Exercise 2:

In this Exercise, we will implement a small application. The main idea is to represent the different places by object, class Place, and the way between places, by objects of class Way.

In graph theory, a directed graph is made of a set of vertices or nodes, V, and a set of edges $E = V\text{X}V$, that joins two vertices. Considering vertices (nodes) as places, an edge (v, w), $v, w \in V$, means there exists a way from place v to place w. Now, we can use existing algorithm (Dijsktra's Algorithm) to determine the shortest path from one place v to another place w.

A Map is made of several Places. To perform the implementation, we will include several libraries as below:

```
#include <iostream>

#include <string>

#include <fstream>

#include <math.h>

using namespace std;
```

To represent the name of places, we will use the string class provided by the standard C++ library. For now, we will use the following definition of the class Place

```
class Place {

public:

Place(void): m_name("unnamed") { }

Place(string name): m_name(name) { }

inline string name(void) const { return m_name; }

private:

string m_name;

};
```

1. Write this class in a file named gps.cpp.

2. Add another class Way to this file which represents the directed way from a place *p1* to another place *p2*. Once a path is created, its source and destination places are not supposed to be changed.

3. Add three member variables to class Way:

◦ dist – distance of the way in kilometers,

◦ speed – maximum allowed speed in kilometers per hour,

◦ traf – traffic factor reflecting the traffic state of the way (when the duration to traverse the way is computed, it has to be multiplied by this factor).

4. Create a class Map that contains the collection of places in an array and a collection of ways. At creation time, the map is considered as empty and therefore, the variables counting the number of used places and ways has to be initialized to 0.

5. Add function addPlace() to class Map which takes as parameter a string (the name of the place), creates and adds a place to the map and return it as reference.

6. Now add another function addWay() that takes two places as parameter, create and adds a way to the map and return it.

7. Now add function load() in order to load the map from a file (attached with name PlaceFile.map). {Complete the code below. Test it by adding a main() function that creates a map and load it.}

```
ifstream in;
in.open("PlaceFile.map");
while(!in.eof()) {
        string name;
        getline(in, name);
        if(name == "")
           break;
// create new place
}
while(!in.eof()) {
        int src, dst;
        float dist, speed, traf;
        in >> src >> dst >> dist >> speed >> traf;
        if(!in.fail()) {
             Place *srcp = _places[src], *dstp = _places[dst];
// create new way
}
}
in.close();
```

8. Modify the class Place to let it store an array of the ways which source is the place itself. Add a function addWay(*way*) to record a way inside its source place.

9. Modify the Way constructor to let it record itself inside the source place. Test the new versions of Place and Way.

10. Add to Map class functions to provide access to the list of places it contains.

11. Add a function to class Way, duration(), that computes the time spent to run through the way from the source place to the destination place.

12. Create a class Path that stores in an array the places that composes a path inside the map.

• add(*place*) – to add a place to the current path,

• print() – that prints to cout the list of ways composing the path.

• invert() – invert the order of places in the path.

13. Add a function to class Map, shortest(*initial place*, *final place*): this function will use the Dijsktra algorithm to compute and return the shortest path between the given initial place and final place. To perform the computation, it will use three arrays:

◦ dura[*v*] – duration from the initial place to the vertex *v*,

◦ prev[*v*] – previous place (as a pointer) in the path to place *v* (initialized to NULL),

◦ done[*v*] – initialized to false, set to true if the vertex *v* has been processed.

The function shortest() applies the Dijkstra algorithm as below:

```
prev[initial] ←NULL
dura[initial] ←0
while it exists a vertex not done do
select the vertex v which duration is minimal
done[v] ←true
for each way (v, w) leaving the vertex v do
alt ←dura[v] + duration(v, w)
if alt < dura[w] then
dura[w] ←alt
prev[w] ←v
```

To build the path, you have to create an object of class Path and traverse the shortest path backward using the prev[] array and starting from the final place.

14. Modify the main program to ask the user to enter a place index (places are displayed first) for initial and final place of the looked path and to display shortest path between both places and its duration.