Code for the Inhabitants :

```cpp
#include <bits/stdc++.h>
using namespace std;
const int inf = 1e7;
const int n = 1e3;
vector<pair<int, int>> g[n];
// { node , cost}
void dijkstra(int source)
{
    vector<int> distance(n, inf);
    vector<bool> visited(n, 0);
    set<pair<int, int>> st;
    // {cost , node} => kept cost on first value to sort based on lowest cost
    st.insert({0, source});
    distance[source] = 0;
    while (!st.empty())
    {
        auto node = *st.begin();
        // will give the minimum weighted pair {cost , node}
        int parent_node = node.second;
        int parent_node_cost = node.first;
        st.erase(st.begin());
        if (visited[parent_node])
        {
            continue;
        }
        visited[parent_node] = 1;
        // Traverse to the child of v,for Relaxation
        for (auto child : g[parent_node])
        {
            int child_node = child.first;
            int edge_cost = child.second;

            // Relaxation
            if ((parent_node_cost + edge_cost) < distance[child_node])
            {
                distance[child_node] = (parent_node_cost + edge_cost);
                st.insert({distance[child_node], child_node});
            }
        }
    }
    cout << "Node\tDistance from " << source << endl;
    for (int i = 0; i < n; ++i)
    {
        if (distance[i] != inf)
        {
            cout << i << "\t" << distance[i] << endl;
        }
    }
}
int main()
{
    int node, edge;
    cin >> node >> edge;
    for (int i = 0; i < edge; i++)
    {
        int u, v, cost;
        cin >> u >> v >> cost;
        g[u].push_back({v, cost});
        g[v].push_back({u, cost});
        // u/v indexed node connected with v/u node with cost
    }
    dijkstra(0);

    return 0;
}

```

Minimum costs output for 0 number city's Inhabitants to travel into other cities using C++ :