CSCI 2270 – Data Structures - Section 100

Instructor: Shayon Gupta

Assignment 2, Sept 2018

WORD ANALYSIS
Array doubling with dynamic memory

OBJECTIVES

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

Write code to complete the following problems. Each problem should be completed in its own file, as they will be graded individually on Moodle.

Problem 1

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 vary, we will implement a dynamically doubling array (known as a *vector*) 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.

CSCI 2270 – Data Structures - Section 100

Instructor: Shayon Gupta

Assignment 2, Sept 2018

What your program will do:

There are two files on Moodle. One contains text to be read and analyzed, and is named *HungerGames_edit.txt*. As the name implies, this file contains the full text from *Hunger Games Book 1*. All the punctuation removed and all the words have been down-cased for your convenience. 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 - 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 them in a dynamically doubling array. It should then calculate and print the following information:

- The *N* most frequent words, along with their frequencies.
- The number of unique words in the file.
- The word count of the file.
- The number of array doublings needed to store all the unique words.

For example, running your program with the command:

```
./Assignment2 10 HungerGames_edit.txt ignoreWords.txt
```

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

```
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
```

CSCI 2270 – Data Structures - Section 100

Instructor: Shayon Gupta

Assignment 2, Sept 2018

Specifics:

1. **Use an array of structs to store the words and their counts**
   There is an unknown number of words in the file, and you will store both the word and their frequency. Because of this, you will need to store these words in a dynamically sized **array of structs**. The struct must have members for the word and its frequency:

```
struct wordItem {
        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, like what was done in Recitation 2.

   Note: Please don't use the built-in std::vector class (from Assignment 1). You're actually writing the code that the built-in vector uses behind-the-scenes!

3. **Ignore the top 50 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.

CSCI 2270 – Data Structures - Section 100

Instructor: Shayon Gupta

Assignment 2, Sept 2018

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

```
<1st highest frequency> - <corresponding word>
<2nd highest frequency> - <corresponding word>
...
<Nth highest frequency> - <corresponding word>
#
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>
```

For example, using the command:

```
./Assignment2 10 HungerGames_edit.txt ignoreWords.txt
```

...should give the output:

```
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
```

CSCI 2270 – Data Structures - Section 100

Instructor: Shayon Gupta

Assignment 2, Sept 2018

7. **You must include the following functions (they will be tested by the autograder):**
   a.
   ```
   void getStopWords(char *ignoreWordFileName, string ignoreWords[]);
   ```
   This function should read the stop words from the file with the name *ignoreWordFileName* and store them in the *ignoreWords* array. You can assume there will be exactly 50 stop words. It has no return value.
   b.
   ```
   bool isStopWord(string word, string ignoreWords[]);
   ```
   This function should return whether *word* is in the *ignoreWords* array.
   c.
   ```
   int getTotalNumberNonStopWords(wordItem list[], int length);
   ```
   This function should compute the total number of words in *list* (of length *length*) by adding up all the frequencies of the individual unique words. It should return this total.
   d.
   ```
   void arraySort(wordItem list[], int length);
   ```
   This function should sort the *list* array (of length *length*) by word frequency. The most frequent word should be first. It doesn't return anything.
   e.
   ```
   void printTopN(wordItem wordItemList[], int topN);
   ```
   This function should print out the first *topN* words of the **sorted** array *wordItemList*. The exact format of this printing is described in part (6) - it only needs to print the first part (the most frequent words). It doesn't return anything.

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