CSCI 2270 – Data Structures and Algorithms
Instructor: Godley
Assignment 2
Due Friday 6/22 11:59pm

Word analysis

Read the entire assignment carefully before beginning. 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.

Description

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.

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 both a dynamically allocated array and a singly linked list. Finally you should 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

Example:

Running your program using:

```
./Assignment2 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
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 a dynamically allocated **array of structs** to store the words and their counts. Your struct needs to have 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 your code needs to read in and populate a common word array. Your code should include a separate function, called *isStopWord()* 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 3 and we'll do in class on Monday.

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>
```

Generate the output with these commands:

```
cout<<numCount<<" – "<<word<<endl;
cout<<"#"<<endl;
cout<<"Array doubled: "<<numDoublings<<endl;
cout<<"#"<<endl;
cout<<"Unique non-common words: "<<numUniqueWords<<endl;
```

Your code needs to include at least the following functions:

```
/*
* Function name: getStopWords
* Purpose: read stop word list from file and store into an array
* @param ignoreWordFile - filename (the file storing ignore words)
* @param ignoreWords - store ignore words from the file (pass by
reference) * @return - none
* Note: The number of words is fixed to 50
*/
void getStopWords(char *ignoreWordFileName, string ignoreWords[]);

/*
* Function name: isStopWord
* Purpose: to see if a word is a stop word
* @param word - a word (which you want to check if it is a stop word)
* @param ignoreWords - the array of strings storing ignore/stop words *
@return - true (if word is a stop word), or false (otherwise)
*/
bool isStopWord(string word, string ignoreWords[]);

/*
* Function name: getTotalNumberNonStopWords
* Purpose: compute the total number of words saved in the words array
(including repeated words)
* @param list - an array of wordItems containing non-stopwords
* @param length - the length of the words array
* @return - the total number of words in the words array (including
repeated words multiple times)
*/
int getTotalNumberNonStopWords(wordItem list[], int length);

/*
* Function name: arraySort
* Purpose: sort an array of wordItems, decreasing, by their count
fields * @param list - an array of wordItems to be sorted
* @param length - the length of the words array
 */
void arraySort(wordItem list[], int length);

/*
* Function name: printTopN
* Purpose: to print the top N high frequency words
* @param wordItemList - a pointer that points to a *sorted* array of
wordItems * @param topN - the number of top frequency words to print
* @return none
*/
void printTopN(wordItem wordItemList[], int topN);
```