ALGORITHMS AND DATA STRUCTURES

PROJECT 1

Julia Czmut 300168

1. DESCRIPTION OF THE DECISIONS

1.1. CONCERNING FUNCTIONS AND CLASSES' INTERFACES

The program consists of two template classes: **Sequence** and **Counter**. Moreover, there is an external function **countWords** which uses a Counter object.

Sequence and Counter are both singly-linked lists. Node, Sequence's nested class holds the key for each element of the list. The elements with the same key can be repeatedly added to the list. Methods in Sequence allow the user to insert new elements anywhere they want.
There are three options: **insertFirst** which inserts the element at the beginning of the list,
insertLast which inserts the element at the end of the list and
insertAfter which takes the following arguments: the key of the element we want to insert, the key of the element after which we want this new element to appear and number of occurrence of this element. We can find out how many times an element appears on the list using **numberOfOccurrences** function.
On the other hand, class Counter contains unique keys and its nested class Node holds not only the key, but also the numberOfOccurrences of each element. So when adding a new element, using the **addKey** function, either a new unique Node is created or numberOfOccurrences of the existing Node on the list is incremented.

Counter and Sequence are connected through composition. Counter has a Sequence object inside. This way, the user has both objects in one place and can see the "unformatted" set of keys and the one reduced to only unique keys and their numbers of occurrences. There are many options of displaying the lists, or just parts of it, for example using **printXNodes** which counts the first X elements in a list. It is useful, because there may be a large number of elements in the list.
It was not necessary to implement iterators, however it would simplify the code.

Function **countWords** takes a Counter object and an istream as arguments. It uses the Counter object to specify how many times each word occurred in a text. Generally, it ignores the characters which are not letters, however it takes into consideration words like "don't", "haven't", "friends'" and treats them like any other word. Moreover, it converts all uppercase characters into lowercase so that it does not influence whether the word is unique or not. At the end, all the counted words are presented. The function also returns the number of unique words in the istream.

1.2. CONCERNING TESTS

The approach I took in testing my program is that I tried to make clear and transparent files with tests for the classes and the external function so that when they are run, they display every case. For classes, basic test cases are operations on empty lists, adding and removing elements. Tests for copy constructors, assignment operators, operators of equality, inequality and different versions of printing functions are also included. For the sake of simplicity of testing files for Counter and Sequence, their specified type was integer, but they will be also implemented as string type lists in testing of the function countWords.

As for countWords function, the testing file is interactive and structured in a way which allows the user to decide on either seeing how it works on my provided text file or supplying their own text through the console.

2. INTERFACES OF CLASSES

```
template <typename valType>
class Sequence{

        private:
        struct Node{
                valType key;
                Node *next;
        };
        Node *head;
        int size;
        Node* findNode(const valType& x, int occurrance = 1);
        Node* findPreviousNode(const valType& x, int occurrance = 1);

        public:
        Sequence();
        ~Sequence();
        Sequence(const Sequence<valType>& );
        void insertFirst(const valType& x);
        void insertLast(const valType& x);
        void insertAfter(const valType& x, const valType& where, int occurrance = 1);
        bool remove(const valType& x, int occurrance = 1);
        bool removeAllOccurrences(const valType& x);     // removes all occurrences of a given key in the sequence
        bool removeFromTo(const valType& x, int start = 1, int end = 0);
                // removes occurrences of a key starting from occurrence number specified by 'start' and ending
                // on the occurrence number specified by 'end'
        bool find(const valType& x, int occurrance = 1);
        int numberOfOccurrences(const valType& x);
        void print();
        void printNode(const valType& x);        // prints the key of a given node
        void printXNodes(int num);          // prints the first X nodes' keys
        int getSize();
        void clear();
        bool isEmpty();
        bool operator==(const Sequence<valType> &otherSequence);
        bool operator!=(const Sequence<valType> &otherSequence);
        Sequence<valType> &operator=(const Sequence<valType> &otherSequence);

};
```

```
template<typename Key>
class Counter{

        private:
        struct Node{
                Key key;
                int numberOfOccurrences = 0;
                Node* next;
        };
        Node* head;
        Sequence<Key> *sequence = new Sequence<Key>();
        int size;
        Node* findKey(const Key& x);
        Node* findPreviousKey(const Key& x);
        void addNode(Node* nodeToAdd);

        public:
        Counter();
        ~Counter();
        Counter(const Counter<Key>& );
        void addKey(const Key& x);     // adds at the end if key is unique, otherwise increments numberOfOccurrences
        bool removeKey(const Key& x);      // removes the node if it occurs only once, otherwise decrements
                                                        numberOfOccurrences
        bool removeNode(const Key& x);    // removes the node and does not care about the numberOfOccurrences
        void printCounter();
        void printSequence();     // prints the sequence object
        void printNode(const Key& x);     // prints the key and numberOfOccurrences of the given node
        void printXCountedNodes(int num);     // prints the first X nodes from Counter
        void printXNotCountedNodes(int num);     // prints the first X nodes from Sequence object
        int getSize();
        void clear();
        bool find(const Key& x);
        bool operator==(const Counter<Key> &otherCounter);
        bool operator!=(const Counter<Key> &otherCounter);
        Counter<Key> &operator=(const Counter<Key> &otherCounter);

};
```

3. EXTERNAL FUNCTION **countWords**

```
int countWords(Counter<string>& counter, istream& source){

        string key;
        stringstream ss;

        while(source){
        ss.put(source.get());
        }

        for(int i=0; i<ss.str().size(); i++){
                if((ss.str()[i] >= 'A' && ss.str()[i] <= 'Z') || ss.str()[i] >= 'a' && ss.str()[i] <= 'z'){
                        key += ss.str()[i];
                        continue;
                }
                if(char(ss.str()[i]) == 39){
                        if((ss.str()[i-1] >= 'A' && ss.str()[i] <= 'Z') || ss.str()[i] >= 'a' && ss.str()[i] <= 'z'){
                                key += ss.str()[i];
                                continue;
                        }
                }
                else{
                        if(key.size() > 0){
                                for(int i=0; i<key.size(); i++){    // switching uppercase letters to lowercase
                                        if(key[i] >= 'A' && key[i] <= 'Z'){
                                                key[i] = key[i] + 32;
                                        }
                                }
                                counter.addKey(key);
                                key.clear();
                        }
                else continue;
                }
        }
        counter.printCounter();
        return counter.getSize();
}
```

4. CONCLUSIONS

Counter and Sequence classes implementations allow the user to create lists of elements of any data type. Their functionalities differ because their expected purpose. Sequence is more likely to be a better choice for a list where the order of elements is important and where their uniqueness is not. It is a very versatile, not only because it is a template class, but also because it has many functionalities which can be used for a range of problems.
Counter has its purpose written into its name. It takes into consideration the uniqueness of elements and essentially counts them. It does not provide the functionalities for adding the elements wherever the user wishes, but considering the idea of its main possible purpose, it is hardly a problem. It can also be implemented for many problems and one of them was presented already in this project – in countWords function. This function could also have its equivalent for other data types. What is more, the class Counter has a Sequence object inside it, so the user does not have to give up on the idea of a general Sequence and has both things in one place.
All in all, there were many other ways of defining these classes, but they are just templates and so their real and interesting objective is what they could be used for later.