Practical 7

```
#include <iostream>
#include <cstring>
using namespace std;
const int MAX_CITIES = 100;
struct Flight {
    string source;
    string destination;
    int cost;
};
class Graph {
public:
    int adjList[MAX_CITIES][MAX_CITIES];
    int numCities;
    Graph() {
        numCities = 0;
        memset(adjList, 0, sizeof(adjList));
    }
    void addFlight(Flight flight) {
        int sourceIndex = getCityIndex(flight.source);
        int destIndex = getCityIndex(flight.destination);
        adjList[sourceIndex][destIndex] = flight.cost;
    }
    // Check if the graph is connected using BFS
    bool isConnected() {
        bool visited[MAX_CITIES];
        memset(visited, false, sizeof(visited));
        // Start BFS from any vertex
        int start = 0;
        visited[start] = true;
        for (int i = 0; i < numCities; i++) {
            if (adjList[start][i] > 0 && !visited[i]) {
                visited[i] = true;
            }
        }
        for (int i = 1; i < numCities; i++) {
            if (!visited[i]) {
                return false;
            }
        }
        return true;
    }
private:
```

```
    int getCityIndex(string city) {
        for (int i = 0; i < numCities; i++) {
            if (city == cities[i]) {
                return i;
            }
        }
        cities[numCities] = city;
        numCities++;
        return numCities - 1;
    }
    // Array to store names of cities
    string cities[MAX_CITIES];
};
int main()
{
    Graph g;
    int choice;
    Flight flight;
    do {
        cout << "\n1. Add a flight\n2. Check if graph is connected\n3. Exit\nEnter choice: ";
        cin >> choice;
        switch (choice) {
        case 1:
            cout << "\nEnter source city: ";
            cin >> flight.source;
            cout << "Enter destination city: ";
            cin >> flight.destination;
            cout << "Enter cost of flight: ";
            cin >> flight.cost;
            g.addFlight(flight);
            break;
        case 2:
            if (g.isConnected())
                cout << "\nGraph is connected";
            else
                cout << "\nGraph is not connected";
            break;
        case 3:
            cout << "\nExiting...";
            break;
        default:
            cout << "\nInvalid choice";
            break;
        }
```

```
    } while (choice != 3);
    return 0;
}
```