Middle East Technical University Department of Computer Engineering

CENG 213

Data Structures

Fall '2018-2019

Programming Assignment 1

Due date: 15 November 2018, Thursday, 23:55

1 Introduction

In this assignment you will implement a program which simulates the backend layer of an imaginary library VideoShare that offers subscription-based streaming media service.

Keywords: *Singly Linked List, Classes, Templates*

2 Problem Definition

The VideoShare allows storing videos and users. In addition, users of the VideoShare are able to subscribe videos and add friendships. The details of the classes are given in the following sections.

3 Video

This class represents a Video entry. For simplicity we only store 2 properties "title" and "genre" of a video. You are not allowed to declare additional private/public members inside this class. You should write the implementations of the methods inside Video.cpp file.

```
class Video {
private:
    /* title is unique */
    string title;
    string genre;
public:
    Video();
    Video(string p_title, string p_genre = "");
    ~Video();
    const string& getTitle() const;
    void setTitle(const string& p_title);
    const string& getGenre() const;
    void setGenre(const string& p_genre);
    bool operator==(const Video & rhs) const;
    friend ostream& operator<<(ostream& out, const Video & video);
};
```

- **bool operator==(const Video & rhs) const** : == operator checks the equalities of two videos. If their titles are same, then the videos are equal.

4 User

This class represents a User entry. A user's status is restricted to "ACTIVE" and "SUSPENDED". You are not allowed to declare additional private/public members inside this class. You should write the implementations of the methods inside User.cpp file.

```
enum Status
{
  ACTIVE,
  SUSPENDED
}
```

4.1 Private Data

We store username, name surname and status of a user. We also store the references of videos a user has subscribed, and references of friends of the user.

```
private:
    string username;
    string name;
    string surname;
    Status status;
    LinkedList< Video* > subscriptions;
    LinkedList< User* > friends;
```

4.2 Public Methods

```
public:
    User();
    User(string username, string name = "", string surname = "");
    ~User();
    const string& getUsername() const;
    const string& getName() const;
    const string& getSurname() const;
    Status getStatus() const;
    void updateStatus(Status st);
    void subscribe(Video * video);
    void unSubscribe(Video * video);
    void addFriend(User * user);
    void removeFriend(User * user);
    LinkedList<Video* > *getSubscriptions();
    LinkedList< User* > *getFriends();
    void printSubscriptions();
    void printFriends();
    bool operator==(const User& rhs) const;
    friend ostream& operator<<(ostream& out, const User & user);
```

- **User(string username, string name = "", string surname = "")** : This parametrized constructor is used to initialize a user object. The status of the new user should be ACTIVE.
- **updateStatus**: Updates the status of the user. Status is either ACTIVE or SUSPENDED.
- **subscribe**: Adds the given video pointer to the list of subscriptions. The new video pointer should be added to the beginning of the list. The actual video data that is pointed by this pointer is in the **videos** list in VideoShare instance.
- **unSubscribe**: Deletes the video pointer from the user's subscription list.
- **addFriend**: Adds the given user pointer to the list of friends. The user pointer should be added to the beginning of the list. The actual user data that is pointed by this pointer is in the **users** list in VideoShare instance.
- **removeFriend**:Deletes the user pointer from the user's friend list.
- **getSubscriptions**: Returns the pointer of the list of subscriptions;
- **getFriends**: Returns the pointer of the list of friends;
- **printSubscriptions,printFriends**: These functions are used to print the subscriptions and friends list. They were already implemented and should not be modified.
- **bool operator==(const User & rhs) const** : == operator checks the equalities of two users. If their usernames are same, then the users are equal.

5 VideoShare

Using VideoShare class, one will be able to create/load users, create/load videos, video subscription etc. You are free to add private helper functions in this class.

5.1 Private Data

We store a list of users and a list of videos. The singly linked list structure to store the data items is defined in 6.

```
private:
    LinkedList<User> users;
    LinkedList<Video> videos;
```

5.2 Public Methods

```
public:
    VideoShare();
    ~VideoShare();
    void createUser(const string & userName, const string & name = "", const string & surname = "")↩
    ;
    void loadUsers(const string & fileName);
    void createVideo(const string & title, const string & genre);
```

```
void loadVideos(const string & fileName);
void addFriendship(const string & userName1, const string & userName2);
void removeFriendship(const string & userName1, const string & userName2);
void updateUserStatus(const string & userName, Status newStatus);
void subscribe(const string & userName, const string & videoTitle);
void unSubscribe(const string & userName, const string & videoTitle);
void deleteUser(const string & userName);
void sortUserSubscriptions(const string & userName);
void printAllUsers();
void printAllVideos();
void printUserSubscriptions(const string & userName);
void printFriendsOfUser(const string & userName);
void printCommonSubscriptions(const string & userName1, const string & userName2);
void printFriendSubscriptions(const string & userName);
bool isConnected(const string & userName1, const string & userName2);
```

- **createUser**: Adds a new user to the list of users. The user entry should be added to the beginning of the users list with ACTIVE status. We will provide users that have unique "username"s.
- **loadUsers**: This method loads the list of user entries from the given file. We will provide records that have unique "username"s. Each line in the file represents a new entry and the fields of an entry are seperated by semicolon(;). From top to bottom, each user entry should be added to the beginning of the users list with ACTIVE status. The line format is as follows:

username	;	name	;	surname

  - The username field is mandatory. A line with empty username should not be accepted. The rest of the fields can be empty.

    E.g. If the file contains the entries below:

user1	;	John	;	Smith
user2	;	Joe	;	Black
user3	;	Mary	;	White

  Then the order of the users in the list should be as follows: $user3 -> user2 -> user1$

- **createVideo, loadVideos**: Similar to create/load users; these methods create/load videos. In these operations video entry should be added to the beginning of the videos list. We will provide videos that have unique "title"s. In case of loading, the line format is as follows:

title	;	genre

  The title field is mandatory. A line with empty title should not be accepted. Genre field can be empty.

- **addFriendship**: Adds a friend entry to the list of friends of both user named userName1 and user named userName2 if they exist. The entry should be added to the beginning of the friends list of the related users. For efficiency, you should only store the user pointer that points to the related user in the **users** list.

- **removeFriendship**: Delete the friendship of 2 users (if they exist) by deleting the corresponding user pointers in their friends list.
- **updateUserStatus**: Update the user's status to either ACTIVE or SUSPENDED.
- **subscribe**: If the user exists and is not SUSPENDED, this method adds a subscription entry to the list of subscriptions of the given user. The entry should be added to the beginning of the subscriptions list of the related user. For efficiency, you should only store the video pointer that points to the related video in the **videos** list.
- **unSubscribe**: Delete the subscription by deleting the corresponding video pointer in the subscriptions list of the user if he was not SUSPENDED.
- **deleteUser**: Deletes the user from the list of users. Do not forget to remove the mutual friendships.
- **sortUserSubscriptions**: Sorts the video subscriptions of the given user according to video titles. You should only change the sort order of the video pointers in the **subscriptions** list, not the actual video data. After the sort operation, the order of the subscriptions belonging to different users should remain unchanged. Also the order of nodes in the **videos** list should not be changed. Please note that, using any of the stl containers(array, vector, list, etc.) is not allowed in this method.

  For sorting operation you should use bubble sort algorithm. In the following code, the bubble sort algorithm given in slides is modified to iterate from left to right to be able to work on singly linked lists.

```
template <class Item>
void bubleSort(Item a[], int n) {
   bool sorted = false;
   for (int i = 0; (i < n-1) && !sorted; i++) {
      sorted = true;
      for (int j=1; j <= n-i-1; j++)
         if (a[j-1] > a[j]) {    //
            swap(a[j],a[j-1]);   // Swap these two
            sorted = false;      // Mark exchange
         }
   }
}
```

- **printAllUsers**: Prints the information of all users. This method was already implemented and should not be modified.
- **printAllVideos**: Prints the information of all viodes. This method was already implemented and should not be modified.
- **printUserSubscriptions**: Prints the videos one video information per line that the given user has subscribed . You should call printSubscriptions method for the given user.
- **printFriendsOfUser**: Prints the users one user information per line that the given user has friendship. You should call printFriends method for the given user.
- **printCommonSubscriptions**: Prints the videos that are common in both given users subscriptions. The output should be one video information per line. We will run this method after sorting the both subscriptions of the given users. Therefore you will **assume** that the subscriptions of both users are sorted.

- **printFriendSubscriptions**: Prints **all the distinct** videos that the given user's all of the friends have subscribed. The output should be alphabetically ascending ordered according to the titles. After the operation, the order of the subscription list of the given user and his friends should remain unchanged. Please note that, using any of the stl containers(array, vector, list, etc.) is not allowed in this method.
- **isConnected**: Checks whether 2 users are linked by some chain of friends even if they are not direct friends. This function returns true if the link exists.

6 Linked List & Node Structures

6.1 Node Template

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

```
class Node {
    private:
        Node<T> *next;
        T data;
    public:
        Node();
        Node(const T& d);
        Node<T>* getNext() const;
        T getData() const;
        T* getDataPtr();
        void setNext(Node<T> *newNext);
        friend ostream &operator<< <> (ostream &out, const Node<T>& n);
};
```

6.2 LinkedList.hpp

This template represents a singly linked list structure. The list uses dummy head for simplifying insertions and deletions.
You are **not** allowed to declare additional private/public members inside this template. You should write the implementations of the methods inside LinkedList.hpp.

```
class LinkedList {
private:
    Node<T>* head;
    int length;
public:
    LinkedList();
    LinkedList(const LinkedList<T>& ll);
    LinkedList<T>& operator=(const LinkedList<T>& ll);
    ~LinkedList();
    Node<T>* getHead() const;
    Node<T>* first() const;
    Node<T>* findPrev(const T& data) const;
    Node<T>* findNode(const T& data) const;
    void insertNode(Node<T>* prev, const T& data);
    void deleteNode(Node<T>* prevNode);
```

```
    void clear();
    size_t getLength() const;
    void print() const;
    void swap(int index1, int index2);
};
```

6.2.1 Private Members

- **head**: This represents the dummy head node and its next pointer will point to the first actual node of the list.
- **length**: This represents the size of the list. Length of the empty list with a dummy head is 0.

6.2.2 Public Methods

- **first()**: Returns the first actual node.
- **getHead()** : Returns the dummy head node.
- **findPrev**:Returns the previous node of the node that contains the data item. If data is not found, then this function returns NULL.
- **findNode**:Return the node that stores the data item. If data is not found, then this function returns NULL.
- **insertNode**: If "prev" node is not NULL, insert a new node that should be placed after the "prev" node.
- **deleteNode**: If "prev" node is not NULL, delete the node that is next to "prevNode".
- **clear**: Clears the content of the list. After this operation the list should contain no nodes except the dummy head. In addition there should be no memory leak.
- **getLength**: Returns the size of the list. Length of the empty list with a dummy head is 0.
- **swap**: Swaps the two nodes whose indexes are given as input. Please note that, index 0 is the first node of the list; not the dummy head.
- **print**: Prints the content of the list. This function was already implemented and should not be modified.

7 Regulations

1. **Programming Language:** You will use C++.
2. **S**tandard **T**emplate **L**ibrary is not allowed, you cannot use *vector* etc. that are available in STL.
3. External libraries are not allowed.
4. **Late Submission:** A penalty of 5 x $daysLate^2$ will apply for late submissions. 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.

8 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.