Assignment 4: File I/O and Containers: Constructing Connectivity Map among Different Cities in an Island

Description: In this assignment, you have to construct a graph showing the connectivity information among different cities in an island. There are 46 cities in the island. The attached file "**Terrain.obj**" contains the topological information of the island. The terrain model of the island (below) shows that its topology is undulated.

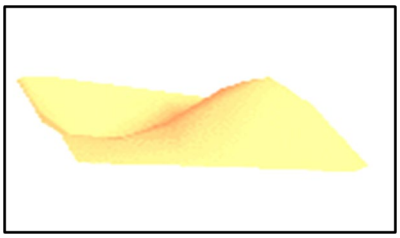

Figure 1: An island consisting of 46 cities.

In the attached file (**Terrain.obj**), lines starting with 'p' contains the 3-dimensional coordinates (i.e., Point3D) of centers of different cities. 'p' stands for position. The first line starting with 'p' refers to the city with index '0' and the corresponding center coordinate (x, y, and z). Similarly, the second line starting with 'p' corresponds to the information for city '1' and so on. You will find there are information for 46 cities. You need to store this information in a vector container of type Point3D (**cityCoordinates**).

Next, you'll also find some lines starting with 'c' and followed by some integers. Each line starting with 'c' contains the connectivity information for 3 cities. Let us take the very first line of the file starting with the character 'c':

```
c 13 28 41 ……(i)
```

This means cities 13, 28, and 41 are connected to each other.

Now, the next line shows more connectivity information:

```
c 13 41 42 ……(ii)
```

That means city 13 has connection with city 42 and vice versa, (41 repeated here);

Down the road you'll see that city 13 has also connection with cities 12 and 28.

```
c 12 28 13 ……(iii)
```

You'll also see some duplicate information about connectivity as shown below. Can you identify why?

```
c 12 13 42 ……(iv)
```

Figure 2 demonstrates the connectivity information of city 13 with its four neighboring cities: 12, 28, 41, and 42. In total, there is connectivity information for 46 cities. Like city 13, you need to find out the connectivity information for other cities.

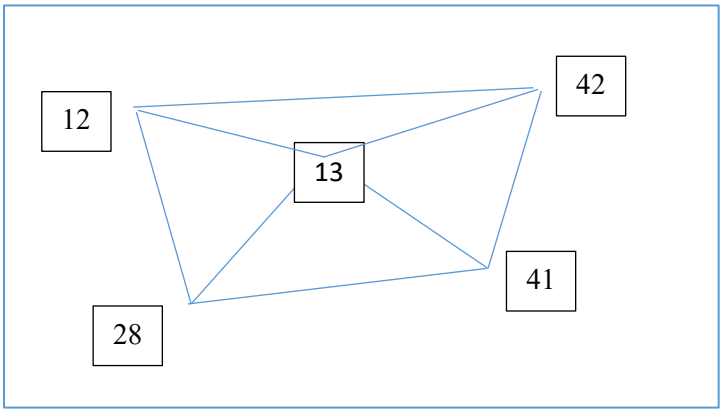

Figure 2: City 13 with its neighboring cities.

Construct a **Graph** class with the center coordinates of different cities and their neighbors. The **Graph** class will have the following member variables:

```
vector<Point3D> cityCoordinates; // See, we are recycling Point3D!
vector<int> cityIndices;
map<int, set<int>> Connectivity; //one key has a set of values
```

You need to parse the file "**Terrain.obj**" to get information about center points (**cityCoordinates**) of different cities and cityIndices (**cityIndices**). The member variable **cityCoordinates** is a vector that stores the center coordinates of cities in Point3D to be read from lines that start with **p**. Similarly, lines starting with '**c**' provides connectivity information among neighboring cities . Use **ifstream/fstream** to read data from **"Terrain.obj".** Next, use **istringstream (**See lecture note for Week 1**)** for extracting necessary information and converting to proper types.

You also need to construct a **map** named **Connectivity** that will store connectivity information of 46 cities. While constructing the map, you need to figure out the key and value pair. Let us again consider the connectivity information of city 13. Here, 13 is the key and the connected cities represented by a set are values, i.e., {12, 28, 41, 42}.

Table 1: Key as the index of a particular city and values as neighboring city indices

City (Key)	Connected Cities (Values)
13	{12, 28, 41, 42}

Note, **set** has been used to avoid duplicate neighboring city indices. You need to construct the connectivity map for 46 cities.

The Graph class can be as simple as shown below:

```
#ifndef GRAPH_H
#define GRAPH_H

#include <vector>
#include <set>
#include <map>
#include "Point3D.h"
        class Graph {
        public:

            // Extracts information from filename to initialize mVertices and vIndices
               bool loadCityInformation(const char *filename);

            // Generates the connectivity map among cities;
            //Calls Link function to generate connectivity information for all cities
               void Generate();

             //prints connectivity information of different cities
               void Print();

              // Displays connectivity information for a particular city "a" and finds
           //distance information of neighboring cities
               void showConnectivity(int a);

        private:
               std::vector<Point3D>cityCoordinates;
               std::vector<int> cityIndices;
               std::map<int, std::set<int>> Connectivity;
};
#endif
```

Figure 3: The **Graph** class.

The **Graph** class does not have **any** constructors. So, when you write '**Graph g'**, the default compiler generated constructor is called.

loadCityInformation function takes a filename as the function argument and extracts information to initialize member variables **cityCoordinates** and **cityIndices**.

Generate function generates the connectivity information among the cities; for each city, its neighbors are figured out. So, this function generates the map with key and corresponding sets of values.

Print function prints connectivity information of different cities in {key, value} pairs. Take a look at the output below (Figure 4) generated by the attached tester file **graph_tester.cpp**.

showConnectivity displays connectivity information of a particular city (key) that is passed as the function argument. It also displays distance information of the connected cities (values) from the key city.

Once you write all functions for the Graph class, the following output will be generated when you test your program with the attached tester file named **graph_tester.cpp**.

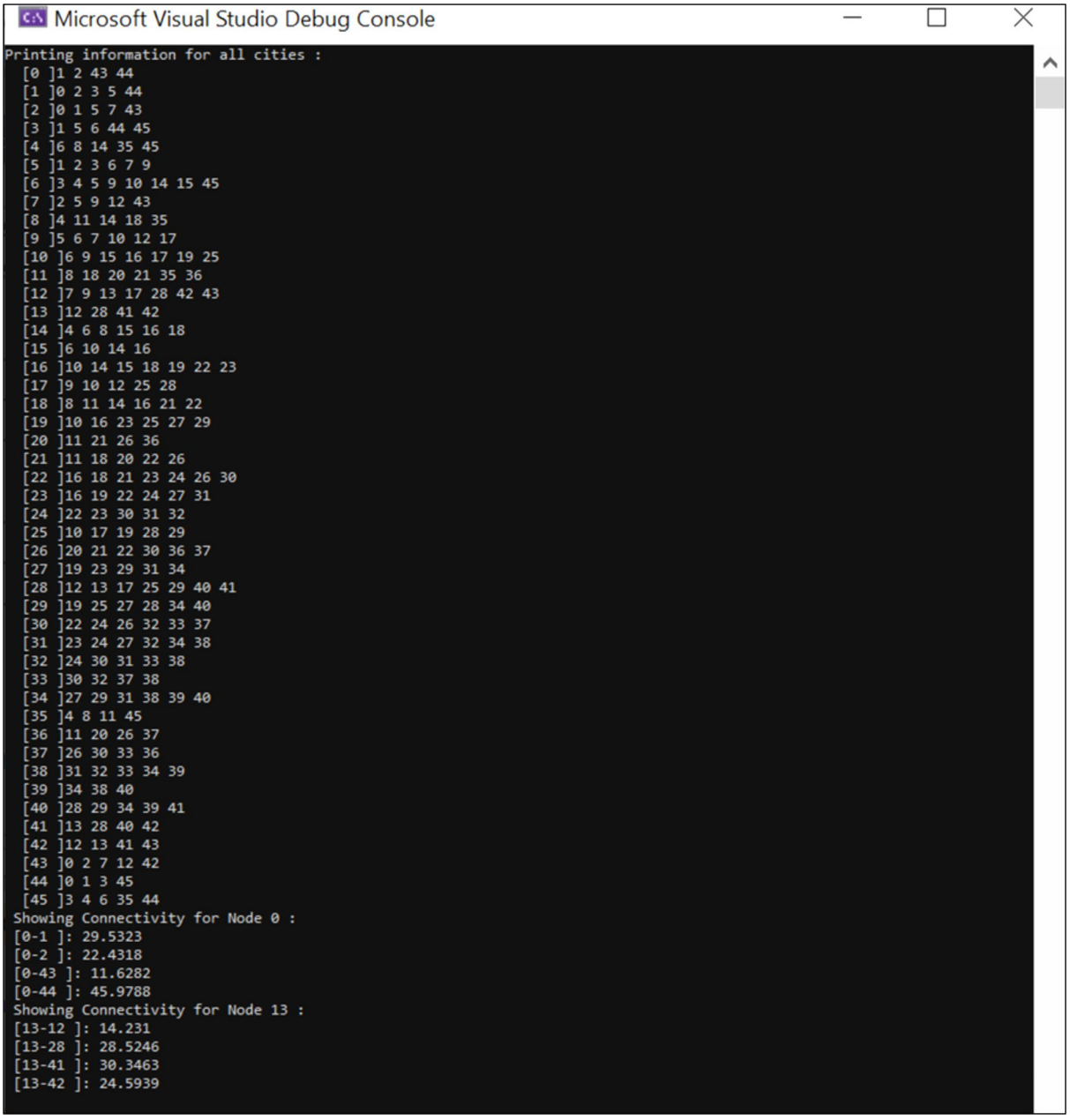

Figure 4: The program output.

Submission:

You need to create at two classes: **Point3D (**you already have it**)** and **Graph**.

Compile and execute your code with g++ or the Microsoft compiler.

Place your solution in a zipped file named with your last name followed by the first initial of your first name followed by 4 (ex: **YasminS4.zip**) and submit the solution via canvas.

Thus, your zip should contain the following:

- **Point3D.h**, **Point3D.cpp**, **Graph.h**, and **Graph.cpp**

Submission deadline is **Friday, May 10, 11:59 pm**.
This assignment weighs **15%** of the overall grade.