CS 201, Fall 2020

Homework Assignment 1

Due: 23:59, Nov 16, 2020

In this homework, you will implement a flight reservation system for an airline company, which operates multiple flights. Each flight has a unique flight number and is operated with an aircraft that may have a different seating plan. According to its type, an aircraft has a specified number of rows and number of seats in each row. (Although these numbers may change from one aircraft type to another, the number of seats is the same for every row in the same aircraft.) You will use dynamically allocated arrays for this purpose. This homework will be graded by **your TA Mert Duman (mert.duman at bilkent edu tr). Please direct your homework related questions to him.**

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

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

Add a flight: The reservation system will allow the user to add a new flight indicating its flight number, the number of rows in its aircraft, and the number of seats in its rows. Since the flight numbers should be unique within the system, you should check whether a flight with the same number exists. If it does, you should not allow the operation and display a warning message. You should also check if the number of rows in its aircraft, and the number of seats are positive. Otherwise, you should add the flight and display a message. At the time of adding a flight, you should allocate its seats all of which should be available for reservation. (Please see the code given below and its output for the format of the output.)

Cancel a flight: The reservation system will allow the users to cancel an existing flight indicating its flight number. If there exists no flight with the specified number, you should display a warning message. If the flight exists, you should delete it from the reservation system and cancel all of the reservations that were made for this flight. You should also display a message that the flight has been canceled. (Please see the code given below and its output for the format of the output.)

Show the list of flights: The reservation system will allow the user to see all of their flights within the reservation system. For each flight, you should output the flight number together with the number of its available seats. (Please see the code given below and its output for the format of the output.)

Show detailed information about a particular flight: The reservation system will allow the user to enter a flight number and see the availability of its seats. If there exists no flight with the

specified number, you should display a warning message. If the flight exists, you should display the seats in the following format where “o” represents an available seat and “x” represents an occupied seat. Note that in the example below, the aircraft has 5 rows and 6 seats in each of its rows.

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

Make a reservation: The reservation system will allow the user to make a reservation in a flight for a given number of passengers. For that, the user will specify the flight number and the number of passengers together with the selected seat numbers. The seat numbers are specified using two arrays. The first array contains integers corresponding to the row numbers and the second array contains characters representing the seats of the corresponding rows. For example, if the user makes a reservation for three passengers and the selected seat numbers are 1A, 5C, and 4E, the first array will contain 1, 5, and 4, and the second array will contain A, C, and E. (Please see the code given below and its output for more examples.)

This function makes a reservation only if the flight with a specified number exists and all of the selected seats are available. If there exists no flight with the specified number, you should display a warning message. Likewise, if even one of the selected seats is not available, you should display a warning message and return the error code -1.

If the flight 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 seats under this code. The reservation code should be unique not only for the specified flight but also for the entire reservation system. Thus, by using only this reservation code, you should identify the flight as well as the seats reserved under this code.

In this function, you may assume that the input arrays for the seat numbers contain only the valid values. For example, if the aircraft has 10 rows and 5 seats in each row, you may assume that the first array will contain the row numbers between 1 and 10, and the second array will contain the seat characters between A and E.

Cancel a reservation: The reservation system will allow the user to cancel a reservation by specifying the reservation code. If there exists no reservation under this code, you should display a warning message. If the reservation exists, you should cancel all of the seats reserved under this code and display the flight number and the seat numbers for which the reservation is canceled. (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 see a reservation 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 flight number and the seat numbers selected for this reservation. (Please see the code given below and its output for the format of the output.)

Below is the required public part of the `ReservationSystem` class that you must write in this assignment. The name of the class **must** be **`ReservationSystem`** and must include these public member functions. The interface for the class must be written in a file called `ReservationSystem.h` and its implementation must be written in a file called `ReservationSystem.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. The interfaces of these classes should be put in their respective header files (i.e. `MyClass.h`), and their implementations should be put in their respective source files (i.e. `MyClass.cpp`). In addition to this, you are supposed to design and implement a `Flight` and a `Reservation` classes for which no details are provided. The interface for the `Flight` class must be written in a file called `Flight.h` and its implementation must be written in a file called `Flight.cpp`. Similarly, the interface for the `Reservation` class must be written in a file called `Reservation.h` and its implementation must be written in a file called `Reservation.cpp`. Finally, you will create a `makefile` that will compile all of your classes and generate an executable.

```
class ReservationSystem {
public:
     ReservationSystem();
     ~ReservationSystem();

     void addFlight( const int flightNo, const int rowNo, const int seatNo );
     void cancelFlight( const int flightNo );

     void showAllFlights();
     void showFlight( const int flightNo );
     int makeReservation( const int flightNo, const int numPassengers,
                          const int *seatRow, const char *seatCol );
     void cancelReservation( const int resCode );
     void showReservation( const int resCode );

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

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

EXAMPLE TEST CODE:

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

int main() {
     ReservationSystem R;

     R.showAllFlights();


     R.addFlight(104, 4, 3);
     R.addFlight(234, 8, 3);
     R.addFlight(76, 6, 2);
     R.showAllFlights();


     R.addFlight(104, 8, 6);
     R.showAllFlights();
     R.showFlight(104);


     int rowRes1[4] = {3, 3, 1, 1};
     char colRes1[4] = {'A', 'B', 'B', 'C'};
     int code1 = R.makeReservation(104, 4, rowRes1, colRes1);
     if (code1 != -1)
        cout << "Your reservation code is " << code1 << endl;

     R.showFlight(104);

     int rowRes2[2] = {2, 4};
     char colRes2[2] = {'A', 'C'};
     int code2 = R.makeReservation(104, 2, rowRes2, colRes2);
     if (code2 != -1)
            cout << "Your reservation code is " << code2 << endl;

     R.showFlight(104);


     int rowRes3[2] = {2, 3};
     char colRes3[2] = {'B', 'A'};
     int code3 = R.makeReservation(104, 2,
     rowRes3, colRes3);
     if (code3 != -1)
            cout << "Your reservation code is " <<code3<< endl;

     R.showFlight(104);


     int rowRes4[7] = {1, 2, 2, 3, 4, 3, 1};
     char colRes4[7] = {'A', 'B', 'C', 'C', 'A', 'B', 'B'};
     int code4 = R.makeReservation(104, 7, rowRes4, colRes4);
     if (code4 != -1)
      cout << "Your reservation code is " << code4 << endl;

     R.showFlight(104);
     R.showAllFlights();
```

```
    R.showReservation(100);
    R.showReservation(code1);


    R.cancelReservation(300);


    R.cancelReservation(code2);
    R.showFlight(104);
    R.showAllFlights();


    R.cancelFlight(234);
    R.showFlight(234);
    R.showAllFlights();


    R.cancelFlight(674);
    R.showAllFlights();

    R.cancelFlight(104);
    R.showReservation(code1);
    R.showAllFlights();

    return 0;
}
```

OUTPUT OF THE EXAMPLE TEST CODE:

```
No flights exist

Flight 104 has been added
Flight 234 has been added
Flight 76 has been added

Flights currently operated:
Flight 104 (12 available seats)
Flight 234 (24 available seats)
Flight 76 (12 available seats)

Flight 104 already exists

Flights currently operated:
Flight 104 (12 available seats)
Flight 234 (24 available seats)
Flight 76 (12 available seats)

Flight 104 has 12 available seats
A B C
1 o o o
2 o o o
3 o o o
4 o o o

Your reservation code is 1

Flight 104 has 8 available seats
A B C
1 o x x
2 o o o
3 x x o
4 o o o
```

```
Flight 104 has 6 available seats
A B C
1 o x x
2 x o o
3 x x o
4 o o x
3A is not available

Flight 104 has 6 available seats
A B C
1 o x x
2 x o o
3 x x o
4 o o x
1B 3B are not available

Flight 104 has 6 available seats

A B C
1 o x x
2 x o o
3 x x o
4 o o x

Flights currently operated:
Flight 104 (6 available seats)
Flight 234 (24 available seats)
Flight 76 (12 available seats)

No reservations under Code 100
Reservations under Code 1 in Flight 104: 1B 1C 3A 3B

No reservations are found under code 300

Reservation for the seats 2A 4C is canceled in Flight 104

Flight 104 has 8 available seats

A B C
1 o x x
2 o o o
3 x x o
4 o o o

Flights currently operated:
Flight 104 (8 available seats)
Flight 234 (24 available seats)
Flight 76 (12 available seats)

Flight 234 and all of its reservations are canceled
Flight 234 does not exist

Flights currently operated:
Flight 104 (8 available seats)
Flight 76 (12 available seats)
```

```
Flight 674 does not exist

Flights currently operated:
Flight 104 (8 available seats)
Flight 76 (12 available seats)

Flight 104 and all of its reservations are canceled

No reservations under Code 1
```

NOTES ABOUT IMPLEMENTATION:

1. You ARE NOT ALLOWED to modify the given parts of the header file. **You MUST use dynamically allocated arrays in your implementation.** You will get no points if you use fixed-sized arrays, linked-lists 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. 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. To detect memory leaks, you may want to use Valgrind which is available at http://valgrind.org.
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. For example, the seat row number is always given as an integer that is between 1 and *rowNo* and the seat letter is always given as an uppercase letter that is between 'A' and the letter corresponding to *seatNo* (e.g., if *seatNo* is 7 then the uppercase letter should be between 'A' to 'G').
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 23:59 on November 16, 2020. This homework will be graded by **your TA Mert Duman (mert.duman at bilkent edu tr). Please direct 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 "`ReservationSystem.h`" and "`ReservationSystem.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 file 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 the following files into a folder and zip the folder (in this zip file, there should not be any file containing the main function): "`ReservationSystem.h`", "`ReservationSystem.cpp`", "`Flight.h`", "`Flight.cpp`", "`Reservation.h`", "`Reservation.cpp`", "`makefile`" and additional files for classes that you implement. The name of this zip file should conform 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. **Then, before 23:59 on Nov 16th, you need to upload your zipped file to Moodle, which contains only your header and source codes (but not any file containing the main function).**

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

6. You are free to write your programs in any environment (you may use either Linux or Windows). However, 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 to 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.