CSCI 2275 – Programming and Data structures
Instructor: Hoenigman
Assignment 3
Due: Wednesday, Sept 23 (Monday lecture) and Friday, Sept 25 (Wednesday lecture)

There are several parts to this assignment. Please don't wait until the last minute to start working on it.

Runtime analysis – 20 points

Using the code you submitted for assignment 2, including both the assignment2.cpp and the MessageBoard.cpp, do the following:

1. Identify the loops in your program and the number of times, best and worst case, that the loop will execute.
2. In each loop, identify the number of constant operations
3. In your main, identify the number of constant operations and loops, and input size for each loop.
4. Write a one-page summary with this information and include it in the zip file that you submit for this assignment

Dynamic memory exercises - 10 points

Each of the following questions will help you in completing the Word Analysis portion of this assignment. You can answer these questions in your main function in Assignment3.cpp

1. In *TextAnalyzer.h*, there is a struct called *wordItem*. In main, use dynamic memory allocation to create an array of *wordItem* with size=10. Set the word and count values for all of the items in the array to any values that you choose. Print the array.
2. Ask the user for an index in the array where you will insert a new *wordItem*. You can use *word="test",* and *count=99*. Shift the array to make room for the new *wordItem*. Since the array is already full, the last *wordItem* in the array will no longer be in the array. Print the array.
3. Ask the user again to enter an index in the array where you will insert a new *wordItem*. Use *word="test2"* and *count=100*. Instead of just shifting the array and losing the last *wordItem*, create a new array that has a size of 20, and shift and copy the array elements into the new array. Print the array. Delete the old array.

Word analysis – 70 points

There are several fields in computer science that aim to understand how people use language. This can include analyzing the most frequently used words by certain authors, and then going one step further to ask a question such as: "Given what we know about Hemingway's language patterns, do we believe Hemingway wrote this

lost manuscript?" In this assignment, we're going to do a basic introduction to document analysis by determining the number of unique words and the most frequently used words in a provided document. Processing the document also provides a vehicle for us to explore key concepts in dynamic memory management and array manipulation.

Please read all directions for the assignment carefully. This write-up contains both the details of what your program needs to do as well as implementation requirements for how the functionality needs to be implemented.

What your program needs to do

There are four files provided that you will need for this assignment:

1. There is a test file on Canvas – *HungerGames_edit.txt* that contain the full text from *Hunger Games Book 1*. We have pre-processed the file to remove all punctuation and down-cased all words.
2. *TextAnalyzer.h* provides the class definition for a *TextAnalyzer* class that you need to build for this assignment. Please don't change the header file unless the changes are requested specifically as part of the assignment.
3. There is a file called *ignoreWords.txt* that contains the 50 most common words in the English language.
4. *Assignment3.cpp* is the driver file that calls your TextAnalyzer methods. You are welcome to modify this file, but I don't think you'll need to.

Overview of program functionality

Your program needs to read in the .txt files, with the names of the files passed to your program as command-line arguments. The words in the *ignoreWords.txt* file need to be stored in the dynamically allocated *ignoreWords* array in your *TextAnalyzer* class. Your program then needs to read each of the words in the HungerGames file and determine if it is a common word. If it is not common, add it to the *words* array in the *TextAnalyzer* class in alphabetical order if it hasn't been previously seen, or increase the count for the word if it has. You will need to use dynamic memory allocation to manage the *words* array.

After the file has been processed, output the following information:

- The top *n* words (*n* is also a command-line argument) and the number of times each word was found
- The total number of unique words in the file
- The total number of words in the file
- The number of array doublings needed to store all unique words in the file

Example:

Running your program using:

```
./Assignment3 10 HungerGames_edit.txt ignoreWords.txt
```

would return the 10 most common words in the file *HungerGames_edit.txt* and should produce the following results.

```
682 - is
492 - peeta
479 - its
431 - im
427 - can
414 - says
379 - him
368 - when
367 - no
356 - are
#
Array doubled: 7
#
Unique non-common words: 7682
#
Total non-common words: 59157
```

Program specifications

An array of structs stores the words and their counts

There are specific requirements for how your program needs to be implemented. For this assignment, you need to use a dynamically allocated **array of structs** to store the words and their counts. In the TextAnalyzer class, the array is defined as a private variable called *words*. The struct has members for the word and count:

```
struct wordItem
{
    string word;
    int count;
};
```

Exclude these top 50 common words from your word counting

Table 1 shows the 50 most common words in the English language. In your code, exclude these words from the words you count in the .txt file. The words are included in a .txt file that you code needs to read in and populate a common word array, called *ignoreWords* in the *TextAnalyzer* class. Your code should include a separate function, called *isIgnoreWord()* to determine if the current word read from the .txt file is on this list and only process the word if it is not.

Table 1. Top 50 most common words in the English language

Rank	Word	Rank	Word	Rank	Word
1	The	18	You	35	One
2	Be	19	Do	36	All

3	To	20	At	37	Would
4	Of	21	This	38	There
5	And	22	But	39	Their
6	A	23	His	40	What
7	In	24	By	41	So
8	That	25	From	42	Up
9	Have	26	They	43	Out
10	I	27	We	44	If
11	It	28	Say	45	About
12	For	29	Her	46	Who
13	Not	30	She	47	Get
14	On	31	Or	48	Which
15	With	32	An	49	Go
16	He	33	Will	50	Me
17	As	34	My		

Use three command-line arguments
Your program needs to have three command-line arguments – the first argument is the number of most frequent words to output, the second argument is the name of the file to open and read, and the third argument is the name of the file that contains the words to ignore, also called *stop words*. For example, running

```
./Assignment2 20 HungerGames_edit.txt ignoreWords.txt
```

will read the *HungerGames_edit.txt* file and output the *20* most frequent words found in the file, not including the words listed in *ignoreWords.txt*.

Use the array-doubling algorithm to increase the size of your array
We don't know ahead of time how many unique words either of these files has, so you don't know how big the array should be. **Start with an array size of 100**, and double the size as words are read in from the file and the array fills up with new words. Use dynamic memory allocation to create your array, copy the values from the current array into the new array, and then free the memory used for the current array. This is the same process you will use in Recitation 4.

Add word to array in alphabetical order
In the *insertWord* method in the *TextAnalyzer* class, if the unique word hasn't been previously seen, you need to insert the word in alphabetical order into the *words* array. To compare two strings, you can use the < or > operators. For example, "dog" < "elephant" returns true. To insert a value into an array, you need to shift the rest of the values to the right. For example, to add a value at words[2], the values at words[3]…words[n] need to be shifted, starting at n and working back to 2.

The code to insert the word and shift the array should be done in the *insertAt* method.

Output the top *n* most frequent words
Write a function to determine the top *n* words in the array. This can be a function that sorts the entire array, or a function that generates an array of the *n* top items. Output the *n* most frequent words in the order of most frequent to least frequent.

Format your output the following way
When you output the top *n* words in the file, the output needs to be in order, with the most frequent word printed first. This should be handled in your *printTopN* method in *TextAnalyzer*. The format for the output needs to be:

Count - Word
#

Next, you need to print the number of times the array was doubled, the number of unique non-common words, and the total number of non-common words. The commands to print this information are in the assignment3.cpp file.

Submitting Your Code:
Log into Canvas and go to the Assignment 3 link. Zip your files and submit them as Assignment3_LastName.zip.

What to do if you have questions
There are several ways to get help on assignments in 2275, and depending on your question, some sources are better than others. There is the slack forum that is a good place to post technical questions, such as how to shift an array. When you answer other students' questions on the forum, please do not post entire assignment solutions. The TAs are also a good source of technical information, especially questions about C++. If, after reading the assignment write-up, you need clarification on what you're being asked to do in the assignment, the TAs and the Instructor are better sources of information than the discussion forum.