Course Name: Data Structures and Algorithms (C++)
Project Name: Word Counter

Group Members:
Name: Anıl KAYAN
Name: Onur ÖZÜDURU

---

1. In General

1.1. What does program do?

-It prints most popular words (amount of user enters) basic histografic way.
-It gives to user numbers of (non repetition ones) words in the text.
-It gives to user how much word which is entered by user repeated in the text.
-It prints all the non repeated words.

1.2. What cannot program do?

- *'s 're 've* cannot be become *is, are, have* For example; *name's* becomes *names*

1.3. Main structure

List.h contains basic functions.
Dictionary.h is derived from list class. It is formed of nodes which are string type distinctly and it has alphabetic order.
WordCounter.h includes Dictionary.h, it creates dictionary object and it fixes words and fills that dictionary with fixed words.

1.4. What are the extra works (bonuses) we have done?

In the list class we have destructor.
In the dictionary class we write copy constructer, overloaded assignment operator, and overloaded control operators.

2. Explanations of the Headers

2.1. List Header

There are struct Node and class List in the List.h. Node is a struct which contains node pointer next, int type counter, and template type data as data members. List is a class which contains Node template pointer head and tail, int type size as data members. It contains basic linked list functions. Basic linked list functions are its member functions. Also it contains a destructor which deletes the entire nodes if it were not, there would be memory leak.

What are the extras in that header?

-We write these classes with using template.

2.2. Dictionary Header

It includes List.h and it derived from List class. Some functions in the List class are modified for alphabetic order in Dictionary class.

There is an important point in this class; destructor is empty but there is no memory leaks since it calls base class' destructor.

What are the extras in that header?

- Copy constructor.
- Assignment operator overloaded.
- Equality operators; equal to and not equal to overloaded.

Member functions are:

- insertOrderly: It is a public function which takes 1 parameter - const string & - and it does not return any value. It takes words and adds to dictionary.
- getNodeByValue: It is a public function which takes 1 parameter - string & - and its return type is int. It returns counter of the given word. There is an important point in that function which is that thanks to 3rd if condition in while loop if given word greater than the comparison word that means searched word is not on the list so function returns 0.
- addToHead: It is a public function which takes const string and if it is alphabetically smaller than the head it calls List class' addToHead function if the new word is equal to current head it will increase head's counter by 1 if the new word isn't smaller than the current head it throws exception.
- addToTail: It is a public function which takes const string and if it is alphabetically greater than the tail it calls List class' addToTail function if the new word is equal to current tail it will increase tail's counter by 1 if the new word isn't greater than the current tail it throws exception.

Operators are:

- Assignment operator (=)
- Operator equal to (==)
- Operator Not equal to (!=)

2.2.1. InsertOrderly (Detailed)

There is a loop which controls if temp is not equal to tail every time and it controls every nodes in the dictionary. Also with *tail->next==NULL || tail->next==tail* statement it enters the loop even if there is only one element in the dictionary.

There are 6 cases which are controlled in that function.

These are:

I. We check whether list is empty or not in the first case. If it is empty, it calls addToHead function to create a new node.
II. We check the case that given word is already in the list if it does, it increases word's counter by 1.
III. We check given word is smaller than the head if it is it calls addToHead function.
IV. We check if list has only one element and given word greater than it. If it is, it calls addToTail.
V. We check if the word less than next node's data if it is, it creates a new node and terminate the function.
VI. If the function was not terminated because of the cases in the loop that means the word is equal to tail or greater than tail. We check the sixth if statement that if it equals to tail, if it is, function increases tail's counter by 1.

At the end of the function if function is not determined by previous cases, the word must be greater than tail so function calls addToTail function.

We realized that if the 4th case and 5th case were reversed order, there would be such an error because program tries to reach the NULL's next.

2.2.2. Assignment operator (=)

We have 6 cases in that operation which are:

I. If both objects are empty it returns *this* pointer.
II. Second case controls if the right object is empty. If it does, it deletes all nodes in the left object.
III. Third case controls if the left object is empty. If it does, it copies all nodes in the right object to the left object.
IV. There is a lucky case which is that if the counters of the both objects are equal. In this case we do not create new nodes we just assign right's data to the left.
V. If size of left less than right first it assigns left's list size from right's list to left and then with creating new nodes it assigns other data.
VI. In the last case we check that if the size of left object is greater than the right one. In this case we assign all data of right to the left object and after that we delete extra nodes in the left object. There is a critical point which is that if we were not deleted extra nodes there will be memory leaks.

2.2.3. Operator equal to (==)

We have 2 cases in that operation first we check that sizes are equal or not if these are not equal function returns false. Secondly if sizes are equal we must check all the node's datas are equal if there is some data which are not equal function returns false thanks to that loop does not check to end of the list. If all data equal it returns true.

2.2.4. Operator not equal to (!=)

It basically returns inverted case of operator equal to.

2.3. WordCounter Header

Purpose of WordCounter header is that opening the text file and adding words to the dictionary.

This header includes Dictionary.h. We created a Dictionary objects in the WordCounter class.

Data members are:

- string fileName
- Dictionary dic

Member functions are:

- fixWord: It is a private function which takes 1 parameter -string &- and its return type is string. It fixes the given words.
- fillDictionary: It is a private void function which takes no parameters.. It firstly check that the file exists if it is not, it throws an exception otherwise it opens the file read word by word send all words to the fixWord one by one and send fixed word to the dictionary object's insertOrderly function.
- getCounter: It returns counter. It called getNodeByValue with given word.
- getSize: It is a public function it basicly return dictionary's size.
- print: It prints all the list.
- listPopular: It prints most popular words on the list user enters that much.

2.3.1. fixWord (Detailed)

It is a helper function which does more than one job. It controls every letter of the given word one by one. It first does all the capital letters to lower case meantime it checks the character is a letter or not. And then checks the character is not letter and either first character or last if it is, program deletes character. If it is not the first or last character and it is not a letter, program checks that is the character is (-) or (‘) if it is, program controls the next index if the next character is a letter program deletes the current character which are (-) or (‘) character (i.e. grown-up becomes “grownup”). If the next character is not a letter than program gets the substring of the word from begin to the current index and sends the substring to the insertOrderly then continues to check other part(i.e. grown--up becomes two words “grown” and “up”) . If character is not (-) or (‘) and not letter program ends the substring to the insertOrderly then continues to check other part(i.e. grown?up becomes two words “grow” and “up”). At the end of the function it returns fixed word or empty string why word might be sent insertOrderly in the loop.

2.3.2. fillDictionary (Detailed)

It is a helper function. It takes the file name and open the file, if the file cannot be opened it throws an exception. It takes strings between spaces one by one and calls fixWord. If the returned string is not empty it calls insertOrderly, if it is empty that means words already added to the dictionary by insertOrderly. At the end it closes the file.

2.3.3. listPopular (Detailed)

It first checks that case given number is greater than the size of the list than throw an exception if it isn't function creates an array which size same as the list it sends nodes list to array and sorts with shell sort technique when the sorting finishes function prints array which size of user enters and then deletes the array.

3. Explanation of the Main.cpp

Main.cpp file includes WordCounter.h. There is menu function which includes:

(1) List the most popular words in the text.
(2) Get the total number of words.
(3) Get the counter of the given word.
(4) Print words.
(q) Quit.

In the main it creates WordCounter object and send file address then it sends object to menu function to give options to the user.