

# LW: Setup with Hello World

<b>Completion Requirements</b>	<b>1</b>
<b>Setup</b>	<b>1</b>
Windows	1
Installing a Linux environment in Windows	1
Installing a C++ compiler and debugger in Windows	5
Files on your Windows computer	6
Installing Visual Studio Code in Windows	6
Configuring Visual Studio Code to Run C++ in Windows	7
Mac	8
Installing a C++ Compiler in Mac	8
Visual Studio Code Installation in Mac	9
Linux	9
Installing a C++ Compiler in Linux	9
Installing Visual Studio Code	10
<b>Write, Compile and Run a C++ Program</b>	<b>10</b>
<b>Using GDB to Debug Your program</b>	<b>15</b>

**You will do this outside of a lab time to get your computer to compile and run C++ code. 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. Ideally during their office hours, but you can also do it before or after your lab session if they have time.

## Setup

Jump to the instructions for your Operating System:

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

## Windows

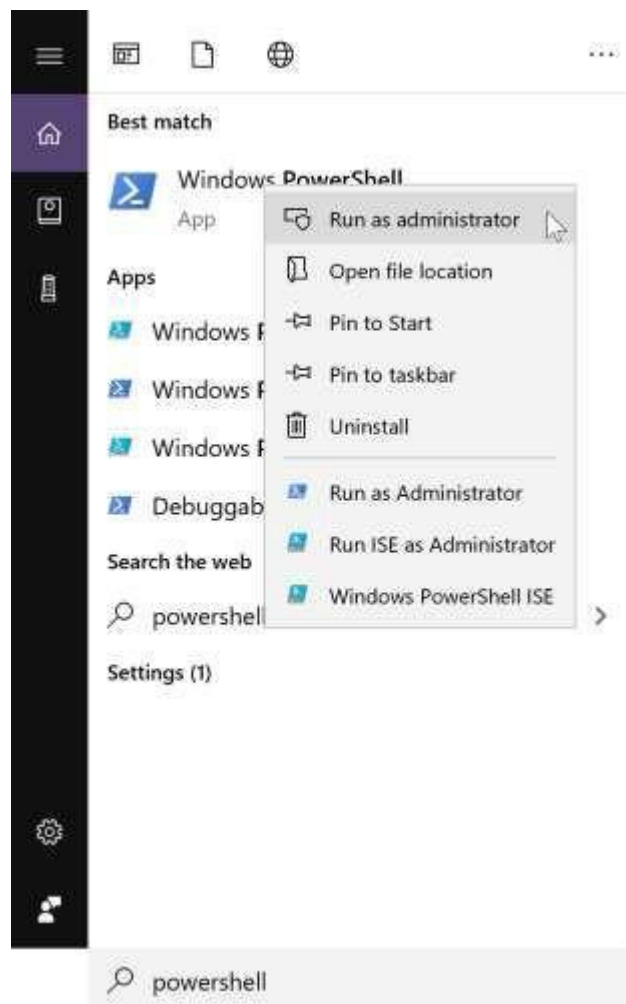
For the first week only, you can complete this labwork on your own and do not have to attend a lab session to get credit.

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”



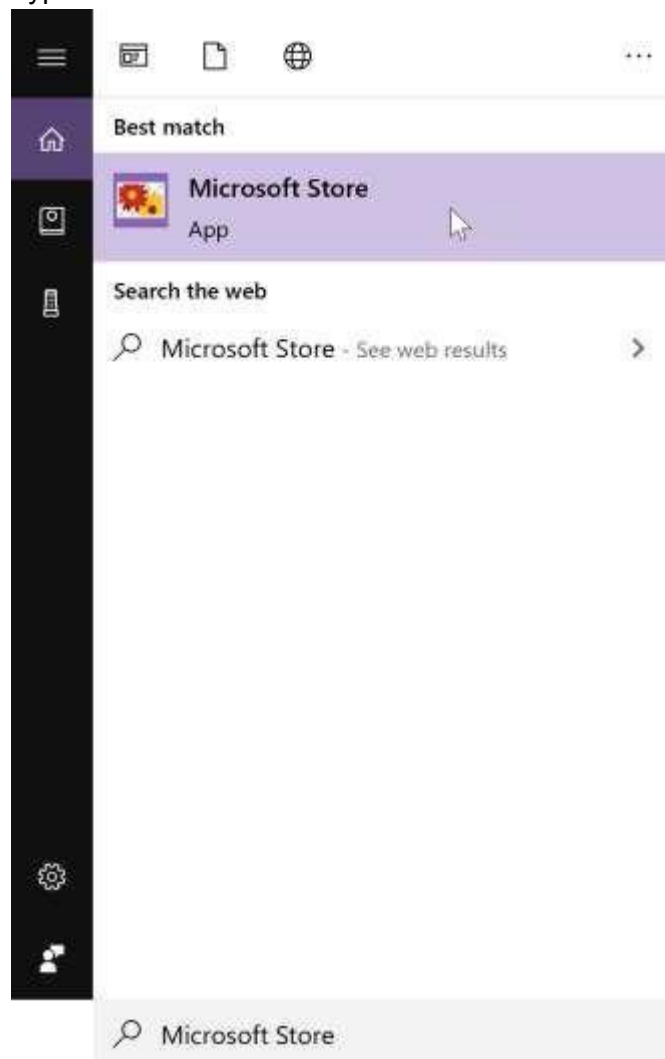
- iv. On the command line run:

```
dism.exe /online /enable-feature
```

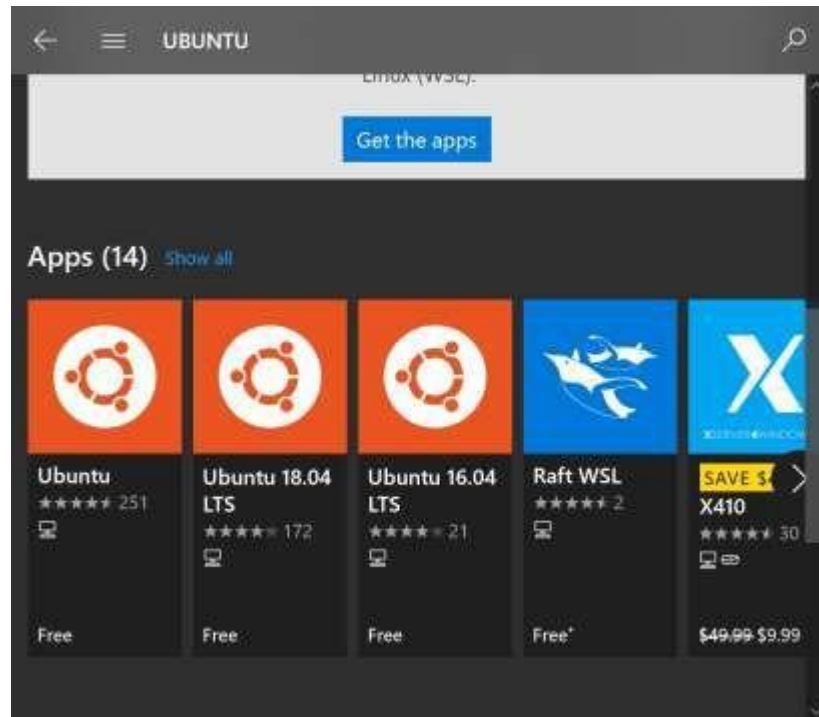
```
/featurename:Microsoft-Windows-Subsystem-Linux /all /norestart
```

\*(if you mis-type above, you do not have to retype from scratch, just hit the up arrow on your keyboard). It is probably easier to copy/paste this command (Ctrl+C, Ctrl+V)

- v. Close Powershell Window
  - vi. Restart your computer (make sure you can get back to this document first!)
3. Install your Linux Distribution of Choice (Ubuntu, if you don't have one yet)
- Most of you will get it from the windows store. If that doesn't work for you, then follow the alternate directions provided and/or talk to a TA or instructor.
- a. Open the Microsoft Store
    - i. Type "Microsoft Store" in the search bar.

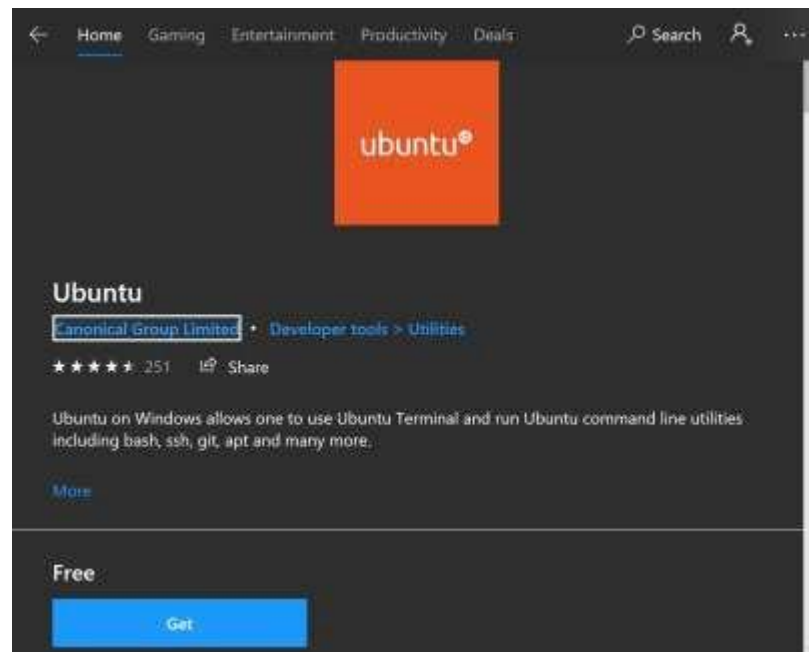


- ii. In the store, search for “Ubuntu”



- iii. Select “Ubuntu”

- iv. From the distro page for the “Canonical Group Limited”, select “Get”

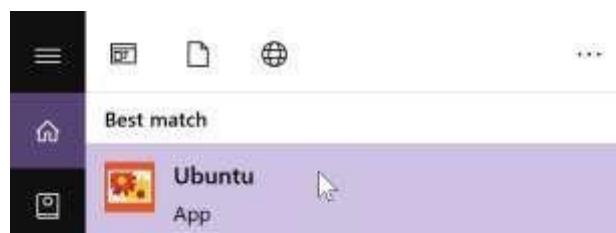


You can launch by selecting “Launch” from the Microsoft Store.

- a. You can type Ubuntu into the search bar and select Ubuntu from there.

4. Initialize

5. Set up a new Linux user account. You can use



any username you want.  
Make sure the password is  
something that you can  
remember since you'll need it  
when you do anything which  
requires super-user privileges,  
like installing and updating  
software .

Note: You will not see characters being typed  
when you enter the password. Just know  
that it is taking whatever you type, so be  
careful.

```
Installing, this may take a few minutes...
Please create a default UNIX user account. The username does not need to match your Windows username.
For more information visit: https://aka.ms/wslusers
Enter new UNIX username: jmichael
Enter new UNIX password: dr. ritchey is the best
Retype new UNIX password: dr. ritchey is the best
passwd: password updated successfully Installation
successful!
To run a command as administrator (user "root"), use "sudo <command>".
See "man sudo_root" for details.

jmichael@WSL: ~$
```

6. Update & upgrade your packages  
In your Ubuntu shell type the following:

```
sudo apt update && sudo apt upgrade
```

`sudo` is the name of the program which lets you *do* something as the *super-user*. `apt` is Ubuntu's package manager, a program that lets you install, update, and remove software. `update` tells `apt` to get the list of available updates. `upgrade` tells `apt` to actually perform the update

Provide the password you used to set up.

Note: If you have something on your clipboard, you can paste it in a command line by right-clicking.

Eventually, it will ask if you want to continue. Type 'Y' or 'y'.

The updates can take some time.

When it asks about restarting services, choose 'Yes'.

## *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 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.

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

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

3. Verify you installed gcc by executing

```
gcc --version
```

The output should be similar to this (image below last updated 8/16/2021):

```
gcc (Ubuntu 9.3.0-17ubuntu1~20.04)
9.3.0 Copyright (C) 2019 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:

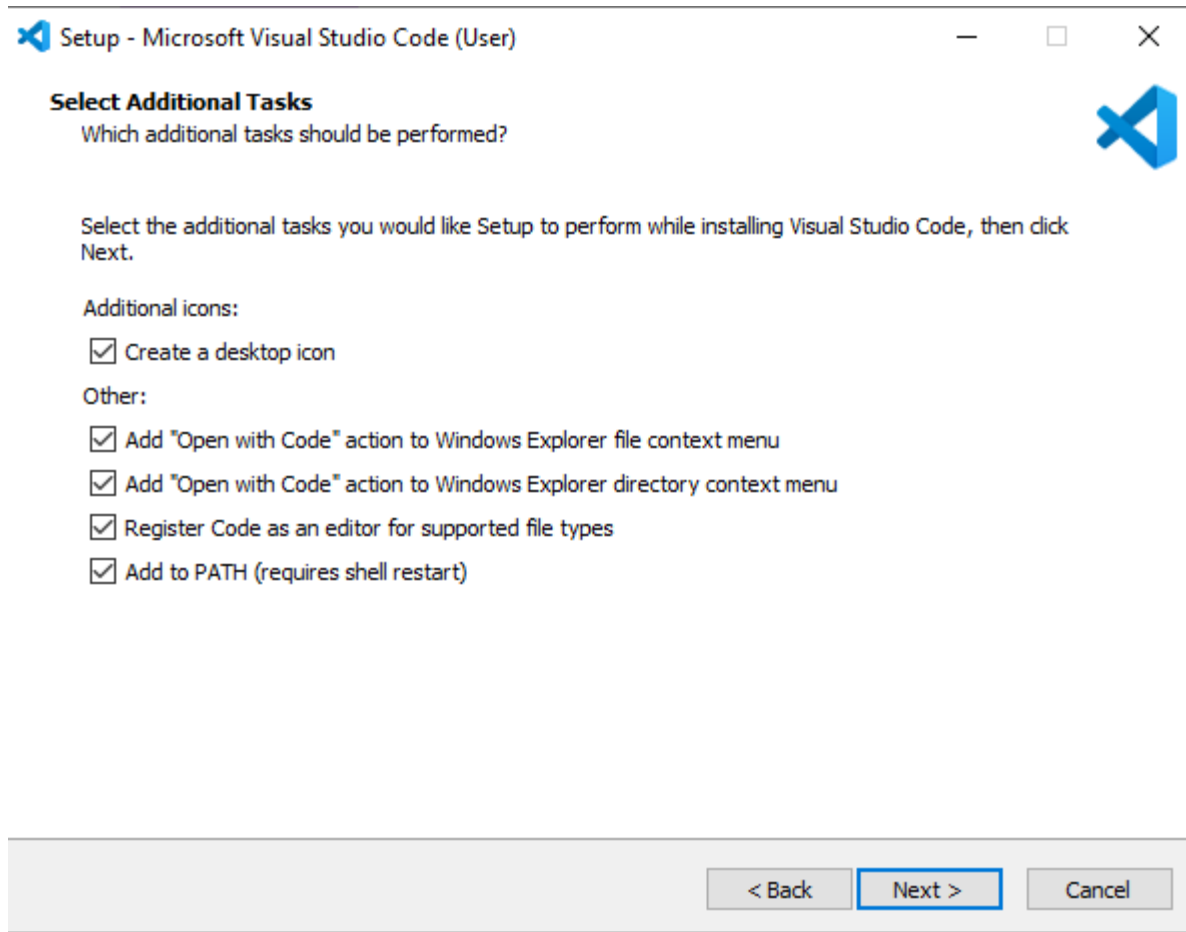
```
df
```

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.



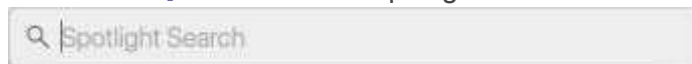
## Mac

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.

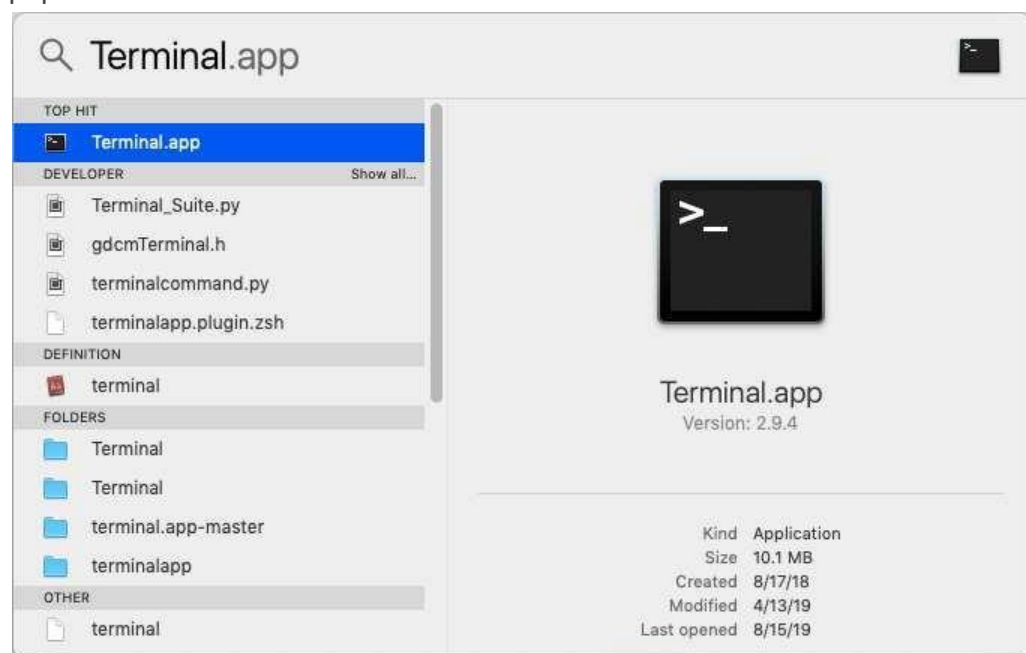
We will be using a common compiling environment for all students in the class, regardless of whether you use macOS, Windows, or a Linux distribution. The way we have decided to do this is to have all students compile in a command-line environment, i.e. the “Terminal” application.

### *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.



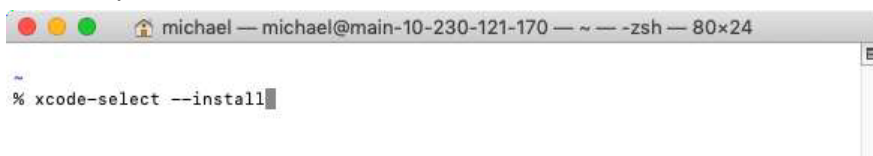
- b. Type `Terminal` in the Spotlight Search field. A list of results should automatically populate.



- 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 @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>

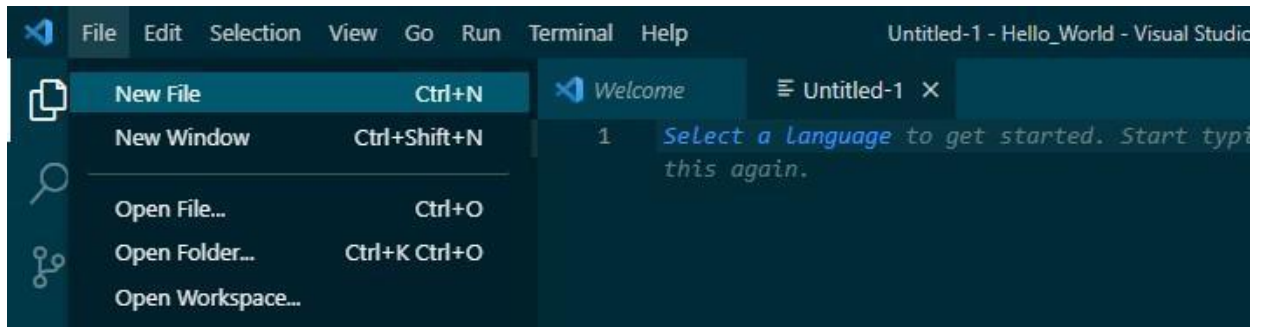
## Write, Compile and Run a C++ Program

You should have already completed the setup for your OS<sup>1</sup> 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 File", then "Select a Language"


---

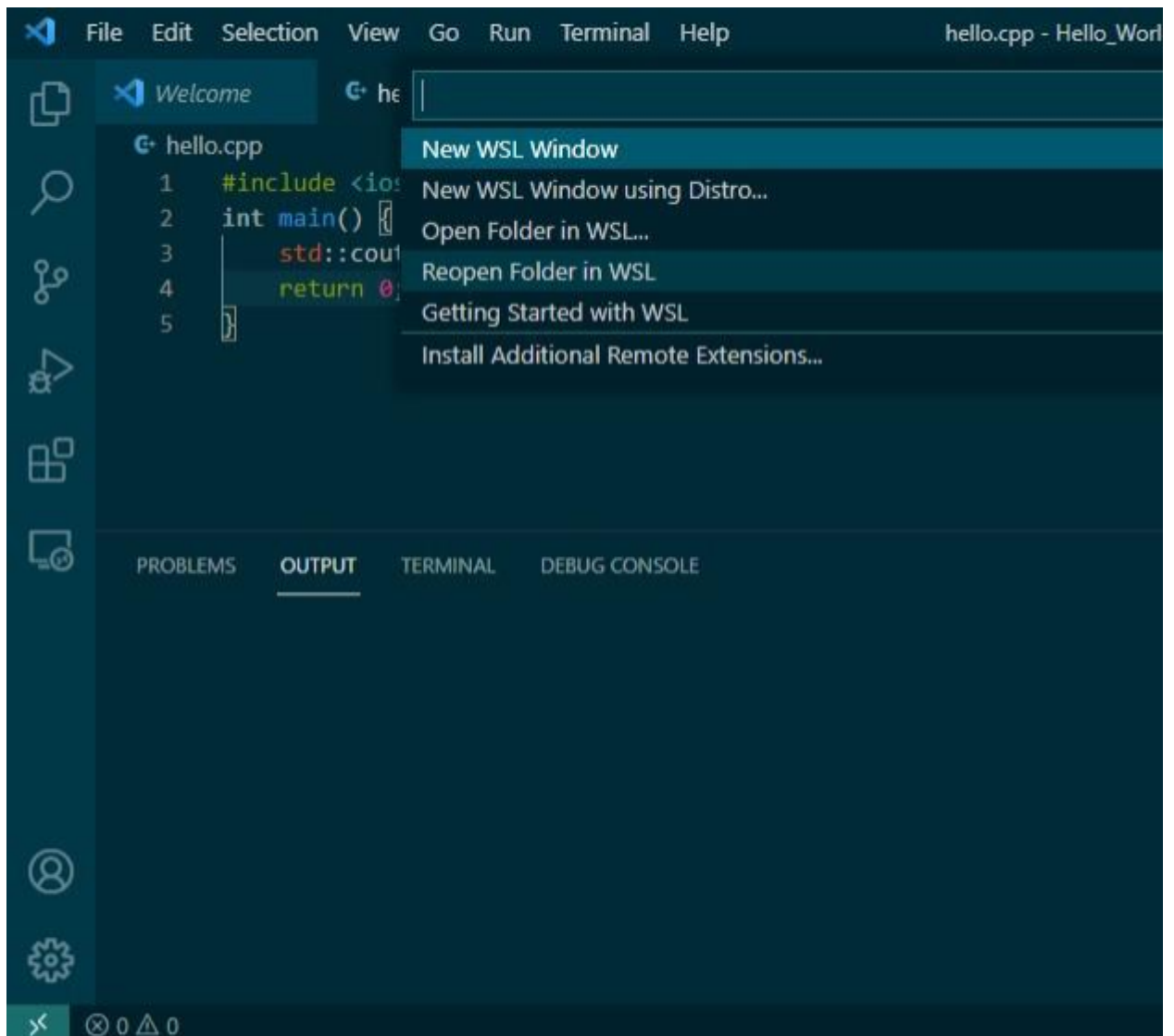
<sup>1</sup> Operating System



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

```
1  #include <iostream>
2
3  int main() {
4      std::cout << "Hello, world!" << std::endl;
5      return 0;
6  }
```

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"



10. (Windows only): You should now see

WSL: Ubuntu

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 **Install in WSL: Ubuntu** 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

- a. In VSCode, press Ctrl+Shift+p
- b. Type "Tasks: Open User Tasks" in the search bar and select the choice (on Mac pick "Other")
- c. Replace the current tasks.json file with this one 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",
        "-Wffc++",
        "-Wno-unused-parameter",
        "-fsanitize=undefined,address",
        "${fileDirname}/*.cpp",
        "-o",
        "${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

- 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.
- 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

- Have your target .cpp file selected in VSCode and select **Run > Start Debugging** or press F5.
- 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.

```
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 might receive an error message after you run your code that says: `==11610==LeakSanitizer has encountered a fatal error`. Ignore this message if you get it.

#### 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.









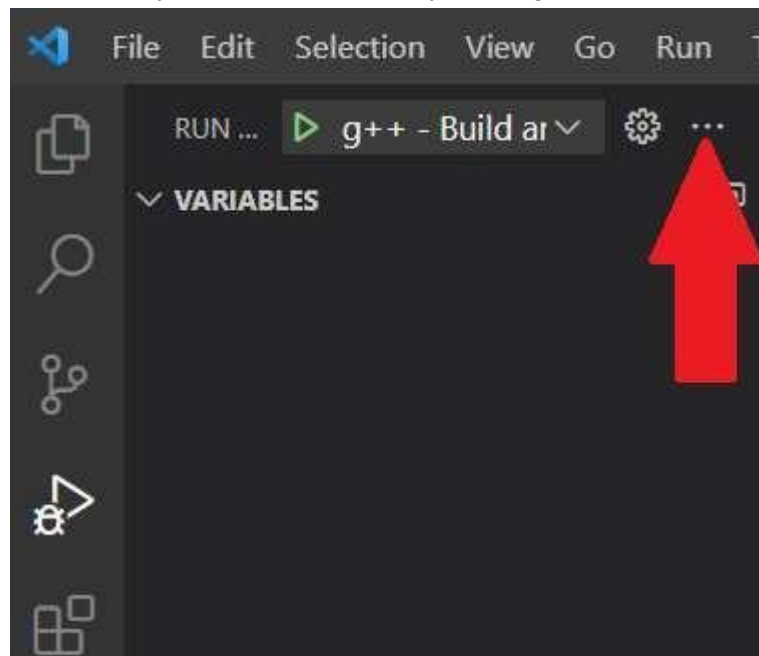
- Once you have your breakpoint set, debug the program select **Run > Start Debugging**.

- 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`.