# Lab 2

**Get checked off for full points of incomplete work from the previous lab within the first 10 minutes of lab.**

**Each lab will begin with a recap of last lab and a brief demonstration by the TAs for the core concepts examined in this lab. As such, this document will not serve to tell you everything the TAs will in the demo. It is highly encouraged that you ask questions and take notes. In order to get credit for the lab, you need to be checked off by the end of lab. For non-zero labs, you can earn a maximum of 3 points for lab work completed outside of lab time, but you must finish the lab before the next lab. For extenuating circumstance, contact your lab TAs and Instructor.**

In this lab, you can form a group of 2-3 individuals. **You must be checked off together as a group at the end of the lab.** Although you perform tasks as a group, ensure that you understand the work and ask questions to TAs as needed.

## (2 pt) More Linux

1. Now, open your secure shell (ssh) client and connect to:
   access.engr.oregonstate.edu

2. In Linux, every command has a manual page, which provides you information on what the command means, how to use the command, and a description of the available options for a command. Linux commands do not have spaces in them and they are lower case. This is important because **Linux is case sensitive!!!** Following a Linux command is a space followed by arguments. Some arguments may be required, and some are optional such as options, which are preceded by a dash.

   **linux_command –option required**

   If an argument is optional in Linux, then it is enclosed in brackets, [], and required arguments do not have brackets. For example, **man ls**, and notice that everything supplied to ls is optional. You can also use the command and --help to get a brief manual page for the command, **i.e. ls --help**

3. In order to get more familiar with the Linux/UNIX environment, you will do a few Linux-type exercises at the beginning of each lab. Today, we will learn the copy,

move, and remove commands, i.e. cp, mv, and rm. First, look at the manual page for each of these commands. **Remember to use the space bar, b, and q for moving around in the manual pages.

**man cp**

**man mv**

**man rm**

4. First, let's use these new commands before moving forward. Copy your hello.cpp program from the labs/lab1 directory to your home directory.

   **cp labs/lab1/hello.cpp ~**

   This says, copy the hello.cpp file located in the labs/lab1 directory into my home directory. Use **ls** to list the directory contents and make sure you have a **hello.cpp file in your home directory.**

5. Now, rename the file to hello2.cpp by using the move command.

   **mv hello.cpp hello2.cpp.**

   Use **ls** to list the directory contents and make sure you no longer have a hello.cpp file and you now have a hello2.cpp file.

6. Create a test directory, and then change into that directory.

   **mkdir test**

   **cd test**

7. Copy the hello2.cpp file from your home directory to the test directory you are currently in. **Remember that .. is up/back a directory. You could also say cp ~/hello2.cpp . because you know that hello2.cpp is in your home directory.

   **cp ../hello2.cpp .**

   You could also say **cp ~/hello2.cpp** . because you know that hello2.cpp is in your home directory.

8. Now, go back to your home directory or up/back a directory, and remove the file hello2.cpp file in your home directory and remove the test directory and its contents, which contains the file hello2.cpp. Use **ls** to make sure you see the hello2.cpp file and the test directory in your home directory.

   **cd ..**

   **ls**

   **rm hello2.cpp** (when prompted press n so you don't remove it)

   **rm –f hello2.cpp** (notice no prompt, -f forcefully removes without asking)

   **rm test** (notice it won't remove a directory, even with -f)

**rm –r test** (notice the prompt, -r recursively descends into a directory to remove it and its contents, note you can use –rf together to avoid all the prompts) **ls** (you shouldn't see hello2.cpp or test)

9. Change into your labs directory, create a lab2 directory, and then change into that directory. **DO NOT use spaces in directory or file names in Linux. **cd labs**
   **mkdir lab2**
   **cd lab2**

10. There are a few shortcuts in Linux that you want to be familiar with using. One is the use of up/down arrows to move through your **history of commands**. At the shell prompt, press the up arrow and note what happens, and then press the down arrow and note what happens.

11. Another useful shortcut is **tab completion**. Go up two directories with **cd ../..**, and then let's change back into the labs directory. This time, after typing cd and l, then press the tab key. This will complete your labs word because it is the only option in your home directory that starts with an l. Now, try changing into the lab2 directory again using tab completion, but this time you'll be presented with two options that start with an l.

Here is a Linux and vim cheat sheet to help you reference some of these commands quickly. http://classes.engr.oregonstate.edu/eecs/winter2020/cs161-010/CheatSheet.pdf
You can find more Linux and vim cheat sheets and tutorials on the links page of our class website: https://oregonstate.instructure.com/courses/1772943/pages/useful-links

**Helpful vim hint**: When you are in vim escape mode, then you can use a command to show the line numbers on the left, which can be helpful for debugging. Show the line numbers in vim by typing **:set number**. **Please make sure you refer to lab 1 for a list of other helpful vim commands.

**(4 pts) Calculate the Size of Things**

Make sure you understand the following equations before moving on.

Number of distinct values that can be stored in x bits: $2^x$

Maximum value for an Unsigned Binary: $2^x - 1$

Maximum value for a Signed Binary: $2^{x-1} - 1$

Minimum value for a Signed binary: $-2^{x-1}$

What does x represent?
Why do we subtract the 1 in the exponent?
Why do we subtract the 1 in the expression?

Now, write a program to

- Ask the user to input a number of bits x.
- Calculate and print the maximum and minimum signed values that can be stored in x bits
- Calculate and print the maximum unsigned values that can be stored in x bits

How are you going to express an exponent? In C++, you need to use a built-in function, pow(base, exp), from the cmath library. For example:

```
#include <iostream>
#include <cmath>

using namespace std;

int main()
{
     int num = pow(2,3);
     cout << "2^3 is: " << num << endl;
     return 0;
}
```

Now, insert statements to add 1 to the unsigned and signed maximum number stored in the variable after you calculate the answer, and print the result of increasing the variable by one. Do the same for the unsigned and signed minimum number calculated by subtracting one from the variable and printing the value afterward. What happened? Why?

**Rand() and conditional statements**

**(1 pts)** In your assignment, you have variables, the need to use rand(), and conditionals. Create a program called **rand_numbers.cpp**, and type the following program into the file.

```
#include <iostream>
#include <ctime> /*included to allow time() to be used*/
#include <cstdlib> /*include to allow rand() and srand() to
be used*/
```

```
using namespace std;

int main()
{
      int x; // variable to hold our random integer
     srand(time(NULL)); // seeds random number generator.
Do this just once

     x = rand();
     cout << "x = " << x << endl;

     x = rand();
     cout << "x = " << x << endl;

     return 0;
}
```

Compile and run the above program 3 times.

- **What is the result of each rand() for the different executions?**

Comment out the srand() function call. Compile and run the above program 3 times. • **What is the result of each rand() for the different executions?**

Now add an additional srand() function call in between the two rand/cout lines of code, so there are two srand() function calls. Compile and run the above program 3 times. • **What is the result of each rand() for the different executions?**

**(3 pts)** Edit your **rand_numbers.cpp** program so that it **chooses a random int that is in the range 0-5** based on the value returned from rand(). I think the mod operator % would be very useful here. What is anything mod 5 or 6? How can this help you? **Print the random 0-5 number to the screen**.

Now, use this number, 0-5, to select which message to print:

- If the number is 0, print “Bummer, I’m zero!!!”
- Else if the number is odd, 1, 3 or 5, print “I’m an odd number!”
- Else it’s an even number, print “I’m an even number!”

**Before writing the code, first create a design document. You can refer to the Example Design Document.**

**Understanding the Problem:**

- Do you understand everything in the problem?
- What assumptions are you making about the problem you are solving, user input, etc.?
- What assumptions are you making about the problem you are solving, user input, etc.?

**Program Design:**

- What does the overall big picture of this program look like? list the specific steps or provide a flowchart of what is needed to create. Be very explicit!!!

**Program Testing:**

- Create a test plan with the test cases (bad, good, and edge cases). What do you hope to be the expected results?

Now, Implement your design.

Use nested if/else statements to print out different results. Example nested if/else:

```
if (num==0)
{
      cout << "Bummer, I'm zero!!!" << endl;
}
else if ((num%2)==1)
{
      cout << "I'm an odd number!" << endl;
      if (num==1)
      {
            cout << "I'm the number 1!" << endl;
      }
      …
}
…
```

**Show your completed work to the TAs for credit. You will not get points if you do not get checked off!**