ESSAY MARKING PROGRAM REPORT

Cand No: 253047

Introduction

The purpose of this program is to scan a given input text file, analyse the file based on certain marking criteria, and return an output text file, containing the outcome of these criteria and various other attributes of the input.

Program description

Key functions.

The program is made up of various elements, designed to achieve the desired result. The first element is the class 'Word'. The class is made up of three private attributes; a string – the word in character form; an int – counting the number of times the word appears in the essay; and another int – the amount of points the word achieves based on its letters. The class is then also made up of public methods which are getters and setters for the private attributes mentioned above. There is also a getLength function which returns the length of a given word, and a getCounterPP function which increments getCounter for each value of itself. getCounterPP is referenced in the main program, when the words in the essay are pushed in the vector, we can check each word in the essay against each word in the vector, given getCounterPP the value of number of times there is a match. Therefore the number of occurrences of each word.

The next element is the struct letter. It too is made up of three attributes; a char to indicate its character value; an int to indicate the number of points the letter has attributed to it; and a bool for whether it is of type v (vowel) or not. These are important as it allows us to initiate the table, we were given in the brief that indicates the points and types of each letter. Because we have and array of the type - letter (struct), we can input each letters' attributes into the array using the create table function. Since, this array contains the character value, points value, and type we can implement it in various other functions.

For example, the calPoints function calculates the total points of a given string. Using the letter array, we can match characters in the string to those in the array, find their respective points and return the total points of that string. This function is called in the public method, getPoint, so the points for a given word can be accessed anytime. In addition, the vCount function calculates the number of vowels in a string using the same method.

One of the most important elements I created was the vector w of type Word. This allowed me to store each of the words in the essay, and their attributes, in a easy to access file. One positive of the vector class is that its size dynamic, meaning it can shrink and grow to any size. This was particularly useful when dealing with essays of varying sizes.

Tasks

The criteria for finding certain words are a little vague, so to be clear on my process I am stating it here. The definition for a certain word, is the ***first*** word to appear in the essay that meets the criteria given. For example, if there are multiple words that are the shortest word, and over 1 character long, e.g., 2 characters long. Then the shortest word would be the first 2 letter word to appear in the essay.

Task 1 required us to mark the inputted text file based on 6 criteria. The first of which was that the shortest word must score between 10 and 50 points. Following the rules stated above, I found the first shortest word to appear in the essay, using the function I created, called shortestWord. Since all the words in the essay are stored in the vector, I can easily access each word and its attributes. Using a for loop and the getLength method, I can iterate through each word comparing its length to ones

that have been checked already. If it is the shortest, it becomes the shortest word for the next word to be checked against until it reaches the end of the vector, and we are left with the shortest word. I used this method to find the word that satisfies each of the minimum and maximum criteria, changing the sign based on whether I needed min or max including the longest word required in criteria 2. Now that I have found the shortest/longest words I was able to check whether they meet the points criteria and return true or false.

The 3rd criterion was that there must be 5 or more palindrome words. For the purpose of this criterion, I included 1 letter words. To solve this, we needed to use a function to check whether a word is palindrome – isPalindrome. This function checks whether the front half of the string is the same as the back half of the string. Returns false if not, otherwise returns true.

The 4th criterion is a simple check on whether the essay is between 50 and 100 words. Because each of the words have been put into the vector w, we can easily find the number of words using w.size.

The 5th criterion again uses a for loop through the whole vector, and using the getPoint and getLength methods, the total points and lengths of the combined words can be found. From this we can compute the average. Whether the essay satisfies the given criteria can then be checked.

The 6th criterion is that the essay cannot have words with over 4 vowels. Here I created another function vowelC. Which uses the letter isVowel attribute to count the number of vowels in a word. We can use a for loop to use this function on all the words in the essay. If there are over 4 vowels in any word, it returns false.

For task 2, the shortest, longest, least frequent, and most frequent word had to be found. As mentioned above, if there is more than one word that has the 'winning' value of length or frequency then the word that appears first in the essay will be selected. The shortest and longest could be found using the functions mentioned above. The least frequent and most frequent word were found using the same method, but instead of comparing length, the functions compare frequency, using the getCounter function.

Task 3, the getCounter function once again had to be used. However, the words also had to be sorted in ascending order and outputted. To achieve this, I created a new vector called countS. The values inputted to this vector were the getCounter (int) value of each word (converted to a string) + the getWord (string) value, creating a vector of strings with the number of occurrences as the first character. This could then easily be sorted by number of occurrences using the .sort argument. The vector also made it simple to output each word to the file using a for loop to iterate through each element.

Task 4 required finding the students name, the key, the secret and the deciphered secret. The student's name is the 10th word in the essay so to find this I just needed to find the word at index 9 of the vector. The is points total of the name, modulo 26. The secret was difficult to find, but the ease of use of the vector class solved this. The secret is the second word the number points for which lies between 90 and 110. To find this I created a new vector of type Word to store all words that scored those points and took the second element (index 1) to find the secret. However, if there isn't 2 words that meet the criteria then the secret is the last word in the essay. The decipher simply takes the secret and shifts left down the alphabet the number of times the key indicates.

Task 5 asks to sort the words by their points and return the median word. If there is no middle (even number of words) then return the two middle words combined. To do so I created a function to initially find the middle index of the vector. Because I was using .size which would return +1 of the

max index, I had to think about this carefully to find the middle index in the case number of words is odd, and the two middle indexes if the number words was even. I think make use of the modulus operator to find if the number of words is even. If so, return the words associated to the two middle indexes combined. Else, return the word associated to the middle index if number of words is odd.

Finally, the results of these had to be outputted to a text file. To do so I used the writeOutput function where I could call on it in the main and feed in a string and a file name to output the file. To write each of the relevant answers I created a new string called output. I could then add all the answers into this single string and use it in the writeOutput function. A while loop was also created to read each of the 100 input files whilst outputting the respective answers to 100 output files.

Conclusion

Overall, this was a difficult and daunting task, particularly for someone with little coding experience. However, after breaking the programs into the various functions I got there (almost). When checking against the answer files, I achieve less than desirable results. However, we can see clearly that the answers are following different rules to me, and even then, some of the answers appear to be wrong. Nevertheless, some of my tasks were not completed effectively. Firstly, in task 3, my output prints duplicates for words that appear more than once. For example, if o appeared 3 times in an essay, then my output would print 1o,2o,3o. I couldn't figure out why this was happening so therefore couldn't find a solution. I also believe at times some of my code is sloppy and that there would be a more efficient way of solving the problem. In addition, if for example my code compiles but it doesn't print what I'd like it to (often an indicator of poor logic), I often find myself solving the issue by applying more logic to function, and not fixing the initial problem. Whilst this might be fine in some cases, I think in many it is reducing efficiency and if dealing with larger programs this sort of thing may result in a bug in the future.