CS 220 – Data Structures and Systems Programming

PEX 1 – Debugging Word Clouds – 75 Points

Due @ 2300 on Lesson 6
M-Day: Wed, 22 Jan 2020
T-Day: Thurs, 23 Jan 2020

Help Policy:

AUTHORIZED RESOURCES: Any, except another cadet's program.
NOTE:

- Never copy another person's work and submit it as your own.
- Do not jointly create a program unless explicitly allowed.
  - **Do not share your error findings with other cadets.**
- You must document all help received from sources other than your instructor or instructor-provided course materials (including your textbook).

Documentation Policy:

- You must document all help received from any source other than your instructor or instructor-provided materials, including your textbook (unless directly quoting or paraphrasing).
- The documentation statement must explicitly describe WHAT assistance was provided, WHERE on the assignment the assistance was provided, and WHO provided the assistance, and HOW it was used in completing the assignment.
- If no help was received on this assignment, the documentation statement must state "None."
- If you checked answers with anyone, you must document with whom on which problems. You must document whether or not you made any changes, and if you did make changes you must document the problems you changed and the reasons why.
- **Vague documentation statements must be corrected before the assignment will be graded and will result in a 5% deduction on the assignment.**

Turn-in Policies:

- On-time turn-in is at the specific day and time listed above.
- Late turn-ins will be accepted up to one week after the due date. The late penalty is a 50% cap on the points you can earn for this assignment.

OBJECTIVES

Upon completion of this programming exercise, students will:

- Demonstrate understanding of file I/O
- Demonstrate debugging proficiency

1. BACKGROUND

A word cloud, (or weighted list/tag cloud) in visual design, is a novelty visual representation of text data. It is typically used to depict keyword metadata (tags) on websites, or to visualize free form text. Tags are usually single words and the importance of each tag is shown with font size or color. This format is useful for quickly perceiving the most prominent terms and for locating a term alphabetically to determine its relative prominence. See image below.

You have been provided an application that opens an article in text format, parses the words, builds a word cloud in html, and displays a sorted list of words to the console minus those identified in an excluded list. The application in its provided form contains several errors.

Your task for this programming exercise is to develop tests to identify and fix any errors found in the word cloud application. Additionally, you must document the errors you find and how you fixed them.

2. PROGRAMMING EXERCISE

For this PEX, you are required to thoroughly test and debug the word cloud application, and document the errors along with your fix actions. The provided source code contains several errors (compile, run-time, and logic) that need to be corrected to ensure a proper word cloud is produced.

When all the errors have been corrected, your console output and html output will match those for the provided text files (USAFA.txt and Graphics.txt).

The program can be run from a terminal window using the following (replace programName with the name of your executable and storyfile.txt with the file to have its words counted):

programName storyFile.txt [ignoredWordList] [outputHTML]

Alternatively, you can configure CLion with the required arguments and not make use of a terminal.

Finally, you will write a short paragraph that paraphrases the ethics.txt article and a one-sentence explanation of the first line of the PEX1.h file.

3. PEX HINTS

- When you run the code, if the process finishes with an exit code other than 1 there are still errors to be found.
- There are no errors in the `fprintf()` function lines.
- Remember that preprocessor definitions are replaced prior to compile time and when used for a constant are typically all capital letters. This PEX uses `#defines STRING char*` and that is not an error.
- Using the play button instead of the debugger will hamper your efforts. The debugger will show segmentation faults more clearly.
- When allocating space for a string, consider how the end of a string is identified.
- Values assigned to an array must be assigned one element at a time. Remembering that a string is an array of characters, consider how to handle this for a string.

4. DEBUGGING THOUGHTS

- Keep a cool head – Errors have almost no reasoning abilities, and you are smarter than they are. Getting frustrated or angry will only make it harder to find the issue.
- Read/Research error messages – This is critical! Error messages can be intimidating, but "the gibberish" contains clues into what is happening. This is especially true in very long stack traces, where you may have to scroll past a page of confusion before finding a reference to your own code.
- Make one change at a time –If you have an error, try a fix, compile, and see if it's fixed. If you have two errors, fix one, compile, test, then go fix the other error. Don't try to fix all your errors at once or you might wind up with more trouble.
- Keep a log of failed attempts – Keeping a log of changes that were attempted to fix an error will prevent you from attempting the same fix twice. Especially if you've had to step away from the code for a while.
- Recompile Constantly – Also known as find errors early! If you spend 5 hours writing code, then you compile it and nothing works, then you may have to sort through 5 hours of work to figure out where you went wrong. *Compile and test as often as possible*. Unit testing can also help with this.
- Read your code (or have someone else read it for you) – Often, when you read your own code, you read it as you meant to write it, not as it's actually written. So it can be hard to catch little things like off-by-one errors. If you take some time away from your code, then read it fresh, or have someone else read it (without your prompting them on what it does), then often you might find your code doesn't do exactly what you thought it did.
- Walk through the code manually – Trace through the code on a piece of paper or white board. Only evaluate one line at a time and look for calculated values vs expected values. This ensures that the code is behaving as expected.
- Correct known problems before tackling unknown errors – When confronted with many errors, you might spot 1-2 that you know the cause of. It's tempting to leave those for later and tackle

the unknown errors first. But sometimes the unknown errors are byproducts of the known errors. Or maybe the known errors make the unknown errors more complicated or confusing.

5. SUBMISSION REQUIREMENTS

- A well-written documented error report submitted via the assignment on Canvas:
  - Type of error
  - Brief description of error/fix
  - File and line of code error occurred on
- Zipped up project containing corrected versions of:
  - PEX1.c
  - PEX1.h
  - WordCloud.c
  - WordCloud.h
  - .html output files for USAFA.txt and Graphics.txt
  - CMakeLists.txt

PEX 1 Grade Sheet **Name:**_____

Criteria	Points Available	Earned
Article Summary	**7%**	
Summary of the ethics.txt article and description of line 1 of PEX1.h	**5**	
Functionality	**93%**	
Compile errors fixed	**15**	
Run-time errors fixed	**30**	
Logic errors fixed	**25**	
Subtotal	**75**	
Vague/missing documentation statement (-5%)	**– 4**	
Submission requirements not followed (-5%)	**– 4**	
Late penalties – 50% Cap	**max score=37.5**	
Total	**75**	

Comments: