LAPORAN PRAKTIKUM

PERTEMUAN 14

GRAPH

Nama :

Resita Istania Purwanto (2311104037)

Dosen :

Yudha Islami Sulistya

PROGRAM STUDI S1 REKAYASA PERANGKAT LUNAK

FAKULTAS INFORMATIKA

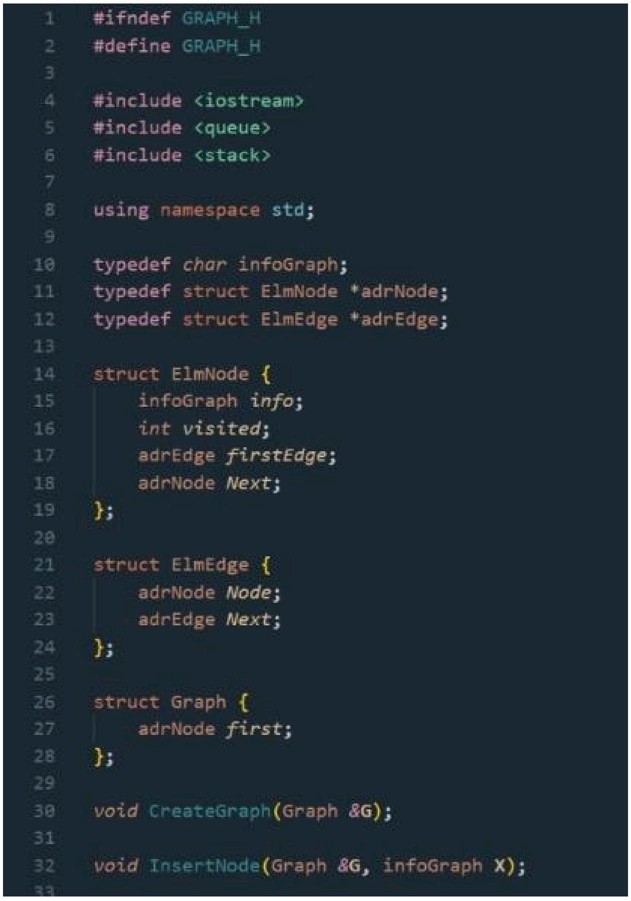
TELKOM UNIVERSITY PURWOKERTO

2024

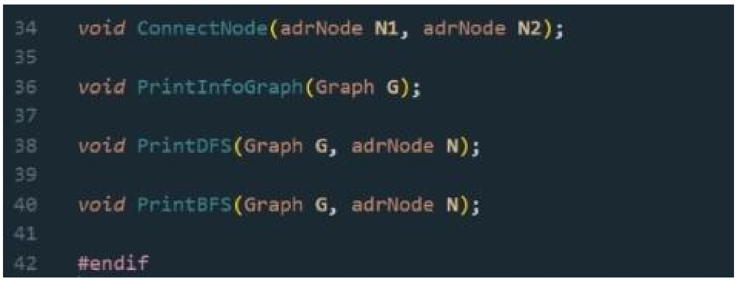
I. GUIDED

kode:

graph.h

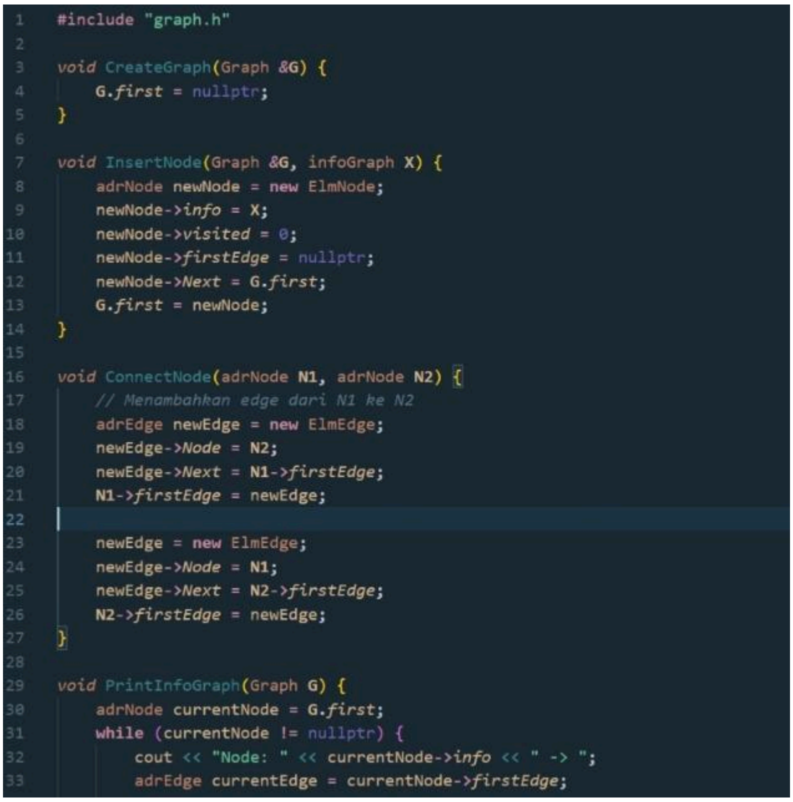


```
#ifndef GRAPH_H
#define GRAPH_H

#include <iostream>
#include <queue>
#include <stack>

using namespace std;

typedef char infoGraph;
typedef struct ElmNode *adrNode;
typedef struct ElmEdge *adrEdge;

struct ElmNode {
    infoGraph info;
    int visited;
    adrEdge firstEdge;
    adrNode Next;
};

struct ElmEdge {
    adrNode Node;
    adrEdge Next;
};

struct Graph {
    adrNode first;
};

void CreateGraph(Graph &G);

void InsertNode(Graph &G, infoGraph X);
```

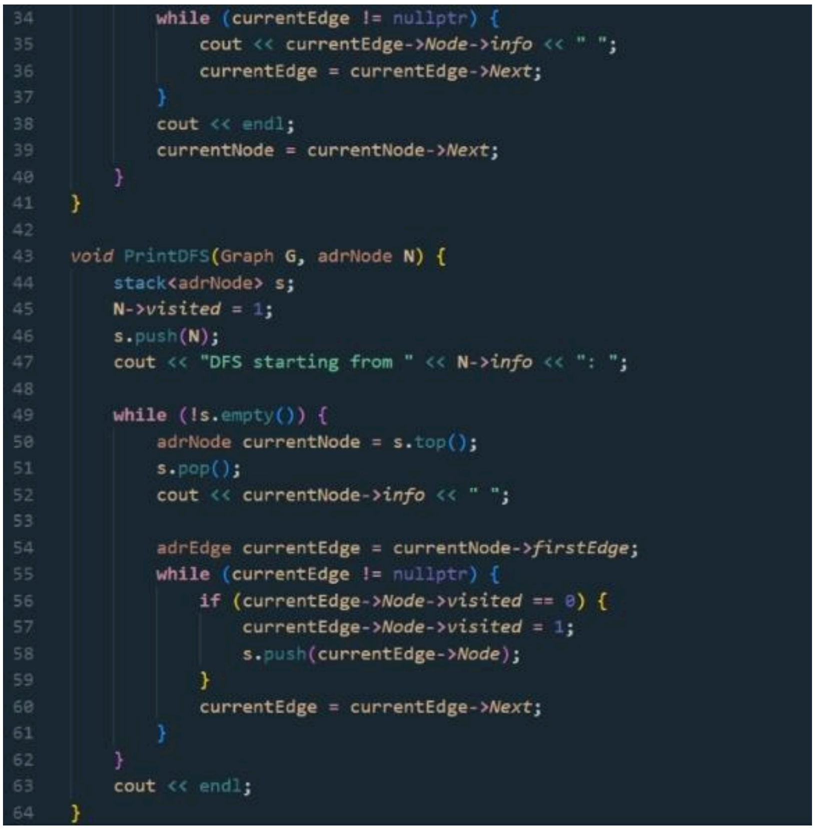


```
void ConnectNode(adrNode N1, adrNode N2);

void PrintInfoGraph(Graph G);

void PrintDFS(Graph G, adrNode N);

void PrintBFS(Graph G, adrNode N);

#endif
```

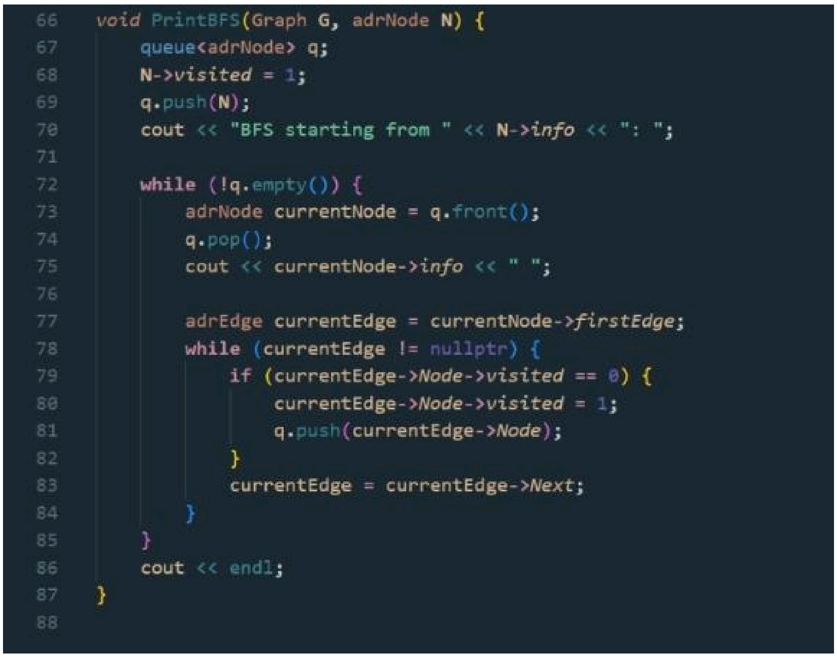
graph.cpp



```cpp
#include "graph.h"

void CreateGraph(Graph &G) {
    G.first = nullptr;
}

void InsertNode(Graph &G, infoGraph X) {
    adrNode newNode = new ElmNode;
    newNode->info = X;
    newNode->visited = 0;
    newNode->firstEdge = nullptr;
    newNode->Next = G.first;
    G.first = newNode;
}

void ConnectNode(adrNode N1, adrNode N2) {
    // Menambahkan edge dari N1 ke N2
    adrEdge newEdge = new ElmEdge;
    newEdge->Node = N2;
    newEdge->Next = N1->firstEdge;
    N1->firstEdge = newEdge;

    newEdge = new ElmEdge;
    newEdge->Node = N1;
    newEdge->Next = N2->firstEdge;
    N2->firstEdge = newEdge;
}

void PrintInfoGraph(Graph G) {
    adrNode currentNode = G.first;
    while (currentNode != nullptr) {
        cout << "Node: " << currentNode->info << " -> ";
        adrEdge currentEdge = currentNode->firstEdge;
```

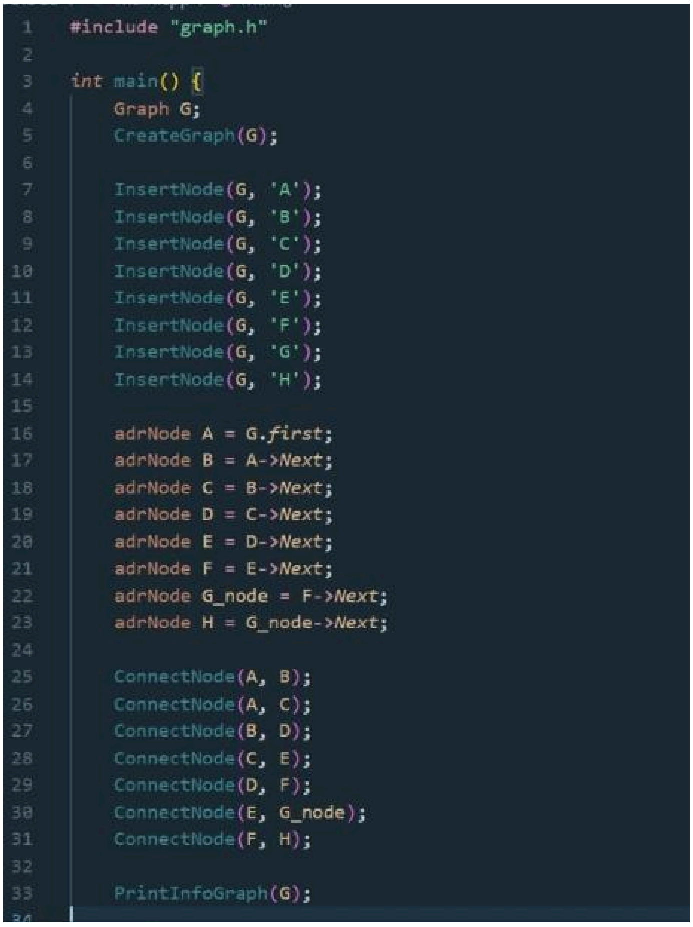


```cpp
        while (currentEdge != nullptr) {
            cout << currentEdge->Node->info << " ";
            currentEdge = currentEdge->Next;
        }
        cout << endl;
        currentNode = currentNode->Next;
    }
}

void PrintDFS(Graph G, adrNode N) {
    stack<adrNode> s;
    N->visited = 1;
    s.push(N);
    cout << "DFS starting from " << N->info << ": ";

    while (!s.empty()) {
        adrNode currentNode = s.top();
        s.pop();
        cout << currentNode->info << " ";

        adrEdge currentEdge = currentNode->firstEdge;
        while (currentEdge != nullptr) {
            if (currentEdge->Node->visited == 0) {
                currentEdge->Node->visited = 1;
                s.push(currentEdge->Node);
            }
            currentEdge = currentEdge->Next;
        }
    }
    cout << endl;
}
```

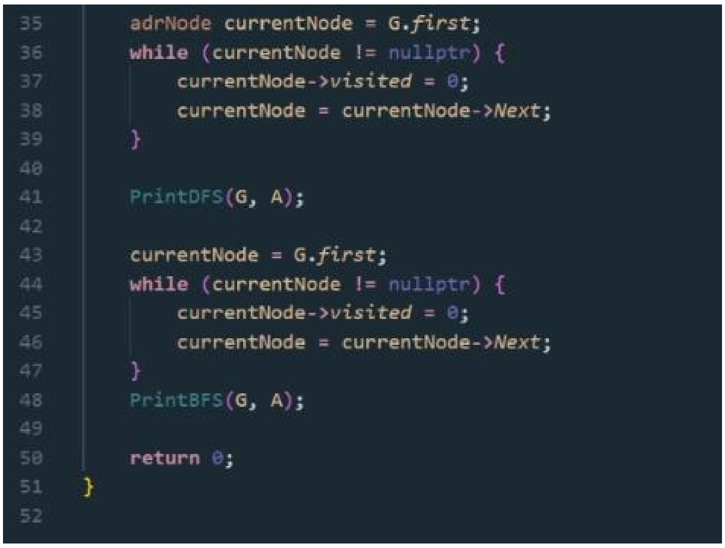


```cpp
void PrintBFS(Graph G, adrNode N) {
    queue<adrNode> q;
    N->visited = 1;
    q.push(N);
    cout << "BFS starting from " << N->info << ": ";

    while (!q.empty()) {
        adrNode currentNode = q.front();
        q.pop();
        cout << currentNode->info << " ";

        adrEdge currentEdge = currentNode->firstEdge;
        while (currentEdge != nullptr) {
            if (currentEdge->Node->visited == 0) {
                currentEdge->Node->visited = 1;
                q.push(currentEdge->Node);
            }
            currentEdge = currentEdge->Next;
        }
    }
    cout << endl;
}
```

main.cpp



```cpp
#include "graph.h"

int main() {
    Graph G;
    CreateGraph(G);

    InsertNode(G, 'A');
    InsertNode(G, 'B');
    InsertNode(G, 'C');
    InsertNode(G, 'D');
    InsertNode(G, 'E');
    InsertNode(G, 'F');
    InsertNode(G, 'G');
    InsertNode(G, 'H');

    adrNode A = G.first;
    adrNode B = A->Next;
    adrNode C = B->Next;
    adrNode D = C->Next;
    adrNode E = D->Next;
    adrNode F = E->Next;
    adrNode G_node = F->Next;
    adrNode H = G_node->Next;

    ConnectNode(A, B);
    ConnectNode(A, C);
    ConnectNode(B, D);
    ConnectNode(C, E);
    ConnectNode(D, F);
    ConnectNode(E, G_node);
    ConnectNode(F, H);

    PrintInfoGraph(G);
```



```
    adrNode currentNode = G.first;
    while (currentNode != nullptr) {
        currentNode->visited = 0;
        currentNode = currentNode->Next;
    }

    PrintDFS(G, A);

    currentNode = G.first;
    while (currentNode != nullptr) {
        currentNode->visited = 0;
        currentNode = currentNode->Next;
    }
    PrintBFS(G, A);

    return 0;
}
```

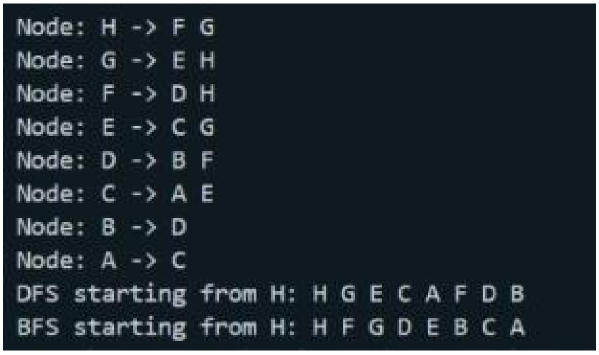
SOAL NO 2
prosedur DFS

```
void PrintDFS(Graph G, adrNode N) {
    stack<adrNode> s;
    N->visited = 1;
    s.push(N);
    cout << "DFS starting from " << N->info << ": ";

    while (!s.empty()) {
        adrNode currentNode = s.top();
        s.pop();
        cout << currentNode->info << " ";

        adrEdge currentEdge = currentNode->firstEdge;
        while (currentEdge != nullptr) {
            if (currentEdge->Node->visited == 0) {
                currentEdge->Node->visited = 1;
                s.push(currentEdge->Node);
            }
            currentEdge = currentEdge->Next;
        }
    }
    cout << endl;
}
```

SOAL 3
prosedur BFS

```
void PrintBFS(Graph G, adrNode N) {
    queue<adrNode> q;
    N->visited = 1;
    q.push(N);
    cout << "BFS starting from " << N->info << ": ";

    while (!q.empty()) {
        adrNode currentNode = q.front();
        q.pop();
        cout << currentNode->info << " ";

        adrEdge currentEdge = currentNode->firstEdge;
        while (currentEdge != nullptr) {
            if (currentEdge->Node->visited == 0) {
                currentEdge->Node->visited = 1;
                q.push(currentEdge->Node);
            }
            currentEdge = currentEdge->Next;
        }
    }
    cout << endl;
}
```

output:



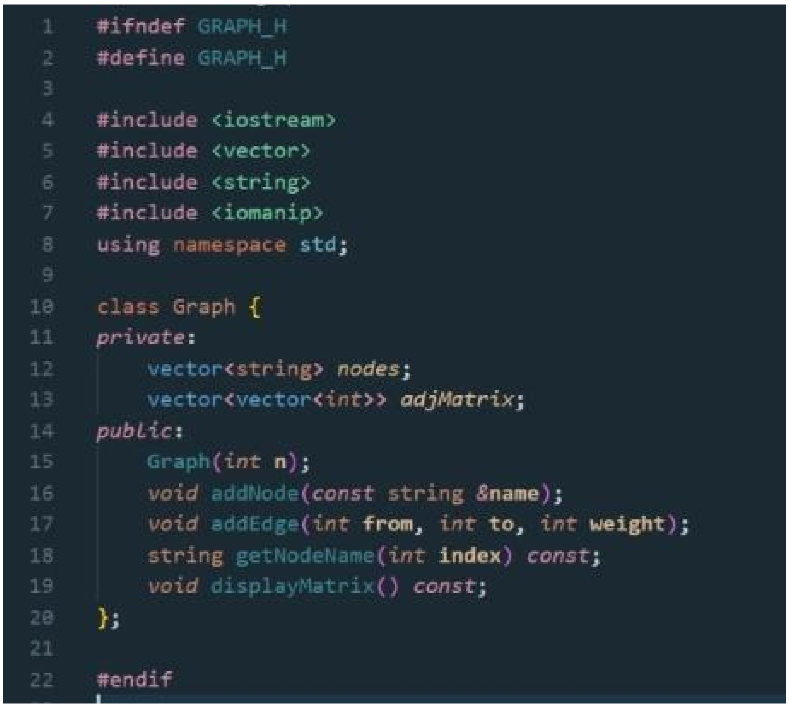
```
Node: H -> F G
Node: G -> E H
Node: F -> D H
Node: E -> C G
Node: D -> B F
Node: C -> A E
Node: B -> D
Node: A -> C
DFS starting from H: H G E C A F D B
BFS starting from H: H F G D E B C A
```

penjelasan:

a. Graph:
    - ElmNode: Menyimpan data dan koneksi.
    - ElmEdge: Menghubungkan node.
    - Graph: Menyimpan node pertama.
b. Fungsi:
    - CreateGraph: Membuat graph.
    - InsertNode: Menambah node.
    - ConnectNode: Menghubungkan node.
    - PrintInfoGraph: Menampilkan graph.
    - PrintDFS/BFS: Penelusuran DFS/BFS.
c. Program Utama: Membuat graph, menambah node, menghubungkan, dan menampilkan DFS/BFS.

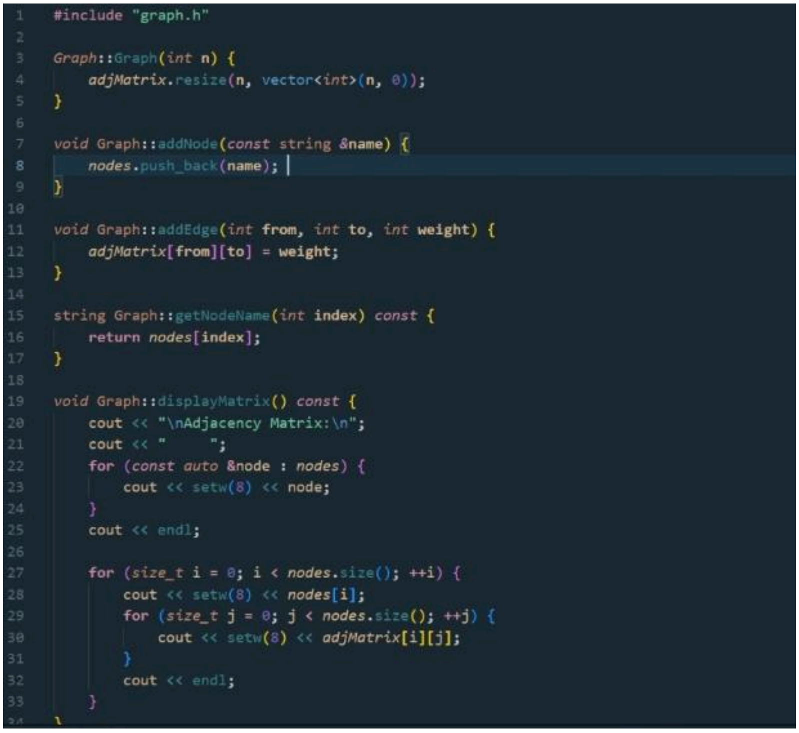
II. UNGUIDED

1. soal 1
   kode:
   graph.h



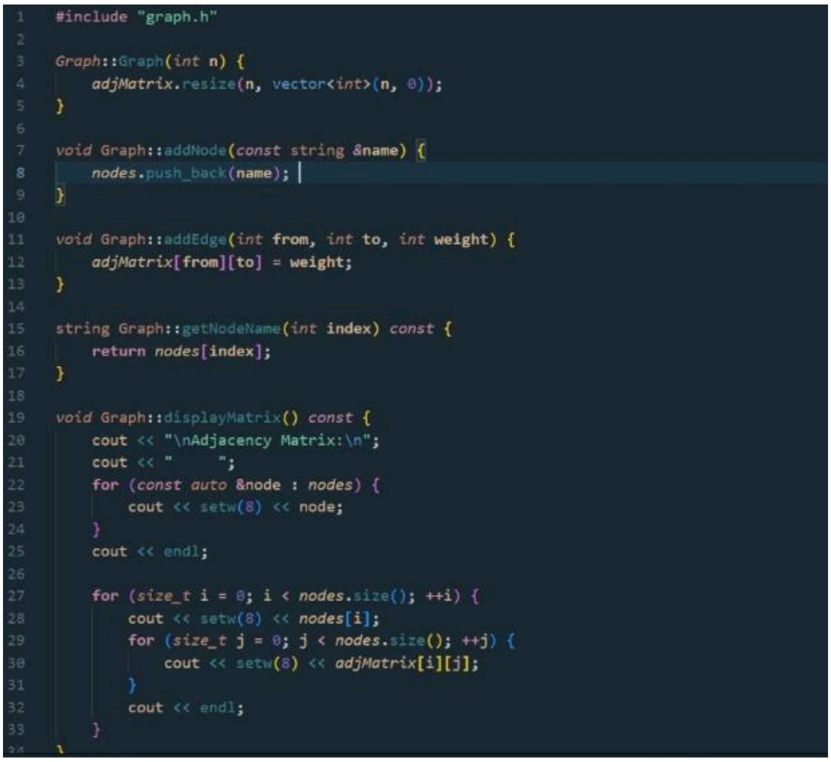
```cpp
#ifndef GRAPH_H
#define GRAPH_H

#include <iostream>
#include <vector>
#include <string>
#include <iomanip>
using namespace std;

class Graph {
private:
    vector<string> nodes;
    vector<vector<int>> adjMatrix;
public:
    Graph(int n);
    void addNode(const string &name);
    void addEdge(int from, int to, int weight);
    string getNodeName(int index) const;
    void displayMatrix() const;
};

#endif
```

graph.cpp



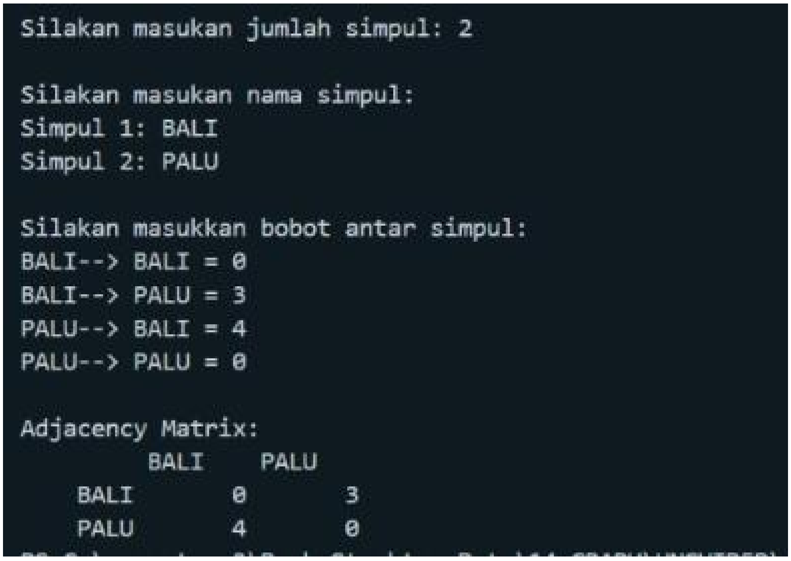
```cpp
#include "graph.h"

Graph::Graph(int n) {
    adjMatrix.resize(n, vector<int>(n, 0));
}

void Graph::addNode(const string &name) {
    nodes.push_back(name);
}

void Graph::addEdge(int from, int to, int weight) {
    adjMatrix[from][to] = weight;
}

string Graph::getNodeName(int index) const {
    return nodes[index];
}

void Graph::displayMatrix() const {
    cout << "\nAdjacency Matrix:\n";
    cout << "      ";
    for (const auto &node : nodes) {
        cout << setw(8) << node;
    }
    cout << endl;

    for (size_t i = 0; i < nodes.size(); ++i) {
        cout << setw(8) << nodes[i];
        for (size_t j = 0; j < nodes.size(); ++j) {
            cout << setw(8) << adjMatrix[i][j];
        }
        cout << endl;
    }
}
```

main.cpp



```cpp
#include "graph.h"

Graph::Graph(int n) {
    adjMatrix.resize(n, vector<int>(n, 0));
}

void Graph::addNode(const string &name) {
    nodes.push_back(name);
}

void Graph::addEdge(int from, int to, int weight) {
    adjMatrix[from][to] = weight;
}

string Graph::getNodeName(int index) const {
    return nodes[index];
}

void Graph::displayMatrix() const {
    cout << "\nAdjacency Matrix:\n";
    cout << "     ";
    for (const auto &node : nodes) {
        cout << setw(8) << node;
    }
    cout << endl;

    for (size_t i = 0; i < nodes.size(); ++i) {
        cout << setw(8) << nodes[i];
        for (size_t j = 0; j < nodes.size(); ++j) {
            cout << setw(8) << adjMatrix[i][j];
        }
        cout << endl;
    }
}
```

output:

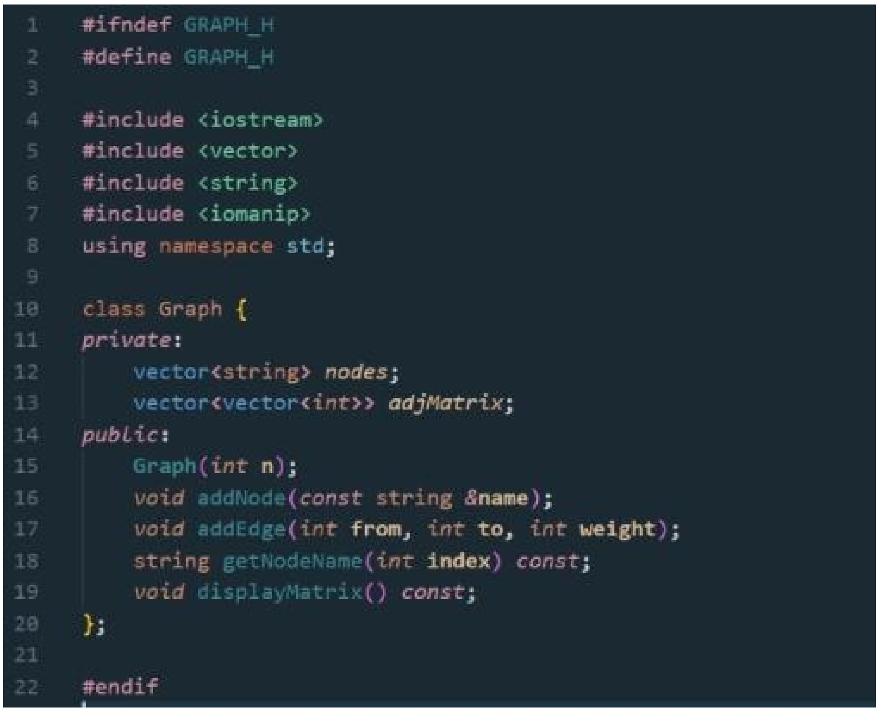


```
Silakan masukan jumlah simpul: 2

Silakan masukan nama simpul:
Simpul 1: BALI
Simpul 2: PALU

Silakan masukkan bobot antar simpul:
BALI--> BALI = 0
BALI--> PALU = 3
PALU--> BALI = 4
PALU--> PALU = 0

Adjacency Matrix:
             BALI    PALU
    BALI        0       3
    PALU        4       0
```

penjelasan:

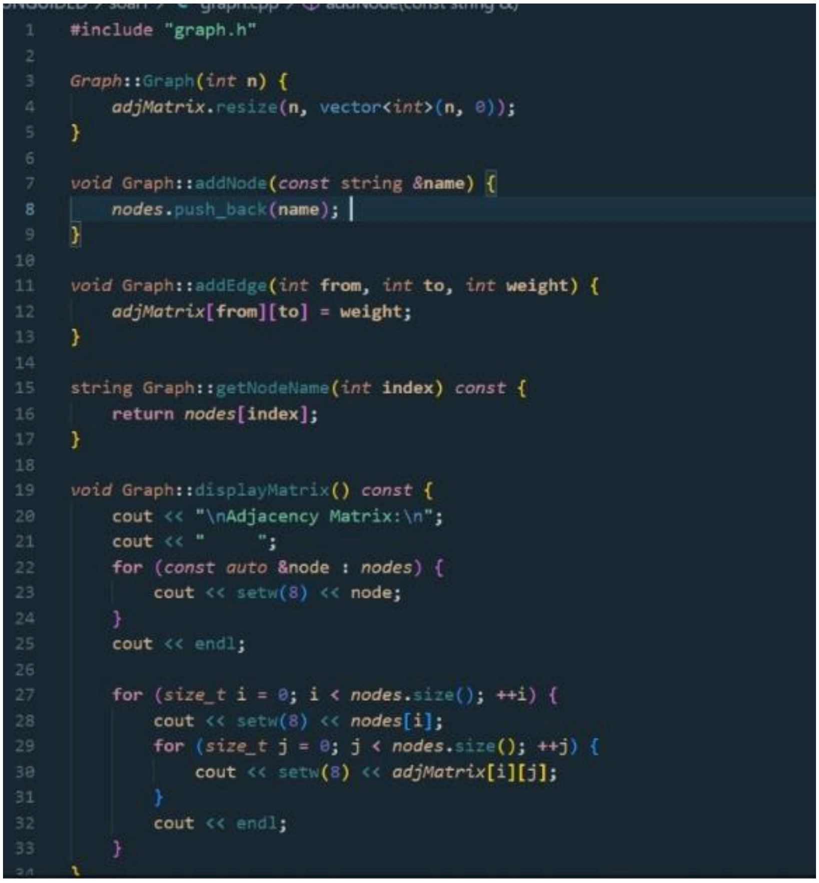
Program ini menggunakan adjacency matrix untuk merepresentasikan bobot (jarak) antar simpul (kota). Inputnya mencakup jumlah simpul, nama simpul, dan bobot antar simpul.

2. soal 2
   kode:



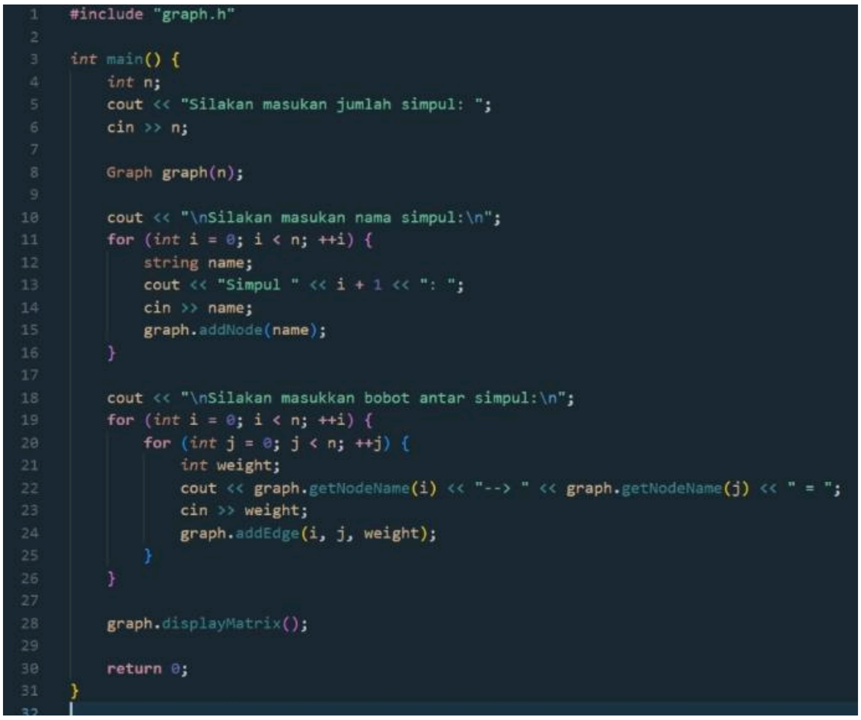
```cpp
#ifndef GRAPH_H
#define GRAPH_H

#include <iostream>
#include <vector>
#include <string>
#include <iomanip>
using namespace std;

class Graph {
private:
    vector<string> nodes;
    vector<vector<int>> adjMatrix;
public:
    Graph(int n);
    void addNode(const string &name);
    void addEdge(int from, int to, int weight);
    string getNodeName(int index) const;
    void displayMatrix() const;
};

#endif
```

graph.cpp



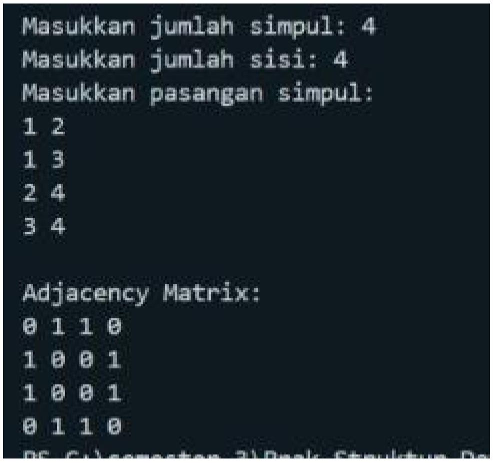
```cpp
#include "graph.h"

Graph::Graph(int n) {
    adjMatrix.resize(n, vector<int>(n, 0));
}

void Graph::addNode(const string &name) {
    nodes.push_back(name);
}

void Graph::addEdge(int from, int to, int weight) {
    adjMatrix[from][to] = weight;
}

string Graph::getNodeName(int index) const {
    return nodes[index];
}

void Graph::displayMatrix() const {
    cout << "\nAdjacency Matrix:\n";
    cout << "        ";
    for (const auto &node : nodes) {
        cout << setw(8) << node;
    }
    cout << endl;

    for (size_t i = 0; i < nodes.size(); ++i) {
        cout << setw(8) << nodes[i];
        for (size_t j = 0; j < nodes.size(); ++j) {
            cout << setw(8) << adjMatrix[i][j];
        }
        cout << endl;
    }
```

main.cpp



```cpp
#include "graph.h"

int main() {
    int n;
    cout << "Silakan masukan jumlah simpul: ";
    cin >> n;

    Graph graph(n);

    cout << "\nSilakan masukan nama simpul:\n";
    for (int i = 0; i < n; ++i) {
        string name;
        cout << "Simpul " << i + 1 << ": ";
        cin >> name;
        graph.addNode(name);
    }

    cout << "\nSilakan masukkan bobot antar simpul:\n";
    for (int i = 0; i < n; ++i) {
        for (int j = 0; j < n; ++j) {
            int weight;
            cout << graph.getNodeName(i) << "--> " << graph.getNodeName(j) << " = ";
            cin >> weight;
            graph.addEdge(i, j, weight);
        }
    }

    graph.displayMatrix();

    return 0;
}
```

output:



```
Masukkan jumlah simpul: 4
Masukkan jumlah sisi: 4
Masukkan pasangan simpul:
1 2
1 3
2 4
3 4

Adjacency Matrix:
0 1 1 0
1 0 0 1
1 0 0 1
0 1 1 0
PS C:\semester 3\Prak Struktur Da
```

penjelasan:Program ini menerima input jumlah simpul dan jumlah sisi, serta pasangan simpul yang terhubung. Adjacency matrix digunakan untuk merepresentasikan graf tidak berarah.