CSCI 2270 – Data Structures

Instructors: Ashraf, Godley

Homework 1

Due Sunday, June 14, 2020, 11:59PM

Array doubling with dynamic memory

OBJECTIVES

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

Overview

In this assignment, we will write a program to analyze the word frequency 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 by your program, and is named ***mobydick.txt***. As the name implies, this file contains the full text from *Moby Dick*. For your convenience, all the punctuation has been removed, all the words have been converted to lowercase, and the entire document can be read as if it were written on a single line. The other file contains the 50 most common words in the English language, which your program will ignore during analysis . 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 the text (**ignoring the words in the second file**) and store all distinct words in a dynamically doubling array (**For file I/O, refer to your first homework assignment…**). It should then calculate and print the following information:

- The number of times array doubling was required to store all the distinct words
- The number of distinct “non-ignore” words in the file
- The total word count of the file (excluding the ignore words)

CSCI 2270 – Data Structures

Instructors: Ashraf, Godley

Homework 1

Due Sunday, June 14, 2020, 11:59PM

- Starting from index N, print the 10 most frequent words along with their probability (***up to 4 decimal places***) of occurrence from the array. The array should be sorted in decreasing order based on probability of occurrence of the words. If two words have the same probability, then those two words should be listed in alphabetical order.

  For example, running your program with the command:

  ```
  ./Homework1 25 mobydick.txt ignoreWords.txt
  ```

  would print the next 10 words starting from index 25, i.e. your program should print the 25th-34th most frequent words, along with their respective probabilities. Keep in mind that these words should **not** be any of the words from ***ignoreWords.txt***.

The full results would be:

```
Array doubled: 8
Distinct non-common words: 13744
Total non-common words: 67327
Probability of next 10 words from rank 25
-------------------------------------
0.0030 - other
0.0030 - over
0.0030 - been
0.0030 - these
0.0029 - sea
0.0029 - said
0.0028 - down
0.0028 - yet
0.0027 - any
0.0027 - whales
```

Specifics:

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

CSCI 2270 – Data Structures

Instructors: Ashraf, Godley

Homework 1

Due Sunday, June 14, 2020, 11:59PM

There is an unknown number of words in the file. You will store each distinct 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 wordRecord {
        string word;
        int count;
};
```

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. (Array-doubling algorithm should be implemented in *main()* function).

   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 the starting index to print the next 10 most frequent words, 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 Next 10 most frequent words starting from index N**
   Your program should print out the next 10 most frequent words - not including the common words - starting index N in the text where *N* is passed in as a command line argument. If two words have the same frequency, then those two words should be listed in alphabetical order.

CSCI 2270 – Data Structures

Instructors: Ashraf, Godley

Homework 1

Due Sunday, June 14, 2020, 11:59PM

E.g. If N=5 then print words from index 5-14 in the array sorted in decreasing order of the probabilities of the occurrence of the words.

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

```
Array doubled: <Number of times the array was doubled>
Distinct non-common words: <Distinct non-common words>
Total non-common words: <Total non-common words>

Probability of next 10 words from rank <N>
---------------------------------------
<Nth highest probability> - <corresponding word>
<N+1 th highest probability> - <corresponding word>
...
<N+10 th highest probability> - <corresponding word>
```

For example, using the command:

```
./Homework1 25 mobydick.txt ignoreWords.txt
```

you should get the output:

CSCI 2270 – Data Structures

Instructors: Ashraf, Godley

Homework 1

Due Sunday, June 14, 2020, 11:59PM

```
Array doubled: 8
Distinct non-common words: 14940
Total non-common words: 71127
Probability of next 10 words from rank 25
---------------------------------------
0.0028 - over
0.0028 - old
0.0027 - these
0.0027 - such
0.0027 - said
0.0026 - though
0.0026 - sea
0.0026 - down
0.0026 - And
0.0026 - any
```

CSCI 2270 – Data Structures

Instructors: Ashraf, Godley

Homework 1

Due Sunday, June 14, 2020, 11:59PM

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: Homework1Solution <number of words>
<inputfilename.txt> <ignoreWordsfilename.txt>" << std::endl;
```

      ii. Get ignore-words/common-words from ***ignoreWords.txt*** and store them in an array (Call your **`getIgnoreWords`** function)
      iii. Array-doubling should be done in the main() function
      iv. Read words from ***mobydick.txt*** and store all distinct words that are not ignore-words in an array of structs
         1. Create a dynamic **`wordRecord`** 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 **`wordRecord`** array is doubled and the number of distinct non-ignore words

   b.

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

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

CSCI 2270 – Data Structures

Instructors: Ashraf, Godley

Homework 1

Due Sunday, June 14, 2020, 11:59PM

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

c.

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

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

d.

```
int getTotalNumberNonIgnoreWords(wordRecord distinctWords[],
int length);
```

This function should compute the total number of words in the entire document by summing up all the counts of the individual distinct words. The function should return this sum.

e.

```
void sortArray(wordRecord distinctWords[], int length);
```

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

f.

```
void printTenFromN(wordRecord distinctWords[], int N, int
totalNumWords);
```

This function should print the next 10 words after the starting index N from **sorted** array of **`distinctWords`**. These 10 words should be printed with their probability of occurrence **up to 4 decimal places.** The exact format of this printing is given below. The function does not return anything.

CSCI 2270 – Data Structures

Instructors: Ashraf, Godley

Homework 1

Due Sunday, June 14, 2020, 11:59PM

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) distinctWords[ind].count / totalNumWords*

`Output format`

```
Array doubled: 8
Distinct non-common words: 13744
Total non-common words: 67327
Probability of next 10 words from rank 25
---------------------------------------
0.0030 - other
0.0030 - over
0.0030 - been
0.0030 - these
0.0029 - sea
0.0029 - said
0.0028 - down
0.0028 - yet
0.0027 - any
0.0027 - whales
```

8. **Submitting your code:**
   Log onto Moodle and go to the Homework 1 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.