CSCI 2270 – Data Structures

Instructors: Shayon Gupta, Ashutosh Trivedi, Maciej Zagrodzki

Assignment 2

Array doubling with dynamic memory

OBJECTIVES

1. Read a file with unknown size and store its contents in a dynamic array
3. Store, search and iterate through data in an array of structs
4. Use array doubling via dynamic memory to increase the size of the array

Overview

There are several fields in computer science that aim to understand how people use language. One interesting example is analyzing the most frequently used words by certain authors, then using those frequencies to determine who authored a lost manuscript or anonymous note.

In this assignment, we will write a program to determine the most frequent words in a document. Because the number of words in the document may not be known a priori, we will implement a dynamically doubling array to store the necessary information.

Please read all the directions *before* writing code, as this write-up contains specific requirements for how the code should be written.

Your Task

There are two files on Moodle. One contains text to be read and analyzed, and is named ***HarryPotter.txt***. As the name implies, this file contains the full text from *Harry Potter and the Sorcerer's Stone*. For your convenience, all the punctuation has been removed, all the words have been converted to lowercase, and the entire document is written on a single line. The other file contains the 50 most common words in the English language, which your program will ignore. It is called ***ignoreWords.txt***.

Your program must take three command line arguments in the following order - a number *N*, the name of the text to be read, and the name of the text file with the words that should be ignored. It will read in the text (ignoring the words in the second file) and store all unique words in a dynamically doubling array. It should then calculate and print the following information:

- The number of array doublings needed to store all the unique words
- The number of unique "non-ignore" words in the file
- The total word count of the file (excluding the ignore words)
- The *N* most frequent words along with their probability of occurrence (***up to 4 decimal places***)

CSCI 2270 – Data Structures

Instructors: Shayon Gupta, Ashutosh Trivedi, Maciej Zagrodzki

Assignment 2

For example, running your program with the command:

```
./Assignment2 10 HarryPotter.txt ignoreWords.txt
```

would print the 10 most common words in ***HarryPotter.txt***, not including any words in ***ignoreWords.txt***. The full results would be:

```
Array doubled: 6
#
Unique non-common words: 5985
#
Total non-common words: 50331
#
Probabilities of top 10 most frequent words
---------------------------------------
0.0241 - harry
0.0236 - was
0.0158 - said
0.0139 - had
0.0100 - him
0.0081 - ron
0.0068 - were
0.0067 - hagrid
0.0065 - them
0.0052 - back
```

Specifics:

1. **Use an array of structs to store the words and their counts**
   There is an unknown number of words in the file. You will store each unique word and its count (the number of times it occurs in the document). Because of this, you will need to store these words in a dynamically sized **array of structs**. The struct must be defined as follows:

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

CSCI 2270 – Data Structures

Instructors: Shayon Gupta, Ashutosh Trivedi, Maciej Zagrodzki

Assignment 2

2. **Use the array-doubling algorithm to increase the size of your array**
   Your array will need to grow to fit the number of words in the file. **Start with an array size of 100**, and double the size whenever the array runs out of free space. You will need to allocate your array dynamically and copy values from the old array to the new array.

   Note: Don't use the built-in std::vector class. This will result in a loss of points. You're actually writing the code that the built-in vector uses behind-the-scenes!

3. **Ignore the top 50 most common words that are read in from the second file**
   To get useful information about word frequency, we will be ignoring the 50 most common words in the English language. These words will be read in from a file, whose name is the third command line argument.

4. **Take three command line arguments**
   Your program must take three command line arguments - a number *N* which tells your program how many of the most frequent words to print, the name of the text file to be read and analyzed, and the name of the text file with the words that should be ignored.

5. **Output the top *N* most frequent words**
   Your program should print out the top *N* most frequent words in the text - not including the common words - where *N* is passed in as a command line argument.

6. **Format your output this way:**

```
Array doubled: <Number of times the array was doubled>
#
Unique non-common words: <Unique non-common words>
#
Total non-common words: <Total non-common words>
#
Probabilities of top <N> most frequent words
---------------------------------------
<1st highest probability> - <corresponding word>
<2nd highest probability> - <corresponding word>
...
<Nth highest probability> - <corresponding word>
```

For example, using the command:

```
./Assignment2 10 HarryPotter.txt ignoreWords.txt
```

you should get the output:

```
Array doubled: 6
#
Unique non-common words: 5985
#
Total non-common words: 50331
#
Probabilities of top 10 most frequent words
---------------------------------------
0.0241 - harry
0.0236 - was
0.0158 - said
0.0139 - had
0.0100 - him
0.0081 - ron
0.0068 - were
0.0067 - hagrid
0.0065 - them
0.0052 - back
```

7. **You must include the following functions (they will be tested by the autograder):**
   a. **In your main function**
      i. If the correct number of command line arguments is not passed, print the below statement and exit the program

```
std::cout << "Usage: Assignment2Solution <number of words>
<inputfilename.txt> <ignoreWordsfilename.txt>" <<
std::endl;
```

      ii. Get stop-words/common-words from ***ignoreWords.txt*** and store them in an array (Call your **`getStopWords`** function)
      iii. Read words from ***HarryPotter.txt*** and store all unique words that are not ignore-words in an array of structs
         1. Create a dynamic **`wordItem`** array of size 100

2. Add non-ignore words to the array (double the array size if array is full)
3. Keep track of the number of times the **`wordItem`** array is doubled and the number of unique non-ignore words

b.

```
void getStopWords(const char *ignoreWordFileName, string ignoreWords[]);
```

This function should read the stop words from the file with the name stored in **`ignoreWordFileName`** and store them in the **`ignoreWords`** array. You can assume there will be exactly 50 stop words. There is no return value.
In case the file fails to open, print an error message using the below cout statement:

```
std::cout << "Failed to open " << ignoreWordFileName << std::endl;
```

c.

```
bool isStopWord(string word, string ignoreWords[]);
```

This function should return whether **`word`** is in the **`ignoreWords`** array.

d.

```
int getTotalNumberNonStopWords(wordItem uniqueWords[], int length);
```

This function should compute the total number of words in the entire document by summing up all the counts of the individual unique words. The function should return this sum. For example, in the quote below, there are 9 total non-stop words since “about” and “the” are stop words and “programmers” has a count of 2.

bad programmers worry ~~about~~ ~~the~~ code good programmers worry ~~about~~ data structures

e.

```
void arraySort(wordItem uniqueWords[], int length);
```

This function should sort the **`uniqueWords`** array (which contains **`length`** initialized elements) by word count such that the most frequent words are sorted to the beginning. The function doesn't return anything.

f.

```
void printTopN(wordItem uniqueWords[], int topN, int
totalNumWords);
```

This function should print out the first **`topN`** words of the **sorted** array **`uniqueWords`**. The exact format of this printing is described in part (6) of this writeup - you only need to print the last part (the most frequent words with their probability of occurrence **up to 4 decimal places**). It doesn't return anything.
Probability of occurrence of a word at position ***ind*** in the array is computed using the formula: ***(Don't forget to cast to float!)***

probability-of-occurrence = ***(float) uniqueWords[ind].count / totalNumWords***

8. **Submitting your code:**
Log onto Moodle and go to the Assignment 2 Submit link. It's set up in the quiz format. Follow the instructions on each question to submit all or parts of each assignment question.