APS 105 — Computer Fundamentals
Lab #0: Introduction to Visual Studio Code and UNIX Basics
Winter 2022

You must use `examify.ca` to electronically submit your solution by 11:59pm on **Saturday, January 15, 2022**.

---

1 Introduction

This lab handout serves as your introduction to the computers in the ECF Labs that we will be using in this course. They are running a variant of the UNIX operating system called Linux, and specifically the version compiled by Red Hat, called Red Hat Enterprise Linux (RHEL). UNIX is significantly different from the Windows operating system that you may have used previously, but similar to the macOS operating system, which is another UNIX variant.

This document is in the form of a tutorial that will gradually take you through the basics of setting up the necessary working environment for compiling, running, and debugging C programs throughout this course, and for interacting with a computer running a variant of the UNIX operating system, such as Linux and macOS. It is highly recommended that you read and follow this document, to be better prepared for Lab 1.

2 Visual Studio Code

There are two approaches to creating, compiling and running the C programs that you will write for this course: Either you can use a separate tool to perform each of these actions — a text editor to create the program file, a compile command to compile it, and another command to run it; Or, you can use an Integrated Development Environment (IDE), which allows you to perform many tasks from a single, usually graphical, application.

In this lab, we will use the second technique to create, code, compile, run and debug your C programs. The official IDE for this course is Visual Studio Code (VS Code). It is the most popular environment tool used in 2021[1].

2.1 Installing Visual Studio Code on macOS

To install and run code in VS Code on macOS, please refer to the video titled "Installing and using Visual Studio Code to compile, run, and debug C programs on macOS" on Quercus or follow the following steps and Section 2.3 for a brief tutorial on developing code using VS Code.

Step 1 Go to `code.visualstudio.com`, and download VS Code. It will take seconds.

Step 2 Once the download is complete, open the zip file downloaded by double-clicking the .zip file. It will extract the zip file and install the application. Drag and drop the produced Visual Studio Code Application into the Applications folder as in Fig. 1.

Step 3 Open the VS Code application by double-cicking it.

Step 4 Before being able to compile, run and debug your program, we need to install the necessary C compiler and debugger. To do so, click on **Terminal** → **New Terminal** next to Run and Window at the top of your screen.

[1] "Stack Overflow Developer Survey 2021 — Integrated Development Environment", `https://insights.stackoverflow.com/survey/2021`

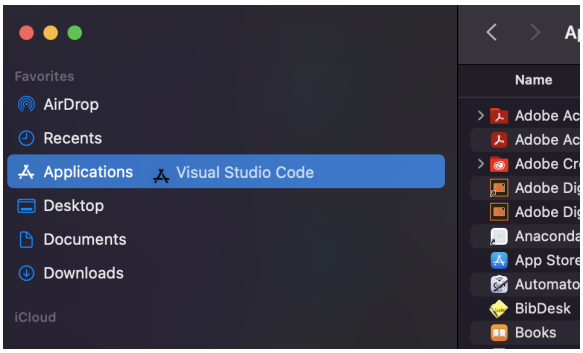


Figure 1: Dragging and dropping Visual Studio Code to the Applications Folder

Step 5 Type in the following command in the terminal:

```
$ xcode-select --install
```

Click "Install" if you are prompted to install Xcode Command Line Tools. This will take time depending on your machine's capacity and Internet connection.

Step 6 Check if your C compiler has been correctly installed by typing:
`$ gcc --version`, to see version number of gcc.
If it is not installed, you will see `gcc: Command not found`

`$ which gcc`, will print out the directory of gcc installation.

2.2 Installing Visual Studio Code on Windows

To install and run code in VS Code, please refer to the video titled "Installing and using Visual Studio Code to compile, run, and debug C programs on Windows" on Quercus or follow the following steps for installation and Section 2.3 for a brief tutorial on developing code using VS Code.

Step 1 Go to `code.visualstudio.com`, and download VS Code. It will take seconds.

Step 2 Click on the downloaded file to open it. Accept the agreement, and press Next and Install when appropriate. It takes a few more seconds. Then press Finish.

Step 3 Go to `https://www.msys2.org/` and follow the installation steps.

Step 4 Go to **Settings** using the search at the bottom left → search for "Edit environment variables for your account". Click on **Path**, and click **Edit**. In the new window, press **New** on the right and add `C:\msys64\mingw64\bin` and click **OK**.

Step 5 Search for *Visual Studio Code* and open the application.

2.3 Developing and Running C Programs in Visual Studio Code

Step 1 Click on The *Explorer* button on the top left as shown in Fig 2. To create a new folder, click on **Explorer** → **Open Folder** → Navigate to your desired folder → press *New Folder*. Name

your folder `aps105` and click *Create* then *Open*. Make sure your desired folder that contains `aps105` does not contain any spaces in its path.

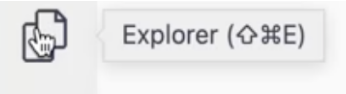


Figure 2: Explorer icon

Step 2 Create a new folder in the `aps105` folder, for example `lab0`, by clicking on **New Folder** icon next to `aps105` on the top left.

Step 3 Click on `lab0`, and press the **New File** icon next to `aps105` to create a new file as shown in the Fig 3.

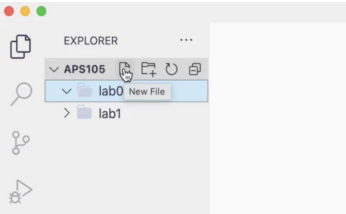


Figure 3: New file creation

Step 4 To compile using VS Code, we need to install two necessary extensions. Go to Extensions below the Explorer icon as shown in Fig. 4.

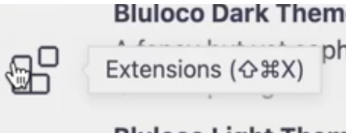


Figure 4: Extensions icon

Step 5 Install `C/C++` and `C/C++ Runner` extensions (as highlighted in Fig. 5) by clicking on "Install" after you click on each of them. Close the tabs when installations are complete.

Step 6 On the bottom left of your window, you should see **Select Folder**. This helps you select a particular working folder. In our case, we want to select `aps105/lab0`.

Step 7 Write a test code in `lab0.c` file. Turn on the option of *Autosave*, so that you do not need to worry about saving your C program every time you make a change. Go to **File** → **Autosave**.

Step 8 To compile your program, press the *Compile* button towards the bottom left of the screen as shown in Fig. 6. To run your program, press the *Run* button. Any output will be observed in the terminal to the lower half of your VS Code window.

Step 9 Walking through individual lines of your code step by step is called debugging. To debug your code, you need to add a breakpoint. A breakpoint is a line of code where your program will pause at when you click the *Debug* button. To create a breakpoint, you need to hover over the line you want your code to pause at and click there as shown in Fig. 7.

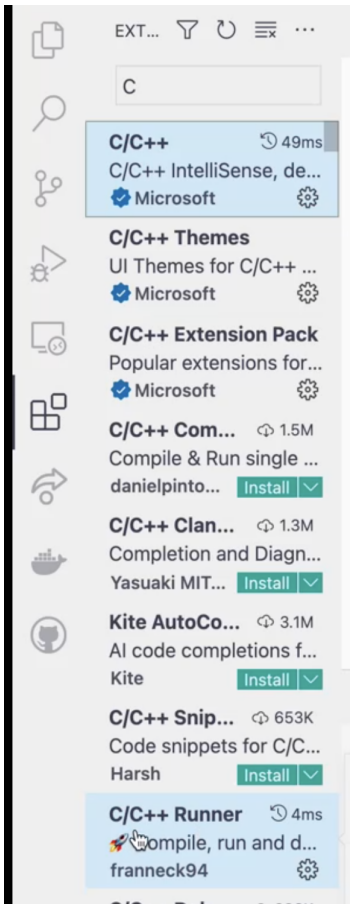


Figure 5: Two extensions that need to be installed.

Figure 6: Handy buttons to compile, run, and debug your C program.

Figure 7: Adding a breakpoint.

Step 10 Click on the *Debug* button to start a debugging session. Your code will stop at your breakpoint without executing it, unless you press **Step over** button as shown in Fig. 8. As you step over, you will see the value of `i` changing from 0 to 1. If you want to continue running the program, without debugging further, you can press the **Continue** button.

Figure 8: A debugging session.

3 Logging Into ECF Remotely

The first thing you must do when you begin your work on the computer is to "log in" to let the system know that it is you who is using the system. Access to the system is controlled to maintain its security; only those with accounts on the Engineering Computing Facility (ECF) system can access it. To login, you must have two pieces of information: a login name and a password. All undergraduate engineering students are automatically given an account when they register. If you do not already have a login name and password, you will be told how to obtain them in the next section.

Your login name (or "login") is a six to eight character string that is your identity on the computer network. Your password is a variable length character string that should be known only to you. Your password serves as a key to your account much as you would use a key to gain entry to your locked house. In fact, not even the system knows your password; it only keeps an encrypted version. Later in the lab you will be shown how to change your password.

Never give your password to someone else or let someone use your account. Doing so is against University policy and the other user can leave open a "back door" into your account to access it later, even if you change your password.

You may choose to work from a remote location, but directly on the ECF computers. You can do so by logging into the ECF computers remotely, using the `ssh` command from any other computer running a UNIX variant, including Linux and macOS. You will need to operate in a terminal. On macOS, press Command + Space and type in *terminal*, and then choose the *Terminal* application. A small window will pop up, and your Terminal application is now open. The `ssh` command is pre-installed on Linux and macOS, you can use it directly.

If you use a Windows computer, you can use the MSYS2 terminal that you used to install minGW, by searching for `MSYS2 MSYS`. Before using the `ssh` command on Windows, however, you will need to first install the `OpenSSH` package by running the command:

```
$ pacman -S openssh
```

To access ECF from your current computer, type in the `ssh` command as follows:

```
$ ssh -l yourUserName remote.ecf.utoronto.ca
```

where `yourUserName` is your ECF login name, `-l` is dash-ell. You will be prompted for your ECF password to log in. Now, you are logged into ECF.

To go back to accessing the terminal of your local computer, just type in `exit`.

4 Changing Your Password

At this point an important task that you should perform is to change your password, if you want. Remember that the password to your account is the only security measure keeping your account private so you should choose your password carefully and keep it an absolute secret. You change the password by using the `passwd` command. Your password should be at least 6 characters long, with 7 or 8 being preferable. Try to use a mixture of numbers and letters in both upper and lower case, making it less likely that someone will be able to guess your password. In particular, do not use any names or single words that could be found in a dictionary. Since computers are good at tedious tasks, people have written programs that can try to guess your password using a dictionary and other guessing schemes.

Use the following command to change your password:

```
$ passwd
```

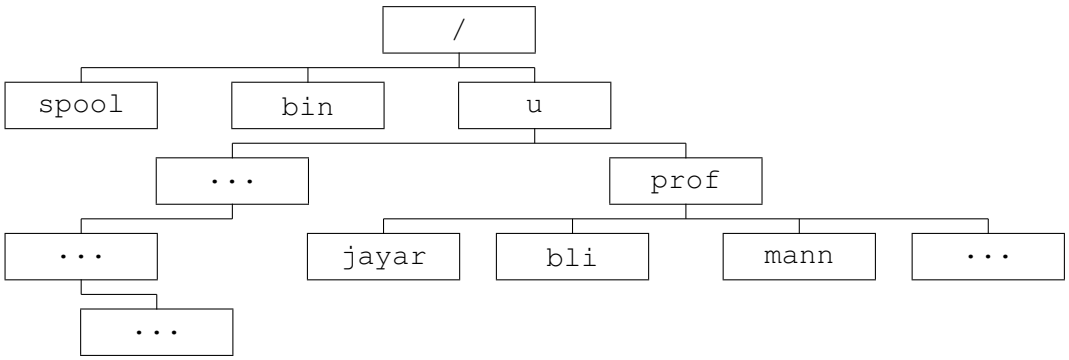
5 File Organization

Data and programs that you use are stored as files on a disk. There are many types of files: some are software programs that can be executed, other files contain text that can be read, while other files contain images that can be displayed on the screen. Two things that all files have in

common are that they can be stored, and that they all have a name. We will soon take a look at some files on the ECF computer system.

As we add more files to computer storage the number of files grows to the point where it is hard to keep track of them all. For this purpose, files are organized into *directories*. You can think of directories as being like file folders. File folders are used to organize papers, articles, and other things and are named according to their contents. You may well be familiar with the concept of folders from the Windows operating system. In UNIX, folders are called directories.

If you mapped out the directory and file organization onto a graphical representation, you would get a tree-like structure as shown in the diagram. At the top is the *root directory*. It is represented by a slash character, which is its name. The root directory may contain files, and it will also contain other directories. The files and directories within the root directory are shown as nodes below and attached to the root directory. These directories may have files and other directories contained within them and these are shown as nodes below and attached to the parent directory. This pattern can be repeated many times.



UNIX file organization

In the figure, `bli` is a *subdirectory* of `prof`, and `u` is the *parent* directory of `prof`.

It is the files that are actually important when it comes to running programs, whereas the directories are only there to organize the files. When you want to work with files, you generally have immediate access to the files in one directory only. This directory is called the *working directory*. After you are logged into ECF using `ssh` command, or using a Terminal in your local computer, find out what the present working directory is by typing in the following command:

```
$ pwd
```

UNIX will respond with a string that looks like `/u/prof/bli`. This string describes a *path* through the directory tree from the root directory to the present working directory. Reading it from left to right, the path starts at the root directory, which contains the directory named `u`, which contains the directory `prof`, which contains the directory `bli`. While your directory path may be different, it always reflects your present working directory.

You can change the present working directory using the `cd` command, which is short for *" change directory."* The `cd` command takes an *argument*, which is an extra term that is input after the command. In this case the argument is the path of the directory to which you want to move. There has to be a space between the command and the argument. Try the following:

```
$ cd /
```

Then enter the `pwd` command. You will see that the present working directory is as you have requested — the root directory. Now try the following:

```
$ cd /u
```

Entering the `pwd` command will show that you are now in the directory requested. Finally, enter:

```
$ cd
```

and this will take you back to your *home directory* as you can verify using the `pwd` command once again. It will print `/u/prof/bli` in my case, where `bli` is my username. The *home directory* is a special directory that is created for you. The home directory is yours and yours alone, and you may freely create files and subdirectories within your home directory. Every time you wish to return to your home directory from anywhere else, you just need to use the command `cd` without any parameters.

Up until now, we have been specifying the path from the root directory to the desired directory as the argument to the `cd` command. This is called the *absolute path*. It is not necessary to always give the absolute path as the argument and usually one does not do so. Rather, one can use what is known as a *relative path*.

To see how relative paths work, start by going to the root directory by entering `cd /`. Now go to the `u` directory by entering the following command:

```
$ cd u
```

If you do a `pwd` command, you will see that you are now in `/u` even though you did not have the slash beginning the argument of the `cd` command. Because the argument did not begin with a slash, UNIX understood you to mean go to the `u` directory from the present working directory, which is `/`. Again, if you enter the `pwd` command you will see how you have moved into the new directory.

There are two important short forms that are used to specify directories. Two consecutive periods (`..`) refers to the parent directory of the current directory, so if you use the command `cd ..` the parent directory will become the working directory. The other important short form is the $\sim$ character, which refers to your home directory. No matter where you are in the directory tree, entering `cd` $\sim$ will make your home directory the working directory. Of course, as we just mentioned, with the `cd` command, you can get even simpler than that and enter just `cd` to make your home directory the working directory.

Practice using these short forms. First, enter a command to go directly to your home directory (`cd` $\sim$ or `cd`), then back up the directory tree, one directory at a time, using the `cd ..` command. Then return to your home directory once again.

Now you should have a good understanding of navigating around directories. But of course, the interesting part is not directories, but what they contain. To determine what a directory contains, one uses the `ls` command, which stands for "list" and instructs UNIX to list the contents of the present working directory, which will include any files and subdirectories. Now let us see what we can find using the `ls` command. Go to the root directory and enter:

```
$ ls
```

You will see some text (strings) arranged in columns. The `ls` command is displaying the names of files and subdirectories contained in the root directory. Try using `ls` in the `/u` directory and in

other directories if you wish.

Now go to your home directory and enter `ls`. There is probably no output. This is because as a new user, you have not created any new files or directories. However, even though `ls` does not show you any files in your home directory, there are some there that are hidden. To show all the files, including those that are hidden, enter:

```
$ ls -a
```

Some filenames will be displayed. The names of hidden files are easily distinguishable as they all begin with a period. When your account was created, your home directory and a few hidden files were automatically created for you. These files are hidden because you would not normally work with them as you would with your program files. Hidden files usually contain special information for the computer system on various things such as how to set up your screen display or how to respond to your commands. You may want to explore some of these files and determine what they do some other time.

One last variation on the `ls` command that you will probably find helpful is:

```
$ ls -al
```

The final `l` stands for long form and entering this command will show all files and directories with more descriptive information for each item.

6 Manipulating Directories and Files

In the previous section, it was shown how to explore the directory tree and how to look at existing directory contents. Now you will learn the basic commands for changing directory and file organization. For the most part, this will be constrained to directories and files within your own home directory.

Changing the directory tree structure is accomplished through the use of two basic commands: one that creates new directories, and one that destroys them. The `mkdir` command makes a new directory. After you are logged into ECF using the `ssh` command, or using a Terminal in your local computer, go to your home directory and create a new directory there by entering

```
$ cd ~

$ mkdir aps105
```

The argument supplied with this command is the name of the new directory, `aps105`. You can verify that a new directory was made by using the `ls` command. Notice that, by default, the new directory is placed within the present working directory.

Removing directories that are no longer required is just as easy. To do so, you go to the directory in which the unwanted directory is contained, then use the command:

```
$ rmdir <directoryName>
```

Note that a directory cannot be removed unless it contains no other files and no other directories.

Basic file manipulation is achieved through the commands that copy, move and delete files. The `cp` (copy) command is used to make a duplicate of an existing file. It takes two arguments: the first argument is the name of the file to be copied, the second argument is the new name of the new

duplicate file. In your home directory, enter the following command:

```
$ cp .bash_history bash_history
```

This command has made a copy of the hidden file `.bash_history` and called it `bash_history`, which is not a hidden file because its name does not begin with a period. Verify that the copy was made using the `ls` command.

The `mv` (move) command changes the name of a file, and can also move a file into a different directory (an idea that will be discussed shortly). Try the following:

```
$ mv bash_history bash_history.txt
```

Again, the `ls` command will show you the change that was made.

Finally, files can be deleted by using the `rm` (remove) command. Delete the new file that you created with the following:

```
$ rm bash_history.txt
```

Up until now, the file and directory manipulation commands have been demonstrated to operate only within the present working directory. These commands are in fact more versatile than this because it is possible to work with files and directories other than those in the present working directory. To work with a file or directory outside of the present working directory, you must supply its path rather than just its name for the command argument. Absolute or relative paths can be used.

For example, `cp` can be used to make a duplicate of a file that is not within the present working directory. Use the following command to copy the file `gcc` which is located in the directory `/usr/bin` to your home directory:

```
$ cp /usr/bin/gcc .
```

The first argument of the `cp` command is the source file; here it contains the absolute path of the `gcc` file. The second argument is a single period, a commonly used short form that represents the present working directory. UNIX interprets this command to mean place a copy of the file `gcc` (located in `/usr/bin`) in the current directory and give it the same name: `gcc`. Now try using the `mv` command to place this file in the `aps105` directory you created earlier:

```
$ mv gcc aps105/gcc
```

The `mkdir`, `rmdir`, and `rm` commands also can take paths as arguments. Experiment with these commands some more until you get the hang of it. Use the UNIX manual as a reference.

Note: All filenames and directory names we discussed so far have no spaces. This is because spaces in commands would mean a different argument. For example, if you are in a working directory named `/u/prof/bli/aps105` and it has a file named `lab 0.c`, and you want to copy it to `/u/prof/bli/aps105/lab0`. You may want to do `cp ./lab 0.c ./lab0`; however, `cp` will consider that you want to copy `./lab` into `0.c` and `./lab0` is an extra argument. Your prompt will look like "lab: No such file or directory and 0.c: No such file or directory". Solution for that is to contain filenames with spaces into quotes and use

```
$ cp "./lab 0.c" ./lab0
```

7 Transferring Files Between Computers: `scp`

When logged in to a computer on the ECF system you are using what is known as a *distributed file system*. This simply means that your files are visible to any ECF machine you happen to be logged in to, be it `p42.ecf.utoronto.ca` or `p178.ecf.utoronto.ca`. As long as you do all your work on ECF machines, you do not need to worry about having to transfer files.

To transfer files form your local computer to ECF, you can use a utility named `scp`, which is a *secure* version of `cp`, and behaves in a very similar manner. To practice using `scp`, we will copy a file from whichever machine you are using to `remote.ecf.utoronto.ca`. Open your terminal — if you are using macOS, type Command + Space, and Search for terminal; and if you are using Windows, you will need to use the MSYS2 terminal that you used for installing minGW by searching for `MSYS2 MSYS`. Then type in the following command:

```
$ scp lab0.c yourLogin@remote.ecf.utoronto.ca:~/lab0.c
```

where `yourLogin` should be replaced with whatever your login name is. You will be prompted for your password on ECF. You may also be given a warning the first time you transfer a file to a new machine, that looks like this:

```
The authenticity of host 'remote.ecf.utoronto.ca (128.100.8.48)' can't be established.
ED25519 key fingerprint is SHA256:sEMRdV52ohJIqszCtAlgL6rn+els5BQUVU2GnO5Vth0.
This key is not known by any other names
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
```

It's OK to answer 'yes'.

The main difference between `scp` and `cp` is the addition of the construct `username@hostname:` at the beginning of the name of either the file being copied or the location the file is being copied to. The '`:`' character is important, as it tells `scp` where the pathname begins. When the first character after the '`:`' is not `/`, `scp` assumes that a pathname relative to the login name's home directory is being used. So, for example, the command

```
$ scp someFile bli@remote.ecf.utoronto.ca:aps105/
```

Copies the file `someFile` from the present working directory on the machine you are currently logged into, to the `aps105` subdirectory of the home directory of user `bli` on `remote.ecf`. The command

```
$ scp someFile bli@remote.ecf.utoronto.ca:/tmp/
```

would transfer the file to the `/tmp` directory on `remote.ecf` — this is an example of using an absolute pathname.

8 Using Examify to Submit your Labs

Throughout this course, developing your source code will be on VS Code; however, submissions of the deliverables for the labs will be on Examify `https://examify.ca/`. Please make an account with your University of Toronto email address. By this time, you should have received an invitation via email to enroll in a course on Examify. *Note:* When you verify your password, check your junk mail for the response. If you login, you will notice an invite to the course on your home page. As you login into Examify, you will see APS105 Jan 2022.

As you click in the APS105 course, under exams on your left, you should find a list called "Exams" under which the labs will be posted as shown below. If you select Lab 0, you should find

a space for you to add your code as shown in Fig. 9. The site contains an editor, compiler and test code. You can cut/paste your C program that you developed in Visual Studio Code.

Figure 9: The Question and Answer editor in Examify.

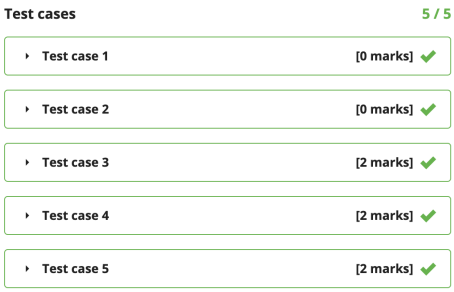


Figure 10: Test cases that have passed.

You can run test cases by clicking "Run test cases". This will run scripts over your code to test it again test cases, and compare the output of your code to the output of the solution code. In Fig. 10, all test cases were passed. While in the following case in Fig. 11, the expected output is different from the output of the student code. As you can see, differences are highlighted in red.

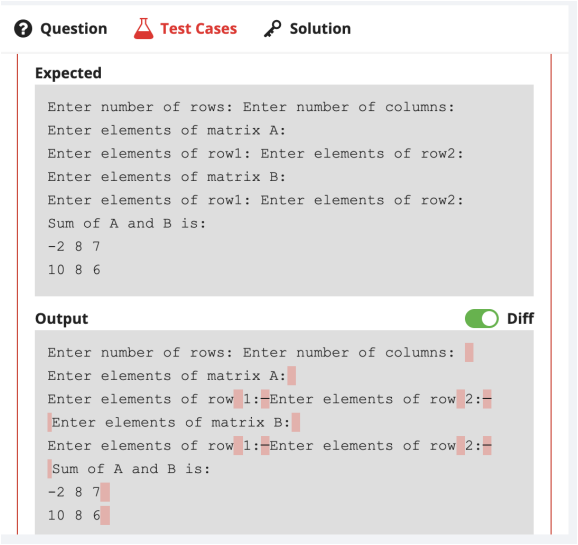


Figure 11: Comparing output in failed test cases

In this lab, you are required to submit a simple C program that prints:

```
Hello World!
```

Although your submission in Lab 0 will not be marked, you should use this opportunity to get familiar with the submission system. Your solution should be submitted by 11:59 p.m. on Saturday, January 15, 2022.