Introduction

Data Structure Airlines (DSA) has finally decided to automate its flight scheduling. They want a new program and they are hiring you to write it for them. You'll have the rights to your new software and since you can sell it to other airlines, you need to make your program flexible.

DSA, or another airline, will provide you with a list of flights. Each flight has the following properties:

- A flight number (integer 0 - 999)
- A departure airport
- A departure date
- An arrival airport
- A seating list

Each seating list has the following properties:

- The number and range of first class seats
  (i.e. 5 seats, seats 1, 2, 3, 4, and 5).
- The number and range of business class seats
  (i.e. 10 seats, seats 6 - 15)
- The number and range of economy class seats
  (i.e. 20 seats, seats 16 - 35)

Note: A particular class can have an arbitrary range of seats (i.e. first class, 5 seats, seats 1, 5, 9, 11, 21).

Once the list of flights is given, you are ready to start scheduling passengers. The scheduling must handle several operations.

Schedule a passenger for a flight: Choose a flight number. Choose a class. If all the seats are full, give the option to put the passenger on a wait list. There should be a wait list for every class (first, business, and economy).

Cancel a passenger from a flight: Choose a passenger. Choose a flight. If there are passengers on the wait list, print the name of the next passenger and give the option to schedule that passenger for the available seat.

Passenger status: print the status of a passenger; Choose a passenger. If the passenger is scheduled for a flight, print the flight information. If the passenger is on a wait list, print the flight information and the passenger's position on the wait list.

Flight information: print the information for a flight; Print the departure airport, the departure date, the arrival airport, and a list of all the seats and the name of the passenger in each seat.

Design and Implementation

There are many ways to implement this program, many different data structures and algorithms can be used, and many ways the project description can be extended and be more realistic. It will be interesting to think about these issues during the project and as we explore more sophisticated data structures in the next few weeks. For now, however, try to keep it as simple as possible. Once you have a working prototype and by then we have covered more data structures, you can extend this program to satisfy the needs of assessment brief by incorporating the advance algorithms we have covered.

Pay particular attention to the design of your classes. Here is an outline of classes you might have:

- A class that represents the seating list. Each seat should have a number, a class, and field for the passenger name.
- A class that represents a flight. It should have the appropriate data structures for storing the departure and arrival information and it should include a seating list.
- A class that represents a passenger. A passenger may be scheduled for several different flights or may be waiting on several different wait lists. The appropriate data structures should be in place for maintaining the different information associated with each passenger.

The member functions defined for each of these class objects should support the above outlined scheduling operations. They should do so in such a way that the implementation details (the lists, queues, etc.) are hidden from the outside world. Start your programming task by determining which classes do what operations and then decide on the implementations within each class (some will be simple, others not).

Program Synopsis and Input Format

The program should have

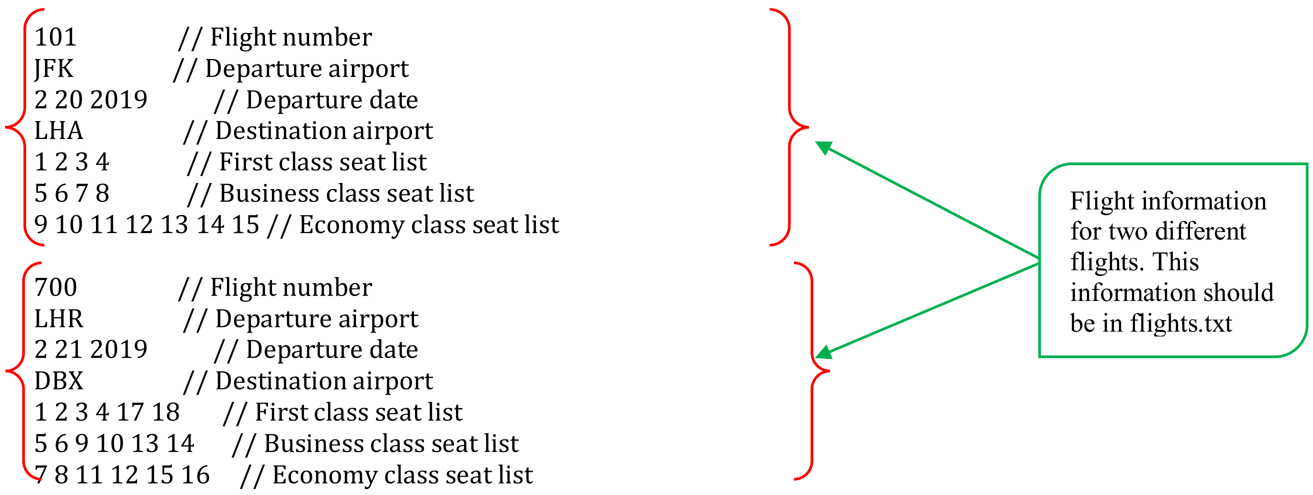
airline.exe (to run the program)
flights.txt (to be loaded in the program)

After reading the flights from flights.txt (or whatever name is provided), the program should interactively query the user for input.

The format of flights.txt is as follows. Note: you can assume for every flight, there are seven lines. Input files will contain several flights.

- Assume flight number can be any integer. But typical fights will be three digits.
- Assume departure and destination airports are single strings. But typical airports are denoted with a three-character abbreviated name.
- Assume departure date is three integers on one line. Implementation of the date can vary. Since the program does not perform any operation on the date, you could simply implement the date as string (it won’t affect your marks).

- Each seat list can contain any range of seats, i.e., seat numbers do not need to be sequential. For instance, first class could be seat number 1, 10, 33, 106. The size of the seat list can vary from flight to flight, i.e., flight 101 could have 4 first class seats while flight 700 could have 6.
- For simplicity, you can assume that input files do not have any comments.



```
101             // Flight number
JFK            // Departure airport
2 20 2019          // Departure date
LHA             // Destination airport
1 2 3 4          // First class seat list
5 6 7 8          // Business class seat list
9 10 11 12 13 14 15 // Economy class seat list

700             // Flight number
LHR             // Departure airport
2 21 2019          // Departure date
DBX             // Destination airport
1 2 3 4 17 18      // First class seat list
5 6 9 10 13 14      // Business class seat list
7 8 11 12 15 16     // Economy class seat list
.
.
.
```

Flight information for two different flights. This information should be in flights.txt

The interactive queries will have the following formats. First the user should type one of the characters `S', `C', `P', `F' or `Q' (to quit). Small letters should be allowed as well so the program isn't case sensitive. The user prompts should be clear.

- For 'S', to schedule a passenger, the program should request the passenger name, the flight number, and the class. If a seat is not available, cancel the schedule operation and give the user the option to put the passenger on the wait list. There should be a wait list for every class.
- For `C', to cancel a passenger, the program should request the passenger name and the flight number. Note: the flight number is necessary because a passenger may be scheduled for more than one flight. If there are passengers on the wait list for that particular class, the program should print out the name of the next passenger on the wait list and give the option to schedule the next passenger.
- For `P', passenger status, the program should request the passenger name. If the passenger is scheduled for a flight then the program should print the flight information. If the passenger is on a wait list then the program should print the

flight information and the passenger's position on the wait list (i.e. the next passenger to be scheduled is number 1).
- For `F', flight status, the program should request the flight number and print the departure date, the departure airport, the arrival airport, and a list all the seats and the name of the passenger in each seat.

Notes and assumptions

- The executable must be called airlines.exe.
- Passenger names are single strings.
- Airport names are single strings.

Implementation environment

You can use any environment to develop your code. Your final submission must be an executable along with the source code submitted in the zip file.