```cpp

#include <iostream>
#include <list>
#include "graph.h"
/*****************************************************************************
Author: Jose Eduardo Morales
graph.cpp
Date: march 22, 2023
*****************************************************************************/

//empty construct
Graph::Graph(){};
//construct that creates an empty graph with size as a parameter
Graph::Graph (int size = 5):m_size(size) { //constructor

    if (m_size == 0){
        std::cout << "error initializing graph at 0" << std::endl;
    }

    graph = new int*[m_size];
    for (int i = 0; i < m_size; i++)
        graph[i] = new int[m_size];

    //std::cout << "graph created" << std::endl;

}
Graph::~Graph(){ // destructor
    //std::cout << "destroying graph" << std::endl;
}

template <typename T>
void Graph::printGraph(T** graph, int rows, int cols) {
    //print titles
    std::cout << "\t";
    for(int i = 0; i < cols; ++i)
    {
        std::cout << "n" << i << "\t" ;
    }
    //print content
    std::cout << std::endl;
    for(int i = 0; i < rows; ++i) {
        std::cout << "n" << i << "\t";
        for(int j = 0; j < cols; ++j) {
            std::cout << graph[i][j] << " \t";
        }
        std::cout << std::endl;
    }
}
// public print
void Graph::print(){
    printGraph(graph, m_size, m_size);

}

int Graph::V(){       //return the # of vertices in the Graph
    return m_size;
}

int Graph::E(){       //return the # of edges in the Graph
    int count_E = 0;
    for(int i = 0; i < m_size; ++i) {
        for(int j = 0; j < m_size; ++j) {
            if ( graph[i][j] > 0)
                count_E ++;
        }
    }
    return count_E;
}

bool Graph::adjacent(int x, int y){ //test if there is an edge from x to y
    return (graph[x][y] > 0);
}

```

```
74  std::list<int> Graph::neighbors(int x){ // list all nodes y with edges to x
75      std::list<int> neigbhborsToX = {};
76
77      for(int i = 0; i < m_size; ++i) {
78          if ( graph[x][i] > 0)
79              neigbhborsToX.push_back(i);
80      }
81      return neigbhborsToX;
82  }
83
84  void Graph::add(int x, int y, int value){ //add the edge from x to y
85      graph[x][y] = value;
86      graph[y][x] = value;    // make bidirectional edge
87  }
88
89  void Graph::remove(int x, int y){ //removes edge from x to y
90      graph[x][y] = 0;
91      graph[y][x] = 0;        // bidirectional edge
92  }
93
94  int Graph::getEdge(int x, int y){
95      return graph[x][y];
96  }
97
98  float Graph::getDensity(){
99      return static_cast<float>(E())*100.0/(V()*(V()-1));
100 }
101 /*
102 int Graph::get_node_value(int x){return x} //return node name
103 void Graph::set_node_value(int x, char a){} //change node name
104
105 int Graph::get_edge_value(int x, int y);
106 void Graph::set_edge_value(int x, int y, int value);
107
108   */
```