# LW: Setup with Hello World

[**Completion Requirements**](https://docs.google.com/document/d/11v-mAAKQKkFWGToYMPZRRo8ct2kf_TeF/edit#heading=h.1hmsyys)[**1**](https://docs.google.com/document/d/11v-mAAKQKkFWGToYMPZRRo8ct2kf_TeF/edit#heading=h.1hmsyys)

[**Setup**](#_2et92p0) **1**

[Windows](#_3dy6vkm) [1](#_3dy6vkm)

[Installing a Linux environment in Windows](#_4d34og8) [1](#_4d34og8)

[Installing a C++ compiler and debugger in Windows](#_17dp8vu) [5](#_17dp8vu)

[Files on your Windows computer](#_26in1rg) [6](#_26in1rg)

[Installing Visual Studio Code in Windows](#_35nkun2) [6](#_35nkun2)

[Configuring Visual Studio Code to Run C++ in Windows](#_44sinio) [7](#_44sinio)

[Mac](#_z337ya) [8](#_z337ya)

[Installing a C++ Compiler in Mac](#_1y810tw) [8](#_1y810tw)

[Visual Studio Code Installation in Mac](#_2xcytpi) [9](#_2xcytpi)

[Linux](#_3whwml4) [9](#_3whwml4)

[Installing a C++ Compiler in Linux](#_qsh70q) [9](#_qsh70q)

[Installing Visual Studio Code](#_1pxezwc) [10](#_1pxezwc)

[**Write, Compile and Run a C++ Program**](#_2p2csry) **10**

[**Using GDB to Debug Your program**](#_ihv636) **15**

**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](#_3dy6vkm)
* Mac
* Linux

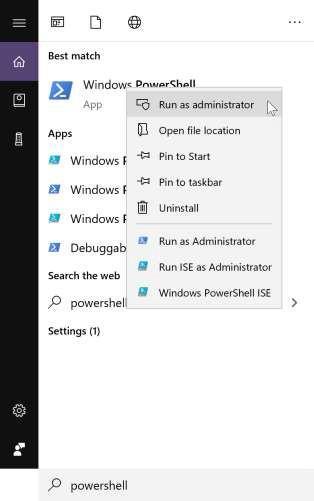
## **Windows**

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](https://docs.microsoft.com/en-us/windows/wsl/install-win10):

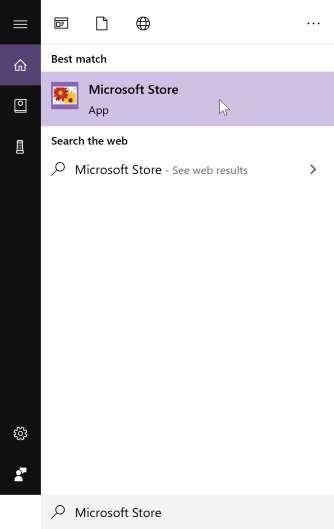
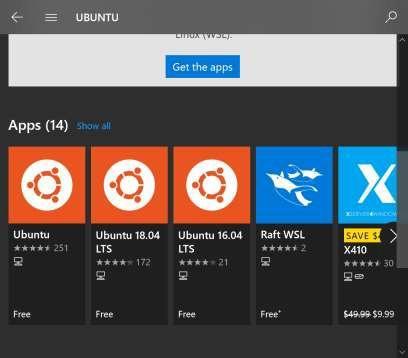
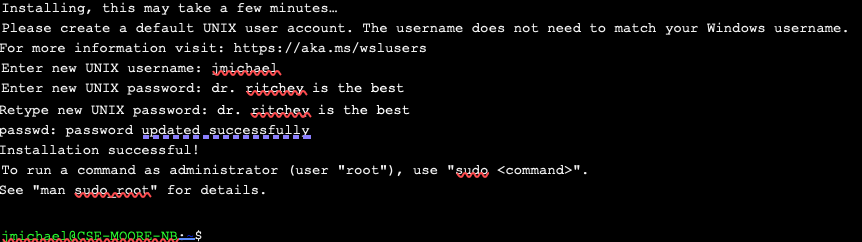
1. It would be prudent to bookmark this Google doc since you will have to reboot your system
2. Install the Windows Subsystem for Linux
   1. Open Powershell as Administrator
      1. Type “powershell” into the search bar
      2. Right-click on “Windows PowerShell”
      3. Select “Run as administrator”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

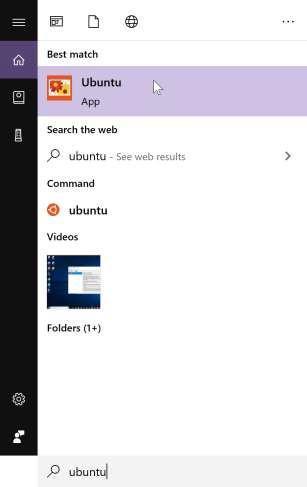
(NOTE) This only works if WSL is not already installed. If you run wsl –install and see WSL help text, then try running wsl --list --online to see a list of available distros and run wsl --install -d <DistroName> to install a distro

(NOTE) To uninstall WSL, see [**Uninstall legacy version of WSL**](https://learn.microsoft.com/en-us/windows/wsl/troubleshooting#uninstall-legacy-version-of-wsl) or [**unregister or uninstall a Linux distribution**](https://learn.microsoft.com/en-us/windows/wsl/basic-commands#unregister-or-uninstall-a-linux-distribution).

1. 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.

* 1. Open the Microsoft Store
     1. Type “Microsoft Store” in the search bar.
     2. In the store, search for “Ubuntu”
     3. Select “Ubuntu”
     4. From the distro page for the “Canonical Group Limited”, select “Get”
     5. You can launch by selecting “Launch” from the Microsoft Store.



1. Initialize
   * + 1. You can type Ubuntu into the search bar and select Ubuntu from there.
2. 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.

1. Update & upgrade your packages

In your Ubuntu shell type the following:



sudo is the name of the program which lets you *do* something as the *s*uper-*u*ser. 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:



Type ‘y’ when it asks if you want to continue. If this window appears to hang with “Processing triggers”, push enter.

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



1. Verify you installed gcc by executing



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



### *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:

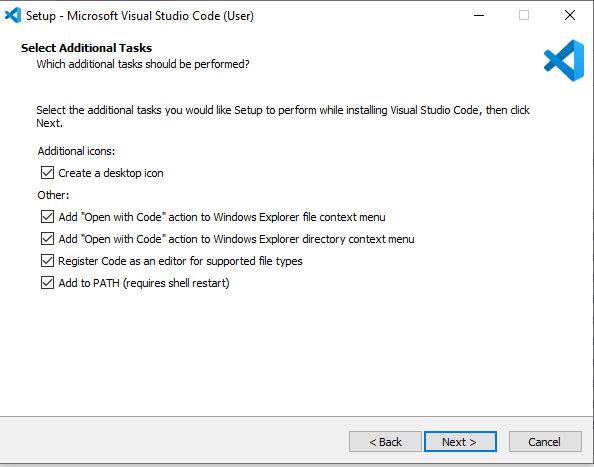


You’ll see /mnt/c. This leads to the C:/ drive on your computer. This is useful for the Unix Tutorial.

### *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](https://code.visualstudio.com/docs/?dv=win)

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](https://marketplace.visualstudio.com/items?itemName=ms-vscode-remote.remote-wsl)
   * 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

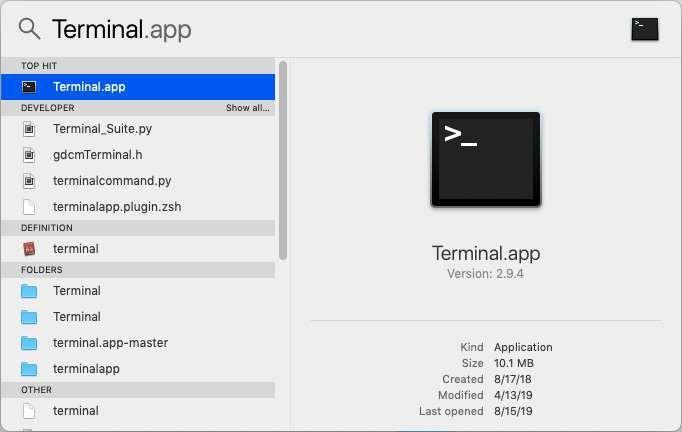
## **Mac**

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.

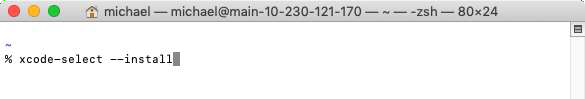
### *Installing a C++ Compiler in Mac*

1. Launch the “Terminal” application. This can be done by the following sequence of steps:
   1. Press cmd+space to launch Spotlight Search.shows spotlight search bar
   2. Type Terminal in the Spotlight Search field. A list of results should automatically populate.



* 1. Click on Terminal.app to launch Terminal.

1. 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](https://code.visualstudio.com/download).

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++
   1. Debian / Ubuntu:

sudo apt install build-essential gdb

* 1. RedHat / CentOS:

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

* 1. Fedora:

sudo dnf install @development-tools

sudo dnf group install "C Development Tools and Libraries"

* 1. Arch:

sudo pacman -S base-devel

* 1. Other: ask your instructor or TA
     1. 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>

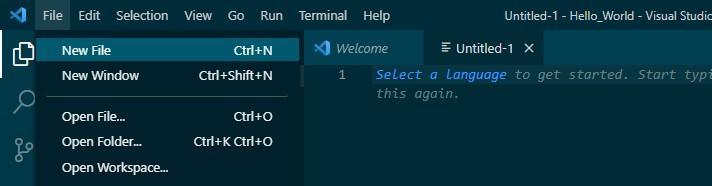
# **Write, Compile and Run a C++ Program**

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

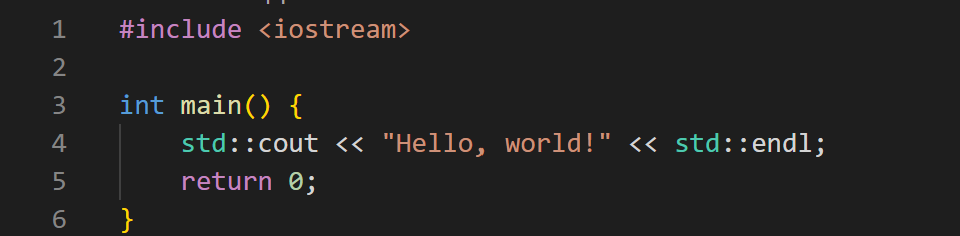
1. Now that you’ve got the compiler installed, let’s go through the steps of writing a program that simply prints “Hello, World!” to the standard output.
2. Create an empty folder somewhere to hold your first assignment (e.g. Documents\CSCE121\Lab0). **\*\*IMPORTANT\*\* DO NOT HAVE ANY SPACES OR PUNCTUATION 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 File”, then “Select a Language”



1 Operating System



1. Select C++ from the menu
2. Do File/Save As, and type hello.cpp
3. Type the below into the file



1. Select File/Save
2. (Windows only) We need to make sure we are using the Windows Subsystem for Linux. Check the bottom left corner. If you see just Shows icon from bottom left corner of VS Code with ><, indicating WSL not selected, click it and pick “Reopen Folder in WSL”



1. (Windows only): You should now see
2. Install the C/C++ extension
   * Click the extensions icon extensions iconor press Ctrl+Shift+X
   * Search the extensions marketplace for the C/C++ extension
   * Click C/C++ extension iconicon 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 button followed by Reload Required.
   * Be sure that the extension is visible (on Windows, it should be under the WSL:UBUNTU - INSTALLED menu) C/C++ extension icon
3. Compile and run your program
4. Creating a tasks.json file
   1. In VSCode, press Ctrl+Shift+p
   2. Type “Tasks: Open User Tasks” in the search bar and select the choice (on Mac pick “Other”)
   3. 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",

"-Weffc++",

"-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

}

}

]

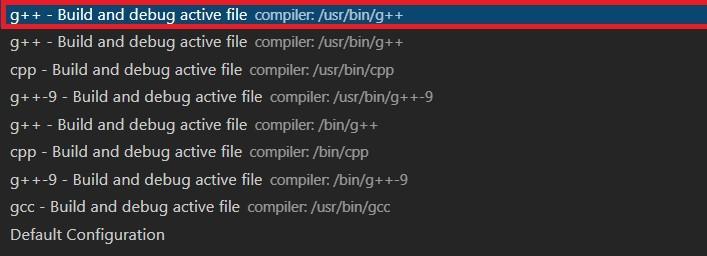
}

* + - This enables us to run certain compiler flags without having

to create a new task.json file every time we create a new program.

1. Creating and running a build file
   1. 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.
   2. Open a new terminal (Ctrl+Shift+`) and execute the command

./hello(where hello is the filename) to run the code.

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

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

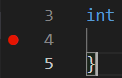
1. Running your program without a debugger

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.debugging toolbar
  + This is the continue button. Pressing it will run the program normally until the next breakpoint is reached.right pointing triangle

➢ dot with curved arrow pointing to the right above itThis is the step over button. Pressing it will advance the program to the next statement.

➢ dot with arrow pointing down into it 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.

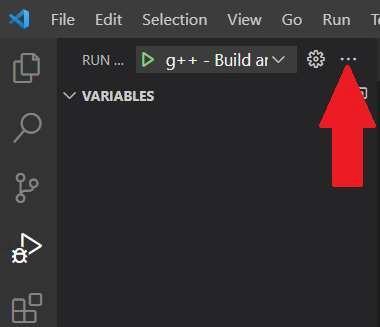
➢ dot with arrow pointing up from itThis is the step out button. Pressing it does the opposite of the step into button, and will return you to where the method was called.

➢ curved arrow going counter-clockwiseThis is the restart button. Pressing it will restart your debugging session.

➢ red rectangleThis 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 any of these windows by clicking the button show below
* \* Helpful GDB → LLDB Commands <https://lldb.llvm.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.