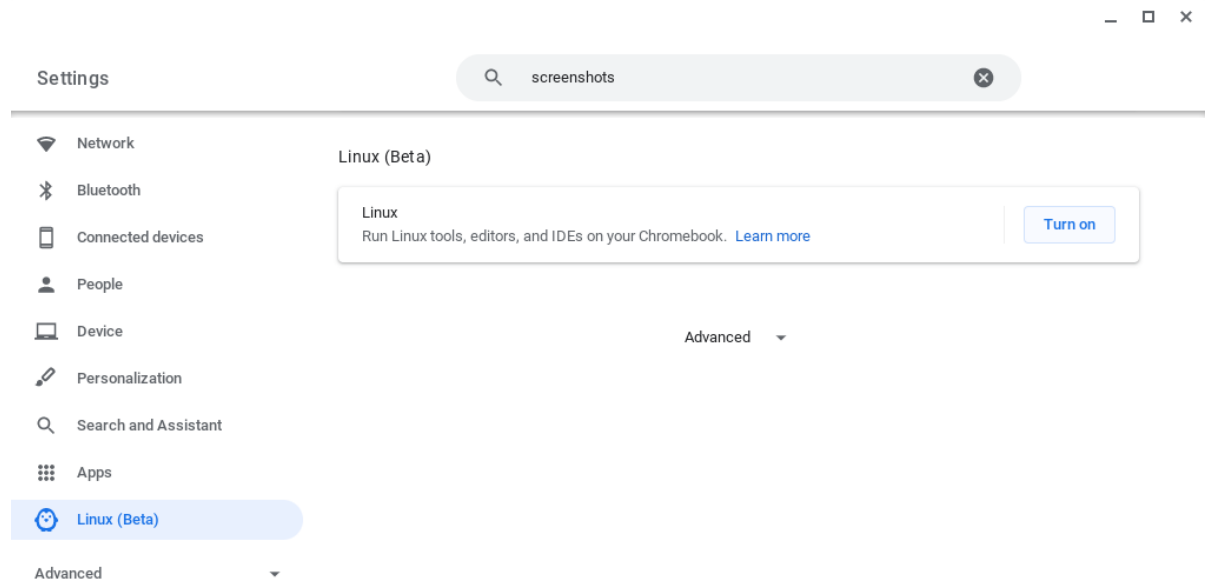


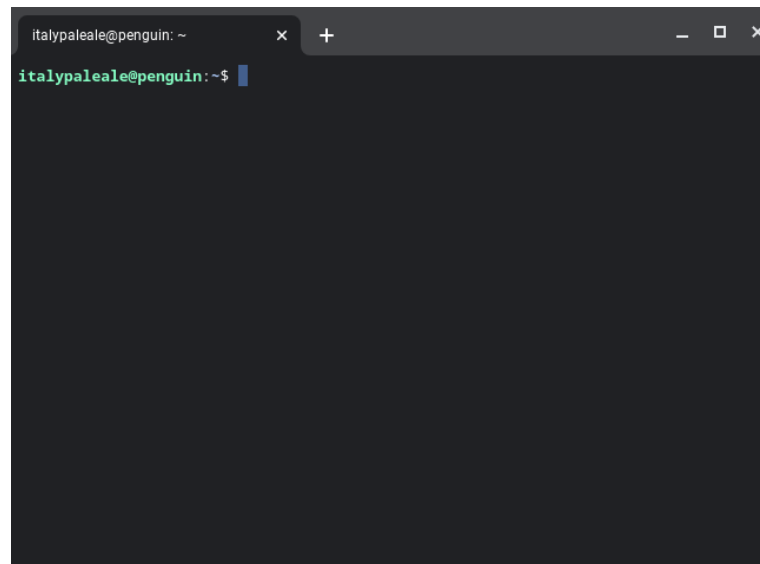
Chromebook VS Code Setup Guide:

Step1: Enable Linux on your Chromebook

- Open the system's Settings.
- Look for **Linux (Beta)** on the sidebar.



- Click on **Turn on**.
- Follow the instructions on screen to configure the Linux environment (accepting the default values should be enough).
- Your Chromebook will then download the tools to create the Linux environment and configure it for you.
- Once the Linux environment has been set up, you'll see a new terminal window popping up.



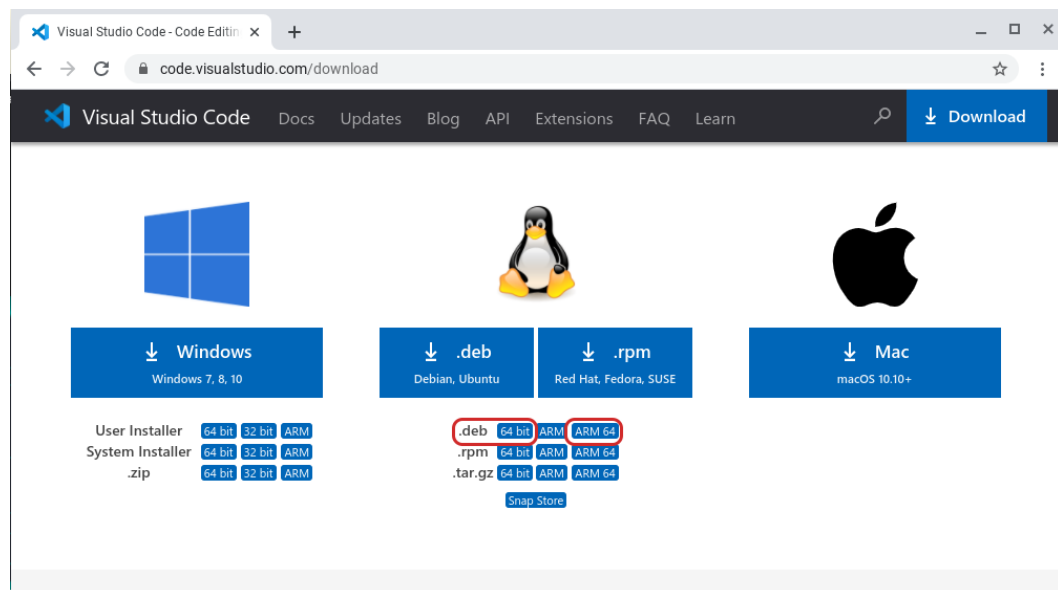
Step2: Setting up Linux

- In the terminal window, type the following commands and press enter.
 - `sudo apt-get update`
 - `sudo apt-get install -y gnome-keyring`
- Output will be similar to the following.

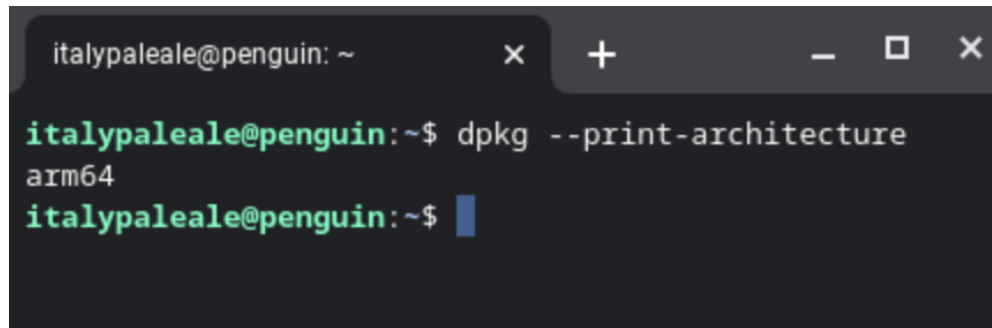
```
italypaleale@penguin: ~  
italypaleale@penguin:~$ sudo apt-get update  
Hit:1 https://deb.debian.org/debian buster InRelease  
Ign:2 https://storage.googleapis.com/cros-packages/86 buster InRelease  
Hit:3 https://deb.debian.org/debian-security buster/updates InRelease  
Hit:4 https://storage.googleapis.com/cros-packages/86 buster Release  
Reading package lists... Done  
italypaleale@penguin:~$ sudo apt-get install -y gnome-keyring  
Reading package lists... Done  
Building dependency tree  
Reading state information... Done  
The following additional packages will be installed:  
  gcr gnome-keyring-pkcs11 libgck-1-0 libgcr-base-3-1 libgcr-ui-3-1 libpam-gnome-keyring libsecret-1-0  
  libsecret-common p11-kit p11-kit-modules pinentry-gnome3  
Suggested packages:  
  pinentry-doc  
The following NEW packages will be installed:  
  gcr gnome-keyring gnome-keyring-pkcs11 libgck-1-0 libgcr-base-3-1 libgcr-ui-3-1 libpam-gnome-keyring  
  libsecret-1-0 libsecret-common p11-kit p11-kit-modules pinentry-gnome3  
0 upgraded, 12 newly installed, 0 to remove and 41 not upgraded.  
Need to get 2,895 kB of archives.  
After this operation, 11.8 MB of additional disk space will be used.  
Get:1 https://deb.debian.org/debian buster/main arm64 libgck-1-0 arm64 3.28.1-1 [81.4 kB]  
Get:2 https://deb.debian.org/debian buster/main arm64 libgcr-base-3-1 arm64 3.28.1-1 [190 kB]  
Get:3 https://deb.debian.org/debian buster/main arm64 libgcr-ui-3-1 arm64 3.28.1-1 [147 kB]  
Get:4 https://deb.debian.org/debian buster/main arm64 gcr arm64 3.28.1-1 [244 kB]  
Get:5 https://deb.debian.org/debian buster/main arm64 p11-kit-modules arm64 0.23.15-2 [208 kB]  
Get:6 https://deb.debian.org/debian buster/main arm64 p11-kit arm64 0.23.15-2 [266 kB]  
Get:7 https://deb.debian.org/debian buster/main arm64 libsecret-common all 0.18.7-1 [25.7 kB]  
Get:8 https://deb.debian.org/debian buster/main arm64 libsecret-1-0 arm64 0.18.7-1 [93.9 kB]  
Get:9 https://deb.debian.org/debian buster/main arm64 pinentry-gnome3 arm64 1.1.0-2 [64.8 kB]
```

Step3: Install VS Code

- Go to the Visual Studio Code [Download](#) page. From there, you need to pick the right package for your Chromebook:
 - For Chromebooks running an Intel or AMD chip, pick the **.deb** in variant **64 bit**.
 - If your Chromebook is running on an ARM64 chip (like the one I'm testing with), pick the **.deb** package in the variant **ARM64** instead.

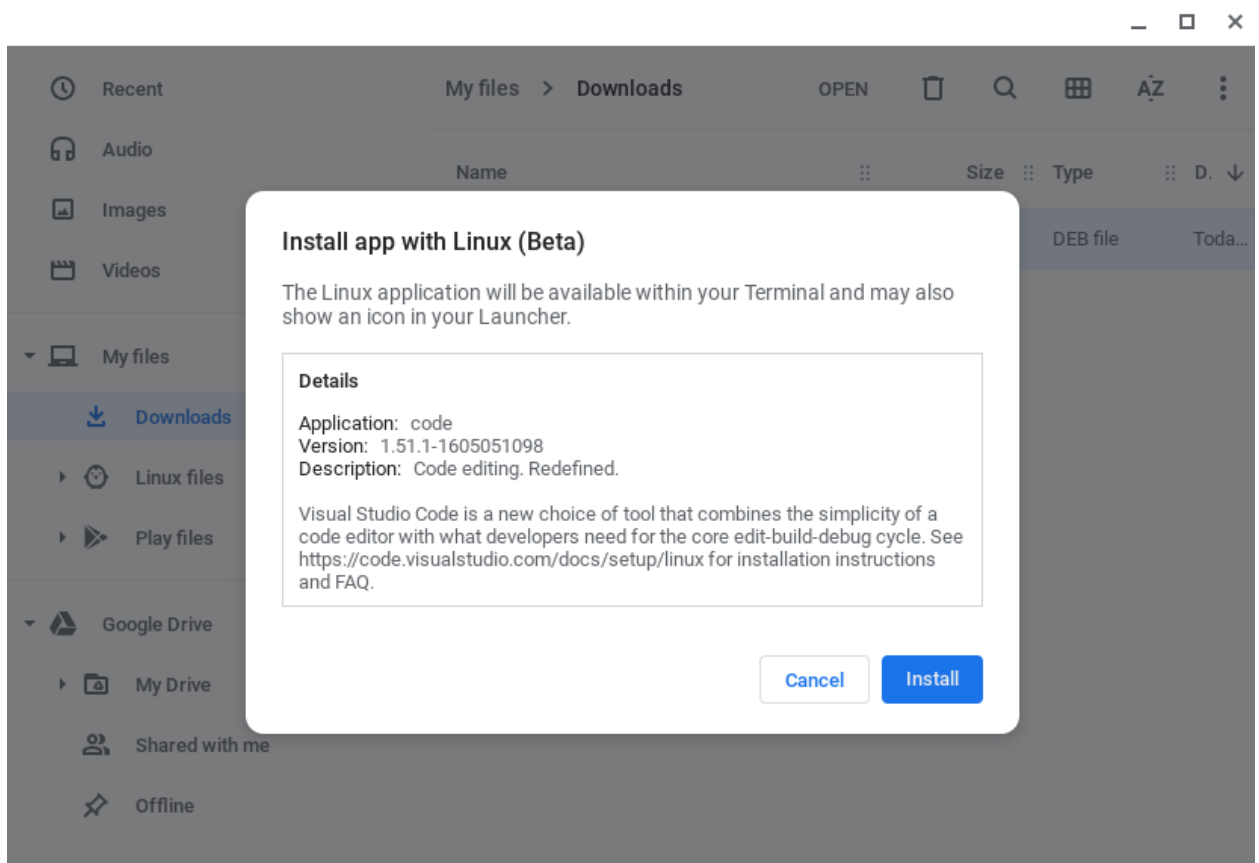


- If you're unsure what kind of chip your Chromebook is using, run the command `dpkg --print-architecture` in the Linux terminal to find out. You'll see either `amd64` (for both Intel and AMD chips: pick the 64 bit variant for VS Code) or `arm64` (pick ARM64).

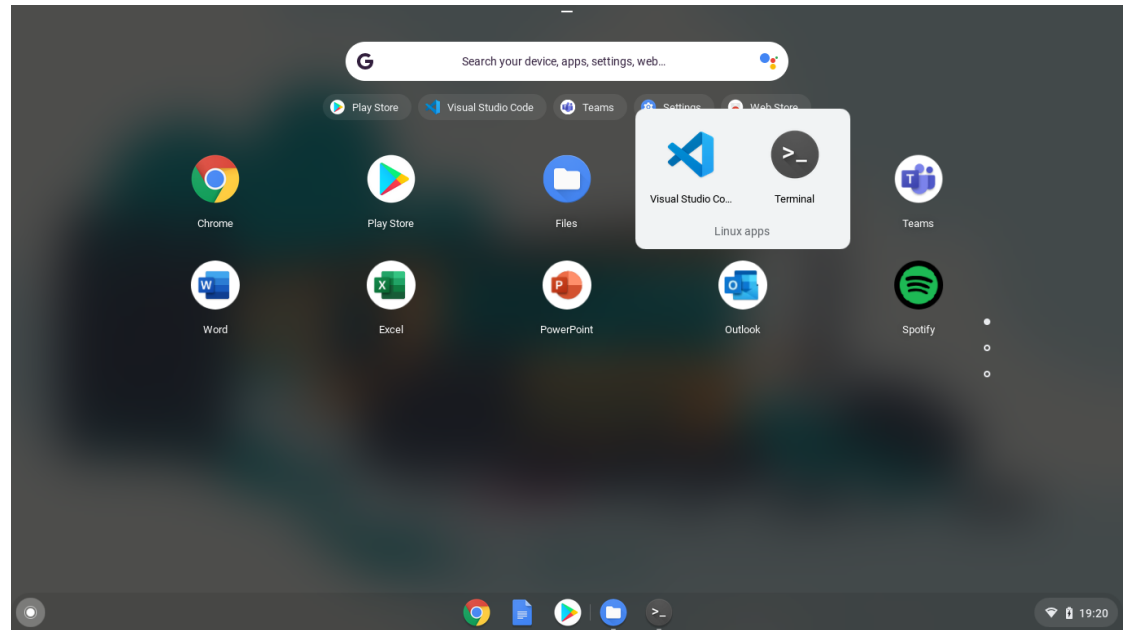


```
italypaleale@penguin: ~  
italypaleale@penguin:~$ dpkg --print-architecture  
arm64  
italypaleale@penguin:~$
```

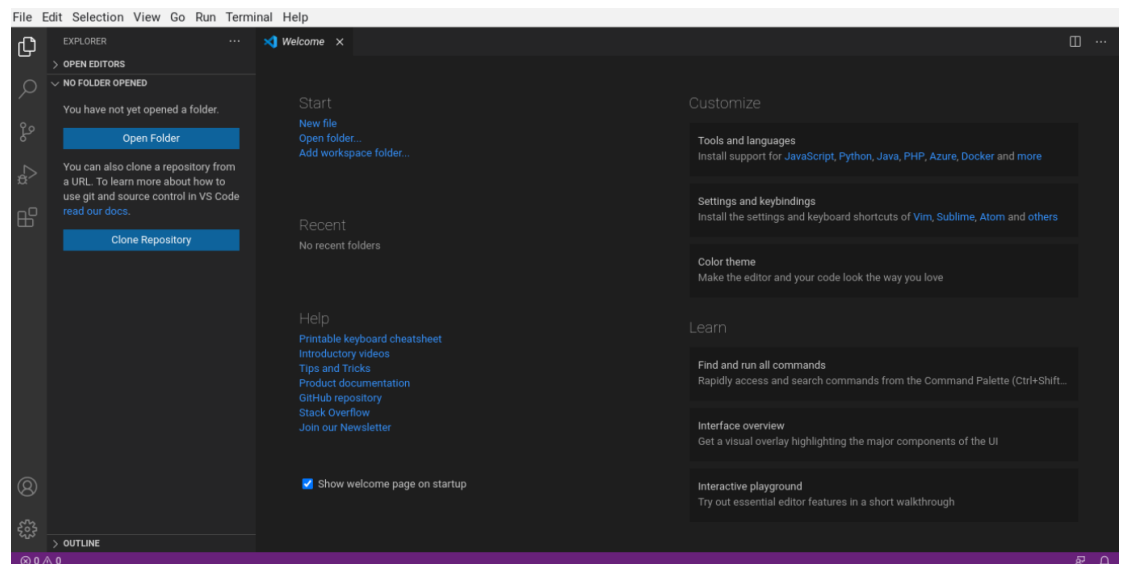
- After you've downloaded VS Code, you'll find the package in your Downloads folder. Double-click on the package to launch the installer and click Install. Your Chromebook will then install VS Code and all other dependencies.



- After the installation is complete, in your list of apps, you'll find Visual Studio Code inside a folder called Linux apps (along with the Linux terminal). You can now launch VS Code.

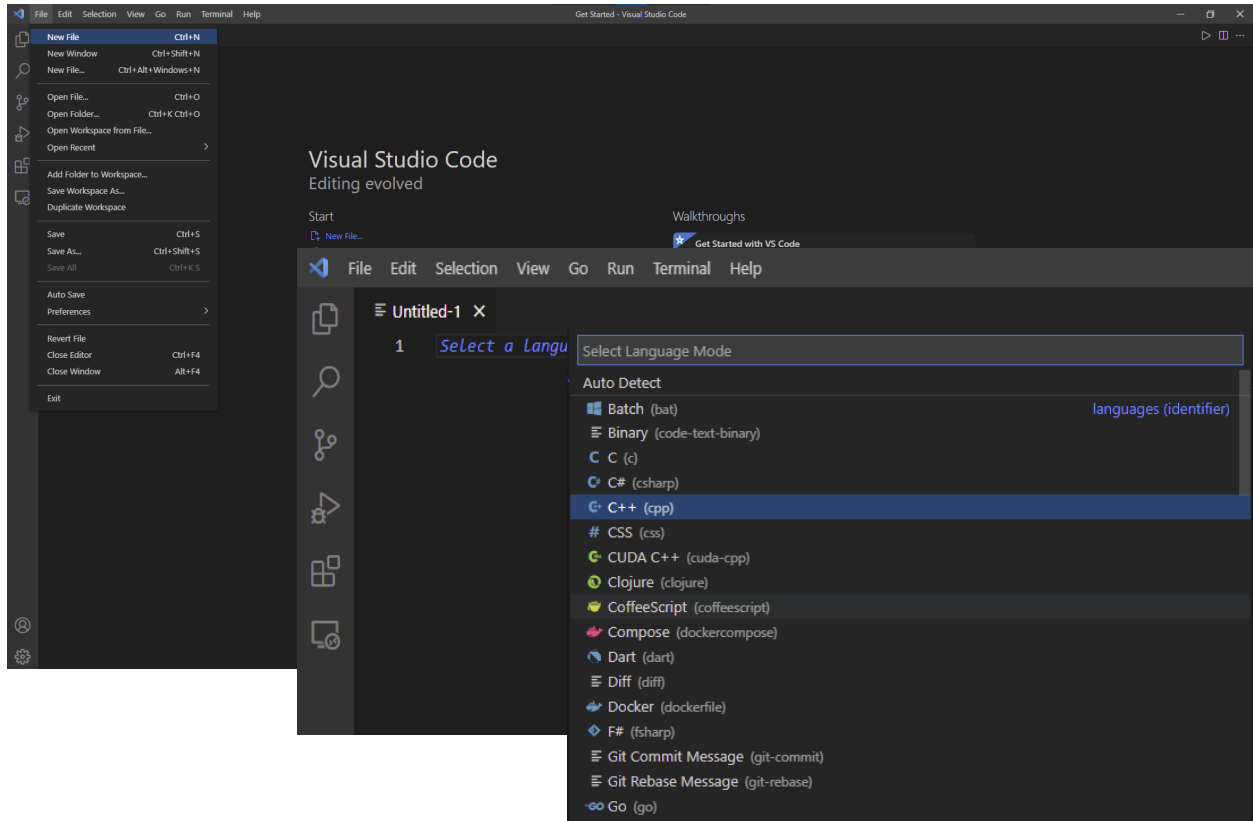


- You should see VS Code running, and at this point you're ready to start coding!



Compiling Code In Terminal

When you open VSCode you should be able to see some options like creating a new file, opening an existing file etc. You will also find these options in the File menu from the top menu bar. Create a new file and save that file as “example.cpp”. The filename can be anything you like, but for now lets keep it simple and name it “example.cpp” as you save(ctrl s). Open that file, select the language c++(cpp) and type the code to be compiled.



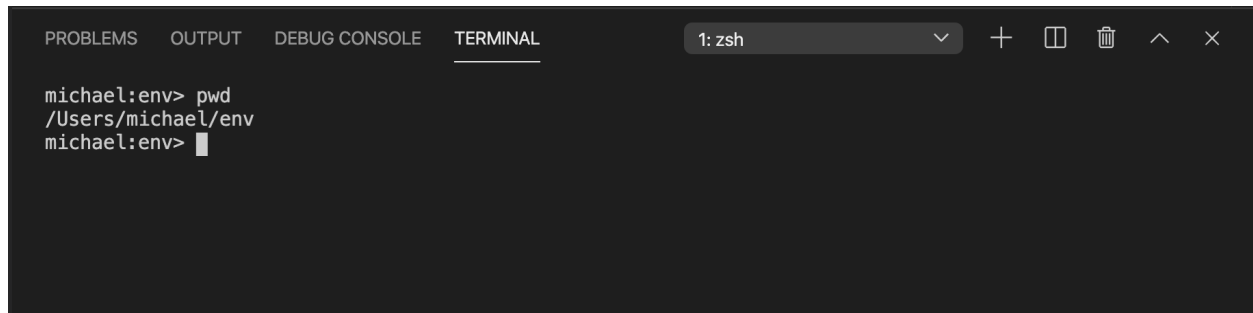
You will write the following code on your VS code.

```
example.cpp x
example.cpp > ...
1  #include <iostream>
2
3  using namespace std;
4
5  int main(){
6      cout << "It works!" << endl;
7  }
8
```

Looks good? Now we will compile and run your code!

In order to compile and run your code you first need to navigate to where your code is saved through the terminal. You do this using the **cd** (change directory) command (see the Homework 0 document on Canvas if you need an explanation on how to do this). If you have a folder open in VSCode by default the terminal will open to the location of that folder.

For instance on my computer I have a folder I made called **env** in VSCode which the terminal automatically opens to. You can check the current path your terminal is working at using the **pwd** command (print working directory) as shown below:

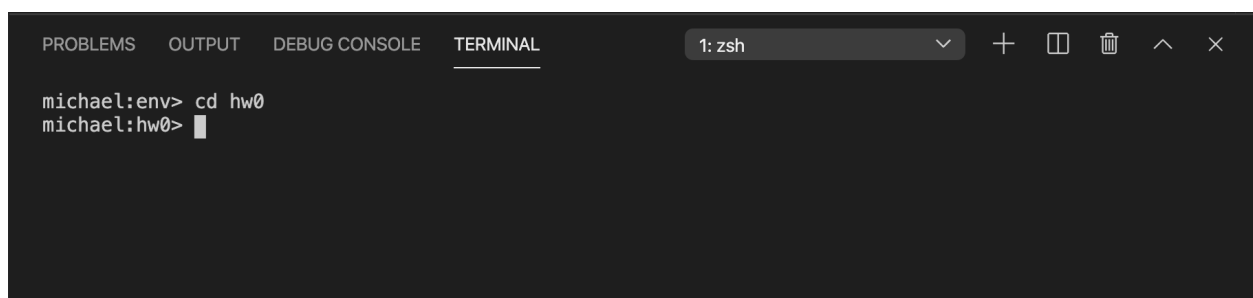
A screenshot of a VS Code terminal window. The terminal has tabs for PROBLEMS, OUTPUT, DEBUG CONSOLE, and TERMINAL, with TERMINAL selected. The terminal title bar says "1: zsh". The prompt is "michael:env>". The user has entered "pwd" and the output is "/Users/michael/env". The prompt is now "michael:env>".

```
michael:env> pwd
/Users/michael/env
michael:env>
```

Note: Your terminal may look slightly different, as everyone's computer is different. The first line in the screenshot above reads "**michael:env> pwd**", **michael** is the username of the person using the terminal, and **env** is the name of the current working directory. And **pwd** is the current command. Any command that users type will be after the > symbol. This symbol might be different depending on your computer, but that's okay.

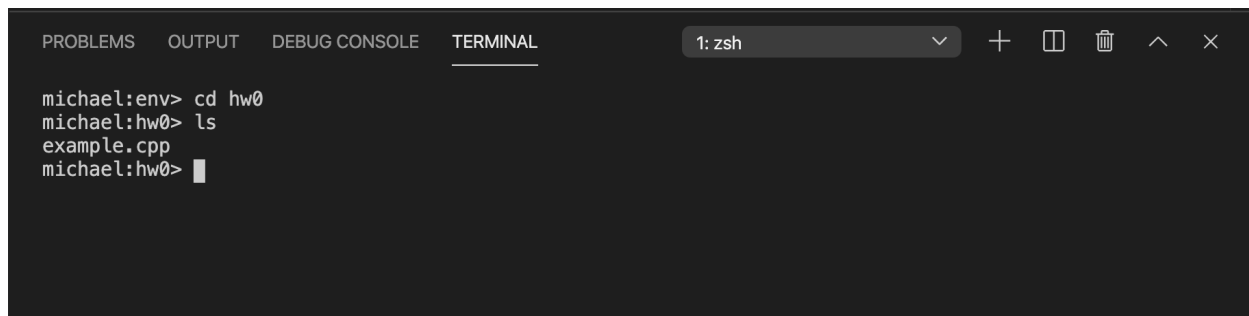
Now the second line shows the output when the **pwd** command was run by pressing enter/return after the command which shows the current working directory. Don't worry about copying this directory path, yours can be different.

Now let's say the code I want to run is located in a subdirectory called **hw0** within the **env** directory, then I can navigate to hw0 using the **cd** command (change directory). This command is of the form "**cd <destination>**". Where <destination> should be replaced by the path to the directory you want to switch to. Here if a folder/directory name is mentioned without a full path (as it is in the example below), it is considered a sub folder/directory of the current working directory.

A screenshot of a VS Code terminal window. The terminal has tabs for PROBLEMS, OUTPUT, DEBUG CONSOLE, and TERMINAL, with TERMINAL selected. The terminal title bar says "1: zsh". The prompt is "michael:env>". The user has entered "cd hw0" and the output is "michael:hw0>".

```
michael:env> cd hw0
michael:hw0>
```

Now let's double check the files we are looking for are actually there by using the **ls** (list) command.

A screenshot of a terminal window within a code editor. The terminal has tabs for 'PROBLEMS', 'OUTPUT', 'DEBUG CONSOLE', and 'TERMINAL', with 'TERMINAL' selected. The title bar shows '1: zsh'. The terminal output shows a user named 'michael' at a prompt 'env' typing 'cd hw0', then at a prompt 'hw0' typing 'ls', which returns 'example.cpp'. The prompt returns to 'env'.

Now to compile our code we use the program **g++** in the following command.

```
"g++ -std=c++17 example.cpp"
```

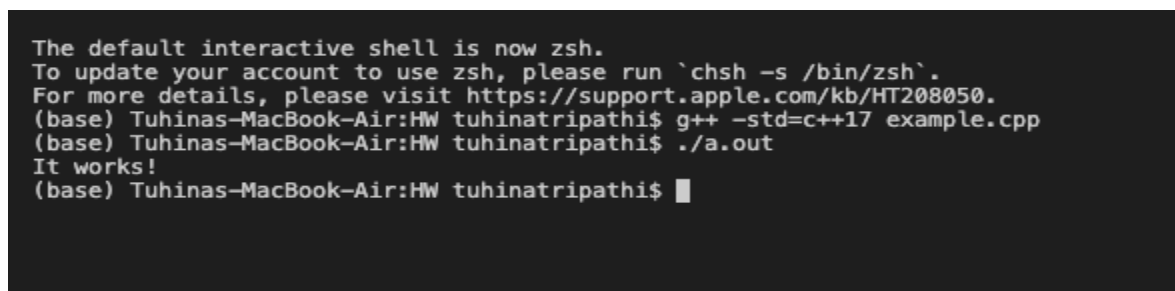
g++ is the compiler program

-std=c++17 specifies the version of C++ we want to use

example.cpp is the file we want to compile

This command creates a file named **a.out** which is the compiled version of the code in **example.cpp**, which can be executed. Now to execute our code we run **a.out** by typing the following and hitting enter.

```
"./a.out"
```

A screenshot of a terminal window. It shows a message about switching to the zsh shell. Then, a user 'tuhinatripathi' runs 'g++ -std=c++17 example.cpp' and then './a.out'. The output says 'It works!'.

=====

You can also specify the name of the output file using the following command format.

```
"g++ -std=c++17 example.cpp -o <output filename>"
```

Where **<output filename>** is replaced with the name you want.

Then running **"./<output filename>"** will run the executable file. **"./a.out"** is the default executable name generated.

=====

Acknowledgements:

Most of the content in this guide is taken from the official installation article available on the VS Code website: <https://code.visualstudio.com/blogs/2020/12/03/chromebook-get-started>.