

# CS 161 Lab #2 – Random Numbers and Conditionals

- **Start by getting checked off for (up to) 3 points of work from the previous lab in the first 10 minutes.**
- Each lab will start with a **recap of the last lab** and a **demo by the TAs of the new concepts in the current lab**. Therefore, this document does not cover everything that will be covered. You are highly encouraged to ask questions and take your own notes.
- **To get credit for this lab, you must be checked off by a TA by the end of lab.** For non-zero labs, you can earn a maximum of 3 points (of 10) for lab work completed outside of lab time, but you must finish this lab before the next lab begins. **For any exceptions, contact your lab TAs and Dr. Wagstaff in advance.**

---

## Goals:

- Expand your list of Linux commands
- Go deeper into min/max values data types can hold; explore overflow and underflow
- Generate random numbers
- Create your first Design Document
- Practice using if/then conditional statements

---

## (2 pts) A. More Adventures with Linux

- 1) **Connect to the ENGR server:** Open your secure shell (ssh) client and connect.

Mac users will open a Terminal window and use:

```
$ ssh <onid>@access.engr.oregonstate.edu
```

Windows users will create a new session using MobaXterm.

Refer to Lab 1 for more detailed instructions.

- 2) **Command arguments and options**

You learned about manual (**man**) pages in Lab #1. In Linux, every command has a manual page, which provides you information on what the command means, how to use the command (**arguments**), and a description of the available **options** for a command. Linux commands are generally lower-case and do not contain spaces. This is important because **Linux is case sensitive**. The Linux command is followed by a space and then any options and arguments.

```
$ linux_command -option1 -option2 ... argument1 argument2 ...
```

For example, in Lab #1 you did:

```
$ cd lab1
```

In which the command **cd** has one (required) argument: the destination directory (**lab1**).

Optional arguments are usually preceded by a dash, like when you did:

```
$ ls -a
```

You can combine arguments with options, e.g.:

```
$ ls -a lab1
```

To list all files in the **lab1** directory. Note that the file/directory argument (**lab1**) is not required for **ls** to run; if omitted, **ls** assumes that the current directory (**.**) is the argument.

You can see this in the “SYNOPSIS” section after typing

```
$ man ls
```

The command is followed by a list of options (in brackets because they are option), optional **arguments** (also in brackets), and any required arguments (not in brackets). Compare with

```
$ man mkdir
```

(Type `q' to exit manual pages.)

You can also use the command and `--help` to get a brief manual page for the command, e.g., `ls --help`

- 3) 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) Copy your `lab1_worst_advice.cpp` program from the `labs/lab1` directory to your home directory (`~`).

```
$ cp labs/lab1/lab1_worst_advice.cpp ~
```

Use `ls` to list the directory contents and make sure you have a `lab1_worst_advice.cpp` file in your home directory.

- 5) Now rename the file to `new_file.cpp` using the move command.

```
$ mv lab1_worst_advice.cpp new_file.cpp
```

Use `ls` to make sure you no longer have a `lab1_worst_advice.cpp` file and you now have a `new_file.cpp` file.

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

```
$ mkdir test
```

```
$ cd test
```

How can you check which directory you are currently in?

- 7) Copy the `new_file.cpp` file from your home directory to the `test` directory you are currently in.

```
$ cp ../new_file.cpp .
```

*\*\*Remember that `..` is up/back a directory. You could also use `cp ~/new_file.cpp .` because you know that `new_file.cpp` is in your home directory.\*\**

- 8) Now go back to your home directory (or up/back a directory) and remove the file `new_file.cpp` file in your home directory.

```
$ cd ..
```

```
$ ls
```

```
$ rm new_file.cpp
```

```
$ cp test/new_file.cpp . (get another copy)
```

```
$ rm -f new_file.cpp (remove it (force) with no safety prompt)
```

Remove the test directory and its contents, which also contains a file `new_file.cpp`.

```
$ rm test (won't work on a directory)
```

```
$ rm -r test (-r means recursive so it will work)
```

```
$ mkdir test (re-create it)
```

```
$ rm -rf test (USE WITH CAUTION! This bypasses all prompts)
$ ls
```

You should no longer see `new_file.cpp` or `test`.

- 9) Change into your labs directory, create a lab2 directory, and then change into that directory. **\*\*Remember not to 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 ../..`, then change back into the `labs` directory. This time, after typing `cd 1` (letter 1, not number 1), press the tab key. This will auto-complete to `labs` because it is the only option in your home directory that starts with an 1. Now try changing into the `lab2` directory using tab completion. This time you'll be presented with two options that start with an 1.

Here is a Linux and vim cheat sheet to help you reference some of these commands quickly.

[http://classes.engr.oregonstate.edu/eecs/winter2020/cs161-020/links/OSU\\_linux\\_vim\\_ref.pdf](http://classes.engr.oregonstate.edu/eecs/winter2020/cs161-020/links/OSU_linux_vim_ref.pdf)

You can find more Linux and vim cheat sheets and tutorials on the links page of our class website:

<http://classes.engr.oregonstate.edu/eecs/winter2020/cs161-020/links/>

**Helpful vim hint:** When you are in vim command 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`. **\*\*Refer to Lab 1 for a list of other helpful vim commands.\*\***

---

### (3 pts) B. Calculate the Size of Things

**Pair Programming:** In this lab, you can **choose a partner** for pair programming on the coding portion of the lab. You must be checked off together, and you only need one computer for pair programming. One person will be the **driver**, who controls the computer and codes, while the other is the **navigator**, who observes and advises. After ~15 minutes, you will **switch driver and navigator**, continuing this pattern until the task is complete. Please read more about pair programming and the benefits:

[http://classes.engr.oregonstate.edu/eecs/winter2020/cs161-020/labs/PP\\_StudentHandout.pdf](http://classes.engr.oregonstate.edu/eecs/winter2020/cs161-020/labs/PP_StudentHandout.pdf)

---

Make sure you understand the following equations before moving on. Ask questions!

- Number of distinct values that can be stored in  $b$  bits:  $2^b$
- Maximum value for an **unsigned** binary value using  $b$  bits:  $2^b - 1$
- Maximum value for a **signed** binary value using  $b$  bits:  $2^{b-1} - 1$
- Minimum value for a **signed** binary value using  $b$  bits:  $-2^{b-1}$

Questions:

1. Why do we subtract 1 in the expressions for maximum values?
2. Why do we subtract 1 in the **exponent** for **signed** values?

Working with your partner, use the vim editor to create a C++ file called `lab2_values.cpp`:

```
$ vim lab2_values.cpp
```

Refer to Lab 1 for a reminder of basic vim commands. You can also use `man vim`.

First, create a comment header at the top of your file that includes both of your names:

```
/*  
*****  
** Program: lab2_values.cpp  
** Authors: Both partners' names  
** Date: 01/13/2020  
** Description: Calculate min/max values.  
******/
```

(2 pts) Write a program to

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

How do you express an exponent? In C++, you can 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;  
}
```

Compile and run your program and confirm that it works.

```
$ g++ lab2_values.cpp -o lab2_values  
$ ./lab2_values
```

(1 pt) Now **switch partners** and expand your program to do the following:

- Store the maximum value for a **signed** 32-bit number in an `int` variable called `max_int` and print it out.
- **Add 1** to your integer and print it out.
- Compile and run your program.
  - What happened? Why? Discuss this with your partner/TAs until you know why.
- Store the minimum value for a **signed** 32-bit number in an `int` variable called `min_int` and print it out.
- **Subtract 1** from your integer and print it out.

- Compile and run your program.
  - What happened? Why? Discuss this with your partner/TAs until you know why.

---

## **(4 pts) D. Random Numbers and Conditional Statements**

**Switch partners** and create a new C++ file called `lab2_hotel.cpp`.

- Remember to create a comment header with both partners' names.

### **(1 pt) Generate random numbers**

First, let's learn how to generate random numbers. Type the following into your program:

```
#include <iostream>
#include <cstdlib> /* include to allow rand() to be used */

using namespace std;

int main()
{
    int x; /* variable to hold our random integer */

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

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

    return 0;
}
```

Compile, then run your program 3 times.

- What numbers did you get? Do they seem random? Discuss this with your partner.

### **Generate different random numbers each time**

Each run of your program will generate random numbers, starting from the **seed**. If you don't specify a seed, it uses the same one on each run. To get different random numbers each time you run, you can specify a different seed. A convenient way to do this is to use the current time (which is always changing).

Add this library to the top of your file:

```
#include <ctime> /*include to allow time() to be used*/
```

Add a statement inside `main()`, before you call `rand()`, to seed the random number generator using the current time:

```
srand(time(NULL)); /* seed the random number generator */
```

You only need to do this once, not before every call to `rand()`.

Compile, then run your program 3 times.

- What numbers did you get? Do they seem random? Discuss this with your partner.

### (1 pt) Generate random numbers in a desired range

Now you'll use random numbers to assign a hotel guest to a random floor. The ground floor is 0, the first floor is 1, etc. The highest floor is 5.

**Switch partners** and modify your program to choose a random integer between 0 and 5 based on the value returned from `rand()`.

- Create a new `int` variable called `hotel_floor`.
- Use the mod operator (`%`) to convert the output of `rand()` into the range of values you want.
- Print out the guest's hotel floor assignment.

### Change program behavior depending on a variable

Your goal is to modify the program so that it prints different output depending on `hotel_floor`.

- If the assigned floor is 0, print "You are assigned to the ground floor."
- Else if the floor is odd (1, 3 or 5), print "You are assigned to an odd floor."
  - Also, if the floor is 5, print "You are assigned to the penthouse!"
- Else it's even (2 or 4), so print "You are assigned to an even floor."

The program should still print the floor assignment itself (from the previous step).

**(1 pt) Before writing the code, first create a Design document** (each person should create their own, but you can discuss it with your partner). **This is good practice for Assignment 2!**

#### Example Design document:

[http://classes.engr.oregonstate.edu/eecs/winter2020/cs161-020/assignments/Example\\_Design\\_Document.pdf](http://classes.engr.oregonstate.edu/eecs/winter2020/cs161-020/assignments/Example_Design_Document.pdf)

#### Understanding the Problem:

- What is the goal? (State the problem/goal in your own words.)
- Do you understand everything in the problem? If not, what is not clear?
- 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? Your choice:
  - Write down the **steps** in pseudocode, or
  - Draw a **flowchart** (can be on paper)

#### Program Testing:

- Normally, you create a test plan with the test cases (bad, good, and edge cases), and write down the expected result in advance.
- This program has no user input, so you can skip it for this exercise.

**(1 pt) Pick one of your designs (yours or your partner's) and implement it with pair programming (Switch partners).**

Use `if/then` statements to print different results depending on the value in `hotel_floor`, as above. Use a **nested** `if/then` statement to achieve the "penthouse" output. Example nested `if/then`:

```
/* Earlier num was declared as an int */
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;  
    }  
    ...  
}  
...
```

Compile and run your program a few times. How often do you get into the penthouse?

---

**G. (1 pt) Submit your programs (lab2\_values.cpp and lab2\_hotel.cpp) to [TEACH](#)**

**Only one partner needs to submit. Both partners' names must be in the comment header to get credit.**

1. Transfer your .cpp files from the engr servers to your local laptop.
2. Connect to TEACH here: <https://teach.engr.oregonstate.edu/teach.php>
3. In the menu on the right side, go to **Class Tools -> Submit Assignment**.
4. Select **CS 161 020 Lab\_2** from the list of assignments and click "SUBMIT NOW".
5. Select your .cpp files (lab2\_values.cpp and lab2\_hotel.cpp).
6. Click the **Submit** button.
7. You are done!

---

**If you finish the lab early, this is a golden chance to start looking at Assignment 2 (with TAs nearby to answer questions!).**  
**Remember, the assignment you submit must be your work alone (no partners).**

---