Praktikum IF411326 - IF311326: Struktur Data

Pelintasan Graf dengan Algorithma BFS

Minggu/Sesi	:	15/2
Tujuan	:	• Mampu membuat langkah-langkah pemecahan masalah untuk graph traversal dengan menggunakan algoritma `breadth first search (BFS)` • Mampu mengimplementasi `BFS`
Setoran	:	**Lembar jawaban dikumpulkan kepada asisten**
Waktu penyetoran	:	akhir sesi praktikum

Petunjuk Praktikum

1. Anda dapat mengerjakan praktikum ini secara berkelompok dengan maksimum 2 mahasiswa per kelompok.
2. Buat folder: mg15_sesi02_topik _praktikum. Semua program pada praktikum sesi ini akan disimpan pada folder tersebut.
3. Ikuti semua prosedur praktikum.

Referensi

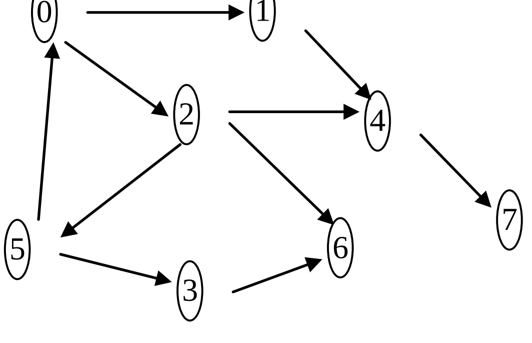
1. T.H. Cormen, C.E. Leiserson, R.L. Rivest and C. Stein, *Introduction to Algorithms*, 2nd Eds., MIT Press 2001.
2. M.A. Weiss, *Data Structures and Algorithm Analysis in C, 2nd Eds.*, Addison-Wesley, 1997.

Ulasan

1. Jelaskan properti dari BFS.

Pendalaman

1. Jika diberikan graf berarah berarah berikut ini:



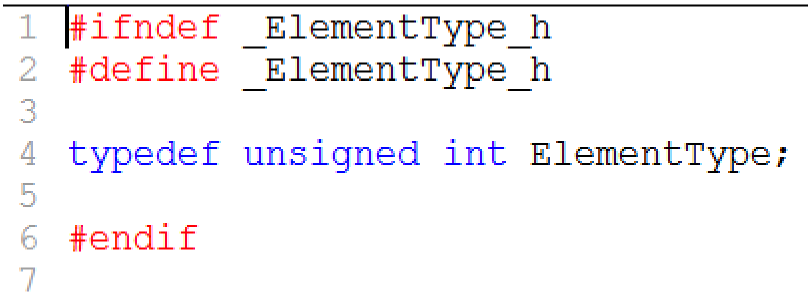
   a. Buatlah representasi *adjacency list* untuk kedua graf tersebut.
   b. Buatlah representasi *adjacency matrix* untuk kedua graf tersebut.
   c. Gambarkanlah illustasi dari langkah-langkah penerapan algorithma `BFS` untuk pelintasan graf di atas.
   d. Gambarkan pula pohon `bread-first (bread-first-tree).`

Pemrograman

Anda telah mempelajari bahwa BFS memerlukan struktur data Queue untuk menampung simpul-simpul yang mengantri, yakni simpul yang baru saja ditemukan, tetapi successors belum di temukan (simpul berwarna abu-abu). Oleh sebab itu, implementasi struktur data Graph yang mendukung pelintasan graph terdiri atas file-file berikut:

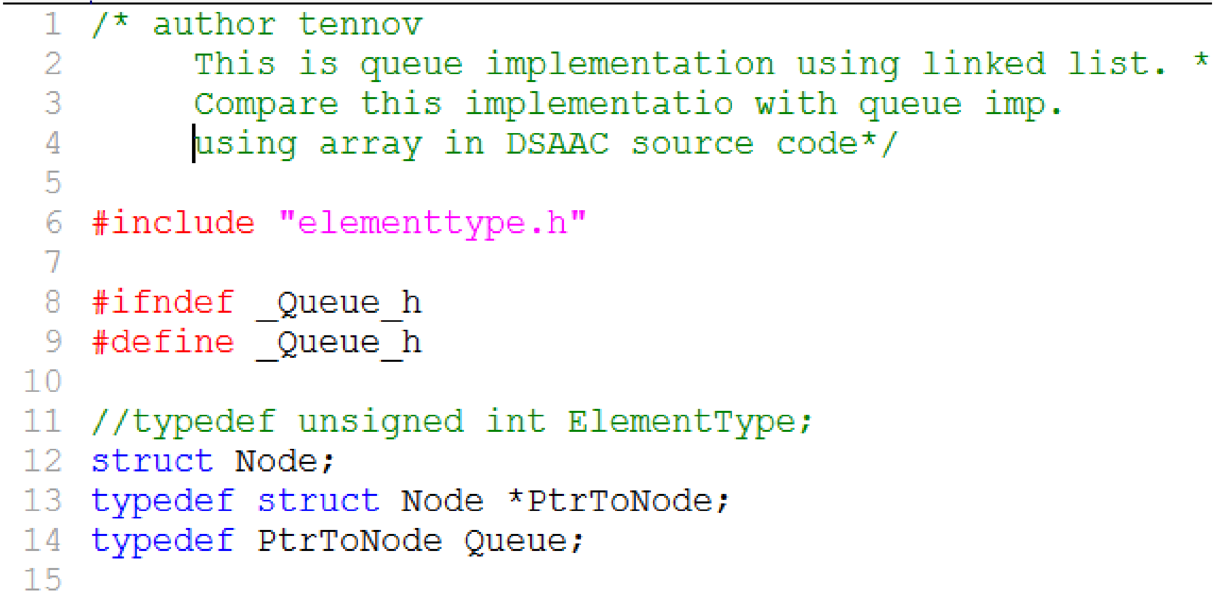
a. elementtype.h
b. queue.h
c. graph.h
d. queue.c
e. graph.c
f. klien_program.c

1. elementtype.h
   a. Buat file antar muka dengan nama elementtype.h.
   b. Edit file tersebut dan tambahkan kode program di bawah ini:

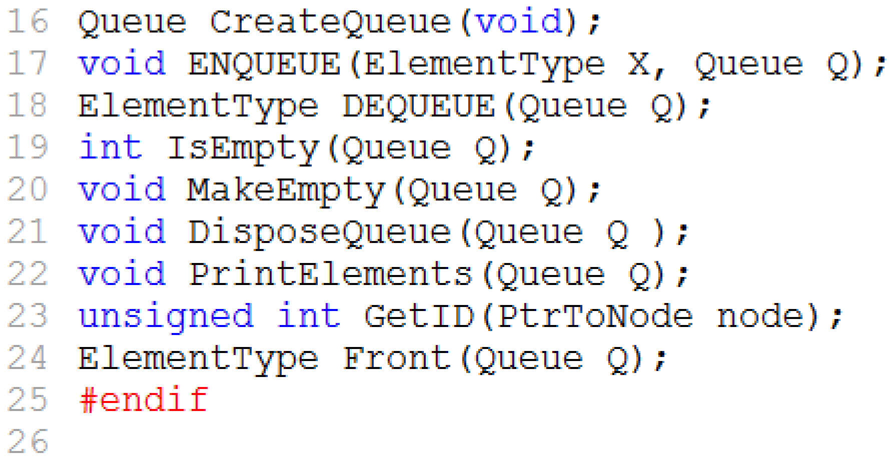


```
#ifndef _ElementType_h
#define _ElementType_h

typedef unsigned int ElementType;

#endif

```

2. queue.h
   a. Buat file antar muka dengan nama queue.h.
   b. Edit file tersebut dan tambahkan kode program di bawah ini:

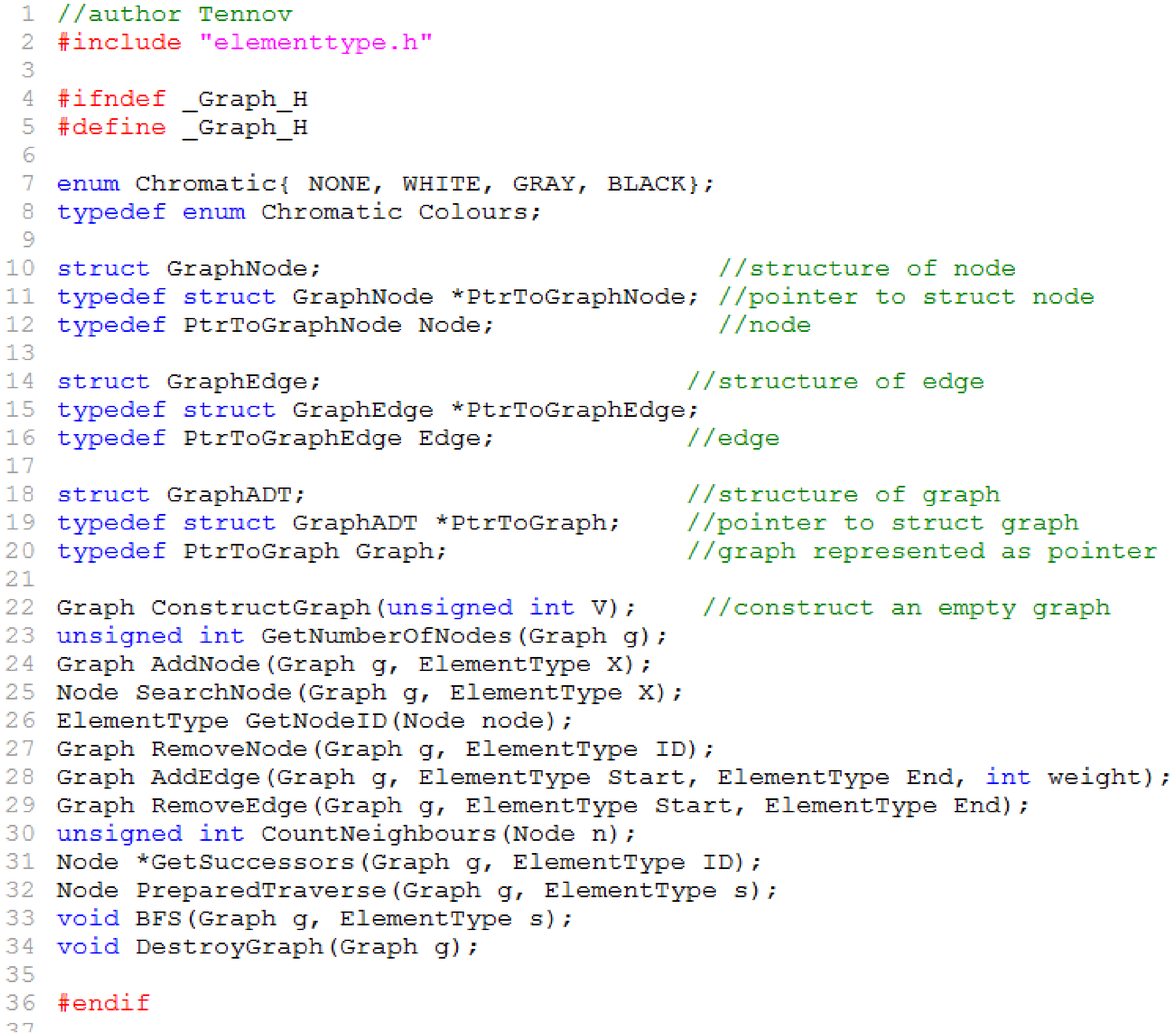


```
/* author tennov
     This is queue implementation using linked list. *
     Compare this implementatio with queue imp.
     using array in DSAAC source code*/

#include "elementtype.h"

#ifndef _Queue_h
#define _Queue_h

//typedef unsigned int ElementType;
struct Node;
typedef struct Node *PtrToNode;
typedef PtrToNode Queue;

```



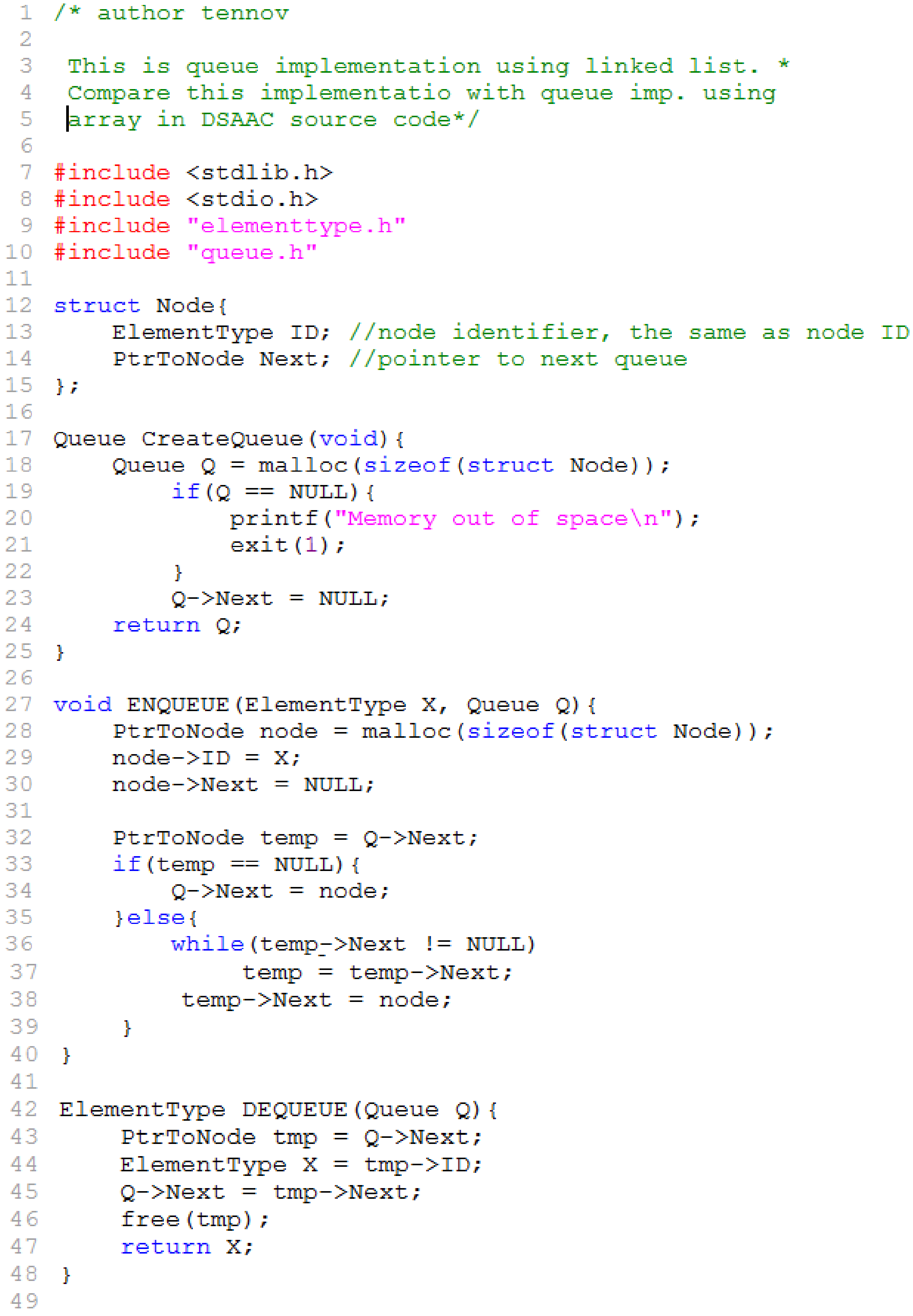
```
Queue CreateQueue(void);
void ENQUEUE(ElementType X, Queue Q);
ElementType DEQUEUE(Queue Q);
int IsEmpty(Queue Q);
void MakeEmpty(Queue Q);
void DisposeQueue(Queue Q );
void PrintElements(Queue Q);
unsigned int GetID(PtrToNode node);
ElementType Front(Queue Q);
#endif

```

3. graph.h
   a. Buat file antar muka dengan nama graph.h.
   b. Edit file tersebut dan tambahkan kode program di bawah ini:

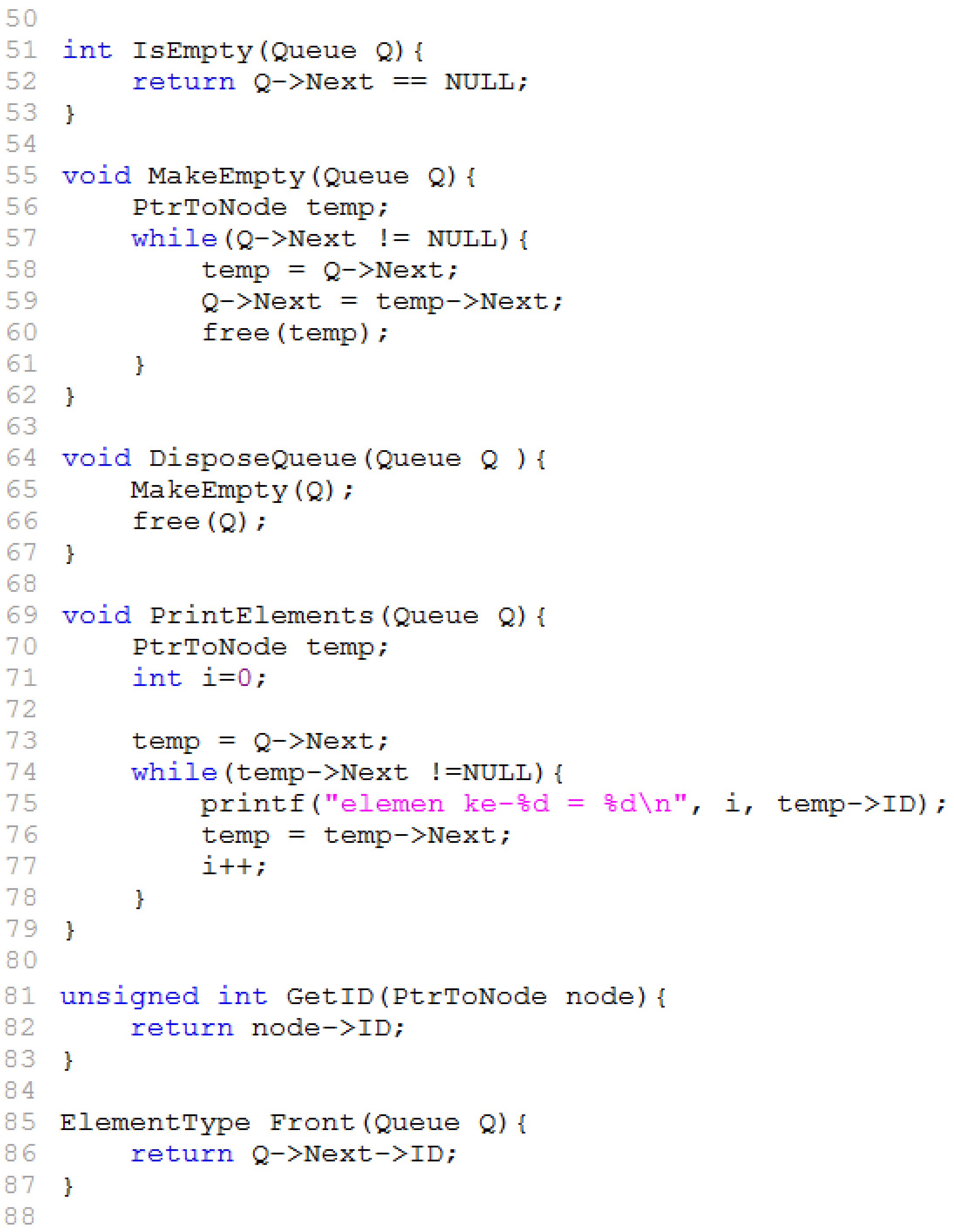


```
//author Tennov
#include "elementtype.h"

#ifndef _Graph_H
#define _Graph_H

enum Chromatic{ NONE, WHITE, GRAY, BLACK};
typedef enum Chromatic Colours;

struct GraphNode;                         //structure of node
typedef struct GraphNode *PtrToGraphNode; //pointer to struct node
typedef PtrToGraphNode Node;              //node

struct GraphEdge;                       //structure of edge
typedef struct GraphEdge *PtrToGraphEdge;
typedef PtrToGraphEdge Edge;            //edge

struct GraphADT;                        //structure of graph
typedef struct GraphADT *PtrToGraph;    //pointer to struct graph
typedef PtrToGraph Graph;               //graph represented as pointer

Graph ConstructGraph(unsigned int V);    //construct an empty graph
unsigned int GetNumberOfNodes(Graph g);
Graph AddNode(Graph g, ElementType X);
Node SearchNode(Graph g, ElementType X);
ElementType GetNodeID(Node node);
Graph RemoveNode(Graph g, ElementType ID);
Graph AddEdge(Graph g, ElementType Start, ElementType End, int weight);
Graph RemoveEdge(Graph g, ElementType Start, ElementType End);
unsigned int CountNeighbours(Node n);
Node *GetSuccessors(Graph g, ElementType ID);
Node PreparedTraverse(Graph g, ElementType s);
void BFS(Graph g, ElementType s);
void DestroyGraph(Graph g);

#endif
```

4. queue.c
   a. Buat file antar muka dengan nama queue.c.
   b. Edit file tersebut dan tambahkan kode program di bawah ini:

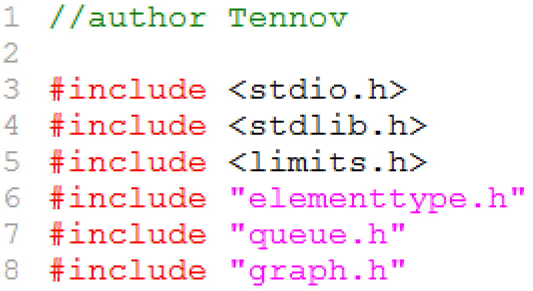


```
/* author tennov

 This is queue implementation using linked list. *
 Compare this implementatio with queue imp. using
 array in DSAAC source code*/

#include <stdlib.h>
#include <stdio.h>
#include "elementtype.h"
#include "queue.h"

struct Node{
    ElementType ID; //node identifier, the same as node ID
    PtrToNode Next; //pointer to next queue
};

Queue CreateQueue(void){
    Queue Q = malloc(sizeof(struct Node));
        if(Q == NULL){
            printf("Memory out of space\n");
            exit(1);
        }
        Q->Next = NULL;
    return Q;
}

void ENQUEUE(ElementType X, Queue Q){
    PtrToNode node = malloc(sizeof(struct Node));
    node->ID = X;
    node->Next = NULL;

    PtrToNode temp = Q->Next;
    if(temp == NULL){
        Q->Next = node;
    }else{
        while(temp->Next != NULL)
            temp = temp->Next;
        temp->Next = node;
    }
}

ElementType DEQUEUE(Queue Q){
    PtrToNode tmp = Q->Next;
    ElementType X = tmp->ID;
    Q->Next = tmp->Next;
    free(tmp);
    return X;
}
```

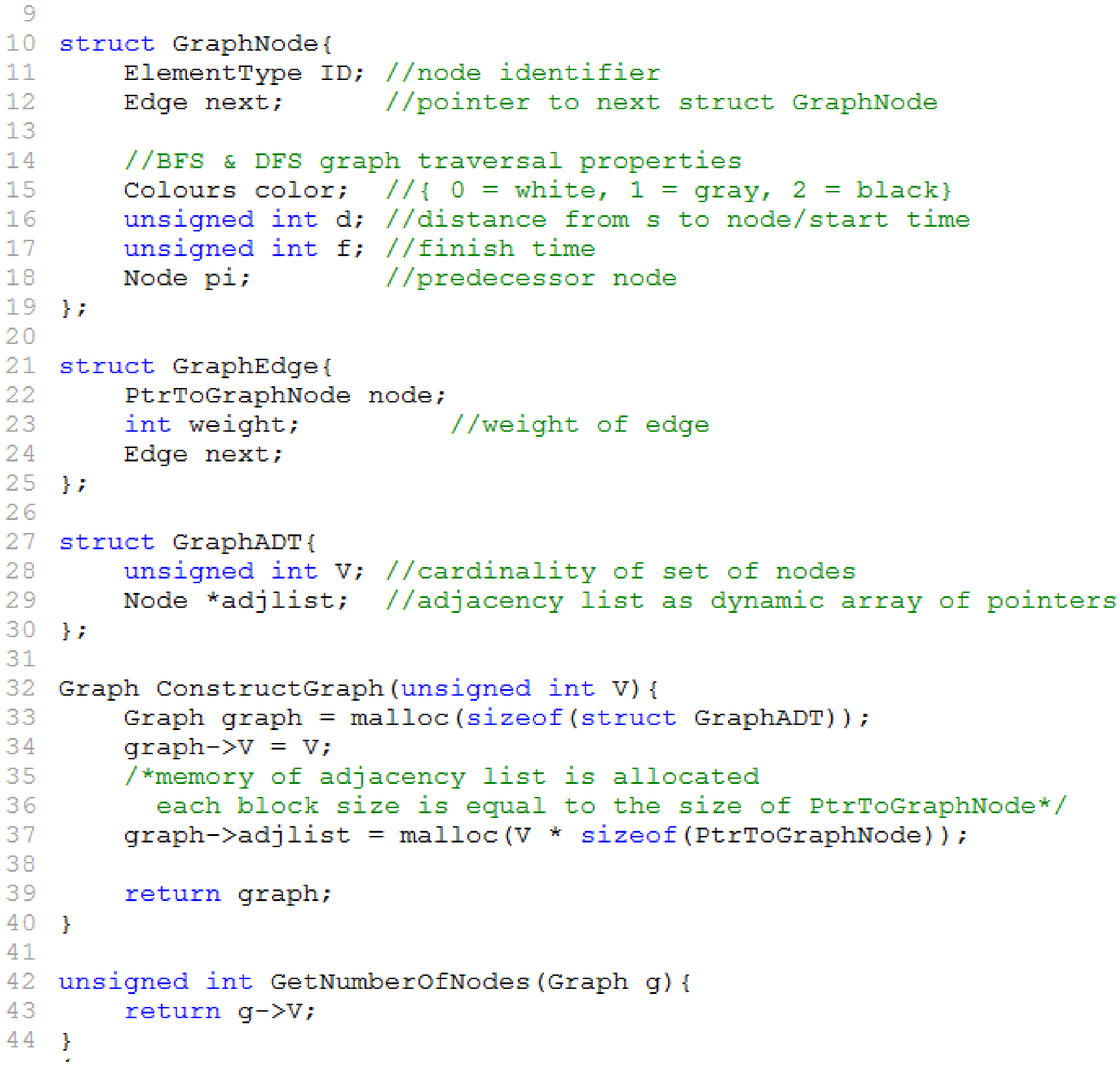


```

int IsEmpty(Queue Q){
    return Q->Next == NULL;
}

void MakeEmpty(Queue Q){
    PtrToNode temp;
    while(Q->Next != NULL){
        temp = Q->Next;
        Q->Next = temp->Next;
        free(temp);
    }
}

void DisposeQueue(Queue Q ){
    MakeEmpty(Q);
    free(Q);
}

void PrintElements(Queue Q){
    PtrToNode temp;
    int i=0;

    temp = Q->Next;
    while(temp->Next !=NULL){
        printf("elemen ke-%d = %d\n", i, temp->ID);
        temp = temp->Next;
        i++;
    }
}

unsigned int GetID(PtrToNode node){
    return node->ID;
}

ElementType Front(Queue Q){
    return Q->Next->ID;
}

```

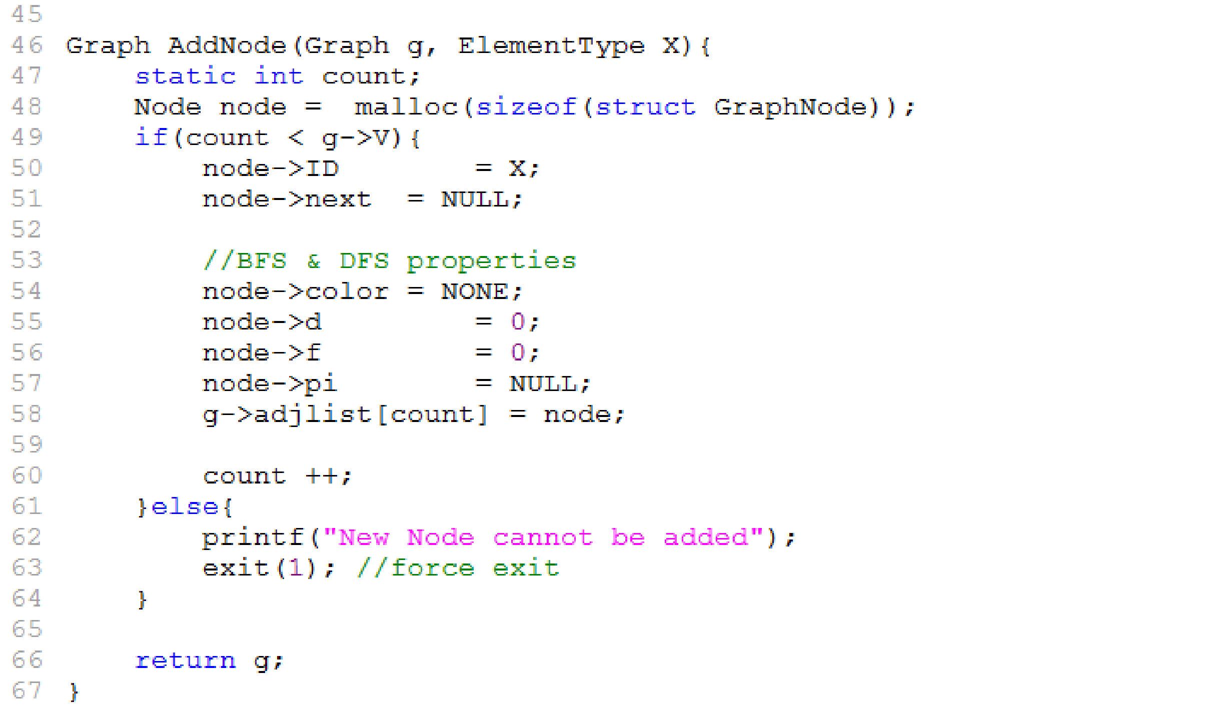
5. `graph.c`
   a. Buat file antar muka dengan nama `graph.c`.
   b. Edit file tersebut dan tambahkan kode program di bawah ini:



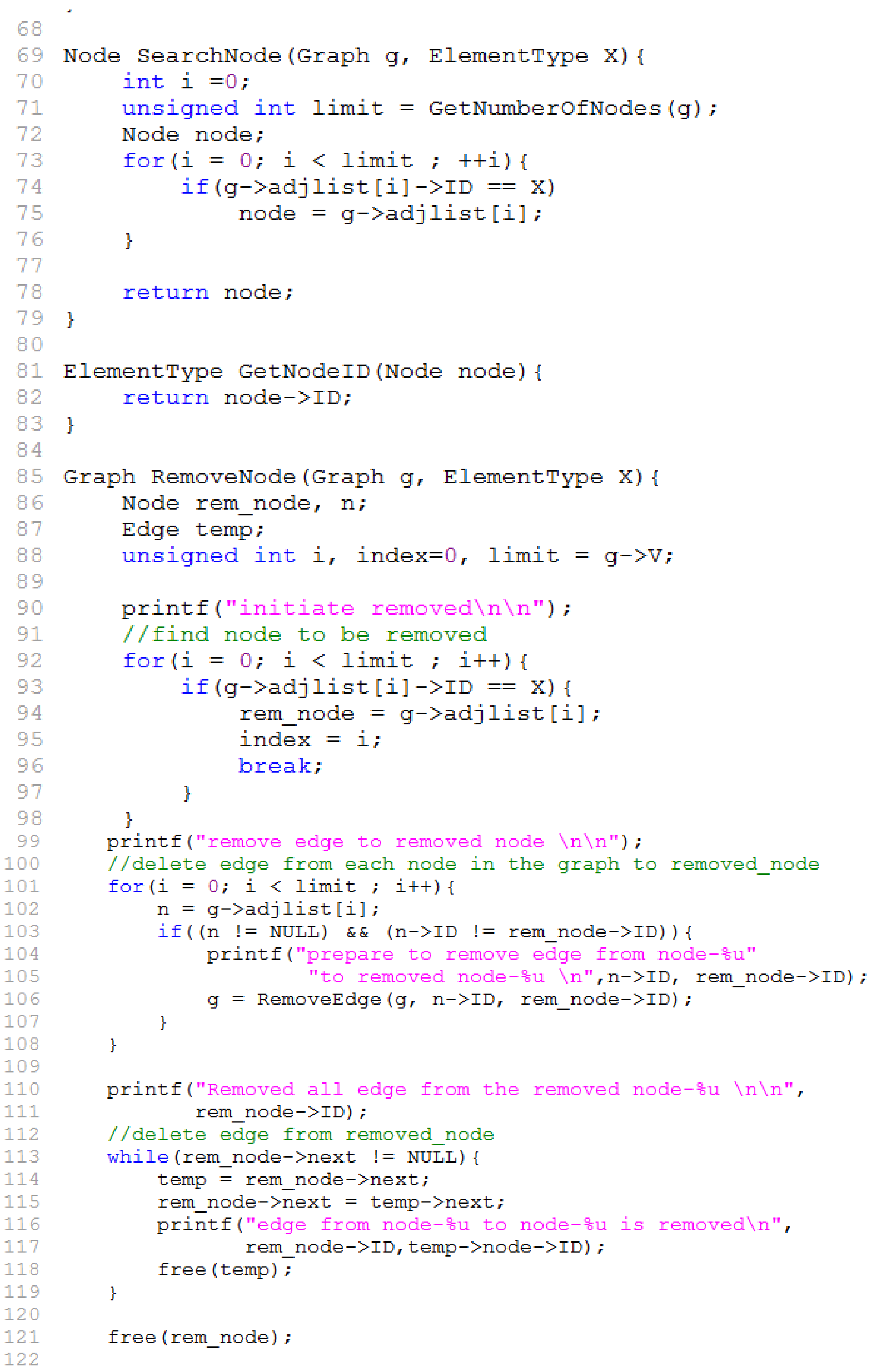
```
//author Tennov

#include <stdio.h>
#include <stdlib.h>
#include <limits.h>
#include "elementtype.h"
#include "queue.h"
#include "graph.h"
```



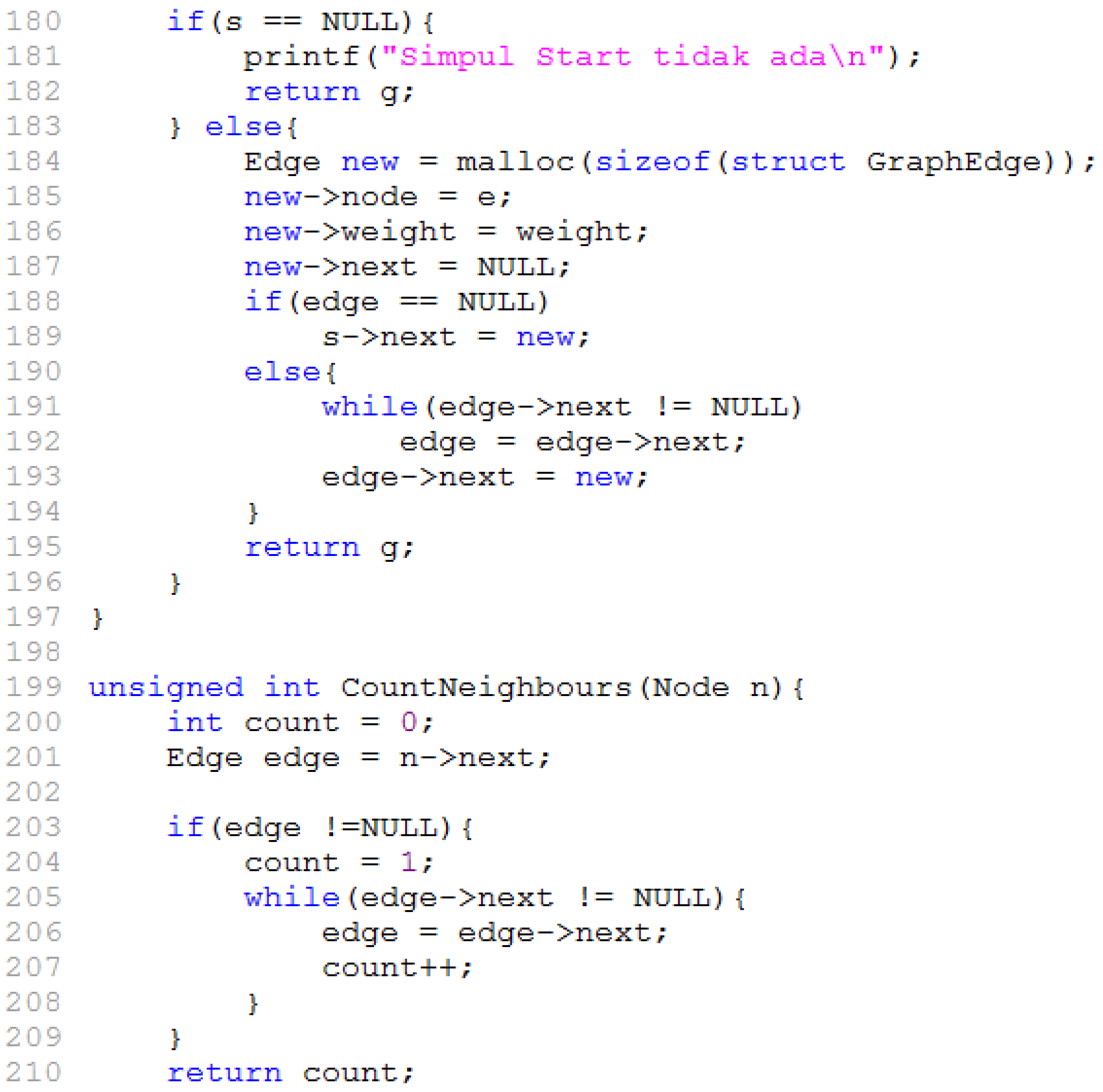
```
struct GraphNode{
    ElementType ID; //node identifier
    Edge next;      //pointer to next struct GraphNode

    //BFS & DFS graph traversal properties
    Colours color;  //{ 0 = white, 1 = gray, 2 = black}
    unsigned int d; //distance from s to node/start time
    unsigned int f; //finish time
    Node pi;        //predecessor node
};

struct GraphEdge{
    PtrToGraphNode node;
    int weight;         //weight of edge
    Edge next;
};

struct GraphADT{
    unsigned int V; //cardinality of set of nodes
    Node *adjlist;  //adjacency list as dynamic array of pointers
};

Graph ConstructGraph(unsigned int V){
    Graph graph = malloc(sizeof(struct GraphADT));
    graph->V = V;
    /*memory of adjacency list is allocated
      each block size is equal to the size of PtrToGraphNode*/
    graph->adjlist = malloc(V * sizeof(PtrToGraphNode));

    return graph;
}

unsigned int GetNumberOfNodes(Graph g){
    return g->V;
}
```



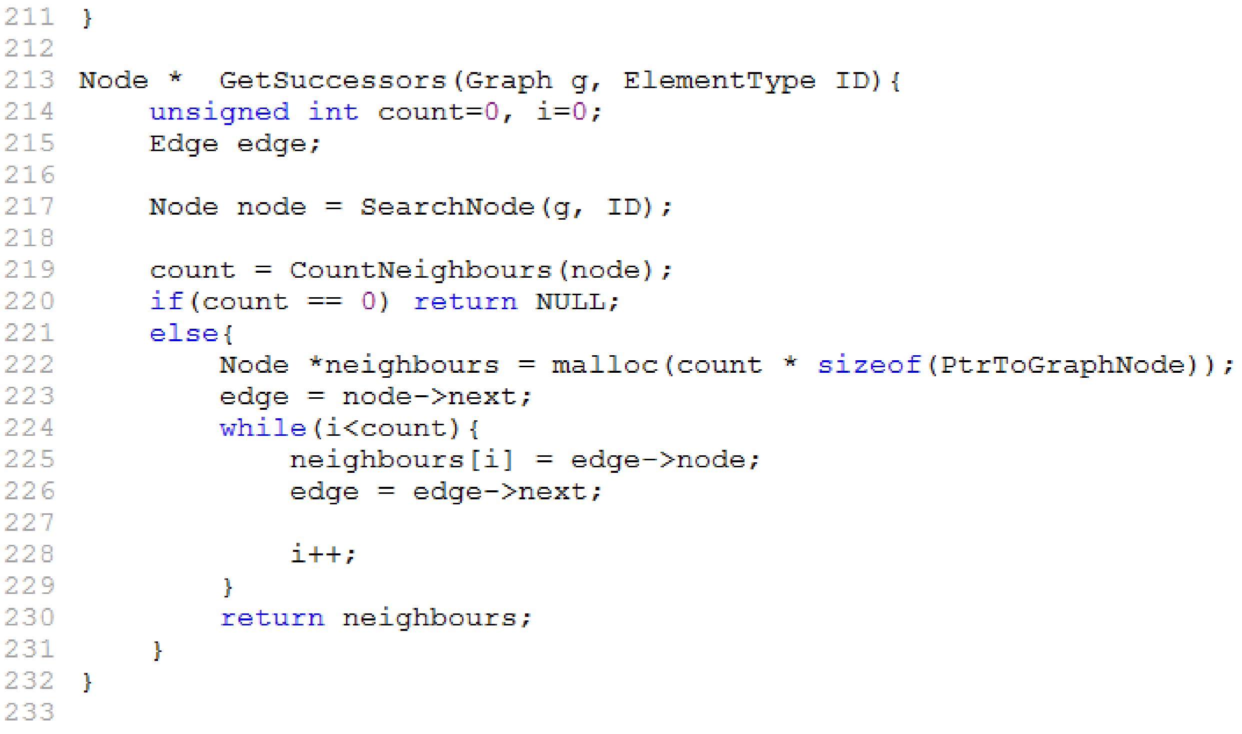
```
Graph AddNode(Graph g, ElementType X){
    static int count;
    Node node =  malloc(sizeof(struct GraphNode));
    if(count < g->V){
        node->ID        = X;
        node->next  = NULL;

        //BFS & DFS properties
        node->color = NONE;
        node->d         = 0;
        node->f         = 0;
        node->pi        = NULL;
        g->adjlist[count] = node;

        count ++;
    }else{
        printf("New Node cannot be added");
        exit(1); //force exit
    }

    return g;
}
```



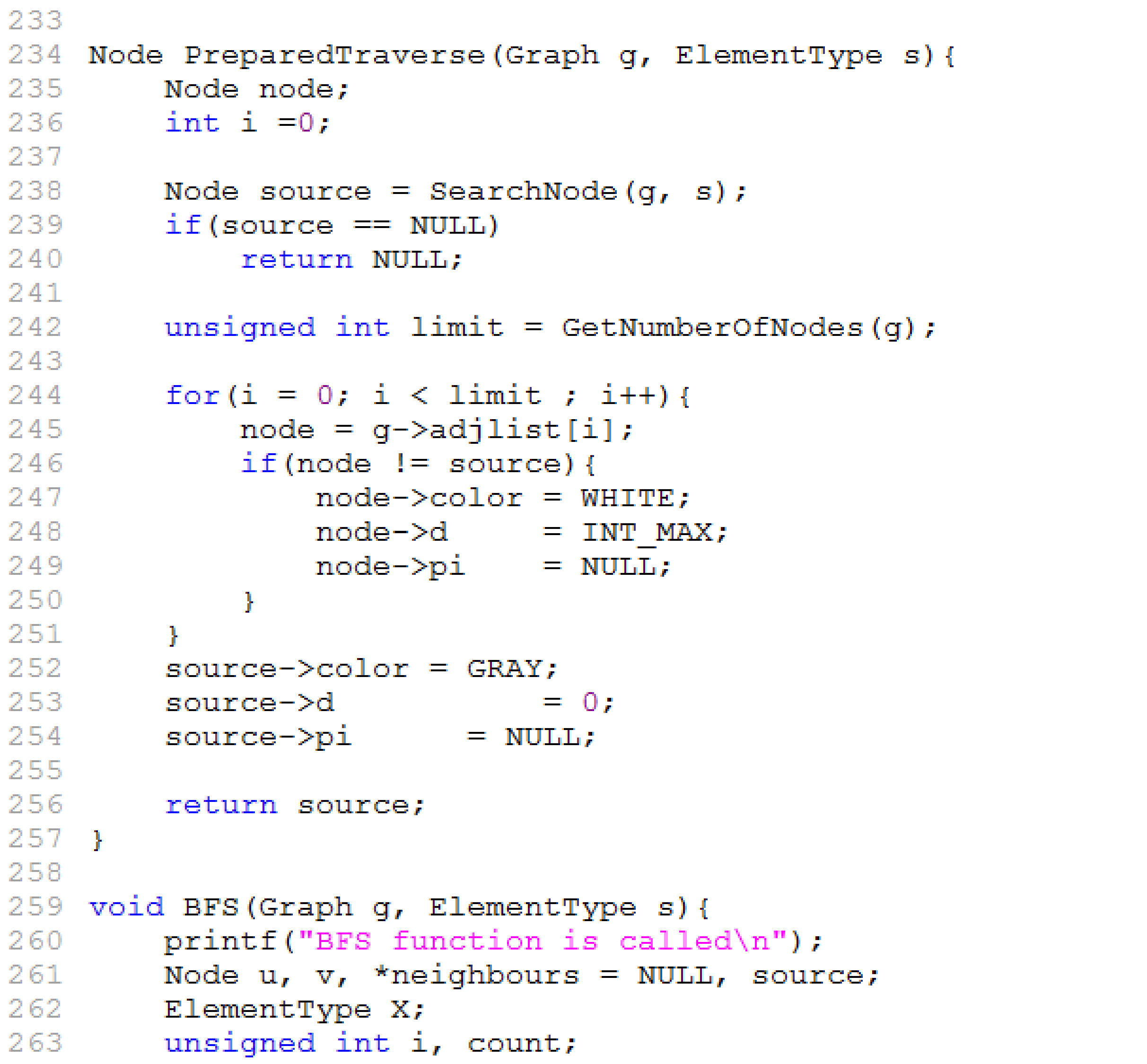
```c

Node SearchNode(Graph g, ElementType X){
    int i =0;
    unsigned int limit = GetNumberOfNodes(g);
    Node node;
    for(i = 0; i < limit ; ++i){
        if(g->adjlist[i]->ID == X)
            node = g->adjlist[i];
    }

    return node;
}

ElementType GetNodeID(Node node){
    return node->ID;
}

Graph RemoveNode(Graph g, ElementType X){
    Node rem_node, n;
    Edge temp;
    unsigned int i, index=0, limit = g->V;

    printf("initiate removed\n\n");
    //find node to be removed
    for(i = 0; i < limit ; i++){
        if(g->adjlist[i]->ID == X){
            rem_node = g->adjlist[i];
            index = i;
            break;
        }
    }
    printf("remove edge to removed node \n\n");
    //delete edge from each node in the graph to removed_node
    for(i = 0; i < limit ; i++){
        n = g->adjlist[i];
        if((n != NULL) && (n->ID != rem_node->ID)){
            printf("prepare to remove edge from node-%u"
                "to removed node-%u \n",n->ID, rem_node->ID);
            g = RemoveEdge(g, n->ID, rem_node->ID);
        }
    }

    printf("Removed all edge from the removed node-%u \n\n",
           rem_node->ID);
    //delete edge from removed_node
    while(rem_node->next != NULL){
        temp = rem_node->next;
        rem_node->next = temp->next;
        printf("edge from node-%u to node-%u is removed\n",
               rem_node->ID,temp->node->ID);
        free(temp);
    }

    free(rem_node);

    /* The adjacency list sliding as consequence of array
       imp for it.
       This can be avoided by implementing adjacency list
       as linked list. */
    if(index != (limit - 1)) {
        for(i = index + 1; i < limit; i++)
            g->adjlist[i-1] = g->adjlist[i];
        g->adjlist[limit-1] = NULL;
    }

    g->V = g->V - 1;
    i = 0;
    while(g->adjlist[i] != NULL){
        printf("adjlist[%u] = node-%u\n", i, g->adjlist[i]->ID);
        i++;
    }
    return g;
}

Graph RemoveEdge(Graph g, ElementType Start, ElementType End){
    Node source = SearchNode(g, Start);
    printf("Find start node = node-%u\n", source->ID);
    Edge temp, e;

    temp = source->next;
    if(temp == NULL)
        return g;
    else if(temp->node->ID == End){
        source->next = temp->next;
        printf("Edge from node-%u  to node-%u is removed\n",
               source->ID, temp->node->ID);
        free(temp);
        return g;
    }else if((temp->node->ID != End) && (temp->next != NULL)){
        while((temp->next != NULL) &&
              (temp->next->node->ID != End))
            temp = temp->next;
        if(temp->next == NULL)
            return g;
        else{
            e = temp->next;
            temp->next = e->next;
            printf("Edge from node-%u  to node-%u is removed\n",
                  source->ID, e->node->ID);
            free(e);
            return g;
        }
    }else
        return g;
}

Graph AddEdge(Graph g, ElementType Start,
              ElementType End, int weight){
    Node s = SearchNode(g, Start);
    Node e = SearchNode(g, End);
    Edge edge = s->next;

```



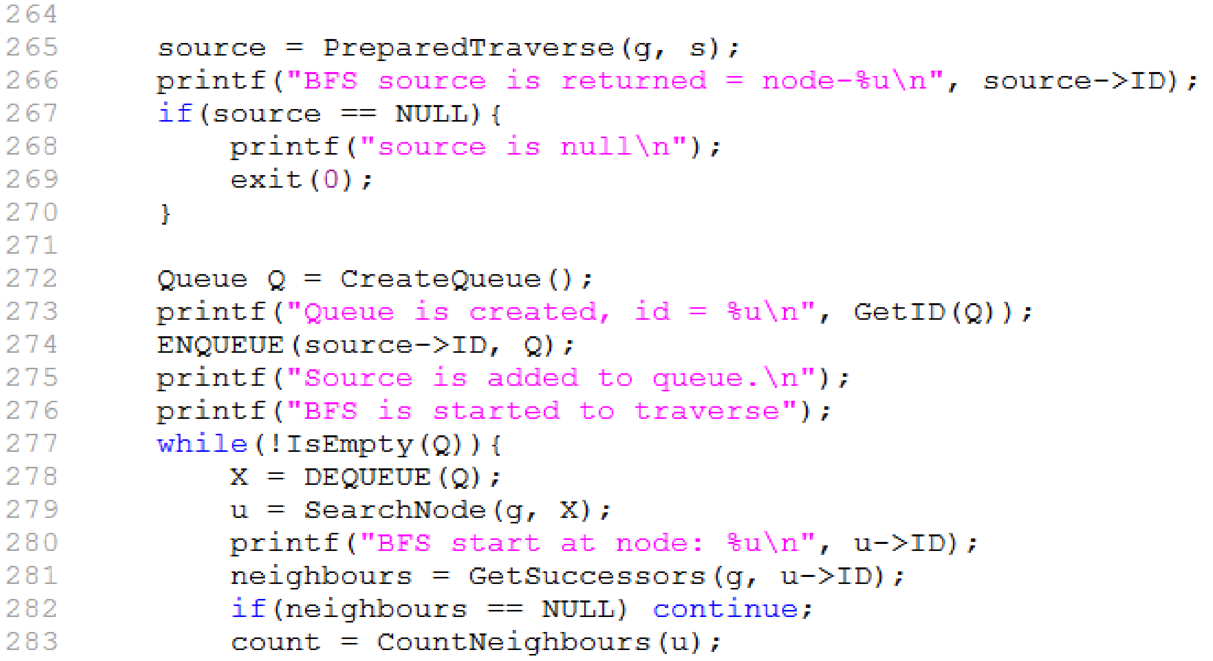
```
    if(s == NULL){
        printf("Simpul Start tidak ada\n");
        return g;
    } else{
        Edge new = malloc(sizeof(struct GraphEdge));
        new->node = e;
        new->weight = weight;
        new->next = NULL;
        if(edge == NULL)
            s->next = new;
        else{
            while(edge->next != NULL)
                edge = edge->next;
            edge->next = new;
        }
        return g;
    }
}

unsigned int CountNeighbours(Node n){
    int count = 0;
    Edge edge = n->next;

    if(edge !=NULL){
        count = 1;
        while(edge->next != NULL){
            edge = edge->next;
            count++;
        }
    }
    return count;
```



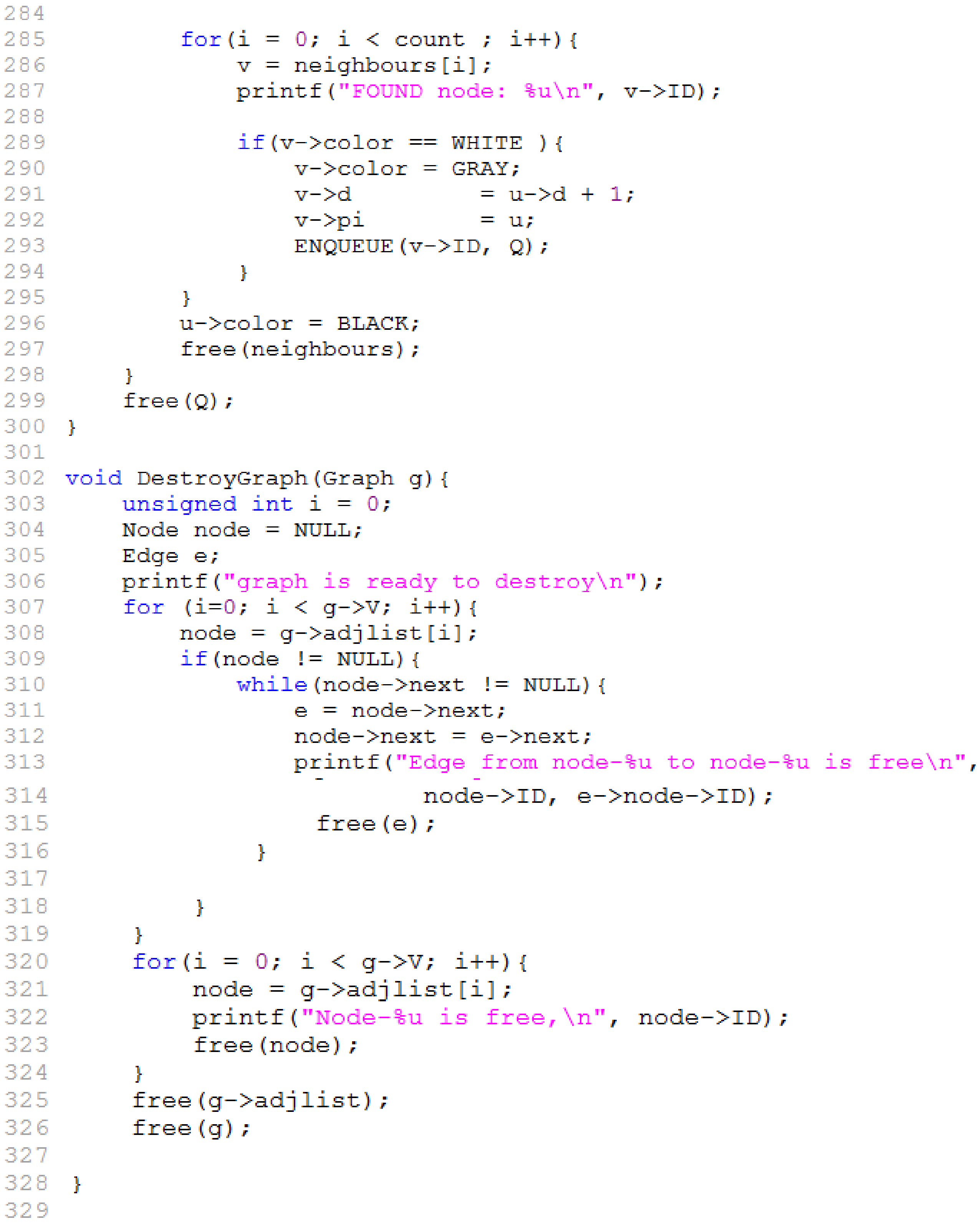
```
}

Node *  GetSuccessors(Graph g, ElementType ID){
    unsigned int count=0, i=0;
    Edge edge;

    Node node = SearchNode(g, ID);

    count = CountNeighbours(node);
    if(count == 0) return NULL;
    else{
        Node *neighbours = malloc(count * sizeof(PtrToGraphNode));
        edge = node->next;
        while(i<count){
            neighbours[i] = edge->node;
            edge = edge->next;

            i++;
        }
        return neighbours;
    }
}

```



```
Node PreparedTraverse(Graph g, ElementType s){
    Node node;
    int i =0;

    Node source = SearchNode(g, s);
    if(source == NULL)
        return NULL;

    unsigned int limit = GetNumberOfNodes(g);

    for(i = 0; i < limit ; i++){
        node = g->adjlist[i];
        if(node != source){
            node->color = WHITE;
            node->d      = INT_MAX;
            node->pi     = NULL;
        }
    }
    source->color = GRAY;
    source->d           = 0;
    source->pi      = NULL;

    return source;
}

void BFS(Graph g, ElementType s){
    printf("BFS function is called\n");
    Node u, v, *neighbours = NULL, source;
    ElementType X;
    unsigned int i, count;
```

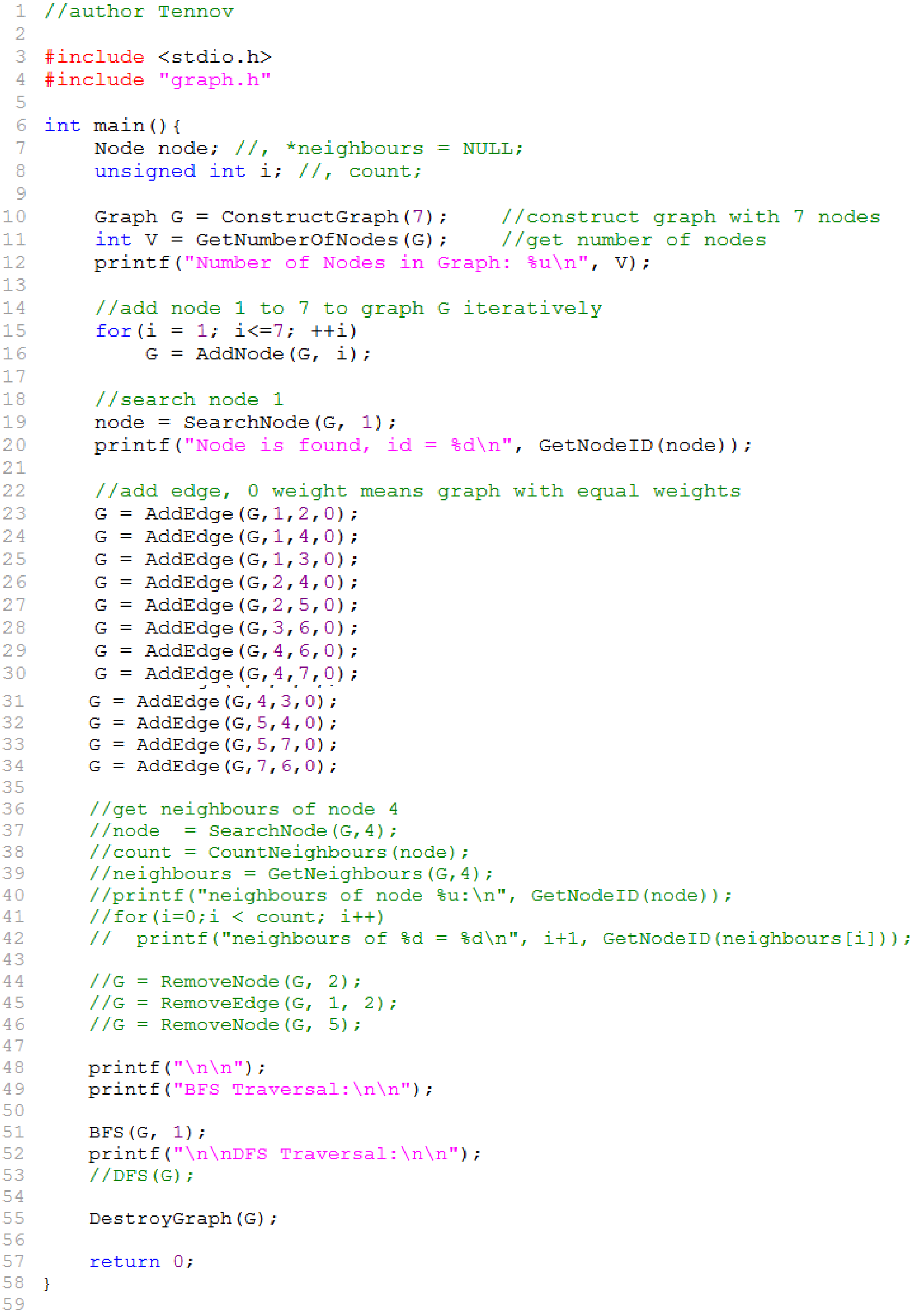


```
    source = PreparedTraverse(g, s);
    printf("BFS source is returned = node-%u\n", source->ID);
    if(source == NULL){
        printf("source is null\n");
        exit(0);
    }

    Queue Q = CreateQueue();
    printf("Queue is created, id = %u\n", GetID(Q));
    ENQUEUE(source->ID, Q);
    printf("Source is added to queue.\n");
    printf("BFS is started to traverse");
    while(!IsEmpty(Q)){
        X = DEQUEUE(Q);
        u = SearchNode(g, X);
        printf("BFS start at node: %u\n", u->ID);
        neighbours = GetSuccessors(g, u->ID);
        if(neighbours == NULL) continue;
        count = CountNeighbours(u);
```



```

        for(i = 0; i < count ; i++){
            v = neighbours[i];
            printf("FOUND node: %u\n", v->ID);

            if(v->color == WHITE ){
                v->color = GRAY;
                v->d          = u->d + 1;
                v->pi         = u;
                ENQUEUE(v->ID, Q);
            }
        }
        u->color = BLACK;
        free(neighbours);
    }
    free(Q);
}

void DestroyGraph(Graph g){
    unsigned int i = 0;
    Node node = NULL;
    Edge e;
    printf("graph is ready to destroy\n");
    for (i=0; i < g->V; i++){
        node = g->adjlist[i];
        if(node != NULL){
            while(node->next != NULL){
                e = node->next;
                node->next = e->next;
                printf("Edge from node-%u to node-%u is free\n",
                         node->ID, e->node->ID);
                free(e);
            }

        }
    }
    for(i = 0; i < g->V; i++){
        node = g->adjlist[i];
        printf("Node-%u is free,\n", node->ID);
        free(node);
    }
    free(g->adjlist);
    free(g);

}

```

6. klien_graph.c
   a. Buat file antar muka dengan nama klien_graph.c.
   b. Edit file tersebut dan tambahkan kode program pada halaman berikut:



```c
//author Tennov

#include <stdio.h>
#include "graph.h"

int main(){
    Node node; //, *neighbours = NULL;
    unsigned int i; //, count;

    Graph G = ConstructGraph(7);     //construct graph with 7 nodes
    int V = GetNumberOfNodes(G);     //get number of nodes
    printf("Number of Nodes in Graph: %u\n", V);

    //add node 1 to 7 to graph G iteratively
    for(i = 1; i<=7; ++i)
        G = AddNode(G, i);

    //search node 1
    node = SearchNode(G, 1);
    printf("Node is found, id = %d\n", GetNodeID(node));

    //add edge, 0 weight means graph with equal weights
    G = AddEdge(G,1,2,0);
    G = AddEdge(G,1,4,0);
    G = AddEdge(G,1,3,0);
    G = AddEdge(G,2,4,0);
    G = AddEdge(G,2,5,0);
    G = AddEdge(G,3,6,0);
    G = AddEdge(G,4,6,0);
    G = AddEdge(G,4,7,0);
    G = AddEdge(G,4,3,0);
    G = AddEdge(G,5,4,0);
    G = AddEdge(G,5,7,0);
    G = AddEdge(G,7,6,0);

    //get neighbours of node 4
    //node  = SearchNode(G,4);
    //count = CountNeighbours(node);
    //neighbours = GetNeighbours(G,4);
    //printf("neighbours of node %u:\n", GetNodeID(node));
    //for(i=0;i < count; i++)
    //  printf("neighbours of %d = %d\n", i+1, GetNodeID(neighbours[i]));

    //G = RemoveNode(G, 2);
    //G = RemoveEdge(G, 1, 2);
    //G = RemoveNode(G, 5);

    printf("\n\n");
    printf("BFS Traversal:\n\n");

    BFS(G, 1);
    printf("\n\nDFS Traversal:\n\n");
    //DFS(G);

    DestroyGraph(G);

    return 0;
}
```

7. TUGAS:
   a. Coba anda analisis implementasi fungsi `ENQUEUE` dan `DEQUEUE` pada queue.c. Apakah ada implementasi yang salah? Jika ada, jelaskan kesalahan yang terjadi.
   b. Hitunglah kompleksitas implementasi algorithma `BFS` pada `graph.c`.

Setoran

1. Lembar jawaban dikumpulkan kepada TA untuk jawaban pertanyaan/tugas pada bagian Ulasan dan Pendalaman.
2. Kode program untuk pengurutan `Graph ADT termasuk traversal BFS`
   - `elementtype.h`
   - `queue.h`
   - `graph.h`
   - `queue.c`
   - `graph.c`
   - `klien_graph.c`