

```

//
// main.cpp
// Lab14
//
// Created by Sergey on 21/05/2019.
// Copyright © 2019 Sergey. All rights reserved.
//
#include
#include // For INT_MAX
#include // Used for labels

using namespace std;

class WGraph
{
public:
static const int
MAX_GRAPH_SIZE = 10, // Default number of vertices
INFINITE_EDGE_WT = INT_MAX; // "Weight" of a missing edge

class Vertex
{
public:
void setLabel( const string& newLabel ) { label=newLabel; }
string getLabel( ) const { return label; }
private:
string label; // Vertex label
};

//-----

WGraph ( int maxNumber = MAX_GRAPH_SIZE ); // Constructors
~WGraph (); // Destructor

// Graph manipulation operations
void insertVertex ( const Vertex& newVertex ); // Insert vertex
void insertEdge ( const string& v1, const string& v2, int wt ); // Insert edge
bool retrieveVertex ( const string& v, Vertex& vData ) const; // Get vertex
bool getEdgeWeight ( const string& v1, const string& v2, int& wt ) const; // Get edge wt.
void removeVertex ( const string& v ); // Remove vertex
void removeEdge ( const string& v1, const string& v2 ); // Remove edge
void clear (); // Clear graph

// Graph status operations
bool isEmpty () const; // Graph is empty
bool isFull () const; // Graph is full

// Output the graph structure -- used in testing/debugging
void show() const;

private:
// Facilitator functions
int getIndex ( const string& v ) const; // Converts vertex label to an
// adjacency matrix index

```

```

int getEdge ( int row, int col ) const; // Get edge weight using
// adjacency matrix indices
void setEdge ( int row, int col, int wt); // Set edge weight using
// adjacency matrix indices
// Data members
int maxSize, // Maximum number of vertices in the graph
size; // Actual number of vertices in the graph
Vertex *vertexList; // Vertex list
int *adjMatrix; // Adjacency matrix (матрица смежности)
};

```

```

WGraph::WGraph(int maxNumber) {
maxSize = maxNumber;
size = 0;
vertexList = new Vertex[maxNumber];
adjMatrix = new int[maxNumber * maxNumber];
for(int i = 0; i < maxSize; i++)
for(int j = 0; j < maxSize; j++)
setEdge(i, j, INFINITE_EDGE_WT);
}

```

```

WGraph::~~WGraph() {
delete[] vertexList;
delete[] adjMatrix;
}

```

```

void WGraph::insertVertex(const Vertex& newVertex) {
//Check to update the vertex
for(int i = 0; i < size; i++) {
//If the vertex exists
if(vertexList[i].getLabel() == newVertex.getLabel()) {
//Update the edge
vertexList[i] = newVertex;
for(int j = 0; j < size; j++)
setEdge(i, j, INFINITE_EDGE_WT);
//Exit when found
return;
}
}
}

```

```

if(isFull()) {
cout << "### Error: The Graph is full";
return;
}
//Add the vertex to the graph
vertexList[size] = newVertex;
for(int i = 0; i <= size; i++)
setEdge(size, i, INFINITE_EDGE_WT);
size++;
}

```

```

void WGraph::insertEdge(const string& v1, const string& v2, int wt) {
int index1 = getIndex(v1), index2 = getIndex(v2);
if( index1 == -1 || index2 == -1 ) //Vertices are not in the graph
cout << "### Error: Vertex not found";
}

```

```

else
setEdge(index1, index2, wt);
}

void WGraph::clear() {
size = 0;
}

bool WGraph::isEmpty() const {
return (size == 0);
}

bool WGraph::isFull() const {
return (size == maxSize);
}

int WGraph::getIndex(const string& v) const {
for(int i = 0; i < size; i++)
if(vertexList[i].getLabel() == v) // Index found
return i;
return -1; // Index not found
}

int WGraph::getEdge(int row, int col) const {
return adjMatrix[(row * maxSize) + col];
}

void WGraph::setEdge(int row, int col, int wt) {
adjMatrix[(row * maxSize) + col] = wt;
adjMatrix[(col * maxSize) + row] = wt;
}

void WGraph:: show() const
// Outputs a graph's vertex list and adjacency matrix.
{
if ( size == 0 ) {
cout << "Empty graph" << endl;
} else {
cout << endl << "Vertex list : " << endl;
for ( int row = 0 ; row < size ; row++ )
{
cout << row << '\t' << vertexList[row].getLabel();
cout << endl;
}

cout << endl << "Edge matrix : " << endl << '\t';
for ( int col = 0 ; col < size ; col++ )
cout << col << '\t';
cout << endl;
for ( int row = 0 ; row < size ; row++ )
{
cout << row << '\t';
for ( int col = 0 ; col < size ; col++ )
{
int wt = getEdge(row,col);

```

```

if ( wt == INFINITE_EDGE_WT )
cout << "- \t";
else
cout << wt << '\t';
}
cout << endl;
}
}
}

```

```

void print_help();

```

```

int main()
{
WGraph testGraph(8); // Test graph
WGraph::Vertex testVertex; // Vertex
string v1, v2; // Vertex labels
char cmd; // Input command
int wt = 0; // Edge weight

```

```

print_help();

```

```

do
{
testGraph.show(); // Output graph

```

```

cout << endl << "Command (H for help): "; // Read command
cin >> cmd;
cmd = toupper( cmd ); // Normalize to upper case
if ( cmd == '+' || cmd == '?' || cmd == '-' )
cin >> v1;
else if ( cmd == '#' || cmd == '!' )
cin >> v1 >> v2;
else if ( cmd == '=' )
cin >> v1 >> v2 >> wt;

```

```

switch ( cmd )
{
case 'H' :
print_help();
break;

```

```

case '+' : // insertVertex
cout << "Insert vertex : " << v1 << endl;
testVertex.setLabel(v1);
testGraph.insertVertex(testVertex);
break;

```

```

case '=' : // insertEdge
cout << "Insert edge : " << v1 << " " << v2 << " "
<< wt << endl;
testGraph.insertEdge(v1,v2,wt);
break;

```

```

case '?' : // retrieveVertex
// if ( testGraph.retrieveVertex(v1,testVertex) )
// cout << "Vertex " << v1 << " exists" << endl;
// else
// cout << "Vertex NOT found" << endl;
// break;

case '#' : // edgeWeight
// if ( testGraph.getEdgeWeight(v1,v2,wt) )
// cout << "Weight = " << wt << endl;
// else
// cout << "No edge between these vertices" << endl;
break;

case '-' : // removeVertex
// cout << "Remove vertex " << v1 << endl;
// testGraph.removeVertex(v1);
break;

case '!' : // removeEdge
// cout << "Remove the edge between vertices "
// << v1 << " and " << v2 << endl;
// testGraph.removeEdge(v1,v2);
break;

case 'C' : // clear
cout << "Clear the graph" << endl;
testGraph.clear();
break;

case 'E' : // isEmpty
if ( testGraph.isEmpty() )
cout << "Graph is empty" << endl;
else
cout << "Graph is NOT empty" << endl;
break;

case 'F' : // isFull
if ( testGraph.isFull() )
cout << "Graph is full" << endl;
else
cout << "Graph is NOT full" << endl;
break;

case 'Q' : // Quit test program
break;

default : // Invalid command
cout << "Invalid command" << endl;
}
}
while ( cmd != 'Q' );

return 0;
}

```

```
void print_help()
{
    cout << endl << "Commands:" << endl;
    cout << " H : Help (displays this message)" << endl;
    cout << " +v : Insert (or update) vertex v" << endl;
    cout << " =v w wt : Insert an edge with weight wt between "
    << "vertices v and w" << endl;
    cout << " ?v : Retrieve vertex" << endl;
    cout << " #v w : Display the weight of the edge between "
    << "vertices v and w" << endl;
    cout << " -v : Remove vertex v" << endl;
    cout << " !v w : Remove the edge between vertices v and w"
    << endl;
    cout << " C : Clear the graph" << endl;
    cout << " E : Empty graph?" << endl;
    cout << " F : Full graph?" << endl;
    cout << " Q : Quit the test program" << endl;
    cout << endl;

}
```