CSCI 2270 – Data structures and algorithms
Instructor: Hoenigman
Assignment 3
Due Friday, February 10 before 3pm

Word analysis

In this assignment, you will implement the word analysis program from assignment 2 using classes and dynamic memory to store the data.

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 two documents.

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 is one test file on Moodle – *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.

Your program needs to read in the .txt file, with the name of the file to open set as a command-line argument. Your program needs to store the unique words found in the file in a dynamically allocated array and calculate and 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

There is a link to a class video on command-line arguments in the Resources section of piazza.

Example:

Running your program using:

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

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

Use an array of structs to store the words and their counts

There are specific requirements for how your program needs to be implemented. For this assignment, you need to use an **array of structs** to store the words and their counts. The members of the struct are left to you, but keep it as simple as possible.

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. Your code should include a separate function 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 two command-line arguments
Your program needs to have two command-line arguments – the first argument is the name of the file to open and read, and the second argument is the number of most frequent words to output. For example, running

```
./Assignment3 HungerGames_edit.txt 20
```

will read the *HungerGames_edit.txt* file and output the *20* most common words found in the file.

Note: There is a link to a class video on command-line arguments in the Resources section on Piazza.

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 used in Recitation 3.

Note: some of you might wonder why we're not using C++ Vectors for this assignment. A vector is an interface to a dynamically allocated array that uses array doubling to increase its size. In this assignment, you're doing what happens behind-the-scenes with a Vector.

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. The format for the output needs to be:

```
Count - Word
#
Array doubled: <number of array doublings>
```

#
Unique non-common words: <number of unique words>
#
Total non-common words: <total number of words>

Create a WordAnalysis class
All of the functionality above, except for command-line arguments, should be included in methods in a *WordAnalysis* class. There is starter code for this assignment on Moodle. The code is in a zip file called Assignment3.zip and includes the header file for the class, called *WordAnalysis.h*, and the *Assignment3.cpp* that COG will use to test your code.

You need to implement the methods defined in the header exactly as they are defined. Your code should be in a .cpp file called *WordAnalysis.cpp*. You can test that your *WordAnalysis.cpp* code works by building the files and running the executable.

```
g++ -std=c++11 WordAnalysis.cpp Assignment3.cpp -o Assignment3
```

The following methods and variables are defined in *WordAnalysis.h*.

```
struct Word{
        int count;
        std::string word;
};

class WordAnalysis{
        private:
            //stores the number of times the array has been doubled in the program
            int timesDoubled;

            //stores the array of words. Memory will be dynamically allocated
            Word *words;

            /*current size of the array. When you double the array, wordCount =
            wordCount * 2 to double the size
            */
            int wordCount;

            //how many unique words found
            int index;

            //call this method when you want to double the array and add the new
            //word to the array. The new word is the input to the method.
            void doubleArrayAndAdd(std::string);

            //call this method to check if a word is in the common word list
            bool checkIfCommonWord(std::string);

            //call this method to sort the words array
            void sortData();
        public:
            //call this method to open a file and read in the data.
            //the filename is the argument
            bool readDataFile(char*); //returns an error if file not opened

            //returns index * count for each word
            int getWordCount();

            //returns index variable
            int getUniqueWordCount();

            //returns timesDoubled variable
            int getArrayDoubling();

            //call this method to print the common words. The argument is the
            //number of words to print
            void printCommonWords(int);

            //constructor. The argument is the initial size of the array
            WordAnalysis(int);
            ~WordAnalysis();
};
```

Submitting Your Code:

Submit your assignment to the COG autograder:
https://web-cog-csci2270.cs.colorado.edu/submit.html.

Login to COG using your identikey and password. Select the "HW 3 - Main Grade" from the dropdown. Zip your WordAnalysis.cpp, WordAnalysis.h, and Assignment3.cpp files together into one Assignment3.zip archive. **Don't add them to a folder and zip the folder. You should not make any modifications to the two files provided (WordAnalysis.h and Assignment3.cpp).** Upload your file and click Submit. COG will run its tests and display the results in the window below the Submit button. If your code doesn't run correctly on COG, read the error messages carefully, correct the mistakes in your code, and upload a new file. You can modify your code and resubmit as many times as you need to, up until the assignment due date.

Submit your assignment to Moodle
In addition to submitting through COG, submit your *Assignment3.zip* file through Moodle using the Assignment 3 Submit link. Make sure your code is commented

enough to describe what it is doing. Include a comment block at the top of your files with your name, assignment number, and course instructor. **If you do not submit your work to Moodle, we will deduct points from your grade, even if COG gives you a perfect score.**

If you do not get your assignment to run on COG, you will have the option of scheduling an interview grade with your TA to get a grade for the assignment. Even if you do get the assignment to run on COG, you can schedule the interview if you just want to talk about the assignment and get feedback on your implementation.

What to do if you have questions

There are several ways to get help on assignments in 2270, and depending on your question, some sources are better than others. There is the Piazza 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 CAs 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 or the CAs.