Assignment 4 Palindrome Detector and Word Frequency

Design Due: Sunday, 11/17/19, 11:59pm on Canvas

Peer Review 4 Due: Friday, 11/22/19, 11:59pm on Canvas

Code Due: Sunday, 11/24/19, 11:59pm on TEACH

Make sure you demo Assignment 3 within two weeks of the due date to receive full credit. If you go outside the two-week limit without permission, you will lose 50 points. If you fail to show up for your demo without informing anyone, then you will automatically lose 10 points.

Problem statement:
You are tasked to write a program that will detect palindrome and word frequency in a sentence (or paragraph) based on the following user choices:

1. **Palindrome Detector**
2. **Frequency of all words**
3. **Frequency of given words**

The program will first ask the user for a C-style string input (array of characters ended by the null character, '\0') that is a sentence (or paragraph) less than 1024 letters, then ask the user for choices 1-3, and output the result accordingly. At the end of each output, the user should be asked if they want to run the program again or exit the program. The program must handle all types of bad inputs from user and recover from the errors. Detailed requirements for each choice are listed below:

1. Palindrome Detector

- A palindrome is a word, phrase, or sequence that reads the same backwards as forward. E.g., racecar, madam.
- Determine whether the input C-style string is palindrome or not. Examples of palindromes that your code needs to detect:
  - "119Radar911" (note the capital letter and numbers)
  - "top spot" (note that your code will need to ignore the space)
  - "Was it a car or a cat I saw?" (note that your code will need to ignore the special character, '?')
- Suggested functions:
  - `bool is_palindrome (char *str);` //return true if str is palindrome, false otherwise
  - `char * purge_string (char *str);` //accepts a c-string, and returns a version where all spaces and special characters have been removed
    (Note: the returned c-string must be on the heap, otherwise this c-string will be deleted once we leave this function)
    OR you may use this:
    `void purge_string (char *str, char* str_new);` //accepts two c-strings, remove all spaces and special characters in str, and store the new string into str_new

2. Frequency of all words

- Letters in words are case insensitive, meaning "word" and "WoRD" are the same
- Ignore all numbers and special characters in the sentence (or paragraph), meaning "wo32rd," "wo]]]rd343.34" and "word" are the same
- Use a static C++ string array to store all words (you can assume that number of words is less than 256)
- Output the frequency of all words in the sentence (paragraph)

3. Frequency of given words

- Letters in words are case insensitive, meaning "word" and "WoRD" are the same word

- Ignore all numbers and special characters in the sentence (or paragraph), meaning “wo32rd,” “wo]]]rd343.34”, and “word” are the same
- Prompt the user for N words that the user wants to search for in the sentence (or paragraph)
- Use a dynamic C++ string array allocated on the heap to store N words
- Output the frequency of N given words in the sentence (paragraph)

***Important note:**
You are not allowed to convert a C-style string to a C++ string object. In other words, you have to treat the input sentence (paragraph) as a C-style string and parse through it.

For frequency of given words, the N words must be entered at one time before searching to see if the words are in the sentence/paragraph. In other words, you cannot ask for a word, search its frequency and then ask for another word. You must ask for all words before searching for the words. You must use a dynamic array allocated on the heap!!! You will **NOT** be given credit for a variable length array, which is not dynamically allocated on the heap, i.e. `string array[num_words];` **is not accepted for choice 3!!!!** In addition, you must not have a memory leaks (use valgrind to help)!

Extra Credit:
(Option 1, 10pts) File input/output: Instead of reading the entire sentence (or paragraph) from the user, the program takes a text filename as input. Your program should first detect whether the file exists or not. If the file does not exist, you need to re-prompt the user until a valid filename is given. If the file exists, then the program stores the content of that .txt file into a C-style string and do the rest. For choices 2 and 3, instead of printing the frequency to the screen, you save them into a text file, called out.txt. Once the program ends, the user can view frequency of words in the out.txt file.

(Option 2, 10pts) Use C-style strings for words: For choice 2 and 3, use an array of C-style strings for the words, instead of C++ string objects. (**Note: Either make the words C-style strings for extra credit OR an array of C++ strings, not both)

Design Document – Due Sunday 11/17/19, 11:59pm on Canvas
Refer to the Example Design Document – Example_Design_Doc.pdf

Understanding the Problem/Problem Analysis:
- What are the user inputs, program outputs, etc.?
- What assumptions are you making?
- What are all the tasks and subtasks in this problem?

Program Design:
- What does the overall big picture of each function look like? (Flowchart or pseudocode)
  - What data do you need to create, when you read input from the user?
  - How to name your variables?
  - What are the decisions that need to be made in this program?
  - What tasks are repeated?
  - How would you modularize the program, how many functions are you going to create, and what are they?
- What kind of bad input are you going to handle?

Based on your answers above, list the **specific steps or provide a flowchart** of what is needed to create. Be very explicit!!!

Program Testing:

Create a test plan with the test cases (bad, good, and edge cases). What do you hope to be the expected results?

- What are the good, bad, and edge cases for ALL input in the program? Make sure to provide enough of each and for all different inputs you get from the user.

Electronically submit your Design Doc (.pdf file!!!) by the design due date, on Canvas.

Program Code – Due Sunday, 11/24/19, 11:59pm on TEACH

Implementation Requirements:

- Your user interface must provide clear instructions for the user and information about the data being presented
- Use of references or pointers is required.
- Use of C-style string is required
- Use of 1D dynamic array is required
- Libraries allowed to use: <iostream>, <string>, <cstring>, <cstdlib>, <cmath>, (<fstream> if doing extra credit 1)
- Your program should be properly decomposed into tasks and subtasks using functions. To help you with this, use the following:
  - Make each function do one thing and one thing only.
  - No more than 15 lines inside the curly braces of any function, including main(). Whitespace, variable declarations, single curly braces, vertical spacing, comments, and function headers do not count.
  - Functions over 15 lines need justification in comments.
  - Do not put multiple statements into one line.
- You are encouraged to use the functions in assignment 2 to do error handling.
- No global variables allowed (those declared outside of many or any other function, global constants are allowed).
- You must not have any memory leaks
- You program should not have any runtime error, e.g. segmentation fault
- Make sure you follow the style guidelines, have a program header and function headers with appropriate comments, and be consistent.

Electronically submit your C++ program (.cpp file, not your executable!!!) by the code due date, on TEACH.

Remember to sign up with a TA on Canvas to demo your assignment. The deadline to demo this assignment without penalty is 12/6/19.