Data Structures and Algorithms, Term Project
(100 marks)
Linked lists, Graphs and Searching

Build an airline reservation system which comprises of several operations of viewing flight information and making reservations.

Write C/C++ functions to be able to perform the following tasks. You are required to use a functional programming approach, thereby writing a function for each of the operations mentioned subsequently and calling them appropriately using a C/C++ program menu.

1. Show a list of all the cities serviced by airline in a tabular form. (5)
2. Show a list of flight departures for a city, sorted by the time of departure. (10)
3. Show a list of flight arrivals for a city, sorted by the time of arrival. (10)
4. Show a list of all the cities which can be reached from a particular city. (10)
5. Show the list of cities in the shortest path between any two cities. (10)
6. Find a route between any two cities. (10)
7. Make an airline reservation for a passenger between two cities. (10)
8. Print a passenger's reservation schedule. (5)
9. Delete a passenger's reservation. (10)
10. Print a list of passengers of a particular flight (in order of last name). (10)
11. Viva/ Quiz, etc. (10)

The airline reservation system consists of the following databases.

1. Flight Schedule Database
2. Passenger Schedule Database

Flight Schedule Database

The flight schedule database consists of a number of flights, each of type `FlightType`.

The structure of FlightType is as follows

```
struct FlightType {
    int FlightNo;              /* flight number */
    char *startCity;           /* departure city */
    int timeDepart;            /* departure time hhmm, e.g. 830, 1220, etc. */
    char *endCity;             /* arrival city */
    int timeArrival;           /* arrival time hhmm, e.g. 950, 1240, etc. */
    int noOfPassengers;        /* number of passengers on the flight */
    FlightType *nextDeparture; /* next departure node from this city */
    FlightType *nextArrival;   /* next arrival node to this city */
};
```

Write down a function names `MakeFlightNode()` to create and initialize a flight. Note: You will need to use *new* for creating space for the node and the strings of the city names. For creating space for strings do not forget to use brackets such as ``new char[strlen(startCity)+1]."

Once a flight node is created, it should be linked to

1. a list of departures for the departure city,
2. a list of arrivals for the arrival city, and
3. a list of flights based on the flight numbers.

The departure and arrival lists are maintained as a dynamic array of cities, named `cityList[]`, with each array entry containing a pointer to the head of the list for departures and arrivals.

```
#define MAXCITY 30                /* maximum number of cities to maintain */
struct CityListType {
    char *cityName;               /* name of the city */
    FlightType *nextDeparture;    /* first departure from this city */
    FlightType *nextArrival;      /* first arrival to this city */
};

    CityListType cityList[MAXCITY];  /* array of cities */
```

Initialize the `cityName` field of all entries to NULL to indicate the entry is not being used.

A hash function will be used to map a city to its index in the array.

You will need to use *new* to allocate space for the city name (if it is a new city) and link the departure into the departure list for the city. Repeat for the arrival city. Each linked list should be sorted by time so that the first departure/arrival of the day is first in the list.

The third structure, relevant to the flights database is `FlightNumberListType`, defined as follows

```
#define MAXFLIGHT 200          /* maximum number of flights to maintain */
struct FlightNumberListType {
    int FlightNo;              /* flight number */
    FlightType *flight;        /* flight node for this type */
};

 FlightNumberListType flightList[MAXFLIGHT];   /* array of flights */
```

`flightList` is a sorted array of elements based on the flight number. The list must be sorted to facilitate searching for flights based upon the flight number. Each element of the array also maintains a link to the flight node. You should maintain a count of the number of flights in the database.

Passenger Schedule Database

The Passenger Schedule Database will maintain a linked list of passenger reservations. Following data structures will be used for this purpose:

```
struct RouteType {
    int Day;                    /* day of travel: mmdd */
    int nHops;                  /* Number of hops (1 or 2) */
    int FlightNo1;              /* Flight number of first hop */
    int FlightNo2;              /* Flight number of second hop (if needed) */
};

#define ROUNDTRIP 0
#define ONEWAY 1
struct ReservationType {
    char *firstName;              /* first name of passenger */
    char *lastName;               /* last name of passenger */
    int tripType;                 /* ROUNDTRIP or ONEWAY */
    RouteType route1;             /* first route */
    RouteType route2;             /* second route (only if ROUNDTRIP) */
    ReservationType *nextReserve; /* next reservation in linked list */
};

ReservationType *reserveHead,*reserveTail; /* head and tail of the reservation list */
```

Once a reservation is read in, a reservation structure should be created for it using the member function `MakeReserveNode()`. This structure should be added to the linked list of reservations (in order of last name and then first name) where `pReserveHead` points to the first entry in the list.

Creating a Reservation

The last major portion of the program is to create a reservation using the function `MakeReservation()`. It needs to be passed the flight database so it can access the flight database functions while making a reservation. This function makes use of the function named `FindRoute()` for computing a route between two cities.

As part of booking a flight you should increment the count of passengers on each flight within the flight database.

Below are some useful functions to be used in the program. Prototypes for these functions are declared. You may need to create additional functions variables. Sample code for `CityDepartureList()` is shown

```
/*  MakeFlightNode → create a flight node from the given info */
FlightType *MakeFlightNode(int FlightNo, char *startCity,
                        int timeDepart, char *endCity, int timeArrival);

/*  ReadFlightData → make a flight database from scratch. Some examples are shown */
void ReadFlightData(){
```

```
        flightList[0].FlightNo = 111;
        flightList[1].FlightNo = 222;
        flightList[2].FlightNo = 333;
flightList[0].flight=MakeFlightNode(111,"Lahore", 700, "Karachi", 830);
flightList[1].flight=MakeFlightNode(222,"Lahore", 800, "Peshawar",845);
flightList[2].flight=MakeFlightNode(333,"Lahore", 900, "Quetta", 1045);

cityList[0].nextDeparture=flightList[0].flight;
flightList[0].flight->nextDeparture = flightList[1].flight;
flightList[1].flight->nextDeparture = flightList[2].flight;
flightList[2].flight->nextDeparture = NULL;

cityList[1].nextArrival = flightList[0].flight;
}

/* DisplayFlightInfo  → display a particular flight info in tabular form  */
void DisplayFlightInfo(FlightType * flight){
/* For example following data is displayed
   111    Lahore    700       Karachi    830
*/
}

/* DisplayFlightInfo  → display all flights info in tabular form  */
void DisplayAllFlightsData(){
/*  makes use of the function named DisplayFlightInfo().
*/
}

/*  FlightByNumber → return info about a flight given its flight number  */
FlightType * FlightByNumber(int FlightNo)

/*  DisplayAllCities → Show a list of all the cities serviced by airline in a tabular form  */
void DisplayAllCities();

/* DisplayCitiesFrom → Show a list of cities which can be reached from a particular city. */
void DisplayCitiesFrom(char *startCity){
/* This function will make use of depth first search (DFS) graph traversal algorithm.*/
}

/* DisplayShortestPath → Show the list of cities in the shorted path between any two cities. */
void DisplayShortestPath(char *startCity, char *endCity){
      /* This function will make use of a shortest path algorithm like Dijkstra's shortest path
      algorithm or Floyd -Warshal Algorithm. The shortest path means the route taking shortest time
      from source to destination.  */
}



/*   CityDepartureList → return a sorted list of departures for a city   */
FlightType *CityDepartureList(char *cityName)
{
    int i, iSave;
    iSave = i = Hash(cityName);         /* compute index in hash table  */
    while (cityList[i].cityName != NULL) {
        if (strcmp(cityName, cityList[i].cityName) == 0)
            return(cityList[i].nextDeparture);
```

```
        i = (i + 1)%MAXCITY;
        if (i == iSave)
            return(NULL);          /* have looped all the way around */
    }
    return(NULL);                  /* city not in hash table */
}


/*CityDepartureList → return a sorted list of departures for a city */
FlightType *CityDepartureList(char *cityName);

/*  CityArrivalList → return a sorted list of arrivals for a city */
FlightType * CityArrivalList(char *cityName);

/* DisplayDepartureList → Show flight departures for a city listed in order of time. */
void DisplayDepartureList(char *cityName){
    /* This function makes use of the function CityDepartureList(cityName) */
}

/*DisplayArrivalList → Show flight arrivals for a city listed in order of time. */
void DisplayArrivalList(char *cityName){
   /* This function makes use of the function CityArrivalList(cityName) */
}

/ * MakeReserveNode → create a reservation node from the given info */
ReservationType MakeReserveNode(char *firstName, char *lastName,
                 int tripType, RouteType route1, RouteType route2);

/* FindRoute → find a route from city1 to city2 and put it in the route structure, which is passed by
                  reference here.
                  Return 0 if a route is found and
                  Return -1 if no route could be found */
int findRoute(char *startCity, char *endCity, RouteType &route){
/* The algorithm for FindRoute() will be as follows:
    1. Get the list of departures for startCity and if there exist any direct flights to endCity then
       pick one of the flights and return it.
    2. If no direct flights exist then compare the departure list for startCity with the arrival list for
       endCity. Try to find a city in common to both lists to use for connections. As a constraint, the
       second flight must leave at least MINLAYOVER minutes (default of 30) after the first flight arrives.
       Return any valid connection.
    3. If no flight combinations exist then report that reservation system will not be able to handle the
       passenger's travel needs and do not make the reservation.
       */
 }

/* MakeReservation → make a reservation for a passenger by reading info from the user. */
void MakeReservation(Flights &flightdb){
/* In the function, you will need to gather the name of the passenger and the departure city along with the
type of travel (round trip or one-way) and the day(s) of travel. For each trip you will need to see if a flight
can be booked based on the flight database. Passengers prefer direct flights between cities, but will accept a
trip with one connecting flight. By default passengers will accept any trip you give them on the day of
travel (no preferences for the time of day).
The function makes use of the functions FindRoute() and MakeReserveNode(). */
}

/* printRoute → prints the route stored in variable route */
```

```
void printRoute(int ifExists, RouteType route);
```

```
/* PrintReservation → prints the reservation schedule, as pointed by pReserve */
PrintReservation(ReservationType *pReserve)

/* PrintAllReservations → prints all the reservations in the database */
PrintAllReservations()

/* PrintPassengers → print the list of all passengers on a particular flight */
void PrintPassengers(int FlightNo);

/* PrintSchedule → prompt the user for a passenger's (first and second) name and print that
passenger's schedule. */
void PrintSchedule(){
   /* This function makes use of the function PrintReservation() */
}

/* DeleteReserve → delete a reservation for a passenger by prompting for name */
void DeleteReserve()
```