CSCE 2014 – Programming Project 5

Midpoint Due Date – Apr 10, 2017 at 11:59pm
Final Due Date – Apr 17, 2017 at 11:59pm

1. Problem Statement:

I am sure you have all used Google to search for documents, but have you ever wondered how Google search works? At a high level they do the following:

- Visit every web page on the internet using a "spider"
- Get information about all of the words in each web page
- Build a giant index using this word information
- Process user queries to find the web pages that match the query words
- Charge a lot of money for the targeted ads that appear on the results page

In this assignment, we will be working on the second step: using file processing and sorting to get statistical information about the words that appear in a web page. To do this, your program must do the following:

- Read an ASCII file and extract the words
- Store these words in an array of strings
- Sort the array of strings in alphabetical order
- Process the array to count how many times each word occurs
- Print out the counts for all words
- Sort the (count, word) pairs in descending order of count
- Print the top N (count, word) pairs based on their counts

For example, if the document contains the following text from the Dr. Seuss Green Eggs and Ham book:

> I would not like them
> here or there.
> I would not like them
> anywhere.
> I do not like
> green eggs and ham.
> I do not like them,
> Sam-I-am.

After sorting, you should have the following array of words: "I I I I I Sam am and anywhere do do eggs green ham here like like like like not not not not or them them them there would would". Notice that we have removed all non-alphanumeric

characters from the input. When you convert this into (count, word) pairs you would get (5, I), (1, Sam), (1, am), (1, and) … (2,would). After sorting these pairs in descending order based on the count values, you would get the following:

(5, I)
(4, like)
(4, not)
(3, them)
(2, do)
(2, would)
etc.

If the user specifies N = 4, then your program would print the top four words based on their frequency in the input file. The “secret sauce” in any search engine is to use this information to quickly find documents that match your input query. You can learn more about how to do this in an information retrieval class.

To test your program, do a Google search for a topic that you are interested in, and save the web page as an html document. Then run your program on this document to see what the "top ten" most common words were, and how many times your query words appeared. Were your query words in the top ten? Try your experiment with a second query, and see if you get similar results. Include your findings in your project report.

2. Design:

Your first design task is to figure out how to read the ASCII file into an array of words. One option is to declare a string variable str and use "din >> str" to read the file. This approach will require some work to remove non-alphanumeric characters from each string. Another option would be to declare a char variable ch, and use "din.get(ch)" to read one character at a time, and build strings up one letter at a time. A third option would be to use din.getline() method to process the input file one line at a time. You are welcome to use any approach you wish.

Your second design task is to sort the array of words. Here you are allowed to use ANY sorting method described in the book, in the lectures, or the labs. You are NOT allowed to use the built-in C++ sort algorithm, the Linux command line sort program, or sorting code you find on the web outside our class website. One of the goals of this assignment is for you to implement and debug sorting functions.

Your final design task is to process the sorted word array to create (count, word) pairs, and then sort this based on the count values. This sounds easy, but it can be tricky to implement correctly. One approach is to use two “parallel arrays” that contain counts and words, and sort the two arrays based on count values.

Make sure you test your code on some small test files (like the Green Eggs and Ham text above which only has 50 unique words) to be sure it is 100% correct. Once this is working correctly, you can go on to process larger input files.

3. Implementation:

You can implement this program using either a bottom-up approach or a top-down approach. If you go for a bottom-up approach, start by creating basic methods and classes, and test theses methods using a simple main program that calls each method. When this is working, you can create the main program that uses these methods to solve the problem above.

If you go for a top-down approach, start by creating your main program that reads user input, and calls empty methods to pretend to solve the problem. Then add in the code for these methods one at a time. This way, you will get an idea of how the whole program will work before you dive into the details of implementing each method and class.

Regardless of which technique you choose to use, you should develop your code incrementally adding code, compiling, debugging, a little bit at a time. This way, you always have a program that "does something" even if it is not complete.

4. Testing:

Test your program to check that it operates correctly for all of the requirements listed above. Also check for the error handling capabilities of the code. Try your program with several input values, and save your testing output in text files for inclusion in your project report.

5. Documentation:

When you have completed your C++ program, write a short report using the project report template describing what the objectives were, what you did, and the status of the program. Does it work properly for all test cases? Are there any known problems? Save this report to be submitted electronically.

6. Midpoint Project Submission:

To encourage students to get an early start on their programming project, students are required to upload into Blackboard a partial solution to their programming project on the midpoint due date shown above. The program does not need to be complete, but it must compile and perform some of the tasks listed above. This midpoint solution is worth 10% of your final grade on the project.

7. Project Submission:

In this class, we will be using electronic project submission to make sure that all students hand their programming projects and labs on time, and to perform automatic plagiarism analysis of all programs that are submitted.

When you have completed the tasks above go to Blackboard to upload your documentation (a single docx or pdf file), and all of your C++ program files. Do NOT upload an executable version of your program.

The dates on your electronic submission will be used to verify that you met the due date above. All late projects will receive reduced credit:

- 10% off if less than 1 day late,
- 20% off if less than 2 days late,
- 30% off if less than 3 days late,
- no credit if more than 3 days late.

You will receive partial credit for all programs that compile even if they do not meet all program requirements, so handing projects in on time is highly recommended.

8. Academic Honesty Statement:

Students are expected to submit their own work on all programming projects, unless group projects have been explicitly assigned. Students are NOT allowed to distribute code to each other, or copy code from another individual or website. Students ARE allowed to use any materials on the class website, or in the textbook, or ask the instructor and/or GTAs for assistance.

This course will be using highly effective program comparison software to calculate the similarity of all programs to each other, and to homework assignments from previous semesters. Please do not be tempted to plagiarize from another student.

Violations of the policies above will be reported to the Provost's office and may result in a ZERO on the programming project, an F in the class, or suspension from the university, depending on the severity of the violation and any history of prior violations.