

LW: Setup with Hello World

LW: Setup with Hello World	1
Completion Requirements	1
Setup	1
Windows	2
Installing a Linux environment in Windows	2
Installing a C++ compiler and debugger in Windows	3
Files on your Windows computer	3
Installing Visual Studio Code in Windows	5
Configuring Visual Studio Code to Run C++ in Windows	5
Mac	6
Installing a C++ Compiler in Mac	6
Visual Studio Code Installation in Mac	7
Linux	7
Installing a C++ Compiler in Linux	7
Installing Visual Studio Code	8
Write, Compile and Run a C++ Program	8
Using GDB to Debug Your program	13
Submitting to Gradescope	14

You need to do this ASAP so you can participate in your lecture, labs, and do homework.

Completion Requirements

- Demonstrate compiling and running your Hello World program to a TA. Hopefully you'll finish during your lab session. Otherwise do it during any TA's office hours.
- Unlike future labs, you do not have to attend this lab to get credit.
 - For example, you added the course after the lab met in the first week.
 - You can do it before your first lab meets so you can use it in your course lectures.

Setup

Jump to the instructions for your Operating System:

- [Windows](#)
- [Mac](#)
- [Linux](#)

Windows

For the first week only, you can complete this labwork on your own and do not have to attend a lab session to get credit.

We will be using a common compiling environment for all students in the class, regardless of whether you use macOS, Windows, or a Linux distribution. The way we have decided to do this is to have all students compile in Visual Studio Code using the Windows Linux Subsystem (Ubuntu). While there are other environments we could use (e.g. Cygwin), using Ubuntu will allow access to tools to better debug your code. This guide allows you to edit your source code in Visual Studio Code and then compile, run, and debug the code with the installed tools in Ubuntu.

Installing a Linux environment in Windows

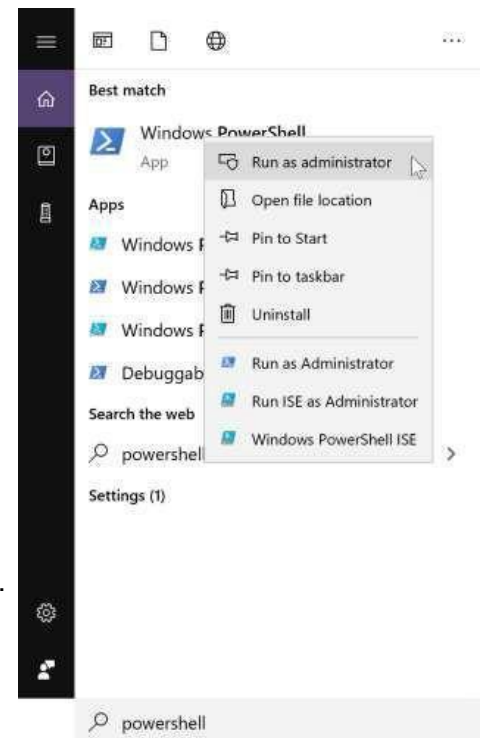
These instructions are based on [the Microsoft webpage for installing WSL on Windows 10](#):

1. It would be prudent to bookmark this Google doc since you will have to reboot your system
2. Install the Windows Subsystem for Linux
 - a. Open Powershell as Administrator
 - i. Type “powershell” into the search bar
 - ii. Right-click on “Windows PowerShell”
 - iii. Select “Run as administrator” in the pop-up menu
 - b. Install WSL
 - i. In the PowerShell command line, run the command `wsl --install`
 - c. Restart PC
 - i. Close PowerShell
 - ii. Restart your computer (make sure you can find this doc after your computer restarts!)

(NOTE) This only works if WSL is not already installed. If you run `wsl -install` and see WSL help text, then try running `wsl --list --online` to see a list of available distros and run `wsl --install -d <DistroName>` to install a distro

(NOTE) To uninstall WSL, see [Uninstall legacy version of WSL](#) or [unregister or uninstall a Linux distribution](#).

(NOTE) WSL will automatically install the Ubuntu operating system, which is the operating system we strongly encourage you to use. .



Installing a C++ compiler and debugger in Windows

These instructions are based on:

<https://linuxize.com/post/how-to-install-gcc-compiler-on-ubuntu-18-04/>

1. In your Ubuntu shell, type:

```
sudo apt update
```

2. In your Ubuntu shell, type:

```
sudo apt install build-essential gdb
```

Type 'y' when it asks if you want to continue. If this window appears to hang with "Processing triggers", push enter.

3. (optional-- this will install help pages for linux commands) Execute

```
sudo apt install manpages-dev
```

4. Verify you installed gcc by executing

```
gcc --version
```

The output should be similar to this (text below last updated 8/21/2023):

```
gcc (Ubuntu 11.4.0-1ubuntu1~22.04) 11.4.0
Copyright (C) 2021 Free Software Foundation, Inc.
This is free software; see the source for copying conditions. There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
```

Files on your Windows computer

You can access your c drive (and probably other drives) through your ubuntu shell. You can list the drives by executing:

```
df
```

You'll see /mnt/c. This leads to the C:/ drive on your computer. This is useful for the Unix Tutorial.

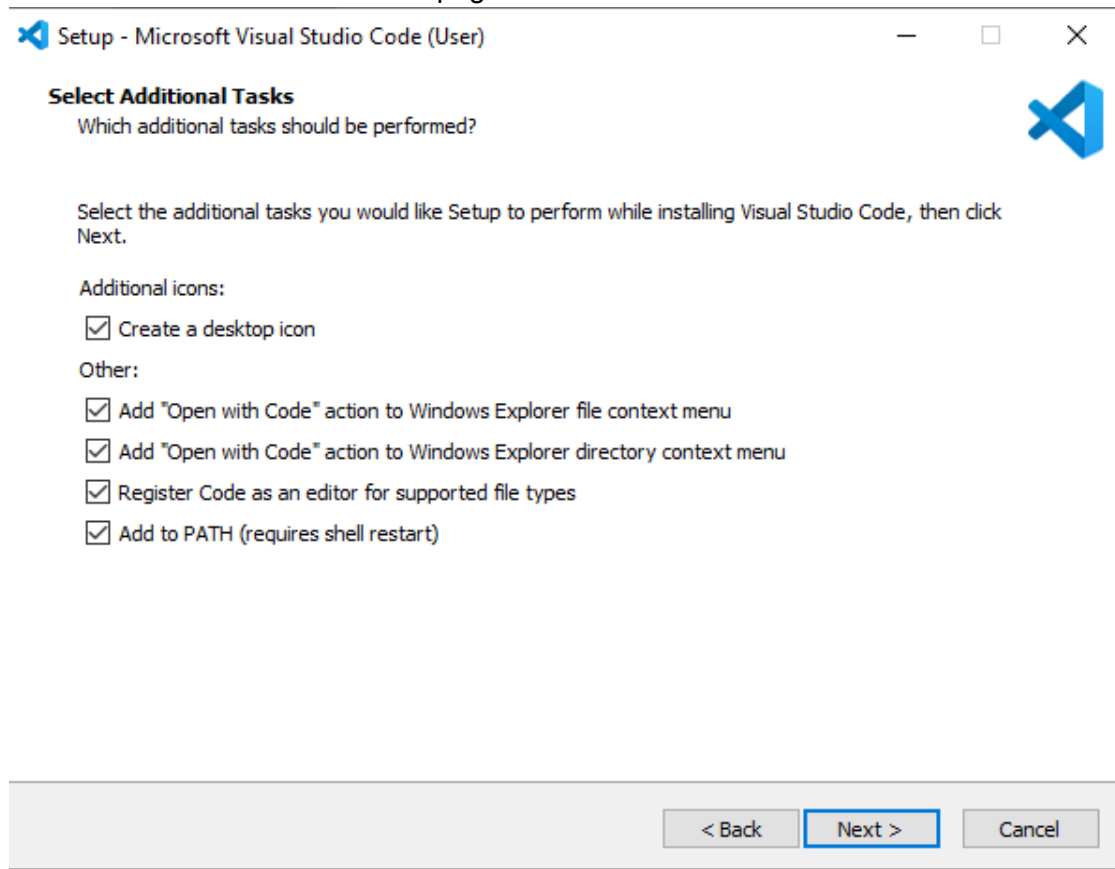
Installing Visual Studio Code in Windows

You will want a text editor with syntax highlighting and a GUI for debugging. Although it is not strictly necessary, it really makes reading and debugging code much easier.

1. Download and install Visual Studio Code from:

- [Visual Studio Code](#)

a. During the installation, make sure all of the checkboxes are selected on the additional tasks page:



Configuring Visual Studio Code to Run C++ in Windows

1. Install the Remote-WSL Extension in VSCode

- [Install Remote-WSL Extension](#)
- When you click on this link, select to open in Visual Studio Code, and then push the install button. You should see "This extension is enabled globally" when you are done

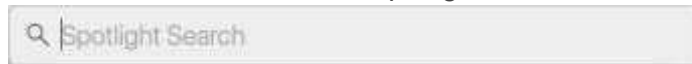
Mac

For the first week only, you can complete this labwork on your own and do not have to attend a lab session to get credit.

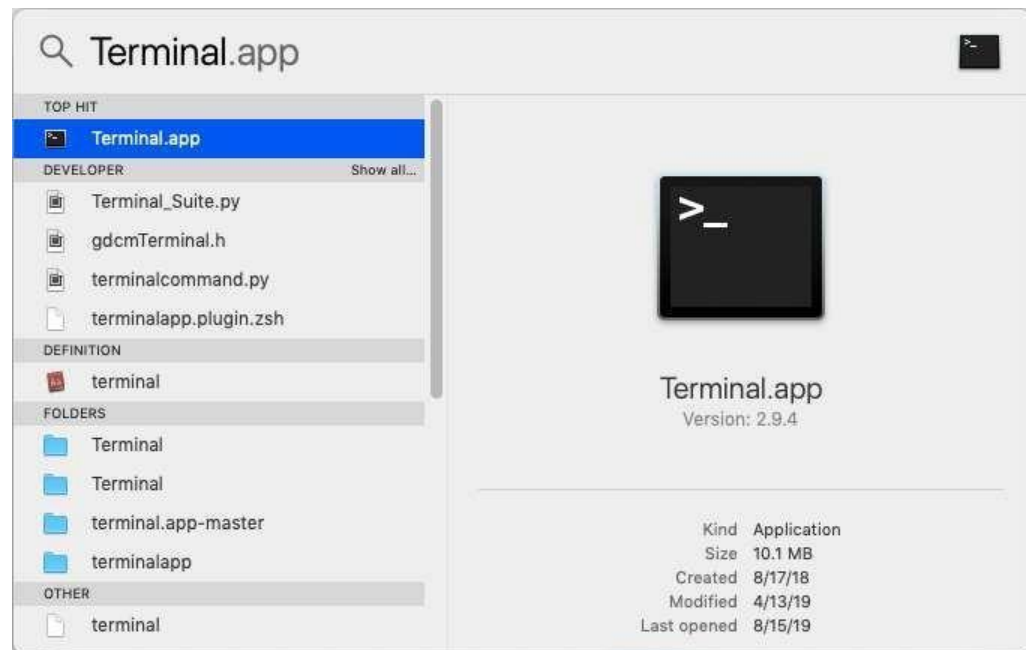
We will be using a common compiling environment for all students in the class, regardless of whether you use macOS, Windows, or a Linux distribution. The way we have decided to do this is to have all students compile in a command-line environment, i.e. the “Terminal” application.

Installing a C++ Compiler in Mac

1. Launch the “Terminal” application. This can be done by the following sequence of steps:
 - a. Press `cmd+space` to launch Spotlight Search.



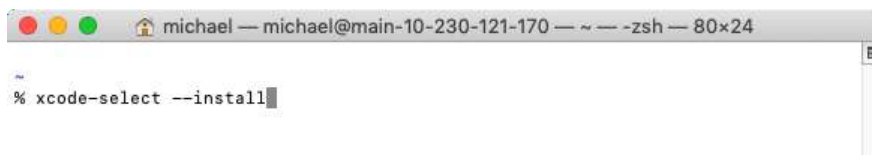
- b. Type `Terminal` in the Spotlight Search field. A list of results should automatically populate.



- c. Click on `Terminal.app` to launch Terminal.



2. In the Terminal application, type `xcode-select --install` and press enter to install the C++ compiler.



Visual Studio Code Installation in Mac

[Download Visual Studio Code.](#)

You will have multiple options. . Select the version that matches the computer architecture of your machine..

Linux

For the first week only, you can complete this labwork on your own and do not have to attend a lab session to get credit.

We will be using a common compiling environment for all students in the class, regardless of whether you use macOS, Windows, or a Linux distribution. The way we have decided to do this is to have all students compile in Visual Studio Code.

Installing a C++ Compiler in Linux

1. Use your package manager to install gcc-g++
 - a. Debian / Ubuntu:

```
sudo apt install build-essential gdb
```

- b. RedHat / CentOS:

```
sudo yum install make automake gcc gcc-c++ kernel-devel gdb
```

c. Fedora:

```
sudo dnf install @development-tools  
sudo dnf group install "C Development Tools and Libraries"
```

d. Arch:

```
sudo pacman -S base-devel
```

e. Other: ask your instructor or TA

- i. Note: We are unlikely to know this outright. But, we will help you figure it out.

Installing Visual Studio Code

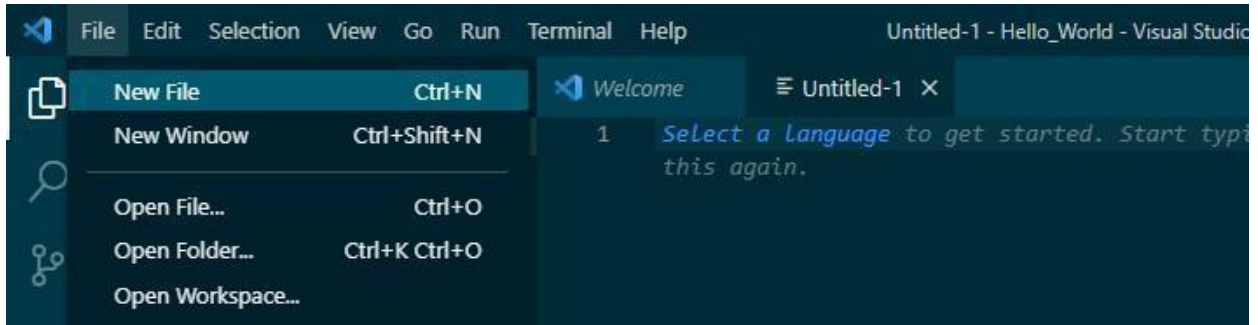
- Download an installer at <https://code.visualstudio.com/Download> for Debian, Ubuntu, Fedora, Red Hat, SUSE.
- For other distributions, try: <https://snapcraft.io/code>

Write, Compile and Run a C++ Program

You should have already completed the setup for your OS¹ before doing this part of the labwork.

1. Now that you've got the compiler installed, let's go through the steps of writing a program that simply prints "Hello, World!" to the standard output.
2. Create an empty folder somewhere to hold your first assignment (e.g. Documents\CSCE121\Lab0).
 - ****IMPORTANT** DO NOT HAVE ANY SPACES OR PUNCTUATION IN YOUR FILENAME/FILEPATH (use only letters, digits and underscores) !!!!!!!**
3. Open this folder with Visual Studio Code (you can right-click on the folder and "Open with Code" or select "Open Folder" from the File menu in Visual Studio Code.
4. Select "File", "New Text File", then "Select a Language"

¹ Operating System



5. Select C++ from the menu
6. Do File/Save As, and type hello.cpp
7. Type the below into the file

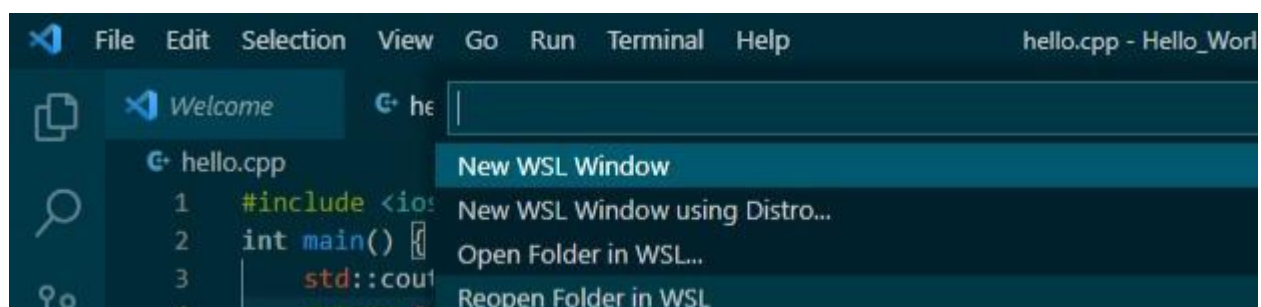
```
#include <iostream>

int main() {
    std::cout << "Hello, World!" << std::endl;
    return 0;
}
```

Note: The autograder wants “Hello, World!” with a capitalized W as opposed to what is listed in the image.

8. Select File/Save
9. (Windows only) We need to make sure we are using the Windows Subsystem for Linux.

Check the bottom left corner. If you see just , click it and pick “Reopen Folder in WSL” (SEE HIGHLIGHTED TEXT BELOW IF OPTION IS NOT AVAILABLE)



If you do not see Reopen folder in WSL "New WSL Window" and navigate to the desired folder using "Open Folder" option NOT the listing in recent. Start your path with `"/mnt/c/"` followed by the rest of the path to your folder. This puts the directory in using the Linux path instead of the Windows path, preventing it from opening your folder without WSL.

Alternatively, type `ctrl+shift+p`. This will open a search bar at the top of the window, in which you can type "WSL". Several options will populate, including:

WSL: Reopen Folder in WSL

WSL: Open Folder in WSL...

Either of these options should allow you to open the desired directory in WSL.

10. (Windows only): You should now see "WSL:Ubuntu" written in the bottom left corner within the green or blue rectangular icon.  OR 

11. Install the C/C++ extension

- Click the extensions icon  or press `Ctrl+Shift+X`
- Search the extensions marketplace for the C/C++ extension
- Click  icon and install the extension. Windows Note: if you already have the extension installed locally, you will need to click the  button followed by Reload Required.
- Be sure that the extension is visible (on Windows, it should be under the

WSL:UBUNTU - INSTALLED menu)



12. Compile and run your program

- Creating a tasks.json file
 - In VSCode, press `Ctrl+Shift+p`
 - Type "Tasks: Open User Tasks" in the search bar and select the choice (if prompted, pick "Other")
 - Copy the text below, paste it in place of the current tasks.json file and save the file

```
{
  "version": "2.0.0",
  "tasks": [
    {
      "type": "cppbuild",
      "label": "C/C++: g++ build active file",
      "command": "/usr/bin/g++",
      "args": [
        "-g",
        "-std=c++17",
        "-Wall",
        "-Wextra",
        "-pedantic-errors",
        "-Wefc++",
        "-Wno-unused-parameter",
        "-fsanitize=undefined,address",
        "${relativeFileDirname}/*.cpp",
        "-o",
        "${fileDirname}/${fileBasenameNoExtension}"
      ],
      "options": {
        "cwd": "${fileDirname}"
      },
      "problemMatcher": [
        "$gcc"
      ],
      "group": {
        "kind": "build",
        "isDefault": true
      },
      "detail": "compiler: /usr/bin/g++"
    },
    {
      "label": "Run",
      "type": "shell",
      "dependsOn": "Build",
      "command": "${fileDirname}/${fileBasenameNoExtension}",
      "args": [],
      "presentation": {
```

```
        "reveal": "always",
        "focus": true
    },
    "problemMatcher": [],
    "group": {
        "kind": "build",
        "isDefault": true
    }
}
]
```

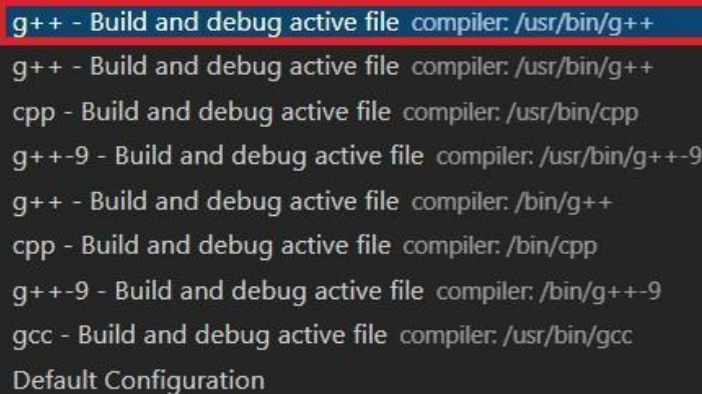
- This enables us to run certain compiler flags without having to create a new task.json file every time we create a new program.

II. Creating and running a build file

- a. With your .cpp file selected, go to: **Terminal > Run Build Task** or press Ctrl + Shift + B to generate an output file. Note: At this point any Compile-Time errors will be displayed in the terminal.
- b. Open a new terminal (Ctrl+Shift+`) and execute the command `./hello`(where hello is the filename) to run the code.

III. Running your program with a debugger

- a. Have your target .cpp file selected in VSCode and select **Run > Start Debugging** or press F5.
- b. The first time you debug, you will be prompted to select an environment. Select **C++ (GDB/LLDB)**
- c. You will also be prompted to select a configuration. Select **g++ build and debug active file** from the dropdown menu.

A screenshot of the VS Code environment selection dropdown menu. The menu is open, showing a list of configurations. The first configuration, 'g++ - Build and debug active file compiler: /usr/bin/g++', is highlighted with a blue background. Below it are several other configurations, including 'g++ - Build and debug active file compiler: /usr/bin/g++', 'cpp - Build and debug active file compiler: /usr/bin/cpp', 'g++-9 - Build and debug active file compiler: /usr/bin/g++-9', 'g++ - Build and debug active file compiler: /bin/g++', 'cpp - Build and debug active file compiler: /bin/cpp', 'g++-9 - Build and debug active file compiler: /bin/g++-9', 'gcc - Build and debug active file compiler: /usr/bin/gcc', and 'Default Configuration'.

```
g++ - Build and debug active file compiler: /usr/bin/g++
g++ - Build and debug active file compiler: /usr/bin/g++
cpp - Build and debug active file compiler: /usr/bin/cpp
g++-9 - Build and debug active file compiler: /usr/bin/g++-9
g++ - Build and debug active file compiler: /bin/g++
cpp - Build and debug active file compiler: /bin/cpp
g++-9 - Build and debug active file compiler: /bin/g++-9
gcc - Build and debug active file compiler: /usr/bin/gcc
Default Configuration
```

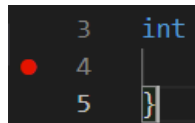
Note: You will receive an error message after you run your code that says: `==11610==LeakSanitizer has encountered a fatal error`. Ignore this message.

IV. Running your program without a debugger

- A. In the “terminal” window, you can type `./hello` and press enter to run the program.

Using GDB to Debug Your program







- Breakpoints
 - Breakpoints are useful debugging tools that allow you to stop the execution of the code at a specific point.
 - To set a breakpoint in VSCode select a line in which you would like to temporarily stop the program and press F9. Alternatively, you can click the margin to the left of this line.
 - If done correctly, you will be able to see a red dot to the left of the line number.



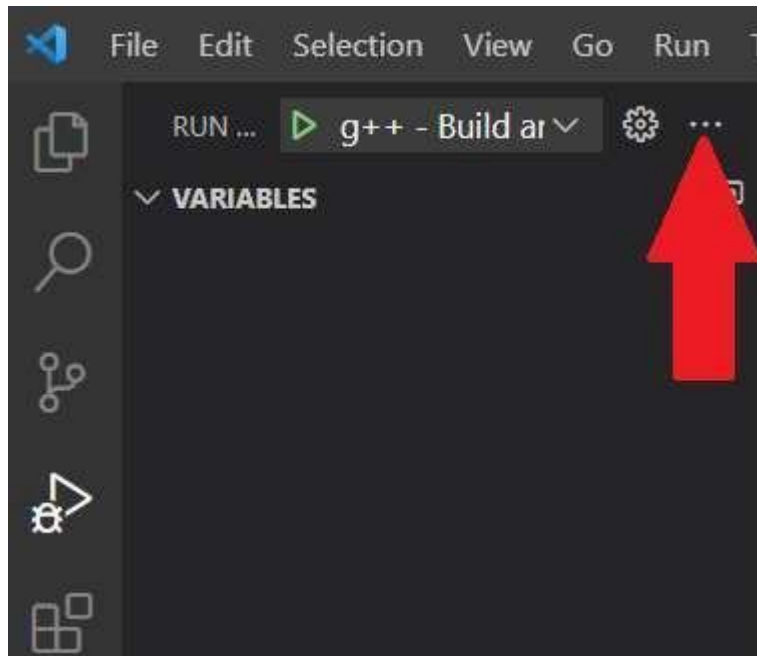
- Once you have your breakpoint set, debug the program select **Run > Start Debugging**.

- Once you start a debugging session, a debugging toolbar will appear.



-  This is the continue button. Pressing it will run the program normally until the next breakpoint is reached.
 -  This is the step over button. Pressing it will advance the program to the next statement.
 -  This is the step into button. This button behaves the same as the step over button except it will step into any method calls, e.g., the debugger enters a called function as opposed to stepping over it.
 -  This is the step out button. Pressing it does the opposite of the step into button, and will return you to where the method was called.
 -  This is the restart button. Pressing it will restart your debugging session.
 -  This is the stop button. Pressing this will terminate the program.

- You will notice several useful windows to the left of the main window.
 - The Variables window allows you to see/edit the variables in the program relative to the current call stack.
 - The Watch window allows you to type in a specific variable and see how the value changes over time. You can also evaluate expressions in the window.
 - The Call Stack window stores information about the order in which the program runs.
 - The Breakpoints window stores the active and inactive breakpoints.
 - You can open any of these windows by clicking the button shown below



- * Helpful GDB → LLDB Commands <https://lldb.lvm.org/use/map.html>

Submitting to Gradescope

- Normally for homework and labworks, you must submit to Gradescope for autograding and feedback.
- For this lab, you do not need to submit to Gradescope for credit, but it is recommended to practice submitting.
- Go to Gradescope and submit hello.cpp.