```
#include<iostream>
#include<vector>        //add respective header files
#include<list>
#include<iterator>
using namespace std;
//helper Classes
class Edge{
  public:
    int DestinationVertexID;
    int weight;
Edge()
{
}
Edge(int DestVid, int w)
{
  DestinationVertexID = DestVid;
  weight = w;
}
//Some setter and getter methods
Int setEdgeValues(int DestVid, int w)
{
  DestinationVertexID = DestVid;
  weight = w;
}
 Void setWeight(int w)
{
    weight= w;
}
Int getDestinationVertexID( )
{
    Return DestinationVertexID;
}
 Void getWeight( )
{
     return weight;
}
};

Class Vertex{
     Public:
       Int state_id;
       string state_name;
       list<edge>edgelist;

Vertex()
{
State_id = 0;
State_name = “”;
}
 Vertex( int st_id, string st_name)
{
```

```
   State_id = st_id;
   state _name = st_name;

}
//some setters and getters here too

   Int getStateId()
{
    Return state_id;
}
  Int getStateName()
{
   Return state_name;
}
Void setStateId(int st_id)
{
  state_id = st_id;
}
Void setStateName(strinf st_name)
{
    state_name = st_name;
}
list<Edge> getEdgeList()
{
   return edgelist;

}
//  void addEdgeToEdgelist(int toVertexID, int weight)
  //  {
  //                  Edge e(toVertexID,weight);
  //                  edgeList.push_back(e);
  //                  cout<<"Edge between "<<state_id<<" and "<<toVertexID<<" added
Successfully"<<endl;
  //  }

  void printEdgeList() {
    cout << "[";
    for (auto it = edgeList.begin(); it != edgeList.end(); it++) {
      cout << it -> getDestinationVertexID() << "(" << it -> getWeight() << ") --> ";
    }
    cout << "]";
    cout << endl;
  }

  void updateVertexName(string sname) {
    state_name = sname;
    cout << "Vertex Name Updated Successfully";
  }
```

```
};
//Main Graph Class
Class Graph{
public:
  vector<vertex> vertices; //storing objects of type vertex into a dynamic array
//All graph operations will be created here

 Void AddVertex(Vertex newVertex)
{
  bool check= checkIfVertex ExistsByID(newVertex.getStateID());
if(check == true)
{
   cout<< ”The Vertex with this ID already exists”<< endl;
}
Else
 {
   vertices.push_back(newVertex);
   cout<<”New Vertex Added Successfully!” << endl;
 }

}
bool CheckIfVertexExistsByID(int vid)
{
bool flag = FALSE;
for( int i= 0; i <vertices.size; i++)
{
    if(vertices.at(i).getStateID() == vid)
  {
   return TRUE;
   }
}
return flag;
}
Vertex getVertexByID(int vid) {
   Vertex temp;
   for (int i = 0; i < vertices.size(); i++) {
     temp = vertices.at(i);
     if (temp.getStateID() == vid) {
       return temp;
     }
   }
   return temp;
  }

bool checkIfEdgeExistByID(int fromVertex, int toVertex) {

   Vertex v = getVertexByID(fromVertex);

   list < Edge > e;
```

```
  e = v.getEdgeList();
  bool flag = false;
  for (auto it = e.begin(); it != e.end(); it++) {
    if (it -> getDestinationVertexID() == toVertex) {
      flag = true;
      return flag;
      break;
    }

  }
  return flag;
}

void updateVertex(int oldVID, string vname) {
  bool check = checkIfVertexExistByID(oldVID);
  if (check == true) {
    for (int i = 0; i < vertices.size(); i++) {
      if (vertices.at(i).getStateID() == oldVID) {
        vertices.at(i).setStateName(vname);
        break;
      }
    }
    cout << "Vertex(State) Updated Successfully " << endl;
```

```
  }
}

void addEdgeByID(int fromVertex, int toVertex, int weight) {
  bool check1 = checkIfVertexExistByID(fromVertex);
  bool check2 = checkIfVertexExistByID(toVertex);

  bool check3 = checkIfEdgeExistByID(fromVertex, toVertex);
  if ((check1 && check2 == true)) {

    if (check3 == true) {
      cout << "Edge between " << getVertexByID(fromVertex).getStateName() << "(" << fromVertex <<
") and " << getVertexByID(toVertex).getStateName() << "(" << toVertex << ") Already Exist" << endl;
    } else {

      for (int i = 0; i < vertices.size(); i++) {

        if (vertices.at(i).getStateID() == fromVertex) {
          Edge e(toVertex, weight);
          //edgeList.push_back(e);
          //vertices.at(i).addEdgeToEdgelist(toVertex,weight);
          vertices.at(i).edgeList.push_back(e);
        } else if (vertices.at(i).getStateID() == toVertex) {
          Edge e(toVertex, weight);
```

```
        //edgeList.push_back(e);
        //vertices.at(i).addEdgeToEdgelist(fromVertex,weight);
        vertices.at(i).edgeList.push_back(e);
       }
      }

     cout << "Edge between " << fromVertex << " and " << toVertex << " added Successfully" << endl;
    }
  } else {
   cout << "Invalid Vertex ID entered.";
  }
}

void updateEdgeByID(int fromVertex, int toVertex, int newWeight) {
  bool check = checkIfEdgeExistByID(fromVertex, toVertex);
  if (check == true) {
   for (int i = 0; i < vertices.size(); i++) {

    if (vertices.at(i).getStateID() == fromVertex) {
      for (auto it = vertices.at(i).edgeList.begin(); it != vertices.at(i).edgeList.end(); it++) {
        if (it -> getDestinationVertexID() == toVertex) {
          it -> setWeight(newWeight);
          break;
```

```
        }


      }


    } else if (vertices.at(i).getStateID() == toVertex) {
      for (auto it = vertices.at(i).edgeList.begin(); it != vertices.at(i).edgeList.end(); it++) {
        if (it -> getDestinationVertexID() == fromVertex) {
          it -> setWeight(newWeight);
          break;
        }
      }
    }
  }
  cout << "Edge Weight Updated Successfully " << endl;
 } else {
  cout << "Edge between " << getVertexByID(fromVertex).getStateName() << "(" << fromVertex << ")
and " << getVertexByID(toVertex).getStateName() << "(" << toVertex << ") DOES NOT Exist" << endl;
 }
}


void deleteEdgeByID(int fromVertex, int toVertex) {
 bool check = checkIfEdgeExistByID(fromVertex, toVertex);
 if (check == true) {
  for (int i = 0; i < vertices.size(); i++) {
```

```
    if (vertices.at(i).getStateID() == fromVertex) {
      for (auto it = vertices.at(i).edgeList.begin(); it != vertices.at(i).edgeList.end(); it++) {
        if (it -> getDestinationVertexID() == toVertex) {
          vertices.at(i).edgeList.erase(it);
          //cout<<"First erase"<<endl;
          break;
        }
      }
    }

    if (vertices.at(i).getStateID() == toVertex) {

      for (auto it = vertices.at(i).edgeList.begin(); it != vertices.at(i).edgeList.end(); it++) {
        if (it -> getDestinationVertexID() == fromVertex) {
          vertices.at(i).edgeList.erase(it);
          //cout<<"second erase"<<endl;
          break;
        }
      }
    }
  }
  cout << "Edge Between " << fromVertex << " and " << toVertex << " Deleted Successfully." << endl;
}
```

```
}

void deleteVertexByID(int vid) {
  int vIndex = 0;
  for (int i = 0; i < vertices.size(); i++) {
    if (vertices.at(i).getStateID() == vid) {
      vIndex = i;
    }
  }
  for (int i = 0; i < vertices.size(); i++) {
    for (auto it = vertices.at(i).edgeList.begin(); it != vertices.at(i).edgeList.end(); it++) {
      if (it -> getDestinationVertexID() == vid) {
        vertices.at(i).edgeList.erase(it);
        break;
      }
    }

  }
  vertices.erase(vertices.begin() + vIndex);
  cout << "Vertex Deleted Successfully" << endl;
}

void getAllNeigborsByID(int vid) {
```

```
    cout << getVertexByID(vid).getStateName() << " (" << getVertexByID(vid).getStateID() << ") --> ";
    for (int i = 0; i < vertices.size(); i++) {
      if (vertices.at(i).getStateID() == vid) {
        cout << "[";
        for (auto it = vertices.at(i).edgeList.begin(); it != vertices.at(i).edgeList.end(); it++) {
          cout << it -> getDestinationVertexID() << "(" << it -> getWeight() << ") --> ";
        }
        cout << "]";

      }
    }

  }


//print Fuction
void printGraph() {
    for (int i = 0; i < vertices.size(); i++) {
      Vertex temp;
      temp = vertices.at(i);
      cout << temp.getStateName() << " (" << temp.getStateID() << ") --> ";// KHI (1) -> []// edgelist
      temp.printEdgeList();
    }
  }
};



int main(){
Graph g;
  string sname;
  int st_id1, st_id2, w;
  int option;
```

```
  bool check;



//driver code
do {
    cout << "What operation do you want to perform? " <<
      " Select Option number. Enter 0 to exit." << endl;
    cout << "1. Add Vertex" << endl;
    cout << "2. Update Vertex" << endl;
    cout << "3. Delete Vertex" << endl;
    cout << "4. Add Edge" << endl;
    cout << "5. Update Edge" << endl;
    cout << "6. Delete Edge" << endl;
    cout << "7. Check if 2 Vertices are Neigbors" << endl;
    cout << "8. Print All Neigbors of a Vertex" << endl;
    cout << "9. Print Graph" << endl;
    cout << "10. Clear Screen" << endl;
    cout << "0. Exit Program" << endl;

    cin >> option;
    Vertex v1;

    switch (option) {
    case 0:

      break;

    case 1:
      cout << "Add Vertex Operation -" << endl;
      cout << "Enter State ID :";
      cin >> st_id1;
      cout << "Enter State Name :";
      cin >> sname;
      v1.setID(st_id1);
      v1.setStateName(st_name);
      g.addVertex(v1);

      break;

    case 2:
      cout << "Update Vertex Operation -" << endl;
      cout << "Enter State ID of Vertex(State) to update :";
      cin >> st_id1;
      cout << "Enter State Name :";
      cin >> sT_name;
      g.updateVertex(st_id1, st_name);

      break;

    case 3:
```

```
  cout << "Delete Vertex Operation -" << endl;
  cout << "Enter ID of Vertex(State) to Delete : ";
  cin >> st_id1;
  g.deleteVertexByID(st_id1);

  break;

case 4:
  cout << "Add Edge Operation -" << endl;
  cout << "Enter ID of Source Vertex(State): ";
  cin >> st_id1;
  cout << "Enter ID of Destination Vertex(State): ";
  cin >> st_id2;
  cout << "Enter Weight of Edge: ";
  cin >> w;
  g.addEdgeByID(st_id1,st_id2, w);

  break;

case 5:
  cout << "Update Edge Operation -" << endl;
  cout << "Enter ID of Source Vertex(State): ";
  cin >> st_id1;
  cout << "Enter ID of Destination Vertex(State): ";
  cin >> st_id2;
  cout << "Enter UPDATED Weight of Edge: ";
  cin >> w;
  g.updateEdgeByID(st_id1, st_id2, w);

  break;

case 6:
  cout << "Delete Edge Operation -" << endl;
  cout << "Enter ID of Source Vertex(State): ";
  cin >> st_id1;
  cout << "Enter ID of Destination Vertex(State): ";
  cin >> st_id2;
  g.deleteEdgeByID(st_id1, st_id2);

  break;

case 7:
  cout << "Check if 2 Vertices are Neigbors -" << endl;
  cout << "Enter ID of Source Vertex(State): ";
  cin >> st_id1;
  cout << "Enter ID of Destination Vertex(State): ";
  cin >> st_id2;
  check = g.checkIfEdgeExistByID(st_id1, st_id2);
  if (check == true) {
    cout << "Vertices are Neigbors (Edge exist)";
  } else {
```

```
      cout << "Vertices are NOT Neigbors (Edge does NOT exist)";
     }

     break;

   case 8:
     cout << "Print All Neigbors of a Vertex -" << endl;
     cout << "Enter ID of Vertex(State) to fetch all Neigbors : ";
     cin >> st_id1;
     g.getAllNeigborsByID(st_id1);

     break;

   case 9:
     cout << "Print Graph Operation -" << endl;
     g.printGraph();

     break;

   default:
     cout << "Enter Proper Option number " << endl;
   }
   cout << endl;

 } while (option != 0);



return 0;
}

//Adding  A vertex into Graph(Psedocode)

Function AddVertex(new vertex)
{
     1. Check = CheckIfVertexExistsByID(vid);
     2.  if check == TRUE;
           2.1 print("Vertex Exists");
     3. Else
          3.1 vertices.push_back(new vertex);
          3.2 print("vertex added");
}

Function CheckIfVertexExistsByID(vid)
{
     1. Loop: (int i=0  to vertices.size )
          1.1 if(vertices.at(i) == vid)
          1.1.1 return TRUE;
     End loop;
     2. Return FALSE;
}
```

```
Int main()
{
Graph g; vertex v1;
cout<<“Add vertex Operation” <<endl;
cout<<”Enter the state id of vertex” << endl;
cin>> st_id1;      // 1
cout<<”Enter the state name of vertex”<<endl;
cin>> st_name; // KHI
v1.setStateID(st_id1);
v1.setStateName(st_name);
d.addVertex(v1);









}
```