Lab 1: Using the shell

The goal of this lab exercise is to get a bit of practice using the shell (command-line environment) and a plain text editor, and to write a very simple C program, compile it, and execute it.
To get a shell in Windows, use the Power Shell application, and type

```
> wsl
```

(without the `>`). To get a shell in MacOS, use the Terminal application.
The commands to execute in the description below use `$` as the prompt, which is the standard prompt for the Bash shell. That is what you should get after starting WSL in Windows. The default shell for recent releases of MacOS Terminal uses `%` for the prompt, but the same commands should work. You should not type the prompt when entering the commands.

1 Learn about the command line

You should have already viewed the following videos that are part of the "Learning Linux Command Line" course on Linked In Learning, which is available through your Wesleyan Portal:

- Chapter 2: Everything.
- Chapter 3: "The Linux file system" through "Copy, move, and delete files and directories." The first video on the Linux file system has details that are specific to Linux. You don't need to remember specific directory names, just the takeaway that different kinds of files reside in particular directories.

This is about one hour's worth of videos. You should be able to follow along these videos by opening a shell as described above. Occasionally the instructor opens a graphical file manager; on Windows you can use Windows Explorer, and on MacOS you can use a Finder window.
Although this Linked In Learning course is (obviously) about Linux, the information in the videos that you are viewing applies to the WSL (for Windows) and Terminal (MacOS) command line environments. I intentionally don't have you viewing the videos in Chapter 1 on setting up your environment, because they don't apply to COMP 211 and will lead you down a very dark hole on creating Linux virtual machines (actually a very interesting hole, but in practice many hours deep with information you do not need). Although there is a section titled "Following along on other platforms," I don't think it is too helpful.

2 Make some directories

1. Using shell commands, change to the directory in which you will keep all your files for COMP 211. For example, if the directory is named `comp211` and it is in your home directory, and the current working directory for your shell is your home directory, then

```
$ cd comp211
```

2. Make a `labs` directory, and in that a `lab1` directory:

```
$ mkdir labs
$ cd labs
$ mkdir lab1
$ cd lab1
```

3 Edit a file

3. Download the `lab1sample.txt` file that is posted for this lab into the `lab1` directory. Copy the file to `lab1stuff.txt` using the `cp` command, which is invoked as

```
$ cp file1 file2
```

to copy `file1` to `file2` (use the actual filenames, of course).

4. Use your plain text editor to edit `lab1stuff.txt` according to the instructions in that file.

4 Write, compile, and execute a program

5. Use your editor to create a file in `lab1` whose contents are exactly those of Program 1. Name the file `hello.c`. Do not copy-and-paste from this file. Type each character. Make sure to change `<Your name here>` to your name.

```
1  /* COMP 211 Lab 1 : Hello world
2  *
3  * <Replace this text with your name>
4  */
5
6  #include <stdio.h>
7
8  int main(void) {
9      printf("Hello world\n");
10     return 0;
11 }
```

Program 1: A "Hello, world" program in C.

6. Compile your program with

```
$ cc hello.c
```

7. Execute your program with

```
$ ./a.out
```

Note that the `./` part is required; it tells the shell to look for the `a.out` program in the current directory. If you try to execute your program with

```
$ a.out
```

you will get an error from the shell to the effect that there is no such program—try it!

8. Remove `a.out` with the `rm` command:

   ```
   $ rm a.out
   ```

9. Re-compile your program with

   ```
   $ cc -o hello hello.c
   ```

   The option `-o hello` tells `cc` to name the executable `hello` instead of `a.out`. Execute your program by telling the shell to run `hello` instead of `a.aout`.