Data Structures

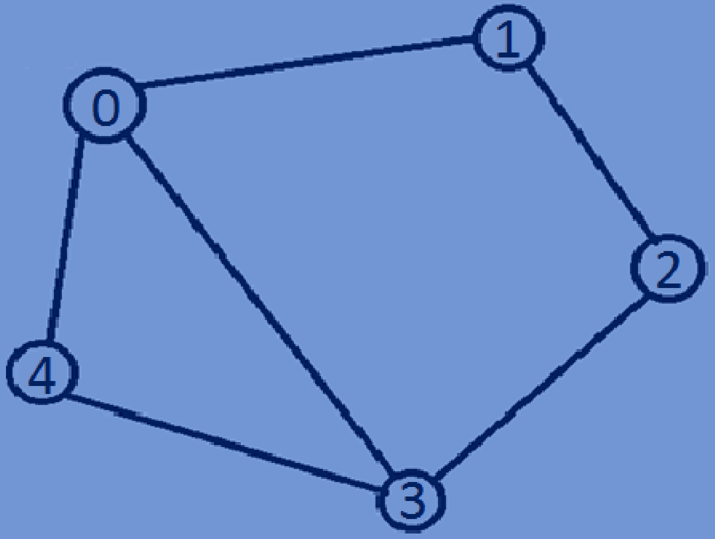
Finding Degree of each vertex of a Graph

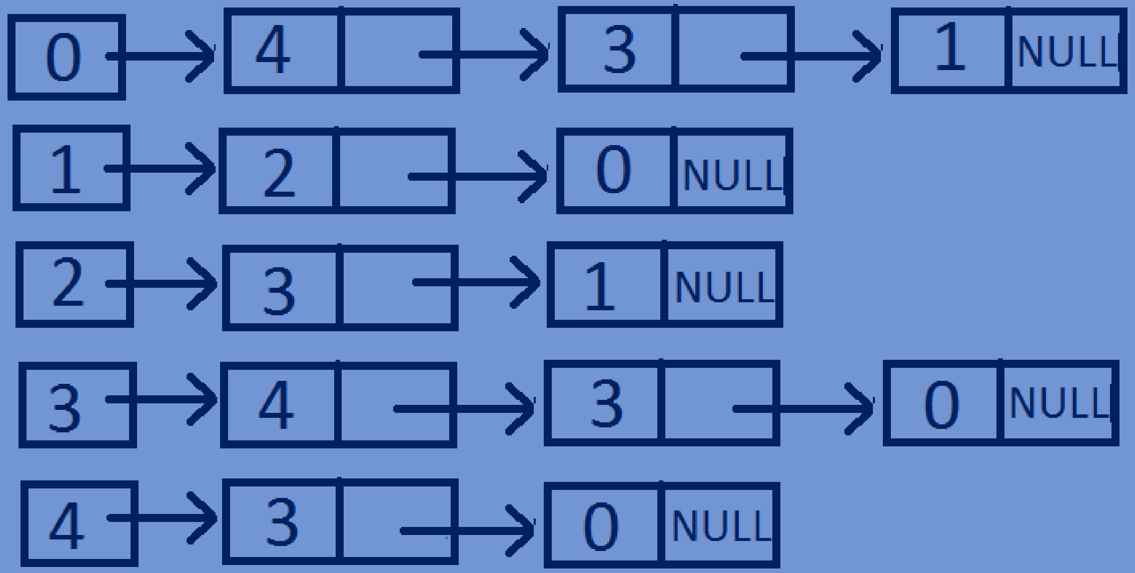
Adjacency List

Degree of a vertex of undirected graph:

Degree of a Vertex: Total no of edges connected to a vertex is called as Degree of That vertex.

Degree(0)=3
Degree(1)=2
Degree(2)=2
Degree(3)=3
Degree(4)=2





- In case of an undirected graph,the degree of a vertex is equal to the total no of Nodes in the adjacency list corresponding to that vertex.

Function for finding the degree of each vertex in an Undirected graph:

```
void FindDegree(Graph *graph){
    for(int i=0;i<graph->totVertices;i++){
        Node *head=graph->array[i].Head;
        int degree=0;
        while(head){
            degree++;
            head=head->next;
        }
        printf("degree of vertex %d is:%d\n",i,degree);
    }
}
```

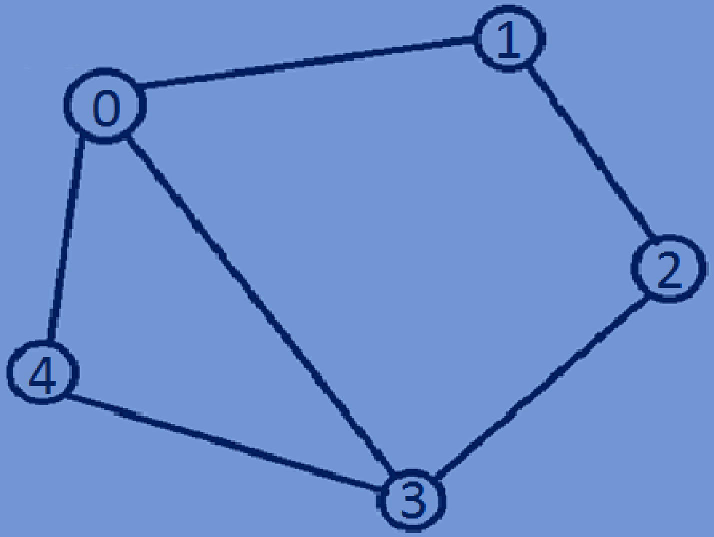
Degree(0)=3

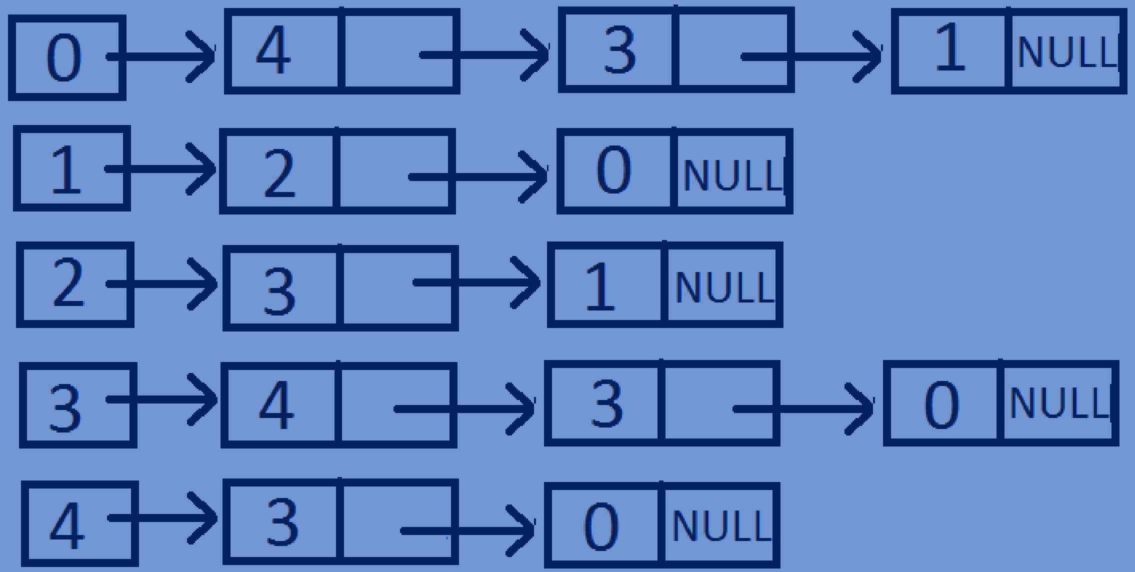
Degree(1)=2

Degree(2)=2

Degree(3)=3

Degree(4)=2





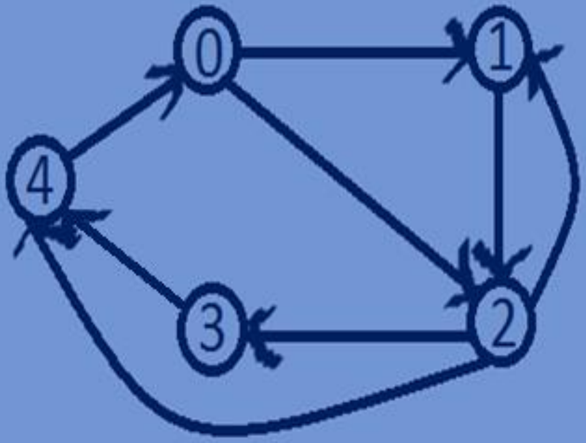
Finding indegree and outdegree of a directed graph:

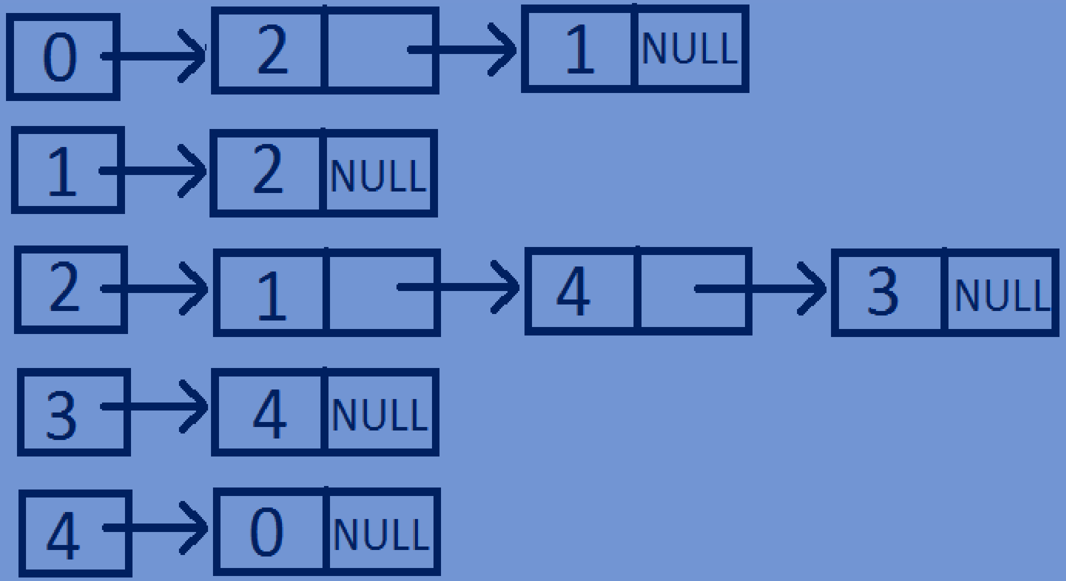
- For a directed graph each vertex has indegree and outdegree.

Indegree: The total no of incoming edges of a vertex is called as indegree of the vertex.

Outdegree: The total no of Outgoing edges of a vertex is called as outdegree of the vertex.

Indegree(0)=1,Outdegree(0)=2
Indegree(1)=2,Outdegree(1)=1
Indegree(2)=2,Outdegree(2)=3
Indegree(3)=1,Outdegree(3)=1
Indegree(4)=2,Outdegree(4)=1



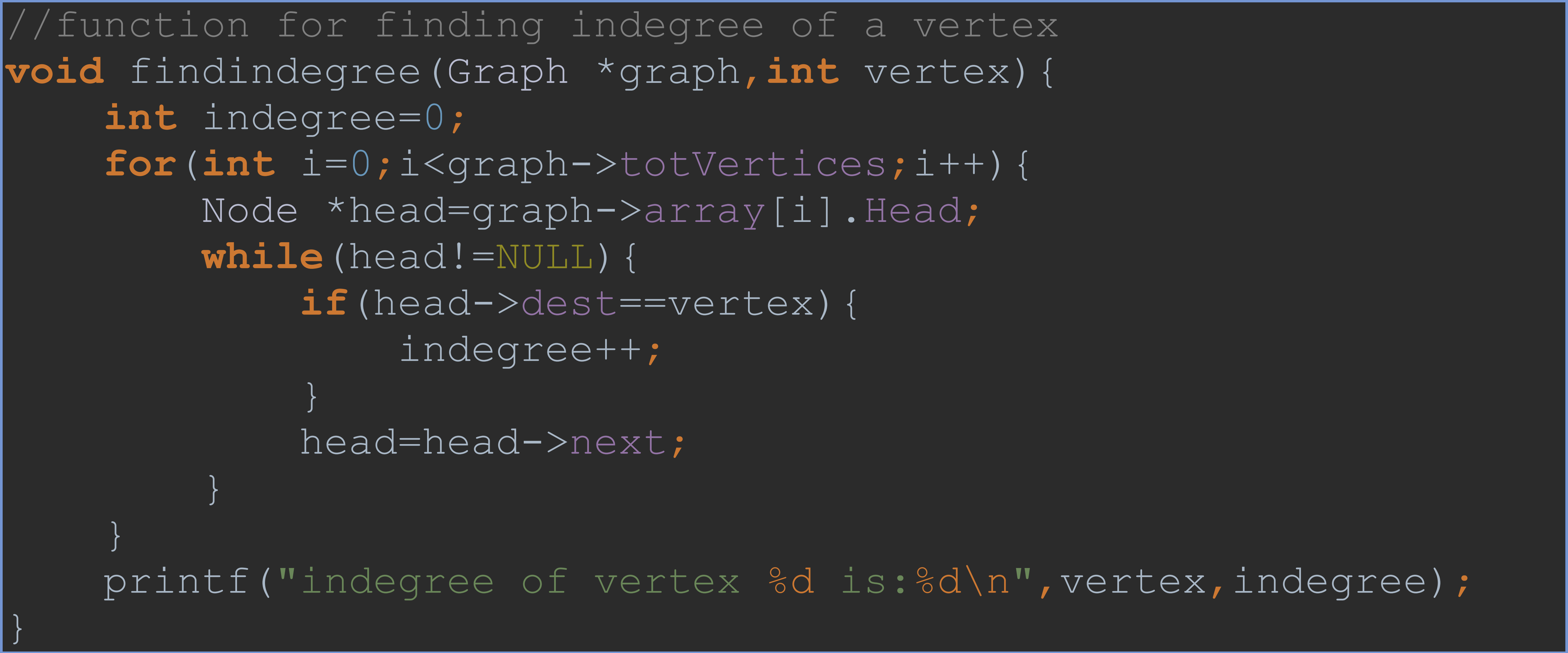


- In case of directed graph outdegree of a vertex is equal to the total no of Nodes in the adjacency list corresponding to that vertex and indegree of a vertex is Equal to the total no of occurrences of that vertex in the adjacency list of all the vertices.

Function for finding outdegree of each vertex of a directed graph :

```c
//function for finding outdegree of a vertex
void findOutDegree(Graph *graph){
    for(int i=0;i<graph->totVertices;i++){
        Node *head=graph->array[i].Head;
        int degree=0;
        while(head){
            degree++;
            head=head->next;
        }
        printf("outdegree of vertex %d is:%d\n",i,degree);
    }
}
```

Function for finding indegree of each vertex of a directed graph:

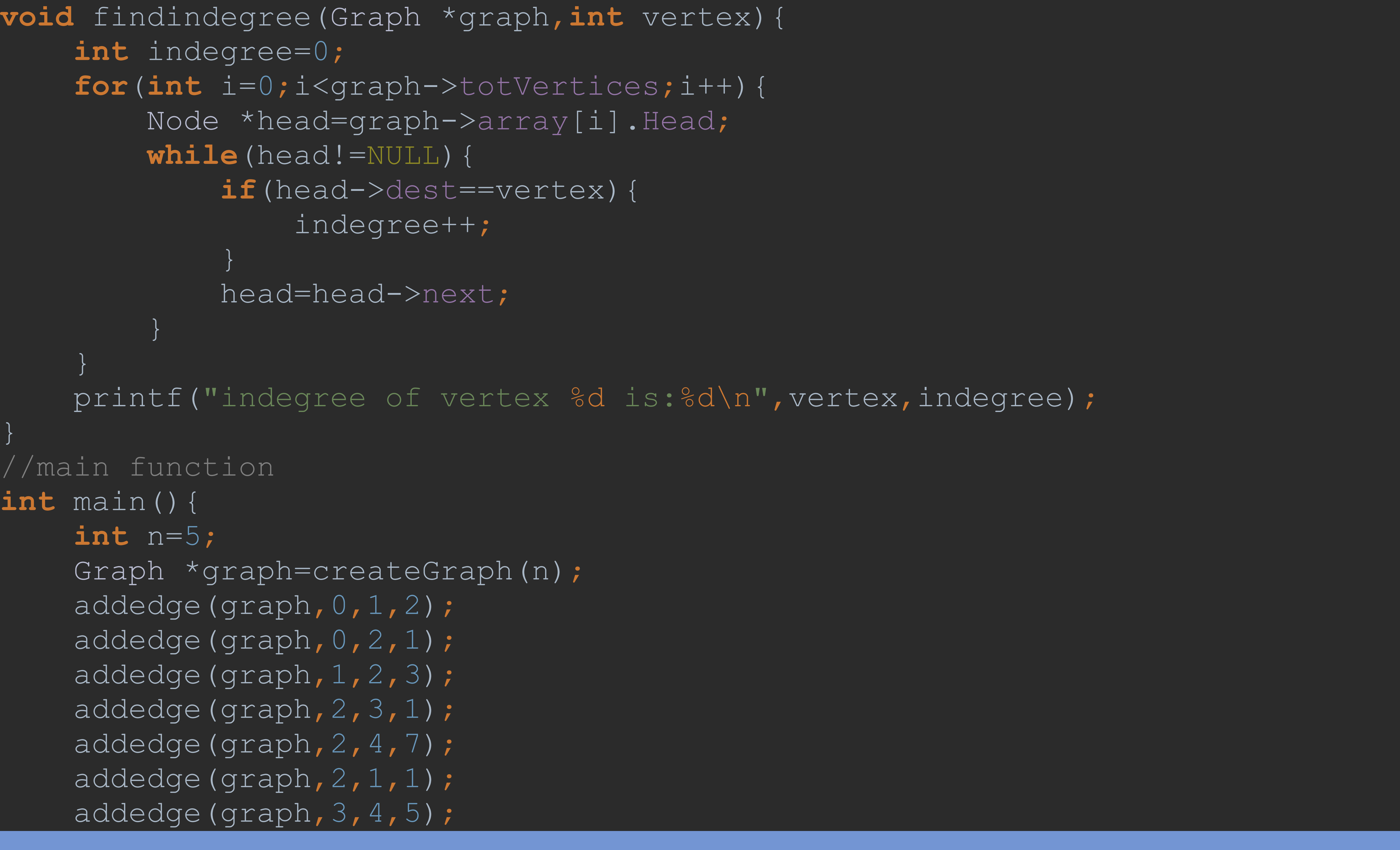


```c
//function for finding indegree of a vertex
void findindegree(Graph *graph,int vertex){
    int indegree=0;
    for(int i=0;i<graph->totVertices;i++){
        Node *head=graph->array[i].Head;
        while(head!=NULL){
            if(head->dest==vertex){
                indegree++;
            }
            head=head->next;
        }
    }
    printf("indegree of vertex %d is:%d\n",vertex,indegree);
}
```

Whole program:

```
#include<stdio.h>
#include<stdlib.h>
//Structure for representing a NODE in the Adjacency List
typedef struct Node{
    int dest;
    int weight;
    struct Node *next;
}Node;
//structure for representing an adjacency liat
typedef struct List{
    Node *Head;
}List;
// A structure to represent a graph - here graph is an array of Adjacency
lists
// size of the array will be  equal to the number of vertices in graph
typedef struct Graph{
    int totVertices;
    List *array;
}Graph;
//function To create a new node in the adjacency list
Node *createNewNode(int dest,int weight){
    Node *newnode=(Node*)malloc(sizeof(Node));
    newnode->dest=dest;
    newnode->weight=weight;
    newnode->next=NULL;
    return newnode;
}
//Function To creates a graph of n vertices
Graph *createGraph(int n){
    Graph *graph=(Graph*)malloc(sizeof(Graph));
    graph->totVertices=n;
    graph->array=(List*)malloc(n*sizeof(List));
    //Initialise each adjacency list as empty by making head as NULL
    for(int i=0;i<n;i++){
        graph->array[i].Head=NULL;
    }
    return graph;
}
//function for Adding an edge to a directed graph
void addedge(Graph *graph,int src,int dest,int weight){
    Node *newnode=createNewNode(dest,weight);
    newnode->next=graph->array[src].Head;
    graph->array[src].Head=newnode;
}
//Function for printing Adjacency list corresponding to each vertex
```

```
void printGraph(Graph *graph){
    for(int i=0;i<graph->totVertices;i++){
        Node *Headnode=graph->array[i].Head;
        printf("connected vertices of vertex %d are:head",i);
        while(Headnode){
            printf("->%d",Headnode->dest);
            Headnode=Headnode->next;
        }
        printf("\n");
    }
}
//function for finding outdegree of a vertex
void findOutDegree(Graph *graph){
    for(int i=0;i<graph->totVertices;i++){
        Node *head=graph->array[i].Head;
        int degree=0;
        while(head){
            degree++;
            head=head->next;
        }
        printf("outdegree of vertex %d is:%d\n",i,degree);
    }
}
//function for finding indegree of a vertex
```



```c
void findindegree(Graph *graph,int vertex){
    int indegree=0;
    for(int i=0;i<graph->totVertices;i++){
        Node *head=graph->array[i].Head;
        while(head!=NULL){
            if(head->dest==vertex){
                indegree++;
            }
            head=head->next;
        }
    }
    printf("indegree of vertex %d is:%d\n",vertex,indegree);
}
//main function
int main(){
    int n=5;
    Graph *graph=createGraph(n);
    addedge(graph,0,1,2);
    addedge(graph,0,2,1);
    addedge(graph,1,2,3);
    addedge(graph,2,3,1);
    addedge(graph,2,4,7);
    addedge(graph,2,1,1);
    addedge(graph,3,4,5);
    addedge(graph,4,0,4);
    printGraph(graph);
    findOutDegree(graph);
    findindegree(graph,2);
    findindegree(graph,0);
    return 0;
}
```

Output:

```
connected vertices of vertex 0 are:head->2->1
connected vertices of vertex 1 are:head->2
connected vertices of vertex 2 are:head->1->4->3
connected vertices of vertex 3 are:head->4
connected vertices of vertex 4 are:head->0
outdegree of vertex 0 is:2
outdegree of vertex 1 is:1
outdegree of vertex 2 is:3
outdegree of vertex 3 is:1
outdegree of vertex 4 is:1
indegree of vertex 2 is:2
indegree of vertex 0 is:1
```