CS 161 Lab #4 – Functions 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:

- Enable you to customize `vim`
- Practice identifying, locating, and fixing bugs
- Practice writing and using functions

(3 pts) A. Lab Quiz (Canvas)

Visit this link on Canvas to take the Lab 4 quiz:

https://oregonstate.instructure.com/courses/1771939/quizzes/2533200

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.

(1 pt) B. Customizing `vim`

For this part of the lab, you will create a `.vimrc` file that will allow you to customize `vim`'s behavior when you write and edit C++ programs. First, create and open a simple `.vimrc` file **in your home directory** on the ENGR (flip) server:

```
$ vim .vimrc
```

In this file, you can insert the following lines to make working with `vim` easier. Add one or more to your `.vimrc`, save the file, quit `vim`, and then re-start `vim` to see the changes. Keep the options you like. Comments are prefaced with a double-quotation mark, ".

```
filetype on
filetype plugin on
filetype indent on
autocmd FileType c, cpp
set cindent                "Use c-like indentation
set sw=3                   "Set the sw for the auto indent to 3 spaces
set number                 "Show the line numbers on the left

"Change the color of the text and turn on syntax highlighting:
"color desert              "Uncomment this option to activate it
color torte
colorscheme evening
```

```
syntax on                   "Turn on syntax highlighting
set showmatch               "Show matching braces
set showmode                "Show current mode

"When the opening character of a string, code block {},
"or function call () is typed, vim will automatically add
"the closing character and move the cursor left for easy editing.
inoremap ' ''<Left>         "These are single quotes
inoremap " ""<Left>         "These are double quotes
inoremap { {}<Left>
inoremap ( ()<Left>
```

There are many more commands you can insert in this file. Here is a reference guide to some of them: http://vimdoc.sourceforge.net/htmldoc/starting.html

To get checked off for this section, show a TA your `.vimrc` file and tell him/her which option(s) you find the most useful.

(2 pts) C. Testing and Debugging

Test your skills as a bug detective!

1. Download the following cpp file.

http://classes.engr.oregonstate.edu/eecs/winter2020/cs161-020/labs/buggyCode.cpp

You can transfer it directly to your ENGR account from linux:

```
$ wget http://classes.engr.oregonstate.edu/eecs/winter2020/cs161-020/labs/buggyCode.cpp
```

2. Add a proper **file header** with both partners' names, assignment, etc.

3. Find as many bugs as possible in this file and fix them. Some are logic errors, and some are syntax errors. Hint: there are 21 bugs, of varying complexity.

- **Add a comment indicating what you changed/fixed for each bug.**
- Use your **bug detection tools:**
  - Visual inspection
  - Compiler error interpretation
  - Create test cases and see if the output matches your expectation
- Use your **bug localization tools:**
  - Compiler error line numbers
  - `cout` to inspect variable values during program execution
  - `cin` to pause program execution
  - `assert()` to declare what must be true (and the program will halt if not)
  - Trace through the code – read it out loud and make sure it does what you expect
  - Comment out problematic code to isolate it

Be sure to re-compile and run the program after you make each single fix to a mistake to make sure you correctly fixed the mistake, didn't introduction new errors, and/or inadvertently eliminated other errors as a result of the fix.

(2 pts) D. Design Functions to Operate on Characters

Pair Programming: In this lab, you can **choose a partner** for pair programming on sections D (design) and E (implementation). 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.

Working with your partner, develop a mini-design for each of the following character operations. Your mini-design for each one will describe the operation of a function. For each one, write down your understanding of the problem, your flow chart or pseudocode, and at least 5 test cases.

- **is_vowel(char c): return true if c is a vowel and otherwise false.**
  You can assume that the input is lower case ('a' through 'z').
- **swap_case(char c): returns character c in the opposite case. That is, if c is lower-case then return the upper-case version of c, and vice versa**.
  Examples (these can be test cases): a -> A, Q -> q.
  You can assume the input is a valid letter ('A' through 'Z' or 'a' through 'z').

Note that you can compare character variables using relational operators like <, >, <=, >= to figure out what case character c is.

(2 pts) E. Implement Functions to Operate on Characters

Once your designs are checked off, proceed to implementation.

1. Create a file named `lab4_char_functions.cpp` .
2. Create a `main()` method that will call your functions using each of the test cases from your design.
3. Implement the `is_vowel()` and `swap_case()` functions **above** `main()` in the file. Include the function headers that are defined for you below.
4. Compile and run your program. Check your test cases and refine your program until all of them pass.

Switch partners every ~15 minutes, and have fun!

```
/*************************************************************
 ** Function: is_vowel
 ** Description: check whether a character is a vowel
 ** Parameters: a single character, c
 ** Pre-conditions: c is a lower-case letter (a-z)
 ** Post-conditions: return true if c is a vowel and otherwise false
 *************************************************************/
bool is_vowel(char c)
{
  ...
}
```

```
/*************************************************************
 ** Function: swap_case
 ** Description: generate and return the opposite case character
 **              (lower-case => upper-case and vice-versa)
 ** Parameters: a single character, c
 ** Pre-conditions: c is a lower-case (a-z) or upper-case (A-Z) letter
 ** Post-conditions: if c is lower-case, return upper-case C,
 **                  and vice-versa
 *************************************************************/
char swap_case(char c)
{
  ...
}
```

E. Submit your programs 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_4** from the list of assignments and click "SUBMIT NOW".
5. Select your .cpp files (`lab4_buggy_code.cpp, lab4_char_functions.cpp`).
6. Click the **Submit** button.
7. You are done!

Point totals: 3 pts (quiz) + 1 pt (.vimrc) + 2 pts (debugging) + 2 pts (design) + 2 pts (implementation)

If you finish the lab early, this is a golden chance to work on your Assignment 3 design (with TAs nearby to answer questions!).

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