Middle East Technical University | Department of Computer Engineering

CENG 213

Data Structures

Fall '2017-2018

Programming Assignment 1

Due date: 16 November 2017, Thursday, 23:55

1 Introduction

This assignment aims to get you familiar with data structure concepts by implementing a playlist editor.

Keywords: *Singly Linked List, Classes, Templates, Stack*

2 Problem Definition

In this homework you are expected to implement a playlist editor. Editing the playlist is possible by inserting/deleting an entry, moving an entry to left or right, reversing the list, sorting the list or shuffle the list. Also merging two playlist entries should also be possible. The playlist editor has also a limited UNDO capability which will be able to undo only insert, delete and reverse operations.

3 Linked List, Node and Stack Templates

3.1 Node Template

This template represents a generic node structure. You are **not** allowed to declare additional private/public members inside this template. You will write the implementations of the methods inside Node.hpp file.

3.2 LinkedList Template

This template represents a singly linked list structure. You are **not** allowed to declare additional private/public members inside this template. Some of the methods in the linked list template are defined as follows:

```
/*Get the previous node of the node that contains the data item.
* If the head node contains the data item, this method returns NULL.*/
Node<T>* findPrev(const T& data) const;
/*Get the node that stores the data item.
* If data is not found in the list , this function returns NULL.*/
```

```
Node<T>* findNode(const T& data) const;
/*Insert a new node to store the data item.
* The new node should be placed after the prev node.
* If prev is NULL then insert new node to the head.*/
void insertNode(Node<T>* prev, const T& data);
/* This method is used to delete the node that is next to prevNode.
* PS:prevNode is not the node to be deleted. */
void deleteNode(Node<T>* prevNode);
```

The details of the rest of the functions can be found in LinkedList.hpp file.
You should write the implementations of the methods inside this file.

3.3 MyStack Template

You are expected to design a generic stack structure. You are free to add private/public members in this template. Your stack template **should** include the following methods:

```
template <class T>
class MyStack{
private:
Node<T> *top;
public:
/*Default constructor*/
MyStack();
/*copy constructor*/
MyStack(const MyStack<T>& rhs);
/*destructor*/
~MyStack();
/*overloaded = operator*/
MyStack<T>& operator=(const MyStack<T>& rhs);
/*returns true if stack is empty*/
bool isEmpty() const;
/*push newItem to stack*/
void push(const T& newItem);
/*pop item from stack*/
void pop();
/*return top item of the stack*/
Node<T>* Top() const;
};
```

You should write the implementations of the methods inside MyStack.hpp file.

4 PLAYLIST

You are expected to design a playlist system. In this system, playlist entries should be stored in the generic singly list structure you have implemented. Also for undo operations, you will store the operations in the generic stack structure you have implemented.

4.1 Entry Class

This class represents a playlist entry. You are not allowed to declare additional private/public members inside this class.

```
class Entry {
private:
std::string title;
std::string genre;
std::string year;
public:
Entry();
/* This constructor is the only way to set private members of the class*/
Entry(std::string _title, std::string _genre="", std::string _year="");
/*return Title of the entry*/
std::string getTitle() const;
/*return Genre of the entry*/
std::string getGenre() const;
/*return year of the entry*/
std::string getYear() const;
/* You are expected to overload the == operator.
* If the titles of two entries are same, this method returns true*/
bool operator==(const Entry & rhs) const;
/*This method will be used to print an entry
It was already implemented and you should not edit this method*/
friend std::ostream &operator<<(std::ostream &out, const Entry& t);
};
```

4.2 HistoryRecord class

We keep track of insert, delete and reverse operations to UNDO these operations. This class represents a record of such operations.

```
class HistoryRecord
{
private:
Entry entry;
Operation operation;
public:
/*default constructor*/
HistoryRecord();
/*constructor*/
HistoryRecord(Operation oper, Entry e=Entry());
/*returns one of the Operations: INSERT, DELETE or DELETE*/
Operation getOperation() const;
/*returns the playlist entry for the corresponding history record*/
Entry getEntry() const;
};
```

You are free to add private/public members to this class.

4.3 Playlist Class

This class represents a playlist editor. To store the playlist entries you are expected to use the LinkedList class and to UNDO some actions you are expected use MyStack class that you have implemented. You are free to add helper functions in this class. The methods in the playlist class are defined as follows:

4.3.1 void load(std :: string fileName)

This method loads the list of playlist entries from the given file.

- Each line in the file represents a new entry.
- Each entry should be inserted at the end of the current playlist.
- You should save the every insert operation for UNDO operation.
- We will provide records that have unique "title"s.
- The fields of an entry are seperated by semicolon(;). The line format is as follows:

Title	;	Genre	;	Year

- The title field is mandatory. The rest of the fields can be empty. The following are valid lines:

```
Billie Jean;dance-pop;1983
Sad but True;;1993
The Four Seasons;Classical;
Nocturne in F major;;
```

4.3.2 void insertEntry(const Entry &e);

Inserts a new entry to the end of playlist entries. For UNDO operation, you should save the insert operation.
PS:We will not insert a new entry that has same "title" of an entry in the playlist.

4.3.3 void deleteEntry(const std :: string & _title)

Deletes the entry with given title from the list. If the delete operation is successful (i.e. the entry with given title is in the playlist and deleted successfully) you should save the this operation for UNDO operation.

4.3.4 void moveLeft(const std::string &title)

Moves the entry with given title to the left. If it is the left-most entry, no action will be performed.

4.3.5 void moveRight(const std::string &title)

Moves the entry with given title to the right. If it is the right-most entry, no action will be performed.

4.3.6 void reverse()

This method is used to reverse the playlist entries. You are encouraged to use stack for the reverse operation. For UNDO operation, you should save the reverse operation.

4.3.7 void sort()

This method is used to sort the entries of the playlist from lowest to highest according to their "title"s.

- You are supposed to use "Selection Sort" algorithm.
- You will assume that all playlist entries have unique titles.
- **Using any of the stl containers(array, vector, list, etc.) is not allowed in this method.**

4.3.8 void merge(const Playlist & pl)

In this method, you will assume that you have 2 sorted playlists; the current playlist(1) and the given playlist as an input(2). You will merge the given playlist(2) into the current one(1). During and after the merging process the current list(1) should remain sorted. You are not allowed to use sort() method inside this method.

4.3.9 void shuffle ()

This method is used to shuffle the playlist entries according to the algorithm given below:

```
-- To shuffle an array a of n elements (indices 0..n-1):
for i from 0 to n-2 do
     j ← random integer such that i ≤ j < n
     exchange a[i] and a[j]
```

You should assume that the index of the first playlist entry is 0 and the index of the last playlist entry is (n-1). To generate a random number, you should use "getRandomNumber" method, which was already implemented and should not be modified.

4.3.10 void undo()

This method is used to UNDO the the operations INSERT, REVERSE and DELETE. You should save every such action to be able to UNDO the operation except for the actions performed in this function.

- to undo INSERT operation: Delete the previously inserted entry from the list.
- to undo DELETE operation : Insert the previously deleted entry **at the end of** the list.
- to undo REVERSE operation: Reverse the playlist again.
- There is no REDO operation of an UNDO operation. After performing UNDO, you should delete the operation from the history stack. Do not save the operations performed in this function at the history.

4.3.11 void print(), printHistory()

The print() method prints the entries of the playlist from left to right. The printHistory() method prints the content of the history stack from top to bottom. These methods were already implemented and should not be modified.

5 Regulations

1. **Programming Language:** You will use C++.
2. **S**tandard **T**emplate **L**ibrary is not allowed, you cannot use *vector*, *stack* etc. that are available in STL.
3. External libraries are not allowed.
4. **Late Submission:** Every student has a total of 7 days for late submission of the assignments. However, one can use at most 3 late days for this assignment. No penalty will be incurred for submitting within late 3 days. Your assignment will not be accepted if you submit more than 3 days late.
5. **Cheating:** We have zero tolerance policy for cheating. In case of cheating, all parts involved (source(s) and receiver(s)) get zero. People involved in cheating will be punished according to the university regulations.
6. **R**emember that students of this course are bounded to code of honor and its violation is subject to severe punishment.
7. **Newsgroup:** You must follow the newsgroup (news.ceng.metu.edu.tr) for discussions and possible updates on a daily basis.

6 Submission

- Submission will be done via Moodle.
- Do not write a *main* function in any of your source files.
- A test environment will be ready in Moodle.
  - You can submit your source files to Moodle and test your work with a subset of evaluation inputs and outputs.
  - This will not be the actual grade you get for this assignment, additional test cases will be used for evaluation.
  - Only the last submission before the deadline will be graded.