CSCI 3110 Project 3

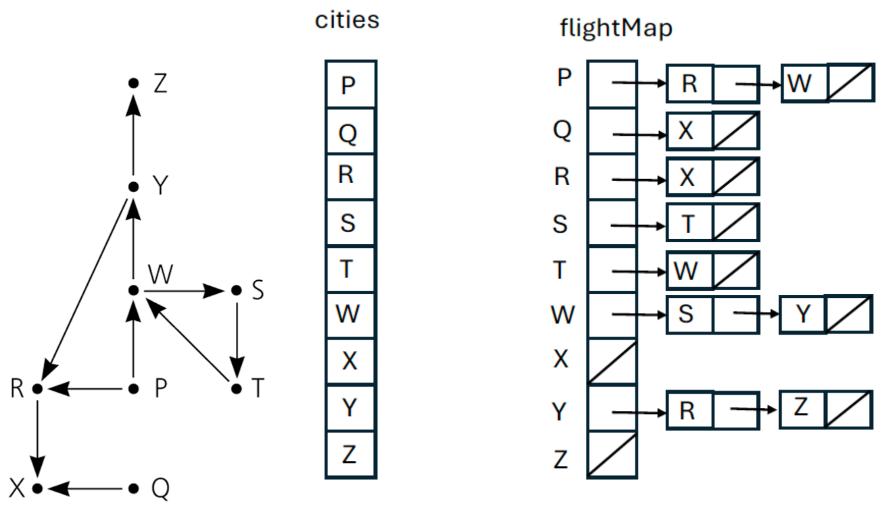
This project implements the preparation code for project 4. Iit is important that this project is implemented accurately.

The BlueSky airline company wants you to help them develop a program that generates flight itinerary for customer requests to fly from some origin city to some destination city. For example, when a customer want to flight from Nashville to San Francisco, a possible complete itinerary is given below:

Request is to fly from Nashville to San-Francisco.
The flight itinerary is:

Flight #	*From*	*To*	*Cost*
178	*Nashville*	*Albuquerque*	*$250*
224	*Albuquerque*	*San-Diego*	*$280*
703	*San-Diego*	*San-Francisco*	*$ 95*
		Total:	*$625*

But first, it is important to have an efficient way to store and maintain the database of all available flights of the company. We want to organize these flights in a manner where all the flights coming out of each city is easily searchable. This is called the *flight map*. The data structure we will use to build the flight map is called an ***Adjacency List***. Adjacency list is a data structure frequently used to represent a graph. The adjacency list consists of an array of head pointers, each pointing to a list of nodes, where each node contains the flight information. The ith array element corresponds to the ith city (***the origin city***) served by the company, and the jth node of that list correspond to the jth city that the origin city can fly to.



Your program should start by reading in a list of city names for which the company currently serves. The list of names is stored in a data file named “cities.dat”. Then, your program reads in a list of flights currently served by the company. The flight information is stored in the data file “flights.dat”.

cities.dat : the names of cities that BlueSky airline serves, one name per line, for example:

Albuquerque
Chicago
San-Diego
…

flights.dat : each flight record contains the flight number, a pair of city names (each pair represents the origin and destination city of the flight) plus a price indicating the airfare between these two cities, for example:

178	*Albuquerque*	*Chicago*	*250*
703	*Chicago*	*San-Diego*	*325*
550	*Nashville*	*San-Diego*	*180*

After reading and properly storing these information in the adjacency list, you program should print out the flight map in a well formatted table. *<u>The origin cities are sorted in alphabetical order, and the destination cities for each origin city is also sorted in alphabetical order:</u>*

```
          Origin           Destination      Flight   Price
   ==========================================

    From Atlanta to:      Chicago          1180     $89
                          New-York         320      $180
                          Seattle          1200     $210
    From Chicago to:      Atlanta          1181     $89
                          WashingtonDC     3400     $67
                          …
```

Program requirements:

1. Define the flight record as a struct type. Put the definition in the header file type.h
   - Overload at least the operators ==, <, assignment =, and << operators for this struct type.
   - Put the implementation of these operators, and any other methods you would like to implement, in type.cpp

2. Implement a FlightMap class, which has the following data and the following methods:
   - Data
     1. Number of cities served by the company
     2. list of cities served by the company
        - The <u>*STL vector*</u> is to be used for the list of cities served by the company**.**
     3. flight map implemented in the form of an adjacency list, e.g., array of lists.
        - The <u>*STL list*</u> needs to be used to implement each list
        - The array needs to be created dynamically. The actual size of the array is based on the number of cities served by the company. Therefore, the array needs to be defined as a pointer to the list of flight records.
        - It should be noted that this array is a parallel array to the list of cities array, e.g., data item 2 above
   - Methods:
     - constructor(s) and destructor
       - default constructor
       - copy constructor
         - make sure to use new operator to allocate space for the flight map before copying the lists
       - destructor – releases memory space dynamically allocated
     - operations
       - read cities (cities.dat)
         - This method takes one parameter: the input file stream opened for the data file: “cities.dat”
         - The input file stream should be opened in the main function and passed in to this method as parameter. Do not open this specific file in the method itself
       - read flight information and build the adjacency list (flights.dat)
         - This is the code that builds the adjacency list with information from the flights.dat file.
         - Dynamically allocate space for the flight map pointer before start reading the flight records and build the adjacency list
       - Overloaded << operator that displays the flight information as shown above.

Additional methods will be added to the FlightMap class in the next project to solve the overall task of flight itinerary generation
Make sure to follow the exact data structure and STL container requirements.