Singly linked list - an EADS project

Maciej Marcinkiewicz

November 2019

1 Introduction

The aim of the project was to implement singly linked list in the C++ programming language with use of class template mechanism. Template was meant to accept two generic variables - `Key` and `Info` which are the fields of each list's node. The first one's purpose is to behave as identifier, i.e. operations such as removing or inserting nodes are based on these. The latter contains information that is meant to be stored in nodes.

The second task of project was to implement funtion `Merge` which combines two lists in specific way and returns the result of that operation. The restrictions were to not include the funtion as a member of `Sequence` class (which is the name of list class) and to not use a keyword `friend`.

2 Overview of funtions

2.1 Constructors and destructor

```
Sequence()
```

Default constructor.

```
~Sequence()
```

Destructor.

```
Sequence(const Sequence<Key, Info> &other)
```

Copy constructor.

```
Node(const Key &key, const Info &info)
```

Constructor for `Node`. Creates new node with given key and information.

2.2 Overloaded operators

```
Sequence<Key, Info> &operator=(const Sequence<Key, Info> &other)
```

Assignment operator.

```
bool operator==(const Sequence<Key, Info> &other)
```

Comparison operator. Returns `true` if compared lists contain the same elements.

```
bool operator!=(const Sequence<Key, Info> &other)
```

Comparison operator. Returns `false` if compared lists contain the same elements.

```
Sequence<Key, Info> operator+(const Sequence<Key, Info> &other)
```

Returns combination of two list.

```
Sequence<Key, Info> &operator+=(const Sequence<Key, Info> &other)
```

Extends list by another list's elements.

2.3 Adding elements

```
void PushFront(const Key &key, const Info &info)
```

Method which adds new node of given key and information on the beginning of list.

```
void PushBack(const Key &key, const Info &info)
```

Method which adds new node of given key and information on the end of list.

```
bool Insert(const Key &key, const Info &info, const Key &after,
            int whichOccurance = 1)
```

Method which adds new node of given key and information after node with specified key. `whichOccurance` tells after which occurance of key `after` new node should be inserted. Returns `true` if operation was successful. Returns `false` if the proper node to insert after does not exist.

2.4 Removing elements

```
bool PopFront()
```

Method which removes the first node. Returns `false` if list is empty.

```
bool PopBack()
```

Method which removes the last node. Returns `false` if list is empty.

```
bool Remove(const Key &key, int whichOccurance = 1)
```

Method which removes node of given key. `whichOccurance` tells which occurance of key in list should be removed. Returns `false` if list is empty or there is no proper node to remove.

```
bool Clear()
```

Removes all nodes. Returns `false` if list is empty.

```
bool RemoveAllOccurances(const Key &key)
```

Method which removes all nodes with `key`. Returns `false` if list is empty or there are no nodes with such a key.

2.5 Other methods

```
bool IsEmpty() const
```

Returns `true` if list contains no nodes.

```
int Size() const
```

Returns private field `size` which conatains number of nodes in list.

```
bool Find(const Key &key, int whichOccurance = 1) const
```

Returns `true` if there is node of given key in the list. `whichOccurance` tells which occurance of key has to be found.

```
int NumberOfOccurances(const Key &key) const
```

Returns number of occurances of nodes with given key.

```
Sequence<Key, Info> GetFragmentOfSequence(int start,
                                          int length) const
```

Returns a list which is a part from the list in which this method has been called. `start` is a position of the first element taken and `length` indicates how many elements should be taken.

```
void Print() const
```

If possible, prints all elements. If list is empty prints message which informs about that.

```
void Print(std::ostream &os) const
```

The same as above with an addition of ostream as an argument. Message about how successful operation of printing was is passed into this ostream to make unit testing easier.

2.6 Merge function

```
Sequence<Key, Info> Merge(const Sequence<Key, Info> &sequence1,
                          int start1, int len1,
                          const Sequence<Key, Info> &sequence2,
                          int start2, int len2,
                          int count)
```

Returns `Sequence` which is a product of two other lists. `start1` and `start2` tell how many nodes from the beginning of lists should be skipped, `len1` and `len2` tell how many nodes should be taken from each of lists and finally `count` tells how many subsequences from each of lists should be taken i.e. how many times should be operation of taking `len1` and `len2` nodes repeated. If both lists are empty function returns an empty list.

Example

```
Input:
//info, keys for every node are the same
s1 = {1, 2, 3, 4, 5, 6, 7, 8, 9}
s2 = {10, 20, 30, 40, 50, 60, 70, 80, 90, 100, 110}

start1 = 2
start2 = 1

len1 = 3
len2 = 2

count = 4

Output:
//only info as keys are the same
Merge(s1, 2, 3, s2, 1, 2, 4) =
{3, 4, 5, 20, 30, 6, 7, 8, 40, 50, 9, 60, 70, 80, 90}
```

3 Decisions and changes

On the early stage my implementation had `operator[]` overloading. The idea was to make accessing nodes more intuitive and to make implementation of other methods easier. However this approach is not suitable for structure like linked list. As list is not a random access structure it is necessary to iterate through nodes of list to obtain specified node. Using this operator for example in removing all occurances of nodes with chosen key could be an operation with very high time complexity. The second problem is breaking encapsulation. With such a public operator node's `next` pointer could be changed in the outside of `Sequence` and it could lead to list's breaking (and in result of memory leak and losing access to some nodes).

With abandonment of `operator[]` new problem arised - accessing nodes of lists in `Merge` function. As it is impossible due to `Node` being private I had to implement a method which returns a fragment of list. Then with use of overloaded `+=` operator it was possible to easily make a new list which is a combination of fragments of two other lists.

To make testing possible I also implemented `Print` method which prints all values stored in the list. There is one restriction - data types stored in the list has to be valuable (or with `<<` operator overloaded). I considered an external method for this purpose but without access to the elements of list it is impossible.

4 Testing

4.1 Introduction

I decided to use Catch2 testing framework to get familiar with using such frameworks. With its help I divided tests on several test cases. These cases are also divided into sections. That makes them easier to be performed as in case of failure framework gives us exact information about test case and section in which assertion has been failed.

Unit test are performed in two source files, one for `Sequence` methods testing and the second for `Merge` function testing.

For the purpose of memory leaks checking I used Valgrind software.

4.2 Structure of tests

Empty list

- Printing list
- Checking if list is empty
- Using removal methods
- Using `Find` method

Filling list

- Adding nodes on the beginning
- Adding nodes on the end
- Adding nodes inside the list

Removing nodes from list

- Removing nodes from the beginning
- Removing nodes from the end
- Removing selected nodes
- Removing all nodes
- Removing all nodes with chosen key

Comparison operators

- Equality operator
- Inequality operator

Copying lists

- Copying with copy constructor
- Copying with assignment operator

Other methods

- Size method
- Find method
- Checking number of occurances

Other methods

- Size method
- Find method
- Checking number of occurances

Merge function

- Merging when both of lists are empty
- Merging when one of lists is empty
- Merging with filled lists