CS 201, Summer 2018
Homework Assignment 1
Due: 23:55, July 2, 2018

In this homework, you will implement a reservation system for a box office. The box office sells tickets for different movie theaters each of which has its unique theater id. Each movie theater could have different seating plan (that is, each movie theater could have different number of rows and different number of seats per row). In your implementation, you **MUST** use dynamically allocated arrays.

The reservation system will have the following functionalities; the details of these functionalities will be given below:

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

This homework will have two parts, whose requirements are explained below.

PART A:

To take the final exam, you MUST submit at least this part and MUST get at least half of its points.

This part includes the following three functionalities of the reservation system:

1. Add a movie theater
2. Remove a movie theater
3. Show the list of movie theaters

PART B:

In this part, you will complete the implementation of the entire reservation system (all of the 7 funcitonalities listed above).

The details of all of the functionalities are given below:

Add a movie theater: The reservation system will allow the user to add a new movie theater indicating

- The unique theater id
- The number of rows in the theater and the number of seats per row, and
- The name of a movie being shown in this theater (note that for simplicity, you may assume that only one movie is shown in a particular theater).

Since the theater ids are unique, the system should check whether or not the specified id already exists (i.e., whether or not it is an id of another theater). If it exists, the system should not allow the operation and display a warning message. At the time of adding a new movie theater, all seats in this theater are available.

Remove a movie theater: The reservation system will allow the user to remove an existing movie theater indicated by its theater id. If the theater does not exist (i.e., if there is no theater with the specified id), the system should display a warning message. Note that this operation will also cancel all reservations for the theater of interest.

Show the list of movie theaters: The reservation system will allow the user to see the id of all existing theaters together with their movie names and the number of their available seats.

<u>Show detailed information about a particular movie theater:</u> The reservation system will allow the user to enter a theater id and see its movie name as well as all of its seats and their availability. If the theater does not exist (i.e., if there is no theater with the specified id), the system should display a warning message. If the theater exists, the system should display the seats in the following format where "o" represents available seats and "x" represents occupied seats. Note that in this example, the theater has 6 rows and 4 seats in each of its rows.

```
        1       2       3       4
A       x       x       o       o
B       o       o       o       x
C       o       o       o       o
D       o       o       o       o
E       x       o       o       o
F       x       o       x       x
```

<u>Make a reservation:</u> The reservation system will allow the user to make a reservation in a movie theater for a particular number of audiences. For that, the user will specify the theater id and the number of audiences together with the seat numbers. Seat numbers are specified using two arrays where the first array contains characters corresponding to row numbers and the second array contains integers representing the seats of the corresponding rows. For example, for three audiences, both of the arrays should have a length of three, the first array can contain the characters A, C, E, and the second array can contain the integers 1, 3, 4, corresponding to the seats A1, C3, and E4.

Given the input, the system should first check whether or not this theater exists; if it does not, it should not allow the operation and should display a warning message. Then it should check whether or not there are enough available seats in the theater; if the number of available seats is less than the requested number of seats, it should not make any reservation and should display a warning message. Last, it should check whether or not the requested seats are available; if at least one of these seats is occupied, it should not make any reservation and should display a warning message.

If all these checks are OK, the system will make the reservation by creating a unique reservation code, and by reserving these seats under this code (how this is done depends on your implementation). The system should return this reservation code to the user. Later, the user can see or cancel his/her reservation using this reservation code. Note that the reservation codes should be unique within the whole system (among all theaters); thus, by taking a reservation code from the user, the system should be able to identify the theater as well as the seats under this reservation code. In this part, you may assume that the user enters the seat numbers correctly. For example, if there are 10 rows and 5 seats per row in the theater, the user will enter the rows between A and J, and the seats between 1 and 5.

<u>Cancel a reservation:</u> The reservation system will allow the user to cancel his/her reservation specifying the reservation code. In this case, the system will cancel all seats under this reservation code. If no reservation is found under this reservation code or there are no seats reserved under this code, the system should display a warning message. Otherwise, it will display the theater id, the movie name, and the seat numbers for which the reservation is canceled.

<u>Show a reservation:</u> The reservation system will allow the user to see his/her reservation specifying the reservation code. In this case, the system will display the theater id, its movie name and all seats under this reservation code. If no reservation is found under this reservation code or there are no seats reserved under this code, the system should display a warning message.

Below is the required public part of the `BoxOffice` class that you must write in this assignment. The name of the class <u>must be</u> `BoxOffice`, and <u>must include</u> these public member functions. We will use these functions to test your code. The interface for the class <u>must be written</u> in a file called `BoxOffice.h` and its implementation <u>must be written</u> in a file called `BoxOffice.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 BoxOffice {
public:
      BoxOffice();
      ~BoxOffice();
      void addTheater( int theaterId, string movieName,
                       int numRows, int numSeatsPerRow );
      void removeTheater( int theaterId );
      void showAllTheaters();
      void showTheater( int theaterId );
      int makeReservation( int theaterId, int numAudiences,
                           char *seatRow, int *seatCol );
      void cancelReservation( int resCode );
      void showReservation( int resCode );

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

NOTES ABOUT IMPLEMENTATION:

1. You ARE NOT ALLOWED to modify the given parts of the header file. You MUST use dynamic arrays in your implementation. You will get no points if you use fixed-sized arrays or data structures such as vector from the standard library. Similarly, you will get no points if you implement your homework using linked lists. However, if necessary, you may define additional data members and member functions. Moreover, you ARE NOT ALLOWED to use any global variables.

2. You may assume that the inputs for the functions (e.g., the seat numbers) are always correct so that you do not need to make any input checks. That is, the seat letter is always given as an uppercase letter that is in between ‘A’ and the letter corresponding to *numRows* (e.g.m if *numRows* is 7 then the uppercase letter will be in between ‘A’ to ‘G’) and the seat column number is an integer that is in between 1 and *numSeatsPerRow*.

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. Once again note that you can define additional public and private member functions and data members in this class. You can also define additional classes in your solution.

Here is an example test program that uses this class and the corresponding output. We will use a similar program to test your solution so make sure that the name of the class is `BoxOffice`, its interface is in the file called `BoxOffice.h`, and the required functions are defined as shown above.

Note that, for Part A, only `addTheater, removeTheater and showAllTheaters funtions will be tested.`

Example test code:

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

int main() {
    BoxOffice R;
    R.showAllTheaters();
    R.addTheater(10425, "Ted", 4, 3);
    R.addTheater(8234, "Cloud Atlas", 8, 3);
    R.addTheater(9176, "Hope Springs", 6, 2);
    R.showAllTheaters();
    R.addTheater(10425, "Skyfall", 8, 6);
    R.showAllTheaters();
    R.showTheater(10425);

    char rowRes1[4] = {'A', 'B', 'B', 'C'};
    int  colRes1[4] = {3, 3, 1, 1};
    int  code1      = R.makeReservation(10425, 4, rowRes1, colRes1);
    if (code1 != -1)
        cout << "Your reservation code is " << code1 << endl;
    R.showTheater(10425);

    char rowRes2[2] = {'A', 'D'};
    int  colRes2[2] = {2, 3};
    int  code2      = R.makeReservation(10425, 2, rowRes2, colRes2);
    if (code2 != -1)
        cout << "Your reservation code is " << code2 << endl;
    R.showTheater(10425);

    char rowRes3[2] = {'B', 'A'};
    int  colRes3[2] = {2, 3};
    int  code3      = R.makeReservation(10425, 2, rowRes3, colRes3);
    if (code3 != -1)
        cout << "Your reservation code is " << code3 << endl;
    R.showTheater(10425);

    char rowRes4[7] = {'A', 'B', 'C', 'C', 'A', 'B', 'D'};
    int  colRes4[7] = {1, 2, 2, 3, 2, 1, 2};
    int  code4      = R.makeReservation(10425, 7, rowRes4, colRes4);
    if (code4 != -1)
        cout << "Your reservation code is " << code4 << endl;
    R.showTheater(10425);

    R.showAllTheaters();
    R.showReservation(code1);
    R.showReservation(100);
    R.cancelReservation(code2);
    R.showTheater(10425);
    R.showAllTheaters();
    R.cancelReservation(300);

    R.removeTheater(8234);
    R.showTheater(8234);
    R.showAllTheaters();
    R.removeTheater(674);
    R.showAllTheaters();

    R.removeTheater(10425);
    R.showReservation(code1);
    R.showAllTheaters();
    BoxOffice R2;
    R2.showAllTheaters();
    // Note that the theater id 9176 exists in the object R but there
    // does not exist any movie theater with this theater id in the object R2
    R2.addTheater(9176, "Skyfall", 3, 3);
    R2. showAllTheaters();

    return 0;
}
```

Output of the example test code:

```
No movie theaters in the system!
Theater 10425 (Ted) has been added
Theater 8234 (Cloud Atlas) has been added
Theater 9176 (Hope Springs) has been added

Theaters currently in the system:
Theater 10425 (Ted): 12 available seats
Theater 8234 (Cloud Atlas): 24 available seats
Theater 9176 (Hope Springs): 12 available seats

Theater 10425 (Ted) already exists

Theaters currently in the system:
Theater 10425 (Ted): 12 available seats
Theater 8234 (Cloud Atlas): 24 available seats
Theater 9176 (Hope Springs): 12 available seats

Theater 10425 (Ted) has 12 available seats
  1 2 3
A o o o
B o o o
C o o o
D o o o

Your reservation code is 1

Theater 10425 (Ted) has 8 available seats
  1 2 3
A o o x
B x o x
C x o o
D o o o

Your reservation code is 2

Theater 10425 (Ted) has 6 available seats
  1 2 3
A o x x
B x o x
C x o o
D o o x

Seat A3 has already been taken

Theater 10425 (Ted) has 6 available seats
  1 2 3
A o x x
B x o x
C x o o
D o o x

Theater 10425 (Ted) does not have enough space for 7 audiences
Theater 10425 (Ted) has 6 available seats
  1 2 3
A o x x
B x o x
C x o o
D o o x

Theaters currently in the system:
Theater 10425 (Ted): 6 available seats
Theater 8234 (Cloud Atlas): 24 available seats
Theater 9176 (Hope Springs): 12 available seats

Reservations under code 1 in Theater 10425 (Ted): A3 B1 B3 C1
No reservations are found under code 100
Reservation for seats A2 D3 is canceled in Theater 10425 (Ted)

Theater 10425 (Ted) has 8 available seats
  1 2 3
A o o x
B x o x
C x o o
D o o o

Theaters currently in the system:
Theater 10425 (Ted): 8 available seats
Theater 8234 (Cloud Atlas): 24 available seats
Theater 9176 (Hope Springs): 12 available seats

No reservations are found under code 300

Theater 8234 (Cloud Atlas) and all of its reservations are canceled
Theater 8234 does not exist

Theaters currently in the system:
Theater 10425 (Ted): 8 available seats
Theater 9176 (Hope Springs): 12 available seats

Theater 674 does not exist

Theaters currently in the system:
Theater 10425 (Ted): 8 available seats
Theater 9176 (Hope Springs): 12 available seats

Theater 10425 (Ted) and all of its reservations are canceled

No reservations are found under code 1

Theaters currently in the system:
Theater 9176 (Hope Springs): 12 available seats

No movie theaters in the system!
Theater 9176 (Skyfall) has been added

Theaters currently in the system:
Theater 9176 (Skyfall): 9 available seats
```

NOTES ABOUT SUBMISSION:

1. This assignment is due by 23:55 on Monday, July 2, 2018. You should upload your homework using the upload link on Moodle before the deadline. This upload page will be available between June 25 and July 2. 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.

2. 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 `BoxOffice` and your files' name MUST BE `BoxOffice.h` and `BoxOffice.cpp`. Note that you may write additional class(es) in your solution. You should email these two files (and any additional files if you wrote additional classes in your solution) as a single archive file (i.e., zip). The name of this zip file should conform the following name convention: Firstname-Lastname-StudentID.zip where X is your section number.

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

3. We also recommend you to write your own driver file to test each of your functions. However, you MUST NOT submit this test code (we will use our own test code). In other words, do not submit a file that contains a function called main.

4. You are free to write your programs in any environment (you may use either Linux or Windows). On the other hand, we will test your programs on "`dijkstra.cs.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.cs.bilkent.edu.tr`" before submitting your assignment.

5. This homework will be graded by your TA Mehmet Başaran (mehmet.basaran at cs bilkent edu tr). Thus, you may ask your homework related questions directly to him.