. The terminal shows the command being run, the build starting, the compiler path, the source and output files, a successful build, and a message indicating the terminal will be reused. The background of the terminal window is dark gray.

```
Hello!
(base) hyeon:
```

Resolve the environment set up errors

Error related to “could not find file”

- If you get an error like the picture below when you execute C code file, follow the solution below.



Solution

- Check the '*preLaunchTask*' part of the '*launch.json*' file.
- If '*preLaunchTask*' is not the same as the '*label*' in the '*task.json*' file, change it to be the same.

```
{  
  "configurations": [  
    {  
      "name": "C/C++: clang.exe build and debug active file",  
      "type": "cppdbg",  
      "request": "launch",  
      ...  
      "preLaunchTask": "C/C++: clang.exe build active file"  
    }  
  ],  
  "version": "2.0.0"  
}
```

Launch.json

```
{  
  "version": "2.0.0",  
  "tasks": [  
    {  
      "type": "cppbuild",  
      "label": "C/C++: clang.exe build active file",  
      "command": "C:\\msys64\\clang64\\bin\\clang.exe",  
      "args": [  
        ...  
      ]  
    }  
  ]  
}
```

task.json



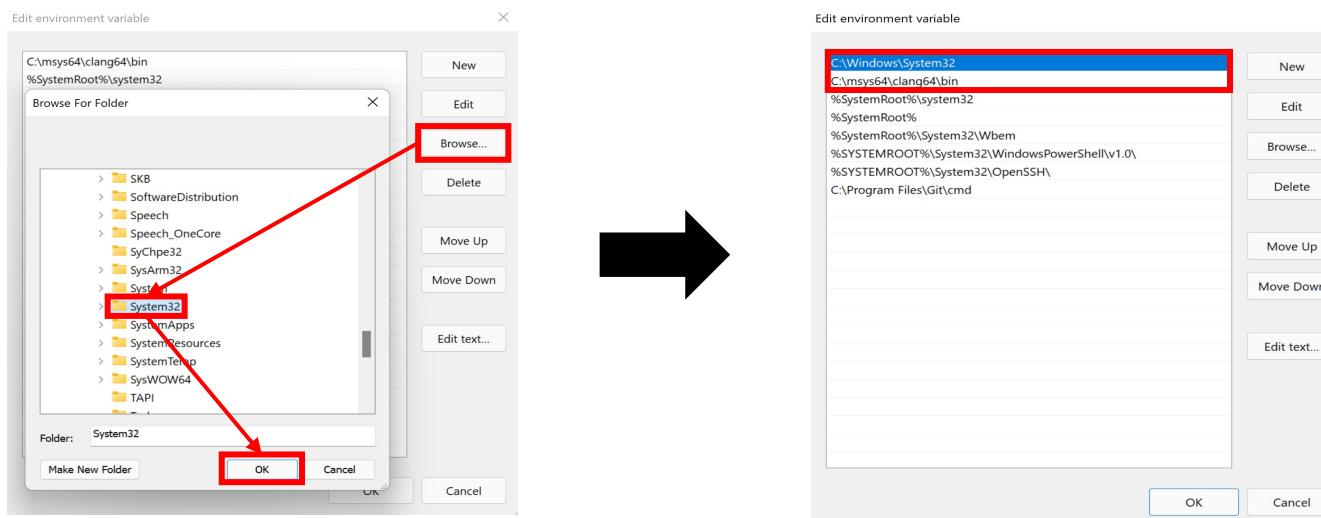
Error related to “cmd”

- If you get an error like the picture below when you execute ‘run task’, follow the solution below.

The screenshot shows a terminal window with the following text:
* Executing task: C/C++: clang.exe build active file
Starting build...
C:\msys64\clang64\bin\clang.exe -fcolor-diagnostics -fansi-escape-codes -g "C:\coding C\test.c" -o "C:\coding C\test.exe"
'cmd' is not recognized as an internal or external command,
operable program or batch file.
Build finished with error(s).
* The terminal process failed to launch (exit code: -1).
* Terminal will be reused by tasks, press any key to close it.

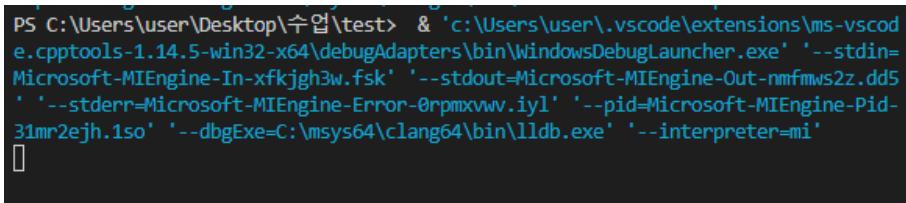
Solution

- Add a path “C:\Windows\System32” to the PATH environment variable.



Error related to “unable to establish a connection to LLDB”

- When you execute ‘start debugging’(= Click ‘F5’) an endless error as shown in the ‘Figure 1’ and then you press ‘Ctrl +C’ on the terminal an error message as shown in the Figure 2.



```
PS C:\Users\user\Desktop\수업\test> & 'c:\Users\user\.vscode\extensions\ms-vscode.cpptools-1.14.5-win32-x64\debugAdapters\bin\WindowsDebugLauncher.exe' '--stdin=Microsoft-MIEngine-In-xfkjgh3w.fsk' '--stdout=Microsoft-MIEngine-Out-nmfms2z.dd5' '--stderr=Microsoft-MIEngine-Error-0rpxmxww.iy1' '--pid=Microsoft-MIEngine-Pid-31mr2ejh.1so' '--dbgExe=C:\msys64\clang64\bin\lldb.exe' '--interpreter=mi'
```

Figure 1

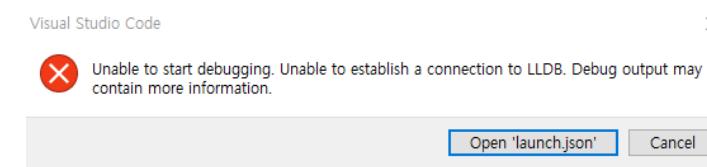
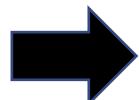


Figure 2

- Solution

- Replace the ‘lldb’ with ‘gdb’ in the launch.json file.

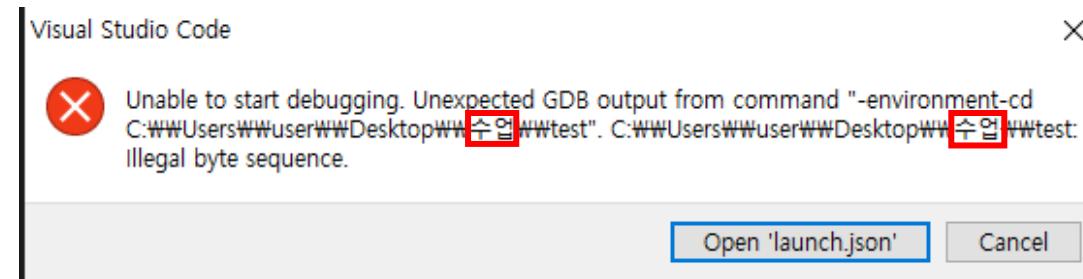
```
//launch.json file
"configurations": [
{
    ...
    "MIMode": "lldb",
    "miDebuggerPath": "C:\\msys64\\clang64\\bin\\lldb.exe",
    ...
    "preLaunchTask": "C/C++: clang.exe build active file"
}
],...
```



```
//launch.json file
"configurations": [
{
    ...
    "MIMode": "gdb",
    "miDebuggerPath": "C:\\msys64\\clang64\\bin\\gdb.exe",
    ...
    "preLaunchTask": "C/C++: clang.exe build active file"
}
],...
```

Error related to “illegal byte sequential”

- If you get an error like the picture below when you execute ‘start debugging’(press ‘F5’), follow the solution below.



- **Solution**

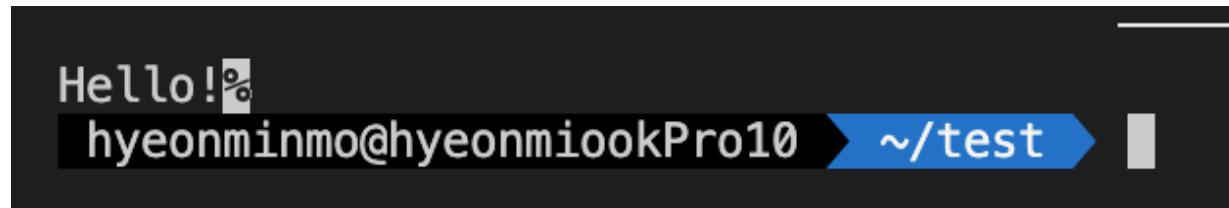
- Please make sure that there is no any letter which is not an English including whitespace
- Move your current project folder to another path without Korean language.
 - Tip. Path of the following picture is equivalent to ‘C:\Users\Public’



FAQ

How to remove “%” at the end of a terminal output

- If you the result of executing the code is ‘%’ as shown in the picture below



```
Hello!%
hyeonminmo@hyeonmiookPro10 ~/test ➜
```

- Solution

- Command +Shift+p (or view> command pallete)
- search for “*Terminal: default Profile*”
- select “*bash*”

