CS 201, Fall 2022
Homework Assignment 3
Due: 23:59, December 11, 2022

In this homework, you will implement a simple movie rental system using linked lists. The movie rental system will consist of a collection of movies and a collection of subscribers. The subscribers will be able to rent movies and will be able to return the movies to the movie rental system as described below. The system will store all rental transactions. Each transaction will contain the subscriber id and the id of the movie that the subscriber rented, and whether or not this movie is returned to the rental system.

In your implementation, you **MUST** keep the subscribers in a sorted linked list where the subscribers are stored as sorted according to their subscriber ids and **MUST** keep the movies in a sorted linked list where the movies are stored as sorted according to their movie ids. Also, for each of these subscribers, you **MUST** use another sorted linked list to store their transactions where the transactions are stored as sorted according to their movie ids. All lists are expected to be linear singly linked lists.

The movie rental system will have the following functionalities. The details of these functionalities are given below:

1. Load movies and subscribers from input files
2. Remove a movie
3. Add a movie
4. Remove a subscriber
5. Rent a movie by a subscriber
6. Return a movie by a subscriber
7. Show the list of all movies rented by a subscriber
8. Show the list of all subscribers who have rented a particular movie
9. Show the list of all movies
10. Show the list of all subscribers

Load movies and subscribers from input files: The movie rental system will load the collection of movies and the list of subscribers from the given two input files given at its declaration: one for the collection of movies and the other one for the list of subscribers. Therefore, you will write a constructor for the `MovieRentalSystem` class below. This constructor will take the names of the input files as parameters, read the contents of the files, and store the movie and subscriber information in linked lists. If at least one of these input files does not exists, your system should give a warning message.

Each movie will have a movie id and can have multiple copies. The input file containing movie information will include the number of movies in the first line of the file, and will have one line per movie in the rest of the file. For the sake of simplicity, after the first line, each line will contain a movie id and the number of copies of that movie. You may assume that the movie ids are unique positive integers and the number of copies for each movie is a positive integer.

Each subscriber will have a subscriber id. The input file containing subscriber information will consist of the number of subscribers in the first line of the file, and will have one line per subscriber in the rest of the file. After the first line, each line will contain a subscriber id. You may assume that the subscriber ids are unique positive integers.

Remove a movie: The movie rental system will allow to remove an existing movie specified with its movie id. If the movie does not exist (i.e., if there is no movie with the specified id), the system should display a warning message. This operation cannot remove the movie at least one copy of which has currently been rented by a subscriber but has not been returned. In this case, it should display a message.

Add a movie: The movie rental system will allow to add a new movie specified with its movie id. If the movie already exists (i.e., if there is a movie with the specified id), the system should display a warning message. Otherwise the system should add the movie sorted by id.

Remove a subscriber: The movie rental system will allow to remove an existing subscriber specified with its subscriber id. If the subscriber does not exist (i.e., if there is no subscriber with the specified id), the system should display a warning message. This operation will remove all of the transactions for this subscriber if all of the movies rented by this subscriber have been returned. If there is at least one movie which has currently been rented by the subscriber but has not been returned, the system will not allow this operation and should display a message.

Rent a movie by a subscriber: The movie rental system will allow to rent a particular movie by a particular subscriber. For this operation, the movie id and the subscriber id have to be specified. The system should first check whether or not this movie exists; if it does not, it should not allow the operation and display a warning message. The system should also check whether or not this subscriber exists; if he/she does not, it should not allow the operation and display a warning message. If both the movie and the subscriber exist and the movie has a copy available for renting, the renting operation must be performed. If both the movie and the subscriber exist but the movie does not have a copy available for renting, a warning message should be displayed. Note that a subscriber can rent multiple movies and multiple copies of the same movie if available.

Return a movie by a subscriber: The movie rental system will allow a subscriber to return a movie. For this operation, the subscriber id and the movie id have to be specified. If there is no transaction corresponding to the given subscriber id and movie id, the system will display a warning message. Note that a transaction is characterized with subscriber id and movie id. Thus, both of these must be matched. If they all match, the system will record that the rental transaction has been completed and that movie can now be rented by other subscribers. The system should store all transaction history and should not remove any transaction information. However, in order to understand that the movie has been returned, it should modify the corresponding transaction accordingly. For example, if a movie was rented by a subscriber and was already returned, a new request for returning the movie should result in a warning message.

Show the list of all movies rented by a subscriber: The movie rental system will allow to display the list of all movies that were rented by a subscriber id. This list includes, for each movie, the movie id and whether it was returned or not. Note that this operation displays all movies – those were rented and returned as well as those were rented but not returned. If the subscriber has not rented any movie, the system should display a warning message. If the subscriber does not exist (i.e., if there is no subscriber with the specified id), the system should display a warning message.

Show the list of all subscribers who have rented a particular movie: The movie rental system will allow to display the list of all subscribers that rented a particular movie. This list includes, for each subscriber, the subscriber id and whether the movie was returned or not. If no one has rented that particular movie, the system displays a warning message. If the movie does not exist (i.e., if there is no movie with the specified id), the system should display a warning message.

Show the list of all movies: The movie rental system will allow to display a list of all movies. The list includes the movie id and the number of copies for these movies.

Show the list of all subscribers: The movie rental system will allow to display a list of all subscribers. The list only includes the subscriber id for these subscribers.

Below is the required `public` part of the `MovieRentalSystem` class that you must write in this assignment. The name of the class must be `MovieRentalSystem`, and must include these public member functions. We will use these functions to test your code. The interface for the class must be written in a file named `MovieRentalSystem.h` and its implementation must be written in a file named

MovieRentalSystem.cpp. You can define additional public and private member functions and data members in this class. You can also define additional classes in your solution.

```
class MovieRentalSystem {

public:
    MovieRentalSystem( const string movieInfoFileName,
                       const string subscriberInfoFileName );
    ~MovieRentalSystem();

    void removeMovie( const int movieId );
    void addMovie( const int movieId, const int numCopies );

    void removeSubscriber( const int subscriberId );

    void rentMovie( const int subscriberId, const int movieId );
    void returnMovie( const int subscriberId, const int movieId );

    void showMoviesRentedBy( const int subscriberId ) const;
    void showSubscribersWhoRentedMovie( const int movieId ) const;

    void showAllMovies() const;
    void showAllSubscribers() const;
};
```

Here is an example test program that uses this class and the corresponding output. We will use a similar (but different) program to test your solution so make sure that the name of the class is MovieRentalSystem, its interface is in the file named MovieRentalSystem.h, and the required functions are defined as shown above. Since we also test your solution with different programs, you should make sure that your solution will work for other programs (main functions) as well. Thus, test your solution with your own driver program. You can assume that the movie ids and subscriber ids are entered as positive integers.

Example test code:

```
#include <iostream>
#include "MovieRentalSystem.h"
using namespace std;

int main( ) {

    MovieRentalSystem firstMRS( "my_movies.txt", "subscribers.txt" );
    cout << endl;

    MovieRentalSystem MRS( "movies.txt", "subscribers.txt" );
    cout << endl;

    MRS.showAllMovies();
    cout << endl;
    MRS.showAllSubscribers();
    cout << endl;

    MRS.removeMovie( 600 );
    MRS.removeMovie( 600 );
    MRS.removeMovie( 70 );
    cout << endl;

    MRS.addMovie( 99, 3 );
    cout << endl;
    MRS.showAllMovies();
    cout << endl;

    MRS.rentMovie( 1111, 10 );
    MRS.rentMovie( 1111, 20 );
    MRS.rentMovie( 7777, 10 );
    MRS.rentMovie( 6666, 10 );
    MRS.rentMovie( 7777, 80 );
    MRS.rentMovie( 6666, 100 );
    MRS.rentMovie( 4444, 40 );
    MRS.rentMovie( 4444, 50 );
    MRS.rentMovie( 1234, 30 );
    MRS.rentMovie( 1111, 33 );
    MRS.rentMovie( 1234, 33 );
    cout << endl;

    MRS.returnMovie( 7777, 80 );
    MRS.returnMovie( 6666, 100 );
    MRS.returnMovie( 1111, 20 );
    MRS.returnMovie( 4444, 10 );
    MRS.returnMovie( 1111, 20 );
    cout << endl;
    MRS.rentMovie( 3333, 80 );
    MRS.rentMovie( 8888, 100 );
    MRS.rentMovie( 1111, 100 );
    MRS.rentMovie( 1111, 100 );
    cout << endl;
    MRS.rentMovie( 1111, 20 );
    MRS.returnMovie( 1111, 20 );
    MRS.rentMovie( 1111, 20 );
    cout << endl;
    MRS.showMoviesRentedBy( 1111 );
    MRS.showMoviesRentedBy( 8888 );
    MRS.showMoviesRentedBy( 2222 );
    MRS.showMoviesRentedBy( 1234 );
    cout << endl;
    MRS.showSubscribersWhoRentedMovie( 10 );
    MRS.showSubscribersWhoRentedMovie( 100 );
    MRS.showSubscribersWhoRentedMovie( 30 );
    MRS.showSubscribersWhoRentedMovie( 70 );
    cout << endl;

    MRS.removeMovie( 100 );
    cout << endl;

    MRS.removeSubscriber( 1111 );
    cout << endl;

    MRS.removeSubscriber( 5555 );
    cout << endl;
    MRS.removeSubscriber( 6666 );
    cout << endl;
    MRS.showAllSubscribers();
    cout << endl;
    MRS.showAllMovies();
    cout << endl;

    return 0;
}
```

The input file “subscribers.txt” has the following content:

```
9
8888
3333
6666
1111
2222
9999
5555
4444
7777
```

The input file “movies.txt” has the following content:

```
15
100 3
400 1
90 1
10 2
30 1
600 1
40 2
60 3
50 6
70 2
500 1
80 1
20 1
200 1
300 3
```

Output of the example test code:

```
Input file my_movies.txt does not exist

9 subscribers and 15 movies have been loaded

Movies in the movie rental system:
10 2
20 1
30 1
40 2
50 6
60 3
```

```
70 2
80 1
90 1
100 3
200 1
300 3
400 1
500 1
600 1

Subscribers in the movie rental system:
1111
2222
3333
4444
5555
6666
7777
8888
9999

Movie 600 has been removed
Movie 600 does not exist
Movie 70 has been removed

Movie 99 has been added

Movies in the movie rental system:
10 2
20 1
30 1
40 2
50 6
60 3
80 1
90 1
99 3
100 3
200 1
300 3
400 1
500 1

Movie 10 has been rented by subscriber 1111
Movie 20 has been rented by subscriber 1111
Movie 10 has been rented by subscriber 7777
Movie 10 has no available copy for renting
Movie 80 has been rented by subscriber 7777
Movie 100 has been rented by subscriber 6666
Movie 40 has been rented by subscriber 4444
Movie 50 has been rented by subscriber 4444
Subscriber 1234 does not exist
Movie 33 does not exist
Subscriber 1234 and movie 33 do not exist

Movie 80 has been returned by subscriber 7777
Movie 100 has been returned by subscriber 6666
Movie 20 has been returned by subscriber 1111
No rental transaction for subscriber 4444 and movie 10
No rental transaction for subscriber 1111 and movie 20

Movie 80 has been rented by subscriber 3333
Movie 100 has been rented by subscriber 8888
Movie 100 has been rented by subscriber 1111
Movie 100 has been rented by subscriber 1111

Movie 20 has been rented by subscriber 1111
Movie 20 has been returned by subscriber 1111
Movie 20 has been rented by subscriber 1111

Subscriber 1111 has rented the following movies:
10 not returned
20 returned
20 returned
20 not returned
100 not returned
100 not returned
Subscriber 8888 has rented the following movies:
100 not returned
Subscriber 2222 has not rented any movies
Subscriber 1234 does not exist

Movie 10 has been rented by the following subscribers:
1111 not returned
7777 not returned
Movie 100 has been rented by the following subscribers:
1111 not returned
1111 not returned
6666 returned
8888 not returned
Movie 30 has not been rented by any subscriber
Movie 70 does not exist

Movie 100 cannot be removed -- at least one copy has not been returned

Subscriber 1111 cannot be removed -- at least one movie has not been returned

Subscriber 5555 has been removed
Subscriber 6666 has been removed

Subscribers in the movie rental system:
1111
2222
3333
4444
7777
8888
9999

Movies in the movie rental system:
10 2
20 1
```

```
30 1
40 2
50 6
60 3
80 1
90 1
99 3
100 3
200 1
300 3
400 1
500 1
```

IMPORTANT NOTES:
Do not start your homework before reading these notes!!!

NOTES ABOUT IMPLEMENTATION:

1. You ARE NOT ALLOWED to modify the given parts of the header file. You MUST use linked lists in your implementation. You will get no points if you use dynamic or fixed-sized arrays or any other data structures such as vectors/arrays from the standard library. However, if necessary, you may define additional data members and member functions. You may also define additional classes.
2. Moreover, you ARE NOT ALLOWED to use any global variables or any global functions.
3. Output message for each operation MUST match the format shown in the output of the example code.
4. Your code MUST use linked lists to store movie, subscriber, and transaction data in your implementations. You must use your own implementations of linked lists. You can adapt the codes and pseudocodes in the Carrano book. However, you are NOT allowed to use any existing codes from other sources (including the codes given in other textbooks, found on the internet, and belonging to your classmates, etc.). Furthermore, you are NOT allowed to use any data structure or related function from the C++ standard template library (STL).
5. Your code MUST NOT have any memory leaks. You will lose points if you have memory leaks in your program even though the outputs of the operations are correct. To detect memory leaks, you may want to use Valgrind which is available at `http://valgrind.org`.
6. Otherwise stated in the description, you may assume that the inputs for the functions are always valid (e.g., number of copies of a movie is a valid positive integer number) so that you do not need to make any input checks.

NOTES ABOUT SUBMISSION:

1. In this assignment, you must have separate interface and implementation files (i.e., separate `.h` and `.cpp` files) for your class. We will test your implementation by writing our own driver `.cpp` file which will include your header file. For this reason, your class' name MUST BE "`MovieRentalSystem`" and your files' name MUST BE "`MovieRentalSystem.h`" and "`MovieRentalSystem.cpp`". You should submit these two files (and any additional files if you wrote additional classes in your solution) **as a single archive file (.zip file)**.
2. The code (`main` function) given above is just an example. We will test your implementation using different scenarios, which will contain different function calls. Thus, do not test your implementation only by using this example code. We recommend you to write your own driver files to make extra tests. However, you MUST NOT submit these test codes (we will use our own test code). In other words, do not submit a file that contains a function named `main`.

3. You should put all of your `.h` and `.cpp` files into a folder and zip the folder (in this zip file, there should not be any file containing a `main` function). The name of this zip file should conform to the following name convention: secX-Firstname-Lastname-StudentID.zip where X is your section number. The submissions that do not obey these rules will not be graded.

4. Make sure that each file that you submit (each and every file in the archive) contains your name, section, and student number at the top as comments.

5. You are free to write your programs in any environment (you may use Linux, Windows, MacOS, etc.). On the other hand, we will test your programs on "dijkstra.ug.bcc.bilkent.edu.tr" and we will expect your programs to compile and run on the dijkstra machine. If we could not get your program properly work on the dijkstra machine, you would lose a considerable amount of points. Thus, we recommend you to make sure that your program compiles and properly works on dijkstra.ug.bcc.bilkent.edu.tr before submitting your assignment.

6. This assignment is due by 23:59 on Sunday, December 11, 2022. You should upload your work to Moodle before the deadline. No hardcopy submission is needed. The standard rules about late homework submissions apply. Please see the course home page for further discussion of the late homework policy.

7. We use an automated tool as well as manual inspection to check your submissions against plagiarism. Please see the course home page for further discussion of academic integrity and the honor code for programming courses in our department.

8. This homework will be graded by your TA Mahmud Sami Aydın (sami.aydin at bilkent.edu.tr). Thus, you may ask your homework related questions directly to him. There will also be a forum on Moodle for questions.