Lab 2

Each lab will begin with 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. Full points may be received at the beginning of the following lab only if 7 or more points were earned for the lab in question. 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 Jennifer Parham-Mocello.

Pair Programming

In this lab we will be pair programming. Please find a partner to work with for the duration of the lab. **You must be checked off together. 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 20 minutes, you switch driver and navigator, continuing this pattern until the task is complete.

(2 pts) 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 mv ~/hello2.cpp . because you know that hello2.cpp is in your home directory.
   cp ../hello2.cpp .

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/winter2016/cs161-001/labs/CheatSheet.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/winter2016/cs161-001/links.html

(3 pts) Design

Design is very important when developing programs and there are a variety of ways to approach it. You may draw pictures, write it out in prose or structured text, use pseudo code, and more! The point of design is to give you a blueprint to follow while you are coding. This saves time debugging your program as you can catch mistakes early. It is better to spend one hour of designing than it is to spend five hours debugging.

For this lab you must design a solution to the following problem statement. You will be implementing your solution!

Problem Statement - Design

The local middle school would like some text adventure games to keep their students occupied during down time. The school is leaving it up to your skill and good judgement to develop a game. It is up to you what the story and theme is but there are some requirements:

*There must be at least three different paths or solutions to complete the adventure
*There must be an element of chance that would change a user's chosen path

You must get your design checked off before moving on to implementing. The paths for success should be obvious in your design. Take no more than one hour to complete the design.

(**5pts) Implementation**

In addition to design, you must implement your program as a C++ program. Here are some implementation requirements:

*There must be an empty line separating each navigation
*The choice system must be based on numbers
*You must use if/else statements and/or switch statements

An example of the run of the program looks like this:

Hello and welcome to the adventure!
To go right enter 1, to go left enter 2: 1

You chose to go right. You have now entered Scandinavia and are being hunted by friendly oxen.
Enter 1 to befriend the oxen, enter 2 to run from the oxen: 1

You attempted to befriend the oxen. You think you can ride them.
Enter 1 to attempt to ride the oxen, enter 2 to walk away: 1

Unfortunately fate was not on your side (due to a random number, coin flip, etc.), and you are forced to walk away.
The adventure has ended.

The rest of the implementation is up to you, but try to make your game as clean and attractive to play as possible.

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.

Show your completed game to the TAs with your partner. You will not get points if you check off as an individual.