KATHMANDU UNIVERSITY

CENTRAL CAMPUS DHULIKHEL, KAVRE

COMP 202

Lab report 6

Submitted by:

Ishan Panta – Roll No: 34
Shaswot Poudel – Roll No: 35

Computer Engineering

2nd Year / 1st Semester

Submitted to:

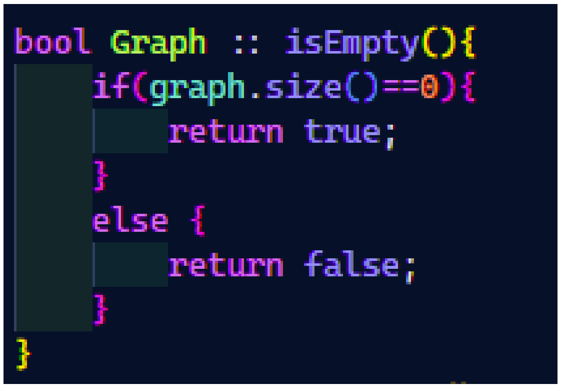
Rajani Chulyadyo

Department of Computer Science and Engineering

Submitted on:

December 06, 2022

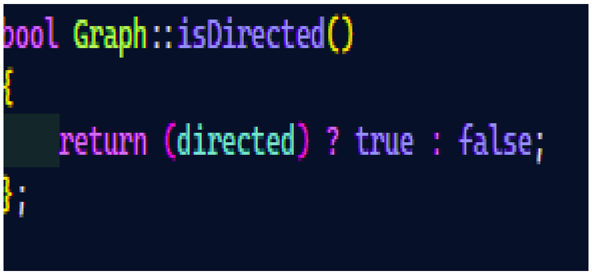
isEmpty()



```
bool Graph :: isEmpty(){
    if(graph.size()==0){
        return true;
    }
    else {
        return false;
    }
}
```

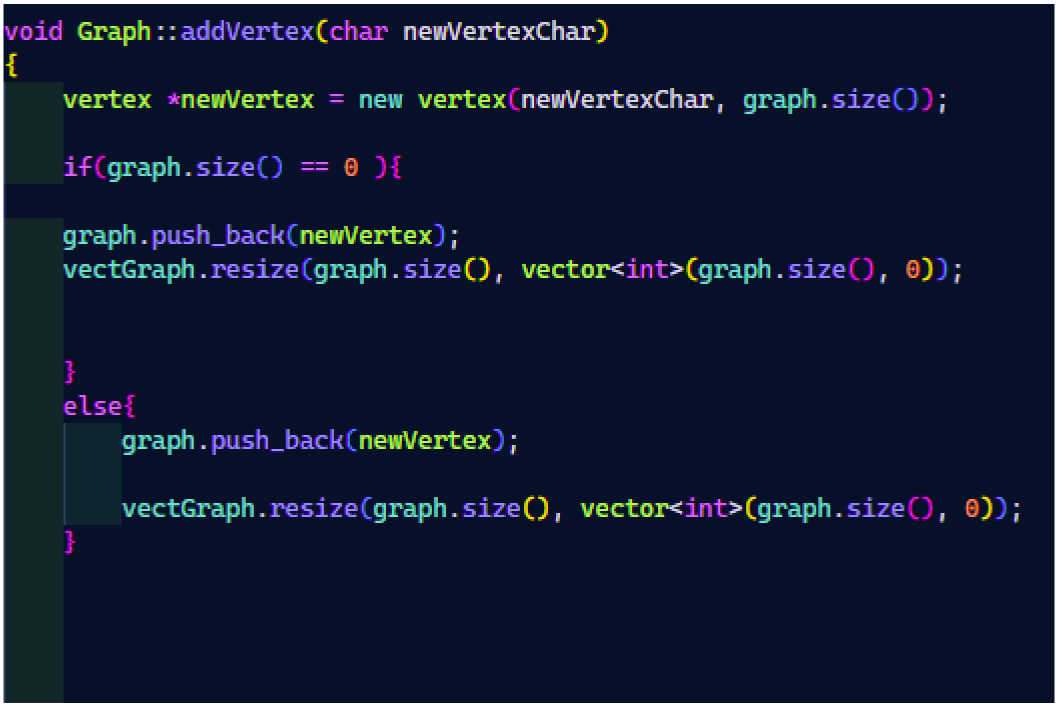
Checks to see whether the graph is empty or not.

isDirected ()



```
bool Graph::isDirected()
{
    return (directed) ? true : false;
};
```

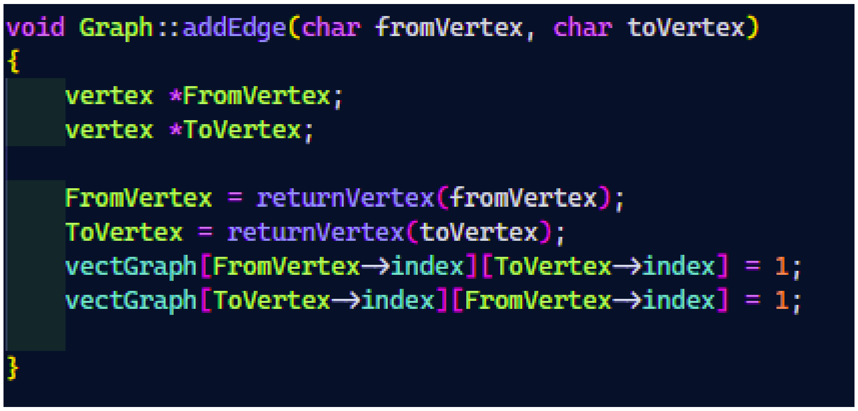
Checks to see if the graph is directed or undirected addVertex(newVertex)



```
void Graph::addVertex(char newVertexChar)
{
    vertex *newVertex = new vertex(newVertexChar, graph.size());

    if(graph.size() == 0 ){

    graph.push_back(newVertex);
    vectGraph.resize(graph.size(), vector<int>(graph.size(), 0));


    }
    else{
        graph.push_back(newVertex);

        vectGraph.resize(graph.size(), vector<int>(graph.size(), 0));
    }
```

Adds a new vertex to the graph with a special character.

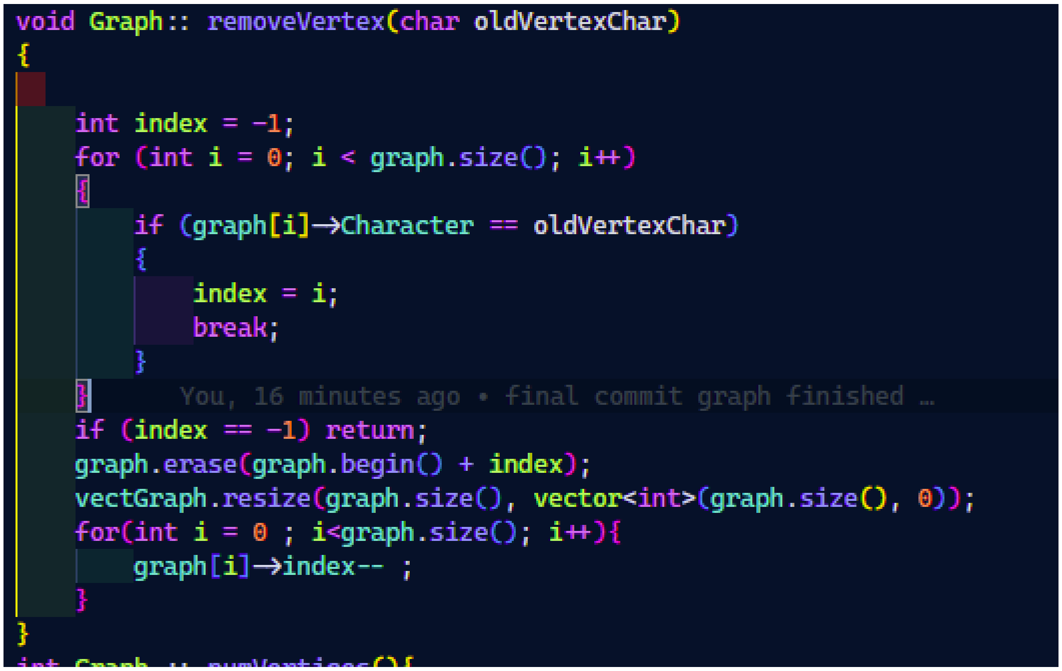
addEdge (vertex1, vertex2)



```
void Graph::addEdge(char fromVertex, char toVertex)
{
    vertex *FromVertex;
    vertex *ToVertex;

    FromVertex = returnVertex(fromVertex);
    ToVertex = returnVertex(toVertex);
    vectGraph[FromVertex->index][ToVertex->index] = 1;
    vectGraph[ToVertex->index][FromVertex->index] = 1;

}
```

Adds edges between two vertices in the graph .

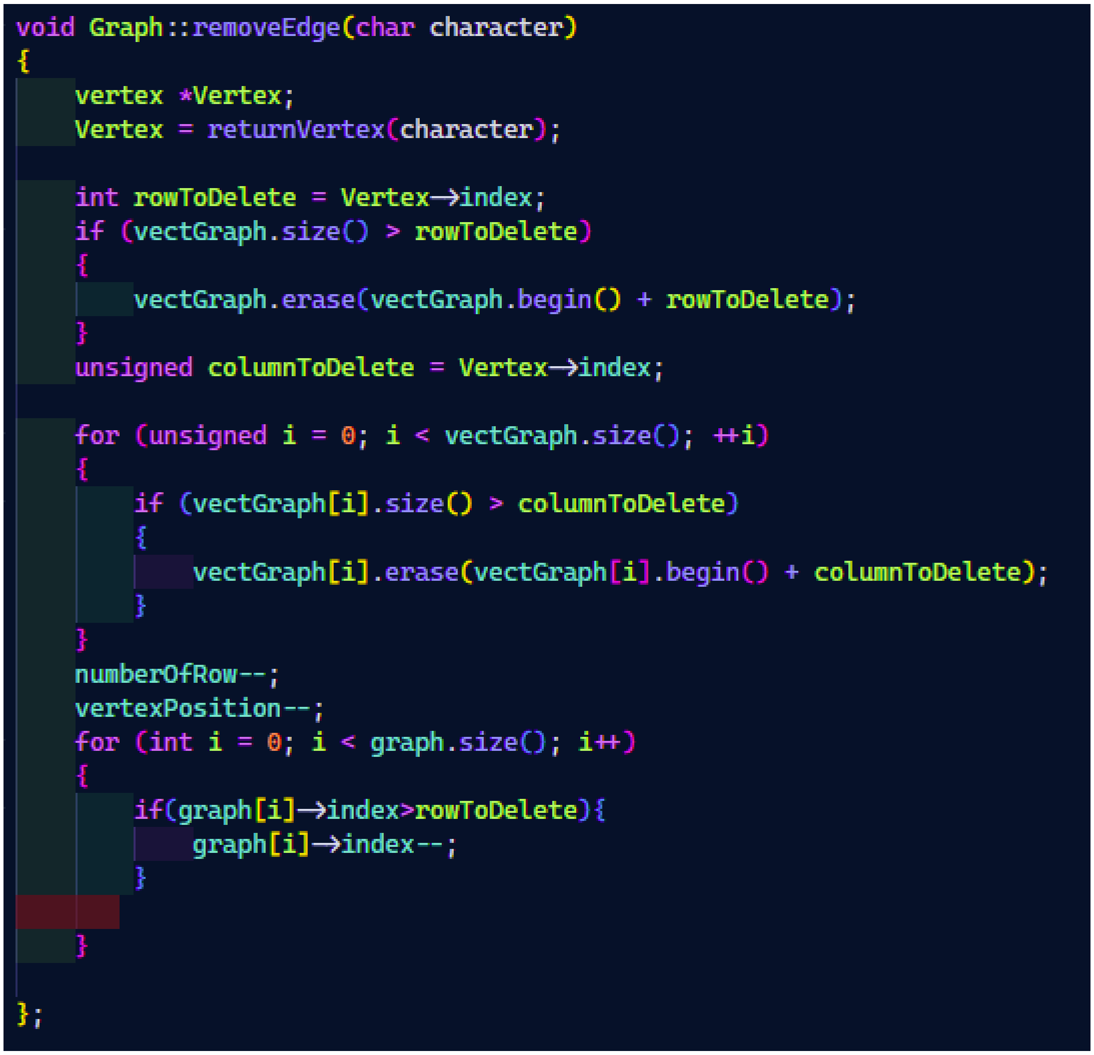
removeVertex(vertexToRemove)



```
void Graph:: removeVertex(char oldVertexChar)
{

    int index = -1;
    for (int i = 0; i < graph.size(); i++)
    {
        if (graph[i]->Character == oldVertexChar)
        {
            index = i;
            break;
        }
    }       You, 16 minutes ago • final commit graph finished …
    if (index == -1) return;
    graph.erase(graph.begin() + index);
    vectGraph.resize(graph.size(), vector<int>(graph.size(), 0));
    for(int i = 0 ; i<graph.size(); i++){
        graph[i]->index-- ;
    }
}
int Graph :: numVertices(){
```

Removes a particular vertex from the graph.

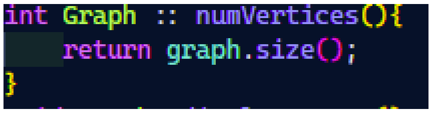
removeEdge (vertex1, vertex2)



```cpp
void Graph::removeEdge(char character)
{
    vertex *Vertex;
    Vertex = returnVertex(character);

    int rowToDelete = Vertex->index;
    if (vectGraph.size() > rowToDelete)
    {
        vectGraph.erase(vectGraph.begin() + rowToDelete);
    }
    unsigned columnToDelete = Vertex->index;

    for (unsigned i = 0; i < vectGraph.size(); ++i)
    {
        if (vectGraph[i].size() > columnToDelete)
        {
            vectGraph[i].erase(vectGraph[i].begin() + columnToDelete);
        }
    }
    numberOfRow--;
    vertexPosition--;
    for (int i = 0; i < graph.size(); i++)
    {
        if(graph[i]->index>rowToDelete){
            graph[i]->index--;
        }

    }

};
```

Removes any edge between two vertices.

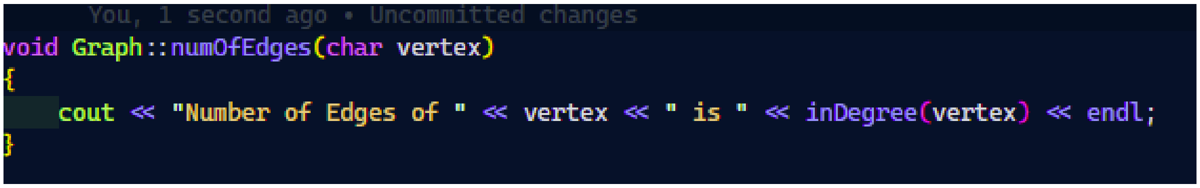
numVertices()



```
int Graph :: numVertices(){
    return graph.size();
}
```

Counts the number of vertices in the graph.

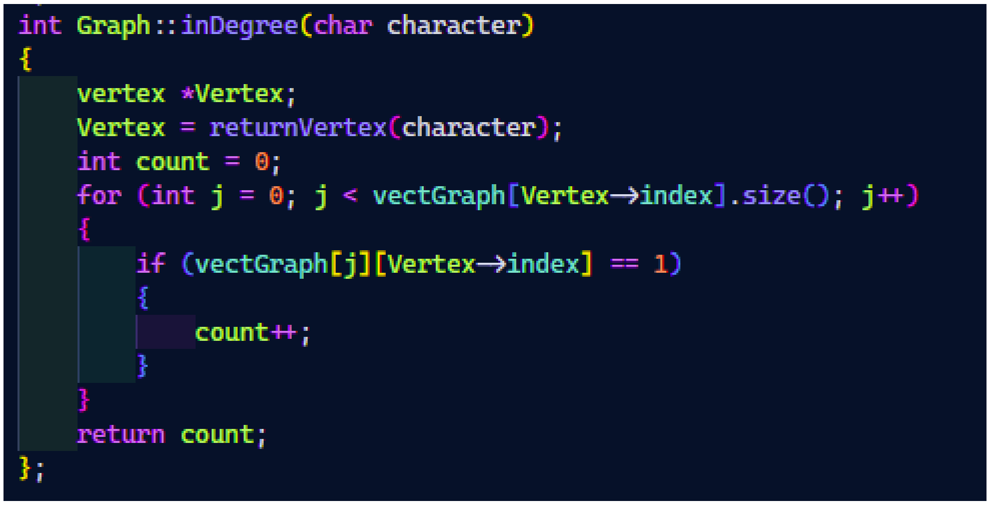
numEdges()



```
      You, 1 second ago • Uncommitted changes
void Graph::numOfEdges(char vertex)
{
    cout << "Number of Edges of " << vertex << " is " << inDegree(vertex) << endl;
}
```

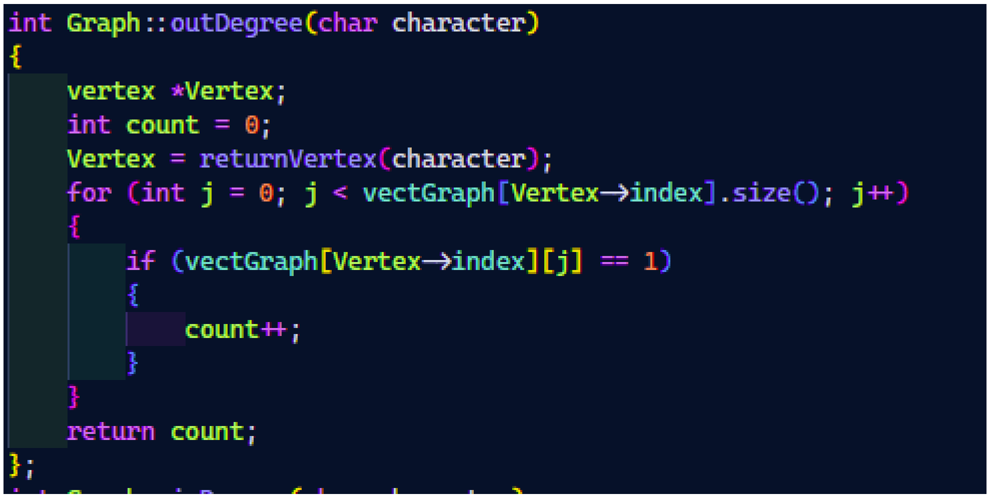
Counts the number of edges in the graph.

indegree(vertex)



```cpp
int Graph::inDegree(char character)
{
    vertex *Vertex;
    Vertex = returnVertex(character);
    int count = 0;
    for (int j = 0; j < vectGraph[Vertex->index].size(); j++)
    {
        if (vectGraph[j][Vertex->index] == 1)
        {
            count++;
        }
    }
    return count;
};
```

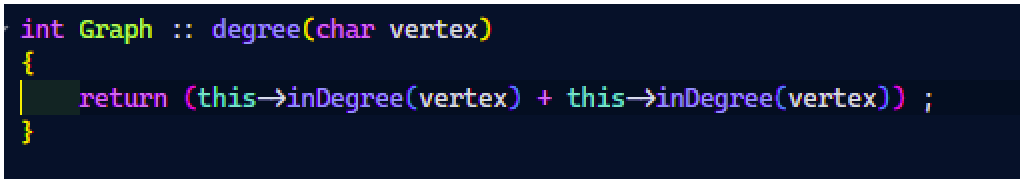
outdegree(vertex)



```cpp
int Graph::outDegree(char character)
{
    vertex *Vertex;
    int count = 0;
    Vertex = returnVertex(character);
    for (int j = 0; j < vectGraph[Vertex->index].size(); j++)
    {
        if (vectGraph[Vertex->index][j] == 1)
        {
            count++;
        }
    }
    return count;
};
```

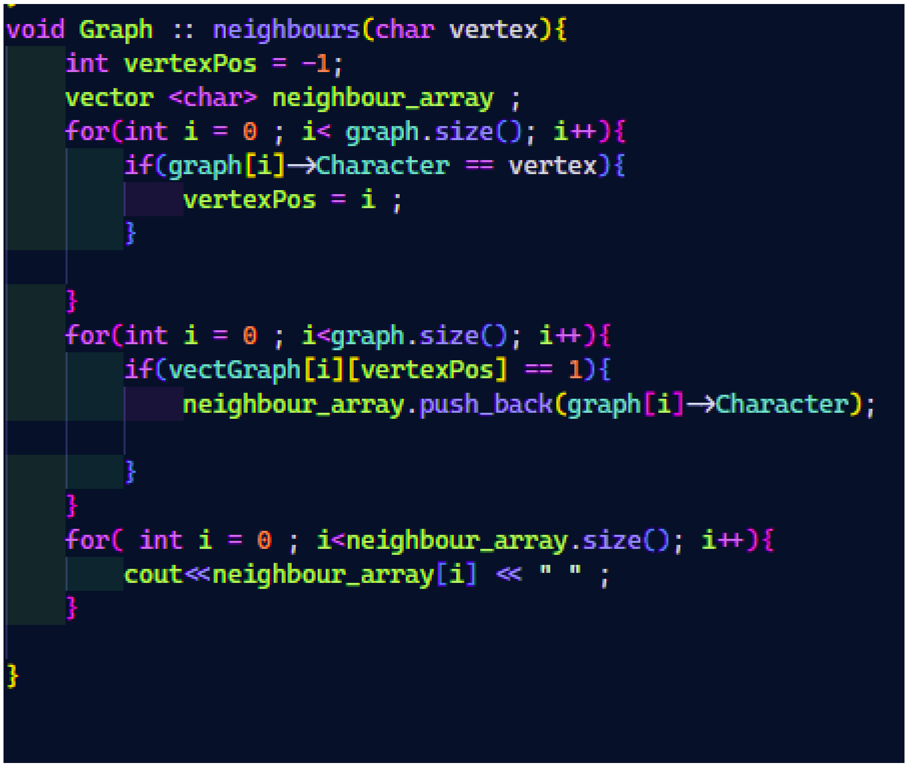
Counts the outdegree of a graph, in a undirected graph outdegree = indegree for a particular vertex.

degree(vertex)



```
int Graph :: degree(char vertex)
{
    return (this->inDegree(vertex) + this->inDegree(vertex)) ;
}
```

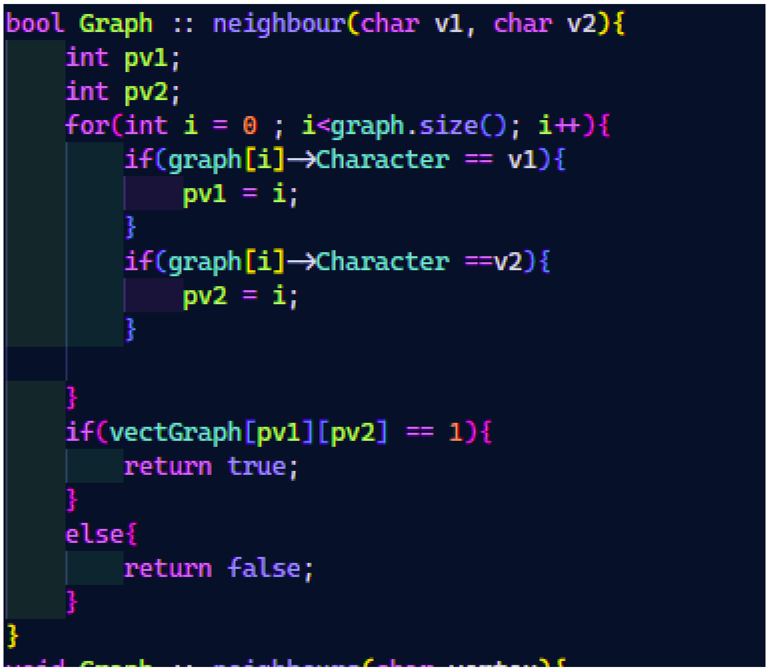
neighbours(vertex)



```cpp
void Graph :: neighbours(char vertex){
    int vertexPos = -1;
    vector <char> neighbour_array ;
    for(int i = 0 ; i< graph.size(); i++){
        if(graph[i]->Character == vertex){
            vertexPos = i ;
        }

    }
    for(int i = 0 ; i<graph.size(); i++){
        if(vectGraph[i][vertexPos] == 1){
            neighbour_array.push_back(graph[i]->Character);

        }
    }
    for( int i = 0 ; i<neighbour_array.size(); i++){
        cout<<neighbour_array[i] << " " ;
    }

}
```

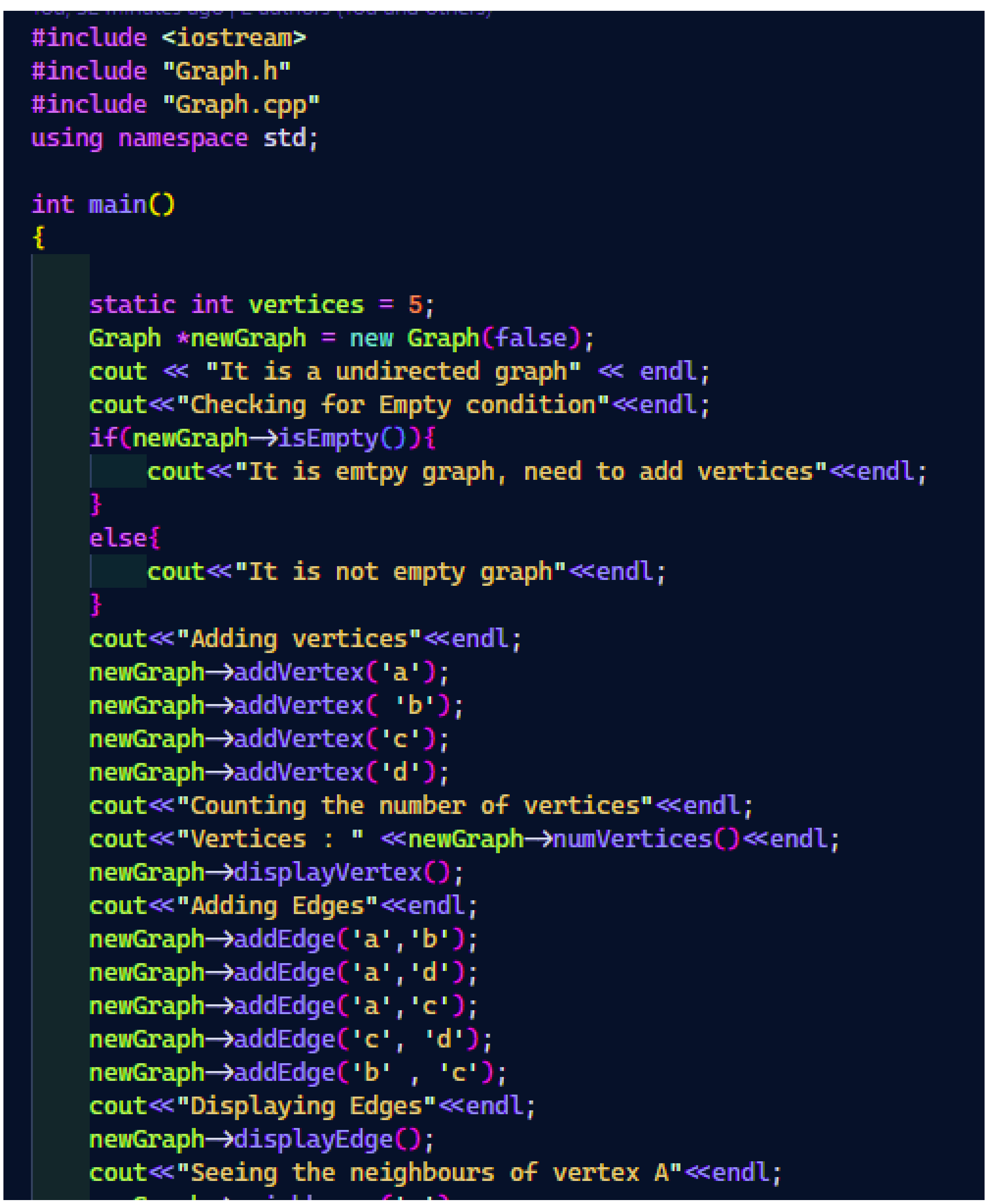
Checks the neighbours of a particular vertex. A neighbour in a graph is a vertex which is directly connected to the mentioned vertex.

neighbour(vertex1, vertex2)

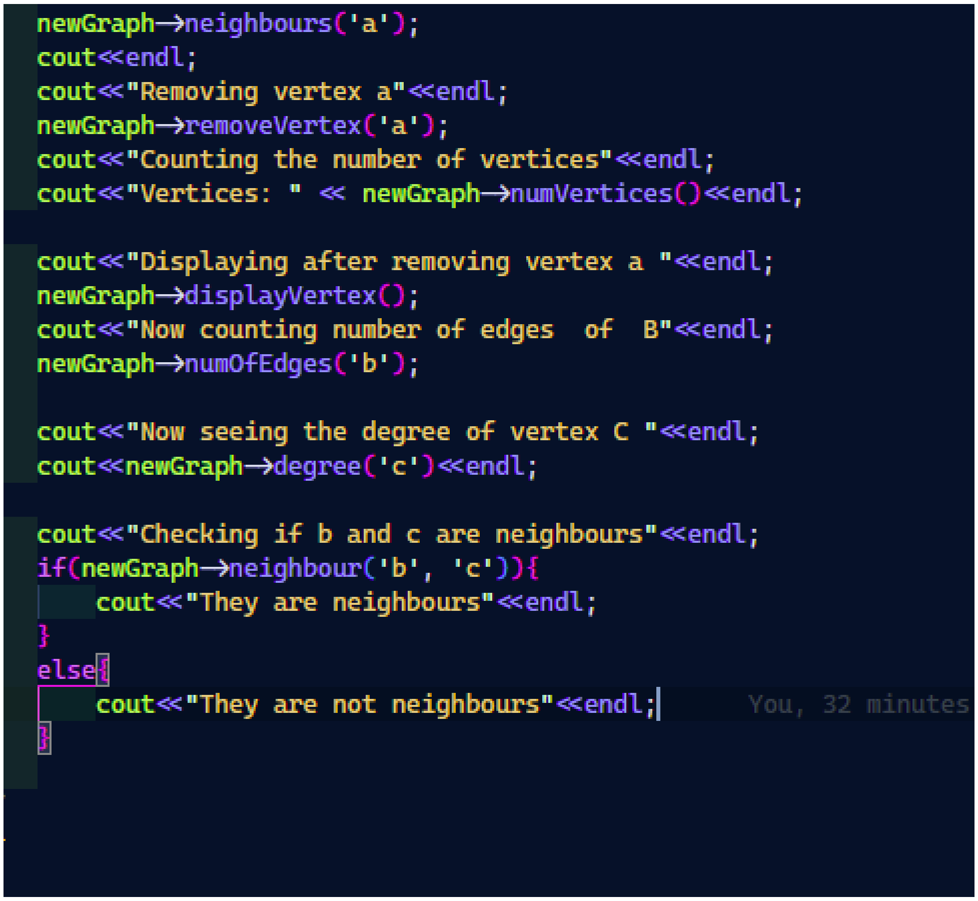


```
bool Graph :: neighbour(char v1, char v2){
    int pv1;
    int pv2;
    for(int i = 0 ; i<graph.size(); i++){
        if(graph[i]->Character == v1){
            pv1 = i;
        }
        if(graph[i]->Character ==v2){
            pv2 = i;
        }

    }
    if(vectGraph[pv1][pv2] == 1){
        return true;
    }
    else{
        return false;
    }
}
void Graph :: neighbours(char vertex){
```

Main.cpp file



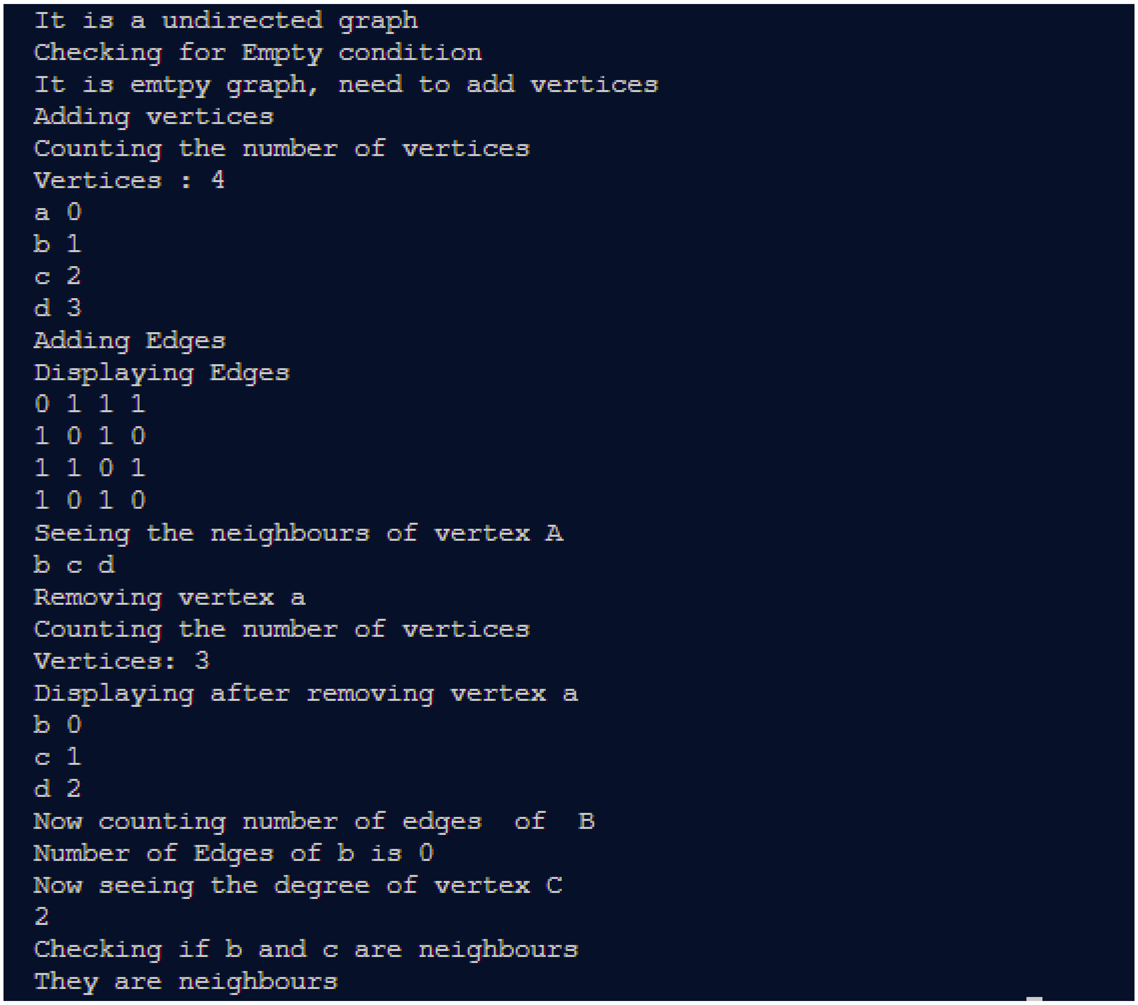
```
You, 32 minutes ago | 2 authors (You and others)
#include <iostream>
#include "Graph.h"
#include "Graph.cpp"
using namespace std;

int main()
{

    static int vertices = 5;
    Graph *newGraph = new Graph(false);
    cout << "It is a undirected graph" << endl;
    cout<<"Checking for Empty condition"<<endl;
    if(newGraph->isEmpty()){
        cout<<"It is emtpy graph, need to add vertices"<<endl;
    }
    else{
        cout<<"It is not empty graph"<<endl;
    }
    cout<<"Adding vertices"<<endl;
    newGraph->addVertex('a');
    newGraph->addVertex( 'b');
    newGraph->addVertex('c');
    newGraph->addVertex('d');
    cout<<"Counting the number of vertices"<<endl;
    cout<<"Vertices : " <<newGraph->numVertices()<<endl;
    newGraph->displayVertex();
    cout<<"Adding Edges"<<endl;
    newGraph->addEdge('a','b');
    newGraph->addEdge('a','d');
    newGraph->addEdge('a','c');
    newGraph->addEdge('c', 'd');
    newGraph->addEdge('b' , 'c');
    cout<<"Displaying Edges"<<endl;
    newGraph->displayEdge();
    cout<<"Seeing the neighbours of vertex A"<<endl;
```



```
newGraph->neighbours('a');
cout<<endl;
cout<<"Removing vertex a"<<endl;
newGraph->removeVertex('a');
cout<<"Counting the number of vertices"<<endl;
cout<<"Vertices: " << newGraph->numVertices()<<endl;

cout<<"Displaying after removing vertex a "<<endl;
newGraph->displayVertex();
cout<<"Now counting number of edges  of  B"<<endl;
newGraph->numOfEdges('b');

cout<<"Now seeing the degree of vertex C "<<endl;
cout<<newGraph->degree('c')<<endl;

cout<<"Checking if b and c are neighbours"<<endl;
if(newGraph->neighbour('b', 'c')){
    cout<<"They are neighbours"<<endl;
}
else{
    cout<<"They are not neighbours"<<endl;      You, 32 minutes
}
```

The main.cpp file of the program

Output of program



```
It is a undirected graph
Checking for Empty condition
It is emtpy graph, need to add vertices
Adding vertices
Counting the number of vertices
Vertices : 4
a 0
b 1
c 2
d 3
Adding Edges
Displaying Edges
0 1 1 1
1 0 1 0
1 1 0 1
1 0 1 0
Seeing the neighbours of vertex A
b c d
Removing vertex a
Counting the number of vertices
Vertices: 3
Displaying after removing vertex a
b 0
c 1
d 2
Now counting number of edges  of  B
Number of Edges of b is 0
Now seeing the degree of vertex C
2
Checking if b and c are neighbours
They are neighbours
```