# CSCI 310 – 02 (Fall 2019)

***Programming Foundations***

Programming Assignment 3

**DUE:** Sun, Oct 6, 11:59 PM (`turnin` time)

## High Planes Airline Company (HPAir)

*Programming Problem 6–11, pp. 222–223, Carrano and Henry 6/e.*
For additional background information on this problem, be sure to read section 5.3.1 (*Searching for an Airline Route*), pp. 172–177 and section 6.4 (*Using a Stack to Search a Flight Map*), pp. 210–215 of the textbook.

## Specifications

You are to complete the solution to the HPAir problem presented in the textbook. Three text files are provided as input to your program:

- `cityFile` Each line contains the name of a city that HPAir serves. The city names are provided in alphabetical order.
- `flightFile` Each line contains a pair of city names that represent the origin city and the destination city of one of HPAir's flights. Pairs of city names are separated by a comma.
- `requestFile` Each line contains a pair of city names that represent a request to fly from some origin city to some destination city.

Begin by implementing the Map ADT as the C++ class `FlightMap` based on the `MapInterface.h` file provided. Use the stack version of `isPath` (see p.210–213).

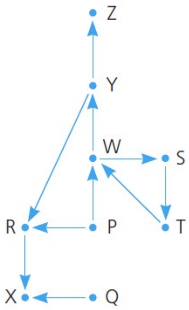

Figure 1: Flight map given in Figure 6–6 of Carrano and Henry, 7e (2017), p.208.

If there are $n$ cities numbered 1, 2, ..., $n$ , you can use $n$ chains of linked nodes to represent the flight map. You place a node on list $i$ for city $j$ if and only if there is a directed path from city $i$ to city $j$. A possible approach is to use an array (or `vector`) of `LinkedBag` objects to represent the flight map. Such a data structure is called an *adjacency list*.

## Input

All input comes from three text files as described in the Specifications section above. Your program should ask for the file names for each of this file in the order

1. `cityFile` filename
2. `flightFile` filename
3. `requestFile` filename

The user will provide the filenames. You may assume that

- The input data from the provided file names is correct.
- HPAir serves at most 20 cities.

For additional information on C++ File I/O, see **Appendix G** on pp. 763–772 of our textbook. Since file I/O can get challenging and progress in the program depends on file I/O, feel free to use the `split()` and the `removeSpecials()` functions. We will discuss these functions in class.

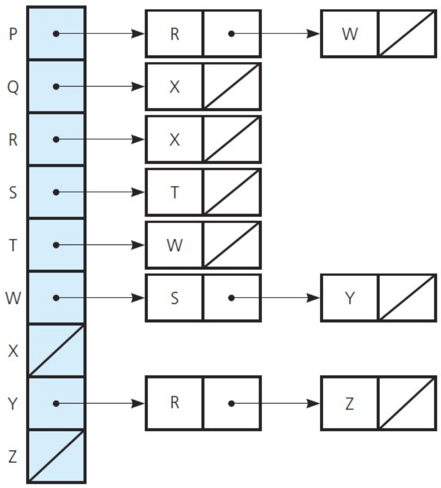

Figure 2: Adjacency list equivalent of flight map above; from Figure 6–14 of Carrano and Henry, 7e (2017), p.222.

### Sample Input

- `cityFile`:

```
Albuquerque
Chicago
San Diego
```

- `flightFile`:

```
Chicago,San Diego
Chicago,Albuquerque
Albuquerque,Chicago
```

- `requestFile`:

```
Albuquerque,San Diego
Albuquerque,Paris
San Diego,Chicago
```

### Output

For each line in the named `requestFile`, your program should first determine if HPAir services both the provided origin city and the indicated destination city. If both cities are served, your program next determines if HPAir can fly from the provided origin city to the indicated destination city. Each request will be numbered starting with `1` (see **Sample Output** below).

All output should be sent to standard output and must exactly follow the format in the sample below.

### Sample Output

For the sample input files given above, the program should produce the following output:

```
Request #1: Fly from Albuquerque to San Diego?
HPAir flies from Albuquerque to San Diego.

Request #2: Fly from Albuquerque to Paris?
Sorry. HPAir does not serve Paris.

Request #3: Fly from San Diego to Chicago?
Sorry. HPAir does not fly from San Diego to Chicago.
```

### Additional Requirements

You are required to design a FlightMap class that inherits from a provided MapInterface abstract class. This FlightMap class will use an *adjacency list* to represent a flight map. To simplify the implementation, your adjacency list will be an array (or vector) of either of the following built–in C++ container classes: forward_list or list. Note that although your driver program does not have to use all the member functions of your FlightMap class to solve this problem, you are still required to implement all the operations as specified in the provided MapInterface abstract class. You may choose to use the display() member function that displays the flight map information when debugging your code.

Use the built–in C++ stack class. Refer to the description of the *stack–based search algorithm* (see searchS on pp.210–213 and the isPath member function on pp. 214–215; the latter is provided below):

```
//  From Carrano and Henry (2016), pp. 214-215.

/** Tests whether a sequence of flights exists between two cities.
    Nonrecursive stack version.
    @pre originCity and destinationCity both exist in the flight map.
    @post Cities visited during the search are marked as visited
        in the flight map.
    @param originCity The origin city.
    @param destinationCity The destination city.
    @return True if a sequence of flights exists from originCity
        to destinationCity; otherwise returns false. */

    bool FlightMap::isPath( City originCity , City destinationCity )
    {
        Stack cityStack;
        unvisitAll(); // Clear marks on all cities

        // Push origin city onto cityStack and mark it as visited
        cityStack.push( originCity );
        markVisited( originCity );

        City topCity=cityStack.peek();
        while( !cityStack.isEmpty() && (topCity!=destinationCity) )
        {
            // The stack contains a directed path from the origin city at
            // the bottom of the stack to the city at the top of the stack

            // Find an unvisited city adjacent to city on top of the stack
            City nextCity=getNextCity( topCity , nextCity );

            if( nextCity==NO_CITY )
                cityStack.pop(); // No city found; backtrack
            else // Visit city
            {
                cityStack.push( nextCity );
                markVisited( nextCity );
            } // end if
            if( !cityStack.isEmpty() )
                topCity=cityStack.peek();

        } // end while

        return !cityStack.isEmpty();  // Path found if stack is non-empty

    } // end isPath
```

This is a fairly involved project, so plan on starting on it early. Make sure you understand all the various components and how they relate to each other. Come up with a good design you understand before writing any code, and be sure to have copies of the necessary files on your working directory before coding. It will also help to implement your program incrementally — you can leave most of your solution as *stubs* and implement them one by one, testing each piece as you go. Attacking a problem like this as one big piece from the very beginning is not a good solution.

### Deliverables

Your submission will consist of the following files, submitted using the Department of Computer Science's `turnin` facility:

- `FlightMap.h` – specification/header file for `FlightMap` class that is derived from the `MapInterface` abstract class
- `FlightMap.cpp` – implementation file for `FlightMap` class
- `HPAir.cpp` – driver code containing `main()` function

Note that the following code will be available on `turnin`:

- `MapInterface.h` abstract class
- `split.cpp`
- `removeSpecials.cpp`

We want you to develop good code documentation habits. Source code solutions submitted without any meaningful documentation will receive a total score of zero (0). You may refer to the *Google C++ Style Guide* section on source code comments as a guide.

Be sure to also review and adhere to the **Coding Standards** for this course.