CSCI-2270
Assignment 2
Instructor: Boese

Word Analysis

Array Doubling with Dynamic Memory

Objectives

- Read a file with unknown size and store in a vector
- Loop through a vector
- Store, search, and iterate through data in an array of struct
- Use array doubling via dynamic memory to increase the size of the array

Background

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. *If you enjoy this, take elective courses on Natural-Language Processing.*

What your program needs to do

There is one test file on the website – HW2-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. *We will test on a different file!*
There is also the ignore words file – HW2-ignoreWords.txt that contain the top 50 common words usually ignored during natural-language processing.

Your program will calculate the following information on any text file:

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

Example:
Your program takes three command-line arguments: the number of most common words to print out, the name of the file to process, and the stop word list file.
Running your program using:

```
./a.out 10 HW2-HungerGames_edit.txt HW2-ignoreWords.txt
```

would return the 10 most common words in the file HW2-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-stop words: 7682
#
Total non-stop words: 59157
```

Program Specifications

The following are requirements for your program:

- Read in the name of the file to process from the *second* command-line argument.
- Read in the number of most common words to process from the *first* command-line argument.
- Write a function named **getStopwords** that takes the name of the ignore-words file and a reference to a vector as parameters (returns void). Read in the file for a list of the top 50 most common words to ignore (e.g., Table 1). These are commonly referred to as 'stopwords' in NLP (Natural Language Processing). *(Create this file yourself)*
    - The file will have one word per line, and always have exactly 50 words in the file. *We will test with files having different words in it!*
    - Your function will update the vector passed to it with a list of the words from the file.
- Store the unique words found in the file that are not in the stopword list in a *dynamically* allocated array.
    - Call a function to check if the word is a stopword first, and if it is, then ignore that word.
    - Use an array of structs to store each unique word (variable name **word**) and a count (variable name **count**) of how many times it appears in the text file.
    - **Use the array-doubling algorithm to increase the size of your array**

        - We don't know ahead of time how many unique words the input file will have, so you don't know how big the array should be. Start with an array size of 100 (use the constant declared in the starter code), and double the size as words are read in from the file and the array fills up with new words.
            - Use <u>dynamic memory allocation</u> 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.
              (Index of any given word in the array after resizing must match index in array before resizing.)
- Output the top n most frequent words
  Write a function named **printTopN** that takes a reference to the array of structs and the value of n to determine the top n words in the array. Generate an array of the n top items sorted from most frequent to least frequent and print these out from most to least.
  Array MUST be sorted before calling printTopN.
- Output the number of times you had to double the array.
- Output the number of unique non-stop words.
- Output the total number of non-stop words.
    - Write a function named **getTotalNumberNonStopWords** that takes a reference to your array of unique non-stop words and returns the total number of words found.
- Format your output the following way *(and reference the example above).*
    - 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-stop words: <number of unique words>
#
Total non-stop 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-stop words: " << numUniqueWords;
cout <<  "#" << endl;
cout <<  "Total non-stop words: ";
cout << getTotalNumberNonStopWords(yourarray, size) <<endl;
```

- Include a comment block at the top of the .cpp file with your name, assignment number, date and course instructor.
- Make sure your code is commented enough to describe what it is doing.

Note: some of you might wonder why we're using array doubling instead of using C++ Vectors for storing the uncommon words. In this assignment, you're doing what happens behind-the-scenes with a Vector.

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 the following starter code (type it in so you get more practice setting up programs):

```
// ==========================================
// Created: August 23, 2016
// @author
//
// Description: Counts unique words in a file
// outputs the top N most common words
// ==========================================

#include <cstdlib>
#include <fstream>
#include <iostream>
#include <sstream>
```

```
#include <string>
#include <vector>


using namespace std;

// struct to store word + count combinations
struct wordItem
{
    string word;
    int count;
};

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

/*
* 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 _vecIgnoreWords - the vector type of string storing ignore/stop words
* @return - true (if word is a stop word), or false (otherwise)
*/
bool isStopWord(string word, vector<string> &_vecIgnoreWords);

/*
* 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, increasing, 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);

const int STOPWORD_LIST_SIZE = 50;

const int INITIAL_ARRAY_SIZE = 100;

// ./a.out 10 HW2-HungerGames_edit.txt HW2-ignoreWords.txt
int main(int argc, char *argv[])
{
    vector<string> vecIgnoreWords(STOPWORD_LIST_SIZE);

    // verify we have the correct # of parameters, else throw error msg &
return
    if (argc != 4)
    {
        cout << "Usage: ";
        cout << argv[0] << " <number of words> <filename.txt>
<ignorefilename.txt>";
        cout << endl;
        return 0;
    }

    /* **********************************
    1. Implement your code here.

    Read words from the file passed in on the command line, store and
    count the words.

    2. Implement the six functions after the main() function separately.
    ********************************** */

    return 0;
}

void getStopWords(char *ignoreWordFileName, vector<string> &_vecIgnoreWords)
{
    return;
}
```

```
bool isStopWord(string word, vector<string> &_vecIgnoreWords)
{
    return true;
}

int getTotalNumberNonStopWords(wordItem list[], int length)
{
    return 0;
}

void arraySort(wordItem list[]], int length)
{
    return;
}

void printTopN(wordItem wordItemList[], int topN)
{
    return;
}
```

Submitting Your Code:

Log into Moodle and go to the Homework 2 link. It is set up in the quiz format. Follow the instructions on each question to submit all or parts of each assignment question. You can check your solution to each question by clicking on the “Check” button. Note that you can only submit your homework **once**.

Note: there is no late period on assignments! If you miss the deadline or do not do well, you can sign up for an optional grading interview to get up to half the points missed back. There is also an optional extra credit assignment at the end of the semester you can use to replace one of your homework scores.