The programming assignments for this week are based on standard library data structures as discussed in the first lecture of week 11 (11.0). The main program given below is meant to keep track of a fare table. You can add entries into the table (command 'a'), or you can query for the fare between some two cities ('g'), or you can ask for all the cities reachable from a certain city ('c'). The last should print the list of cities in sorted order. To exit input, type 'x'. If you have an entry in the fare table from mumbai to pune, it does not mean that the fare between pune to mumbai is the same or even that you can travel between pune to mumbai at all (you need a explicit fare entry). You are to write the required functions. As usual, the main program has already been entered.

```cpp
int main()
{
    faretabletype faretable; // create faretabletype using typedef
    while(true){
        char command; cin >> command;
        if(command == 'x') break;
        if(command == 'a'){
            string origin, destination; cin >> origin >> destination;
            double fare; cin >> fare;
            addfare(faretable,origin, destination, fare);
        }
        else if(command == 'g'){
            string origin, destination; cin >> origin >> destination;
            double fare;
            bool found = getfare(faretable,origin, destination,fare);
            if(found) cout << fare <<endl;
            else cout << "Not found."<<endl;
        },
    }
}```
else if(command == 'c'){
    string origin; cin >> origin;
    cout << connections(faretable,origin) << endl;
}
else cout <<"Illegal command."
} OTHERWISE:

Note that it is probably better to write the faretable as a class, and implement the operations as member functions. However, given that we are grading using the computer itself, we would not be able to tell whether you are implementing the internals using a map -- you could do it entirely without maps!

A MINOR POINT. Think about which arguments must be called by reference.

A VERY IMPORTANT POINT: before indexing into a map, you must first check if the index is defined (before writing A[x] you should check A.count(x) unless you know A[x] is defined). You have to do this for both the indices. Note that C++ allows you to write A[x] even if A[x] has not been defined and it creates A[x]. In fact this behaviour is required: if you write fare[x][y] = z; C++ will create fare[x], first initialising it to an empty map from strings to double, and then set the y index for that map to z.

The 'faretabletype' is already defined along with the 'main' function & other code that is automatically included in the program - which may not be all visible to you. You don't have to create this type again but this is specified below:

```cpp
typedef map<string,map<string,double> > faretabletype;
```

---

Sample input:
a mumbai pune 500
a mumbai nashik 600
a pune nashik 700
g mumbai nashik
g nashik pune
c mumbai
x
---

Output:
600
Not found.
nashik pune

**Sample Test Cases**

<table>
<thead>
<tr>
<th>Input</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>a earth moon 100</td>
<td>Not found.</td>
</tr>
<tr>
<td>a earth mars 500</td>
<td>mars sun venus</td>
</tr>
</tbody>
</table>
Test Case 2

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>a mumbai pune 500</td>
<td></td>
</tr>
<tr>
<td>a mumbai nashik 600</td>
<td></td>
</tr>
<tr>
<td>a pune nashik 700</td>
<td></td>
</tr>
<tr>
<td>g mumbai nashik</td>
<td></td>
</tr>
<tr>
<td>g nashik pune</td>
<td></td>
</tr>
<tr>
<td>c mumbai</td>
<td></td>
</tr>
</tbody>
</table>

The due date for submitting this assignment has passed.
As per our records you have not submitted this assignment.

Sample solutions (Provided by instructor)

```cpp
#include <iostream>
#include <map>
#include <cmath>
#define repeat(x) for(int _iterator_i = 0, _iterator_limit = x; _iterator_i < _iterator_limit; _iterator_i++)

using namespace std;
typedef map <string, map<string, double>> faretabletype;

void addfare(faretabletype &f, string &origin, string &destination, double fare)
{
    if(f.count(origin) == 0) return false;
    if(f[origin].count(destination) == 0) return false;
    fare = f[origin][destination];
    return true;
}

bool getfare(faretabletype &f, string &origin, string &destination, double &fare)
{
    string connections(faretabletype &f, string &origin)
    {
        string res = "";
        for(auto df : f[origin])
        {
            res = res + df.first + " ",
            res = res.substr(0, res.length()-1);
        }
        return res;
    }

    int main()
    {
        faretabletype faretable;
        while(true){
            char command; cin >> command;
            if(command == 'x') break;
            if(command == 'a')
            {
                string origin, destination; cin >> origin >> destination;
                double fare; cin >> fare;
                addfare(faretabele, origin, destination, fare);
            }
            else if(command == 'g')
            {
                string origin, destination; cin >> origin >> destination;
                double fare;
                bool found = getfare(faretable, origin, destination, fare);
                if(found) cout << fare << "\n";
                else cout << "Not found.\n";
            }
            else if(command == 'c')
            {
                string origin; cin >> origin;
                cout << connections(faretable, origin) << "\n";
            }
            else cout << "Illegal command." << "\n";
        }
    }
```