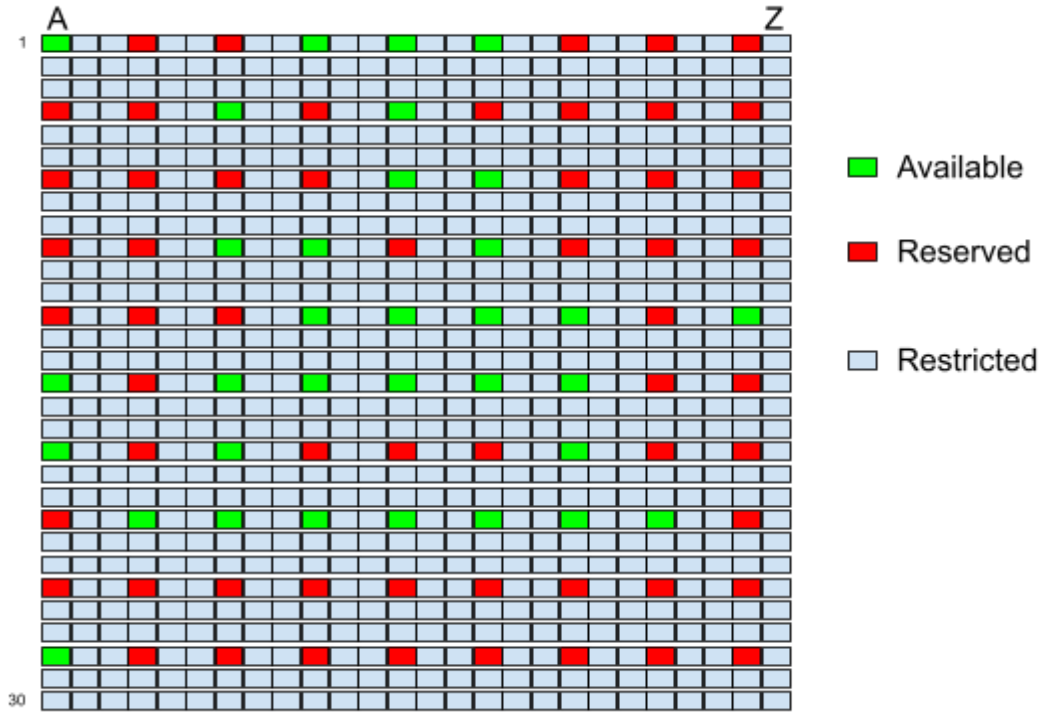
CS 201, Spring 2021

Homework Assignment 3

Due: 23:55, May 3 (Monday), 2021

In this homework, you will implement a ticket booking system for a movie theater, which stages multiple movies. **This time you will be using LINKED LISTS.** Each movie has a unique movie id. Although all movies are shown in the same movie theater, because of COVID-19 policies, some movies have different seating plans. According to policy at the time of the movie, some rows and columns should be empty in the plan. The theater has 26 columns which are indexed by the English uppercase alphabet and rows which are numbered 1 to 30 (e.g. C29). Audience radius, which is the number of empty seats between two horizontal or vertical neighbor audiences, should be indicated for each movie. Also for all seating plans, seat A1 is occupiable. "Occupiable" seat means making a reservation is possible for that seat if no one made a reservation yet and no restriction in the plan for that seat. A seat is available if it is occupiable and not reserved.



An example of a seating plan with audience radius 2 is presented below:

The booking system that you will implement should have the following functionalities. The details are given below.

1. Add a movie
2. Cancel a movie
3. Show the list of movies
4. Show detailed information about a particular movie
5. Make a reservation
6. Cancel a reservation
7. Show a reservation

Add a movie: The booking system will allow the user to add a new movie indicating movie id which is an integer and audience radius. Since the movie numbers should be unique within the system, you should check whether a movie with the same number exists. If it does, you should not allow the operation and display a proper warning message. Also, the audience radius is an integer and should be in the range [0,7]. If the radius is invalid, again you should not allow the operation and display another proper warning message. If both conditions are valid, you should add the movie and display a message. At the time of adding a movie, you should allocate its occupiable seats all of which should be available for reservation. Also, you shouldn't

allocate memory for restricted seats which are empty because of COVID-19 policies. (Please see the code given below and its output for the format of the output.)

Cancel a movie: The booking system will allow the user to cancel an existing movie indicating its movie id. If there exists no movie with the specified id, you should display a warning message. If the movie exists, you should delete it from the booking system and cancel all of the reservations that were made for this movie. You do not need to display the cancelation of reservations. You should also display a message that the movie has been canceled. (Please see the code given below and its output for the format of the output.)

Show the list of movies: The booking system will allow the user to see all of the movies within the booking system. For each movie, you should output the corresponding date and time to the movie id together with the number of its available seats. To find the corresponding date and time use the `ctime(&rawtime )` function in the `<ctime>` library . In this case you use movie id as a rawtime. It is not an obligation but if you eliminate the newline from the function it would be good exercise for arrays in C++. (Please see the code given below and its output for the format of the output.)

Show detailed information about a particular movie: The booking system will allow the user to enter a movie id and see the availability of its seats. If there exists no movie with the specified movie id, you should display a warning message. If the movie exists, you should display the seats in the following format where "o" represents an available seat and "x" represents an occupied seat. You should only show occupiable seats. Note that in the example below, the audience radius is 4.

```
    A  F  K  P  U  Z
 1  x  x  o  o  x  x
 6  o  o  o  x  o  x
11  o  o  o  o  x  x
16  o  o  o  o  o  o
21  x  o  o  o  x  o
26  o  x  x  o  o  o
```

Make a reservation: The booking system will allow the user to make a reservation in a movie given for one audience. For that, the user will specify the movie id which is represented by long integer, and seat number which is represented by row (int) and column (char). This function makes a reservation only if the movie with a specified movie id exists and the selected seat is occupiable and available. If there exists no movie with the specified movie id, you should display a warning message. Likewise, if the selected seat is not occupiable nor available, you should display a warning message accordingly. For example, the user can make a reservation for A6 in this plan but cannot do it for A7. (Please see the code given below and its output for the format of the output.)

If both the movie exists and all of the selected seats are available, you should make this reservation by creating a unique reservation code, and by reserving the selected seat under this code. The reservation code should be unique not only for the specified movie but also for the entire booking system. Thus, by using only this reservation code, you should identify the movie as well as the seat reserved under this code. Also, you should display a message that the reservation is done. (Please see the code given below and its output for the format of the output.)

In this function, you may assume that the input arrays for the seat numbers contain only the valid values which are A-Z in the column and 1-30 in the row. However, you should check whether the seat is occupiable or not before checking availability. It is a good idea to use modular arithmetic to check occuiablity.

Cancel reservations: The booking system will allow the user to cancel the specified linked list of the reservation codes. If some of the codes do not exist, you should display a warning message. If all reservations exist, you should cancel all reservations and display the time of the movie and the seat number for which the reservation is canceled. You are expected to preserve the order of reservation codes in input when displaying the canceled reservations. Also, you can assume that reservation codes are unique in the list. (Please see the code given below and its output for the format of the output.)

Show a reservation: The reservation system will allow the user to display a reservation by specifying its reservation code. If there exists no reservation under this code, you should display a warning message. If the

reservation exists, you should display the time of the movie and the selected seat number for this reservation. (Please see the code given below and its output for the format of the output.)

This time, you should implement your list of movies, their seats, and the reservation codes as linked lists. For the list of the reservation codes, a class definition. You can re-use code on lecture slides. You may add some other data into **ReservationNode** and some other functions into the **ReservationList** according to your design but you shouldn't change the part given below.

```
class ReservationList{
public:
        ReservationList();
        ReservationList( const ReservationList& aList );
        ~ReservationList();

        bool isEmpty() const;
        int getLength() const ;
        bool retrieve(int index, int& resCode) const;
        bool insert(int index, int resCode);
        bool remove(int index);

private:
         struct ReservationNode{
              int Code;
              ReservationNode* next;
         }
         int size;
         ReservationNode *head;
         ReservationNode *find(int index) const;

        // ...
        //you may define additional member functions and data members, if
necessary
}
```

Below is the required public part of the `MovieBookingSystem` class that you must write in this assignment. The name of the class **must** be **`MovieBookingSystem`** and must include these public member functions. The interface for the class must be written in a file called `MovieBookingSystem.h` and its implementation must be written in a file called `MovieBookingSystem.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 MovieBookingSystem{
public:
        const static int numOfRow = 30;
        const static int numOfColumn = 26;
        MovieBookingSystem();
        ~MovieBookingSystem();

        void addMovie( const long movieID, const int audienceRadius );
        void cancelMovie( const long movieID);
        void showAllMovies() const;
```

```
        void showMovie( const long movieID) const;
        int  makeReservation( const long movieID, const int row, const char
col);
        void cancelReservations( ReservationList resCode);
        void showReservation( const int resCode) const;
private:
        ReservationList reservationCodes;
        // ...
        //you may define additional member functions and data members, if
necessary.

}
```

Here is a test program that uses this class. The corresponding output is also given below.

EXAMPLE TEST CODE:

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

int main(){
    MovieBookingSystem M;
    int code = new int[2];
    ReservationList RL;

    M.showAllMovies();
    cout << endl;
    M.addMovie(1614791273,0);
    M.addMovie(1614801273,3);
    M.addMovie(1615801273,1);
    cout << endl;
    M.showAllMovies();
    cout << endl;
    M.addMovie(1614791273,2);
    cout << endl;
    codes[1] = M.makeReservation(1614801273,1,'A');
    codes[0] = M.makeReservation(1614801273,9,'A');

    RL.insert(0, codes[1]);
    RL.insert(0, codes[0]);

    cout<<endl;
    M.showMovie(1614801273);
    cout<<endl;

    M.showReservation( codes[0] );
    cout<<endl;

    M.cancelReservations( RL );
    cout<<endl;
```

```
    codes[0] = M.makeReservation(1614801273,1,'A');
    M.makeReservation(1614801273,1,'A');
    M.makeReservation(1614801273,5,'E');
    M.makeReservation(1614801273,5,'M');

    cout<<endl;
    RL.remove(0);
    RL.remove(0);
    RL.insert(0,-1);
    RL.insert(0, codes[0] );
    M.cancelReservations( RL );
    M.showReservation( codes[0] );
    cout<<endl;

    M.makeReservation(1614801273,12,'D');
    cout<<endl;
    M.showMovie(1614801273);
    cout<<endl;

    M.cancelMovie(1614801273);
    M.showReservation( codes[0] );

    cout<<endl;
    M.cancelMovie(1614801273);
    M.addMovie(1614801273,4);
    M.showMovie(1614801273);
    return 0;
}
```

OUTPUT OF THE EXAMPLE TEST CODE:

```
No movie on show

Movie at Wed Mar  3 20:07:53 2021 has been added
Movie at Wed Mar  3 22:54:33 2021 has been added
Movie at Mon Mar 15 12:41:13 2021 has been added

Movies on show:
Movie at Wed Mar  3 20:07:53 2021 (780 available seats)
Movie at Wed Mar  3 22:54:33 2021 (56 available seats)
Movie at Mon Mar 15 12:41:13 2021 (195 available seats)

Movie at Wed Mar  3 20:07:53 2021 already exists

Reservation done for A1 in Movie at Wed Mar  3 22:54:33 2021
Reservation done for A9 in Movie at Wed Mar  3 22:54:33 2021
```

```
Movie at Wed Mar  3 22:54:33 2021 has 54 available seats
   A E I M Q U Y
 1 x o o o o o o
 5 o o o o o o o
 9 x o o o o o o
13 o o o o o o o
17 o o o o o o o
21 o o o o o o o
25 o o o o o o o
29 o o o o o o o

Reservation with Code 2: Seat A9 in Movie at Wed Mar  3 22:54:33 2021

Reservation on Code 2 is canceled: Seat A9 in Movie at Wed Mar  3
22:54:33 2021
Reservation on Code 1 is canceled: Seat A1 in Movie at Wed Mar  3
22:54:33 2021

Reservation done for A1 in Movie at Wed Mar  3 22:54:33 2021
Seat A1 is not available in Movie at Wed Mar  3 22:54:33 2021
Reservation done for E5 in Movie at Wed Mar  3 22:54:33 2021
Reservation done for M5 in Movie at Wed Mar  3 22:54:33 2021

Some reservation codes do not exist. Cancelation is failed
Reservation with Code 3: Seat A1 in Movie at Wed Mar  3 22:54:33 2021

Seat D12 is not occupiable in Movie at Wed Mar  3 22:54:33 2021

Movie at Wed Mar  3 22:54:33 2021 has 53 available seats
   A E I M Q U Y
 1 x o o o o o o
 5 o x o x o o o
 9 o o o o o o o
13 o o o o o o o
17 o o o o o o o
21 o o o o o o o
25 o o o o o o o
29 o o o o o o o

Movie at Wed Mar  3 22:54:33 2021 has been canceled
No reservation with Code 3

Movie at Wed Mar  3 22:54:33 2021 does not exist
Movie at Wed Mar  3 22:54:33 2021 has been added
Movie at Wed Mar  3 22:54:33 2021 has 36 available seats
   A F K P U Z
 1 o o o o o o
 6 o o o o o o
11 o o o o o o
16 o o o o o o
21 o o o o o o
26 o o o o o o
```

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 arrays, fixed--sized arrays or any other data structures such as `vector/array` from the standard library**. However, if necessary, you may define additional data members and member functions.
2. Moreover, you ARE NOT ALLOWED to use any global variables or any global functions..
3. 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.
4. Otherwise stated in the description, you may assume that the inputs for the functions (e.g., the seat numbers) are always valid so that you do not need to make any input checks.
5. Make sure that each file that you submit (each and every file in the archive) contains your name and student number at the top as comments.

NOTES ABOUT SUBMISSION:

This assignment is due by 23:55 on May 3 2021. **This homework will be graded by your TA Mahmud Sami Aydın (sami.aydin at bilkent edu tr). Please direct all your homework related questions to him.**

1. In this assignment, you must have separate interface and implementation files (i.e., separate .h and .cpp files) for your class. The file names should be "`MovieBookingSystem.h`" and "`MovieBookingSystem.cpp`". You should also submit other .h and .cpp files if you implement additional classes. We will test your implementation by writing our own main function. Thus, you should not submit any function that contains the main function.

   Although you are not going to submit it, we recommend you to write your own driver file to test each of your functions. However, you SHOULD NOT submit this test code (we will use our own test code).

2. The code (main function) given above is just an example. We will test your implementation also using different main functions, which may contain different function calls. Thus, do not test your implementation only by using this example code. Write your own main functions to make extra tests (however, do not submit these test codes).

3. You should put your "`MovieBookingSystem.h`" and "`MovieBookingSystem.cpp`" (and additional .h and .cpp files if you implement additional classes) into a folder and zip the folder (in this zip file, there should not be any file containing the 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, for example Sec01_MahmudSami_Aydin_21501578.zip .

   The submissions that do not obey these rules will not be graded.

4. **Then, before 23:55 on May 3, you need to upload this zipped file containing only your header and source codes (but not any file containing the main function) to Moodle.**

   No hardcopy submission is needed. The standard rules about late homework submissions apply. Please see the course syllabus for further discussion of the late homework policy as well as academic integrity.

5. You are free to write your programs in any environment (you may use Linux, Mac OS X or Windows). 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. Therefore, we recommend you to make sure that your program compiles and properly works on "`dijkstra.ug.bcc.bilkent.edu.tr`" before submitting your assignment.