CS 161 Lab #3 – Loops and Debugging

- 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.

Goals:

- Expand your list of Linux and Vim commands
- Practice using loops
- Practice identifying, locating, and fixing bugs

(3 pts) A. Lab Quiz (Canvas)

Visit this link on Canvas to take the Lab 3 quiz:
https://oregonstate.instructure.com/courses/1771939/quizzes/2532412
Re-take the quiz until you get all of the questions right! Canvas saves your last score, not your highest score. If you don't get 6/6 within the time available, finish outside of lab.

(2 pts) B. More Adventures with Vim

First, create a **`lab3`** directory in your **`labs`** directory, and change into the **`lab3`** directory.

```
$ cd labs
$ mkdir lab3
$ cd lab3
```

Now, let's practice using the vim editor following the instructions below.
(If needed, refer to Lab #1 for a reference guide to the basic commands)

1. Open a test file for practice:
   `$ vim_test.cpp`
2. Save the file using **:w** <Press Return>
3. Go into insert mode by pressing **i**
4. Type the standard input/output header and main function in your **`test.cpp`** file.
5. Switch from insert mode to command mode by pressing <Esc>.
6. Show the line numbers in vim by typing **:set number** (shortcut: **:set nu**)
7. Go to the third line (or whichever line has `main()`) in your code by typing **:3**
8. Delete this line of code by typing **dd**
9. Put the line of code back underneath the first line by moving your cursor to the top line and typing **p**
10. Redo the delete by typing **u** to undo the paste.
11. Paste the line back using **p**
12. Now, copy the line of code by typing **yy** (If you want to copy more than 1 line, you type one less than the number of lines you want to copy after the **y**, e.g., **y4** copies 5 lines).

13. Paste the line of code by moving your cursor to a line and typing **p**
14. Undo the paste by typing **u**
15. Try using the **j, k, h**, and **l** keys to move around the screen. You can also use the arrow keys, but these don't always work with all possible terminals/connections.
16. Search for "main" in vim with **/main**
17. *Switch from command mode to insert mode,* **i**, and type some random letters, such as **dkfjdsl**, then press <Esc> to *switch back to command mode*.
18. Use **x** to delete all the characters from the random string you typed.
19. Press **i** to *switch to insert* mode to begin typing text again.
20. Now write a small program that does the following:
    a. Ask the user if she/he likes vim as an editor. Instruct the user to type 1 if the answer is yes and 0 if the answer is no.
    b. Read the user's input and store it in a variable.
    c. If the user likes vim, then display a message, `"You love vim!"`
    d. If the user doesn't like vim, then display a message, `"You hate vim!"`
21. Press <Esc> to *switch back into command mode*, and go to the first line in your program by typing **:0**
22. Type **=G** to auto-indent your program. You can set the number of spaces you prefer by typing **:set sw=3**, and then re-auto-indenting with **=G**. (If you want to auto indent while you type, then use **:set cindent**)
23. You can change your color scheme or background by using one of these commands:
    :colorscheme evening
    :set background=dark
24. Write (save) the file and quit by typing **:wq**

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

In the next lab you will add some of these useful commands to a vim system resources file so you don't have to re-type them every time you run vim.

C. Create a Password Generator

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

STOP: You can do the Design and Implementation individually or with a partner. Do pair with someone if you feel you are struggling with concepts or simply want to bounce ideas around with someone else. Pair programming can be fun!

(2 pts) Design

Design is very important when developing programs, and there are a variety of ways to approach it. You can draw pictures, describe your design in prose or in structured text, use pseudocode, and more. The point of design is to give you a plan to follow while you are coding. This saves time later debugging your program because you can catch mistakes early (sometimes before any code is written). 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 implement your solution only after finishing the design.

Problem Statement

A good password has many requirements. Humans have a hard time meeting these requirements on their own. You are tasked with creating a program that will generate a strong password based on what the user wants in the password.

The user should be able to choose if they want a password with:

- upper-case letters (A to Z)
- lower-case letters (a to z)
- numbers (0 to 9)

The user should also provide the length of the password and how much of the password should be comprised of their selections**. All upper-case letters come first, then lower-case letters, then numbers.** For example:

```
Welcome to Password Creator!

How many characters do you want the password to be? 10
How many letters (out of 10)? 5
How many upper-case letters (out of 5)? 2
The rest of the letters (3) will be lower-case.
The rest of the characters (5) will be numbers.

Your random password is: ACbzd12594

Would you like to create another password (0-no, 1-yes)? 1

How many characters do you want the password to be? 10
How many letters (out of 10)? 0
The rest of the characters (10) will be numbers.

Your random password is: 7328628592
Would you like to create another password (0-no, 1-yes)? 0
```

Your Design

(1 pt) Draw a flowchart to show how this program will execute. Use rectangles for command/statements and diamonds for decision points. Use arrows to connect elements in the flowchart.

(1 pt) Write down at least 6 test cases that include at least 1 good, 1 bad, and 1 edge case. The first row shows an example.

Test setting / prompt	User input	Expected result
How many upper-case letters (out of 6)?	3	Password will contain 3 upper-case and 3 lower-case letters.

Your design (flowchart and test cases) must be checked off by a TA before proceeding to implementation.

(3 pts) Implementation

To achieve the outermost structure (repeatedly asking the user if they want to create a password), you may find a `while` loop to be useful. The `while` loop is structured like an `if` statement, but it continues to repeat a block of code while the condition is true, instead of doing it only once.

```
int input;
cout << "Enter a number greater than 1: ";
cin >> input;
while (input <= 1)
{
  cout << input << " is not a good number!" << endl;
  cout << "Enter a number greater than 1: ";
  cin >> input;
}
/* now we know we have good input greater than 1 */
```

Notice the indentation to show which lines are part of the `while` loop.

In contrast, to generate characters and numbers for the password length specified by the user, a `for` loop will be more suitable because we know (from the user) exactly how many times to execute the loop. A `for` loop is used to repeat something a specific number of times.

```
for (int i=0; i < pass_length; i++)
{
  /* do one thing */
  /* do some more things */
  /* these commands will be repeated each time through the loop */
}
```

Notice the indentation to show which lines are part of the `for` loop.

This is what is going on with the pieces of information in the `for` loop:

Step 1: create and initialize a counter variable `i`

Step 2: check if the counter variable `i` is less than `pass_length`

If so: **Step 3**: do what is in the curly braces and part of the for loop

Step 4: increment the counter variable `i` using `i++`

Step 5: go back to Step 2 to check the counter variable `i` again

If not: Step 3: do not repeat and go to the bottom of the for loop

In your implementation, you must use the ASCII chart, and you must use `rand()` and `srand()` to generate random values. Think about how we used rand to get a range of numbers in class. Notice characters and numbers are in ranges in the ASCII chart: http://www.asciitable.com/

D. Submit your program (lab3_password.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 file 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_3** from the list of assignments and click "SUBMIT NOW".
5. Select your .cpp file (`lab3_password.cpp`).
6. Click the **Submit** button.
7. You are done!

Point totals: 3 pts (quiz) + 2 pts (vim) + 2 pts (design) + 3 pts (implementation)

If you finish the lab early, this is a golden chance to continue working on Assignment 2 (with TAs nearby to answer questions!).

Remember, the assignment you submit must be your work alone (no partners).
