CS 161 Assignment #4 – Text Surgeon

Design Document due: Sunday, 2/16/2020, 11:59 p.m. (Canvas)
Design Peer Review due: Weds., 2/19/2020, 11:59 p.m. (Canvas)
Assignment due: Sunday, 2/23/2020, 11:59 p.m. (TEACH)
Demo/Grade Your Assignment by: 3/8/2020, 11:59 p.m. (In person)

Goals:

- Practice good software engineering design principles:
  - Design your solution before writing the program
  - Develop test cases before writing the program
  - Run tests and assess results to iteratively improve the program
- Use one-dimensional arrays to store information
- Use string functions to access and analyze characters within C-style strings
- Allocate memory from the stack and the heap, and clean up memory when done
- Use functions to modularize code to increase readability and reduce repeated code

Part 0. Sign Up for Your Demonstration

You are required to **meet with a TA within the two weeks following the due date** to demo. Please sign up for your slot now (yes, before you begin the assignment!) to ensure your work will be graded. You can schedule a demo with a TA on the course website (http://classes.engr.oregonstate.edu/eecs/winter2020/cs161-020) in the far right column of the bottom table labeled "Grading Hours". You can select one of your lab TAs, or a different TA.

Demos for Assignment 4 will be 15 minutes long (not 10 minutes).

Submissions that do not compile on the ENGR servers will receive a 0 for the implementation part. Please test your code before submitting! Comment out parts that do not work if necessary.

Your goal in the demonstration is to show your TA how to use your program, explain how it works, and answer questions. Be proud of your work and ready to show it off!

Part 1. (10 pts) Design a Text Surgeon

This is your chance to write functions that analyze, manipulate, and modify C-style strings. You will need to use good memory management to avoid your program filling up the computer's memory.

Design a program that reads in a sentence or paragraph (max length 1027 characters) from the user, stores it as a C-style string (think: how big should this be?), and gives the user several options for operations to perform on their text (or to quit). After performing an operation, the user should be prompted to select another operation to perform on the same string (until they select the option to quit).

To get full credit, implement 3 of the following 4 operations, plus #5 (create your own operation). If you implement all 5, you will receive extra credit.

1. **Check if the number of vowels equals the number of consonants**. Ignore any non-letter characters.
   Example: User enters "`banana`"; program prints "`# vowels = # consonants`."
   Example: User enters "`ban ana!`"; program prints "`# vowels = # consonants`."

Example: User enters "`zip`"; program prints "`# vowels != # consonants.`"
Example: User enters "`@#$!57`"; program prints "`# vowels = # consonants.`"

2. **Letter swap:** the user inputs two letters (should be from a-z or A-Z), and the C-style string is **modified** to replace the first letter with the second letter.
   Example: User enters "`Off we go, into the wild blue yonder!`". The program prompts the user for two characters: one to find and one to replace the first one with. The user enters characters '`o`' and '`3`'. The program **modifies the string** to: `Off we gi, inti the wild blue yinder!` and prints it out. Letters are case-sensitive (so "`Off`" did not change). If the first character does not exist in the string, it does not change.
3. **Temporarily flip (reverse) the C-style string:** The program should **create a new C-style string from the heap**, rather than changing the user's input string. Remember to clean up your heap! (After printing out the new string, the program should delete it.)
   Example: User enters "`ban ana!`"; program prints: "`!ana nab`"
   Example: User enters "`zip`"; program prints: "`piz`"
4. **Compute character frequency distribution:** The program prompts the user for a second string that contains from 1 to 10 characters of interest, stores it in a char array, then dynamically creates an integer array to store the number of occurrences of each character in the first string. Remember to clean up your heap!
   Example: User enters "`banana`", then the string "`an5`", which is stored in a char array. The program creates an integer array containing the count for each character and then prints the distribution showing each character that was counted and its count:
   ```
   a) 3
   n) 2
   5) 0
   ```
5. **Propose your own text operation here:** Design another text operation that interests you. Your operation could report properties or statistics of the user's string (as in #1 and #4), or it could modify the user's string (as in #2), or it could create new strings (as in #3). Consider what you can do with color, random variables, etc. **Have fun and be creative!**

Implementation requirements:

- No use of string type variables, or any string functions, from the C++ string library. **You must use `char[]` arrays to store your strings, and functions from the <cstring> library.** See this useful reference: https://www.cprogramming.com/tutorial/lesson9.html
- To modify individual characters in a C-style string, use array indexing: `c_string[3] = 'r';`
- No use of global variables, goto, recursion, or classes/objects.
- Use good memory management. Delete memory off the heap when you are done using it.
  - Use valgrind to check for memory leaks, and then fix them.
- Use functions to divide the program into logical units. No function should be more than 25 lines long (*excluding lines that consist only of comments and whitespace*). If you find your functions are getting too long, break them into multiple smaller functions.
  - Do not put more than one statement on each line.
- Based on your choice of which operations to implement, use these exact prototypes:
  - ```
    /* return true if #vowels == #consonants */
    bool check_vowel_cons(const char* s);
    ```
  - ```
    /* change char a to char b in s */
    ```

```
void letter_swap(char* s, const char a, const char b);
```
- ```
  /* flip order of s and return it in a new char* string */
  char* flip_str(const char* s);
  /* OR flip order of s and store the new string in s2 */
  void flip_str(const char* s, char* s2);
  ```
- ```
  /* return int array with #occurrence for each char in chars */
  int* count_chars(const char* s, const char* chars);
  ```
- ```
  /* your new operation here! */
  <return type> <name>(<parameters...>);
  ```

- You will probably want to create more functions as helpers that perform smaller parts of each operation, like:
  - `bool is_vowel(const char c);`
  - `char* strip_string(const char* s);` create a new version of string s that consists only of letters (no numbers, punctuation, etc.)
  - and so on.
- When there is a constraint on what the user can input, check that they met the requirements (e.g., letter_swap() only swaps letters a-z or A-Z, not all characters), and prompt the user to re-enter an input if it isn't valid.
- Employ good programming style and follow the course style guidelines.

Your first step is to **write a Design Document** to plan your program. You can start on this right away, since it does not require any programming, just thinking.
Consult the Design and Peer Review Guidelines:
http://classes.engr.oregonstate.edu/eecs/winter2020/cs161-020/assignments/cs161-design-review-guidelines.pdf
and the sample Design Document:
http://classes.engr.oregonstate.edu/eecs/winter2020/cs161-020/assignments/Example_Design_Document.pdf

Your Design Document must include three sections:

1. **Describe the problem** in your own words (what does your program need to do?) and any assumptions you are making to help make the problem more concrete.
2. **Devise a plan**: design your solution and show how it will work with either (1) pseudocode (not C++) and/or (2) a flowchart diagram. You can create your flowchart using software or sketch it on paper/whiteboard, take a picture, and insert it into your Design Document. **You should have a separate section of pseudocode, or a flowchart, for each function**.
3. **Identify at least 8 test cases**. Each one should include
   - a user string
   - the operation they chose
   - what the program output should be.

   Your tests must be different from the examples included above. The more tests you think of now, the better your program will be later.

Submit your Design Document on **Canvas** (not TEACH):
https://oregonstate.instructure.com/courses/1771939/assignments/7820844

Part 2. (10 pts) Provide a Peer Review of 2 Text Surgeon Designs

You can work on this part in parallel with your work on Part 3. In this part, you will provide feedback on your classmates' designs (and receive feedback on your own design to help improve your final submission). **Your peer reviews do not affect the other student's grade.**
  ```

Canvas will randomly assign you two other students' design documents to review. Please enter your review as **comments (not rubric)** on the Canvas page where their submission is.

Peer reviews must be written **constructively and positively**. Your goal is to identify good aspects of the design, plus ways it can be improved. Negative, harsh, or mean comments are not appropriate and will cause you to lose points.

1. (3 pts) Describe the problem:
   - Does the document list assumptions? Do you have suggestions to clarify the assumptions?
   - If any parts are unclear, write down questions that will help the student clarify their description.
2. (3 pts) Devise a plan:
   - Can you understand the pseudocode or flowchart diagram? If hand-drawn, is the handwriting legible? Does it match the planned design? Are there parts that seem incomplete or confusing?
   - Write down at least one suggestion for how to improve the design. **There is always a way to improve any design!**
3. (4 pts) Test cases:
   - Do you understand the test cases, and do they match with the design as described? If so, write "All test cases look good." If not, suggest changes or improvements to clarify the test cases.
   - Propose one new test case not included by the student (e.g., use one of their "Test case settings" but change what the user types). You do not need to write the "Expected result" for this test case because that is up to the student to decide.

When you receive your peer reviews, **read them carefully and decide if you would like to modify your design (and implementation).** Often other people will spot issues you did not anticipate!

Part 3. (110 pts) Implement Your Text Surgeon

Implement your Text Surgeon in C++ following the design you have developed. **Note: It is normal (in fact, expected) that your design will evolve as you write the program and figure out new details.** The design provides a starting point, but you can deviate from it.

- Name your file `assign4_text.cpp`
- Before you submit, read the peer reviews you received and ensure that you have addressed any issues they identified. This will help you improve your score on the assignment.
- Ensure that your program (1) compiles and (2) does not generate run-time errors.

Part 4. (25 pts) Analyze Your Work (in a Word/text file, section "Part 4")

(A) (2 pts) As you worked to implement your Text Surgeon, you probably thought of new test cases after your Design Document was already submitted. List one here. (If you didn't think of any new test cases while implementing, create one now.)

(B) (9 pts) Try each of your 9 test cases (there are 9 now) and for each one, report whether or not the behavior is as expected. If not, state whether (1) your design has evolved and now the expected behavior is different (state what the new expected behavior is) or (2) this test helped you find (and fix) a bug in your program.

- If you didn't have 8 test cases in your Design Document, make them up now to allow you to get full credit here.

(C) (5 pts) When you ran valgrind, did you find a memory leak? If so, copy and paste valgrind's message about the memory leak. If not, create a memory leak on purpose to generate a valgrind message for you to copy.

(D) (5 pts) How did you fix the memory leak you found? (If you created it on purpose, describe how you created it.) Either way, explain what was wrong with the C++ code. Fix the error before submitting your code.

(E) (4 pts) What was the hardest part of this assignment?

Part 5. Extra Credit (write in the Word/text file, section labeled "Part 5")

- **(up to 2 pts)** Implement all 4 of the operations described above (plus your custom operation), instead of only 3 (plus your custom operation.
- (**up to 2 pts**) Ask another person (friend, family member, roommate, coffee shop stranger) to try out your program. In "Part 5" of your file, copy/paste their full interaction (program outputs and what they typed). Did the program crash or behave in a way you did not expect? Describe anything you found surprising.
- (**up to 2 pts**) Describe one improvement you would recommend to make your program better (more robust, more entertaining, more attractive, anything).
- (**up to 2 pts**) Add another operation to your Text Surgeon that counts the occurrences of a given **word** (prompting the user for this word too):
  `int count_word(char* s, char* word);`

  In this setting, a "word" is defined as a sequence of letters and or numbers not including whitespace (space, tab, newline, etc.), punctuation mark, or other non-letter, non-number character. Results are not case-sensitive.
  Example: User enters "`Off we go, into the wild blue yonder!`", then enters word "`off`"; program prints:
  `1 occurrence of "off".`
  Example: User enters "`The heart of the eye of the world`", then enters word "`the`"; program prints:
  `3 occurrences of "the".`

Submit Your Assignment Electronically

(A) Before you submit, read the peer reviews you received and ensure that you have addressed any issues they identified. This will help you improve your score on the assignment.

(B) Ensure that your program (1) compiles, (2) does not generate run-time errors, (3) has no memory leaks.

(C) Submit your C++ program (.cpp file, not your executable) and your Word/text file (**must be converted to a PDF file**) with written answers (Part 4 and optionally Part 5) before the assignment due date, using **TEACH**.

(D) Remember to sign up with a TA on the course website to demo your assignment. **The deadline to demo this assignment is 03/08/2020.**