**Programming Project Report**

Name: Riley Nadwodny

Date: 4/17/2020

**Academic Integrity Statement:** I pledge that I have neither given nor received unauthorized help on this programming assignment.

**Problem Statement:**

The goal of this programming project was for us, as students, to gain experience using sorting algorithms and document analysis. We had to calculate the frequency of all the words in a book; the minimum number of books required to do this for was three books. This program does that, with a little menu included into the programming. The inputs in this program are the user’s menu selection, and the title of the book that they wish to calculate; the titles available to the user are provided by the program in the menu after the select the “Read” option. If the user’s input is not an “E” or an “R”, the program continues to prompt them for a valid input.

The program outputs are all of the words in a scanned book, along with their frequencies, or how many times that they appear in that specific book. The primary error handling done in this program is with user selection in the main menu, as well as failure to open an input or output file. If the program fails to open an input or output file, a message is sent to the user and the program ends.

**Design:**

For this program, no separate class was built, as Professor Gauch said that it would be an extra hassle to do that and it was not necessary for this project. This program contains seven different functions, each with a different task. The seven functions and their tasks are as follows:

* string FixWord(string word)
  + Loop over characters to fix the words read in. Saves all lowercase letters and keeps apostrophes. Saves uppercase letters as lowercase letters.
* void ReadWords(string filename, vector<string> &data)
  + Opens a text file to read in. Reads in one word at a time. Calls the FixWord function to fix each word, and then if the length of the string is greater than 0, i.e. not a space or blank character, pushes the word back into an unsorted vector which contains all the words read in from the book.
* void insertion\_sort(vector<string> &data, int low, int high)
  + Inserts each element of the unsorted data into a sorted list, then puts the new value into an array.
* void partition(vector<string> &data, int low, int high, int &mid)
  + Splits the array into two parts. Scans left to right, and then right to left, swapping data values. Uses a pivot point to swap to mid as it loops through and swaps values in the partition.
* void quick\_sort(vector<string> &data, int low, int high)
  + If low is less than high, it calls partition and makes recursive calls to itself to sort array. Else, it calls insertion\_sort to sort the array.
* void AlphabetSort(vector<string> &data, vector<string> sorted[])
  + Scans through every word in the data vector. Sorted is an array of 26 vectors, 0-25, each corresponding to a letter of the alphabet. The character in position zero in the data element being looked at will determine which vector in the array to insert the data element in to. For example, if data[0].at(0) == ‘a’, data[0] would be pushed into sorted[0]. Then, inside of a for loop, quick\_sort is called for each sorted[]. Then, the data vector is cleared before another for loop that inserts each sorted vector back into the data vector in order.
* void Frequency(vector<string> &data)
  + This function opens an output file, and then has a for loop to read the data in the data vector. If the element being scanned is equal to the next element, the frequency int is incremented by one. When the element being scanned is NOT equal to the next element, the element being scanned is pushed back into a word vector, and the frequency is pushed back into a vector containing all of the word frequencies in the same slot as the word. Then the function writes the frequency number and the word to book.count.

The pros of doing this is that I was able to avoid creating a whole new class and class methods for storing data and manipulating it. The big downside to this is that all of the functions are in main.cpp, and brought the total lines of code in one file to 364 lines of code, whereas if it was just the main function inside of main.cpp, there would only be 65 lines of code.

**Implementation:**

To begin, I started with John Gauch’s code for FixWord, ReadWords, insertion\_sort, partition, and quick\_sort. Insertion\_sort, partition, and quick\_sort had to be modified to work with strings instead of int values, which was as simple as changing a few ints to strings. Next, I began by filling out the user menu and implementing the choices. I then worked on reading in books to a word vector, and then worked on the method for sorting the word vector into the alphabetical array of word vectors. Once the words were sorted into the array of vectors, each vector in the array was sorted so the a’s were alphabetically sorted, the b’s, and so on. After that, each vector was inserted back into the text vector, so it contains all the words of the book sorted alphabetically.

The development timeline on this took the full two weeks. For the midpoint, I had the user interface working and reading in a text file. In the last 48 hours, I implemented the AlphabetSort and Frequency functions to sort all of the words and calculate the frequency of each word. Coding and implementation, as well as debugging of these functions took roughly 12-14 hours.

**Testing:**

This program was tested with invalid menu selections and invalid text file names for error checking. Then, it was tested with valid and invalid selections and text file names, and lastly, all valid selections and file names. The program works mostly as expected- a few of the frequency counts are off for a few words, but they are within the +/-5% range provided in an email by John Gauch. Sample input is included in the typescript attached in the zip folder, and sample output is included in the typescript and the book.frequency, book.frequency2, and book.frequency3. These are the words sorted by frequency for Anne Gables, Dracula, and Tale of Two Cities respectively.

**Conclusions:**

Overall, the result of this assignment was a success. I am pleased with the results and pleased with how the functions turned out and how efficiently the code ran on turing when reading from a text file containing over 100,000 words. Next time, I would like to find a way to do the comparison and sort more efficiently rather than with 25 if-else statements. Other than that, I would the rest of the project the same. This project took the full two weeks to complete.