Aim: Utilization of structures and dynamic memory allocation in a realistic reservation system.

Task:
In this assignment you will create an airline reservation system. You will define a structure of Flights and Passengers. You **have to use structures and dynamic memory allocation** in this assignment. Flights have *number*, *departure city*, *arrival city*, *departure day*, *departure time*, *arrival day* and *arrival time*. Assume that each city has only one airport and there is only one company. Passengers have *first name*, *last name*, *departure city*, *arrival city*, *departure day and flight number*.

Your program should have the following functionalities:

addFlight: Adds a new flight to the system
searchFlight: Look for a flight using various search info. The user should be able to search for a flight using flight number, departure city and arrival city.
printFlight: Prints out the flight info with the number of reservations.
printFlightwPassengers: Prints out the flight info with the number of reservations and the passengers' first and last names on this flight.
makeReservation: Makes a reservation for a flight. The flight that is suitable for the passenger will be reserved.
printReservation: Prints out the reservation info on the screen.

You will be reading flight information from std. input in the given order:

- Flight number
- Departure city
- Arrival city
- Departure day
- Departure time
- Arrival day
- Arrival time

You will be reading passenger information from std. input in the given order:

- First name
- Last name
- Departure city
- Arrival city
- Departure day

The number of flights and number of passengers will be provided at run-time, so you HAVE TO make memory allocations at run-time, using malloc function. The number of characters in a city, first name and last name will not exceed 15 characters (so you may use char arrays for these fields, if you like).

After getting the data, your program must handle the following queries:

1- Flight queries: Your program should be able to search a flight using flight number, departure city and arrival city (***search flightNumber*** flightNumber, ***search departure*** departure city, ***search arrival*** arrival city).
2- Print queries (***print*** flightNumber, ***printwP*** flightNumber, ***printR*** firstName lastName).

Check-out the input and output format before proceeding further, to understand the problem better:

Input Format:

<number of flights:N>
<f1_id>[SPACE]**<f1_dep.city>**[SPACE]**< f1_ar.city>**[SPACE]**<f1_dep_day>**[SPACE]**<f1_dep_time>** [SPACE] **<f1_ar_day>**[SPACE]<**f2_ar_time**>
<f2_id>[SPACE]**<f2_dep.city>**[SPACE]**< f2_ar.city>**[SPACE]**<f2_dep_day>**[SPACE]**<f2_dep_time>** [SPACE] **<f2_ar_day>**[SPACE]<**f2_ar_time**>
…
<fN_id>[SPACE]**<fN_dep.city>**[SPACE]**< fN_ar.city>**[SPACE]**<fN_dep_day>**[SPACE]**<fN_dep_time>** [SPACE] **<fN_ar_day>**[SPACE]<**fN_ar_time**>
<number of passengers:M>
<p1_firstName>[SPACE]**<p1_lastName>**[SPACE]**<p1_dep.city>**[SPACE]**<p1_ar.city**>[SPACE]**<p1_dep_day>**
<p2_firstName>[SPACE]**<p2_lastName>**[SPACE]**<p2_dep.city>**[SPACE]**<p2_ar.city**>[SPACE]**<p2_dep_day>**
…
<pM_firstName>[SPACE]**<pM_lastName>**[SPACE]**<pM_dep.city>**[SPACE]**<pM_ar.city**>[SPACE]**<pM_dep_day>**
search[SPACE]**flightNumber**[SPACE]**<flightNumber>**
…
search[SPACE]**departure**[SPACE]**<departure_city>**
…
search[SPACE]**arrival**[SPACE]**<arrival_city>**
…
print[SPACE]**<flightNumber>**
…
printwP[SPACE]**<flightNumber>**
…
printR[SPACE]**<firstName>**[SPACE]**<lastName>**
…
END

<u>**Output Format:**</u>

<p1_firstName>[SPACE]**<p1_lastName>**[SPACE]**<p1_dep.city>**[SPACE]**<p1_ar.city**> No reservation

<f_id>[SPACE]**<f_dep.city>**[SPACE]**< f_ar.city>**[SPACE]**<f_dep_day>**[SPACE]**<f_dep_time>** [SPACE]
<f_ar_day>[SPACE]<**f_ar_time**>
...
<f_id>[SPACE]**<f_dep.city>**[SPACE]**< f_ar.city>**[SPACE]**<f_dep_day>**[SPACE]**<f_dep_time>** [SPACE]
<f_ar_day>[SPACE]<**f_ar_time**>
...
<f_id>[SPACE]**<f_dep.city>**[SPACE]**< f_ar.city>**[SPACE]**<f_dep_day>**[SPACE]**<f_dep_time>** [SPACE]
<f_ar_day>[SPACE]<**f_ar_time**>
...
<f_id>[SPACE]**<f_dep.city>**[SPACE]**< f_ar.city>**[SPACE]**<f_dep_day>**[SPACE]**<f_dep_time>** [SPACE]
<f_ar_day>[SPACE]<**f_ar_time**>[SPACE]**<NumberofPassengers>**
...
<f_id>[SPACE]**<f_dep.city>**[SPACE]**< f_ar.city>**[SPACE]**<f_dep_day>**[SPACE]**<f_dep_time>** [SPACE]
<f_ar_day>[SPACE]<**f_ar_time**>[SPACE]**<NumberofPassengers>**
<firstName1>[SPACE]**<lastName1>**
<firstName2>[SPACE]**<lastName2>**
...
...
<firstName1>[SPACE]**<lastName1>**[SPACE]**<f_id>**[SPACE]**<f_dep.city>**[SPACE]**<f_ar.city>**[SPACE]
<f_dep_day>[SPACE]**<f_dep_time>** [SPACE] **<f_ar_day>**[SPACE]<**f_ar_time**>
...

Detailed Specs:

Your program first reads the number of flights and then the flight information given as in the input and output format. Then, you read the number of passengers and their information. ***When you read the passenger information, your program makes a reservation for each passenger to possible flight according to the given flight information.*** If there is no available flight, your program print output that passenger info with "**No reservation**" keyword. If the reservation is made without problem, nothing is printed as output. After reading all flight and passenger information, the queries will start.
The query keywords:
search flightNumber <flightNumber>: Prints all the matching flight info. If no match is found, prints "**No flight with number** <flightNumber>".
search departure <departure_city>: Prints all the matching flight info. If no match is found, prints "**No departure city** <departure_city>".
search arrival <arrival_city>: Prints all the matching flight info. If no match is found, prints "**No arrival city** <arrival_city>".
print <flightNumber>: Prints the flight information and the number of passengers in that flight.
printwP <flightNumber>: Prints the flight information and the number of passengers in that flight and the passenger names.
printR <firstName> <lastName>: Prints the passenger first name, last name and his/her flight information.

Please pay attention to the sample output file in that, when the printed records for a query is finished, you need to print an empty line to separate different query results.

Your program will understand the end of queries with the keyword **END** at the end.

For example:

Sample Input:

```
15
1011 Ankara Istanbul 22/12/2015 08:30 22/12/2015 09:15
1013 Ankara Trabzon 19/12/2015 08:30 19/12/2015 09:45
1018 Istanbul Adana 21/12/2015 09:30 21/12/2015 11:15
1021 Istanbul Chicago 19/12/2015 08:30 19/12/2015 12:15
1027 Istanbul Berlin 19/12/2015 08:30 19/12/2015 13:30
1028 Ankara Istanbul 20/12/2015 10:30 20/12/2015 11:15
2032 Ankara Istanbul 21/12/2015 14:30 21/12/2015 15:15
3011 Bursa Izmir 19/12/2015 08:30 19/12/2015 09:15
3019 Istanbul Barcelona 21/12/2015 08:30 21/12/2015 14:15
3024 Izmir Istanbul 19/12/2015 11:30 19/12/2015 12:45
3056 Ankara Berlin 27/12/2015 08:30 27/12/2015 14:15
4048 Ankara Adana 25/12/2015 09:30 25/12/2015 10:30
4097 Adana Istanbul 19/12/2015 12:30 19/12/2015 14:15
5089 Ankara Istanbul 23/12/2015 08:30 23/12/2015 09:15
6013 Istanbul Barcelona 22/12/2015 10:30 22/12/2015 16:15
7
Mehmet Kaya Ankara Istanbul 23/12/2015
Ahmet Tas Ankara Istanbul 23/12/2015
Murat Kum Istanbul Barcelona 21/12/2015
Fatma Aslan Istanbul Barcelona 22/12/2015
Ayse Bahar Adana Istanbul 20/12/2015
Hasan Yaz Ankara Istanbul 21/12/2015
Cihan Kartal Istanbul Barcelona 22/12/2015
search flightNumber 4048
search flightNumber 6015
search departure Istanbul
search departure Ankara
search arrival Ankara
print 5089
printwP 6013
printR Murat Kum
END
```

<u>Sample Output:</u>

```
Ayse Bahar Adana Istanbul 20/12/2015 No reservation

4048 Ankara Adana 25/12/2015 09:30 25/12/2015 10:30


No flight with number 6015

1018 Istanbul Adana 21/12/2015 09:30 21/12/2015 11:15
1021 Istanbul Chicago 19/12/2015 08:30 19/12/2015 12:15
1027 Istanbul Berlin 19/12/2015 08:30 19/12/2015 13:30
3019 Istanbul Barcelona 21/12/2015 08:30 21/12/2015 14:15
6013 Istanbul Barcelona 22/12/2015 10:30 22/12/2015 16:15

1011 Ankara Istanbul 22/12/2015 08:30 22/12/2015 09:15
1013 Ankara Trabzon 19/12/2015 08:30 19/12/2015 09:45
1028 Ankara Istanbul 20/12/2015 10:30 20/12/2015 11:15
2032 Ankara Istanbul 21/12/2015 14:30 21/12/2015 15:15
3056 Ankara Berlin 27/12/2015 08:30 27/12/2015 14:15
4048 Ankara Adana 25/12/2015 09:30 25/12/2015 10:30
5089 Ankara Istanbul 23/12/2015 08:30 23/12/2015 09:15

No arrival city Ankara

5089 Ankara Istanbul 23/12/2015 08:30 23/12/2015 09:15 2

6013 Istanbul Barcelona 22/12/2015 10:30 22/12/2015 16:15 2
Fatma Aslan
Cihan Kartal

Murat Kum 3019 Istanbul Barcelona 21/12/2015 08:30 21/12/2015 14:15
```

Testing:

As usual, use *input redirection* mechanism of your operating system to test your programs. For example, if your executable is called as Lab6, redirect the input.txt file to standard input using **<** operator and redirect your outputs to a file using **>** operator such as:

```
> ./PA3<input.txt>output.txt
```

This kind of execution enables your programs to read inputs from a file without writing any file related functions (e.g. fopen(), fscanf() etc.). In other words, the getchar() or scanf() functions in your code reads data from the redirected files instead of the std. input in this way (e.g. keyboard).

Have fun ☺