9) Design and implement C/C++ Program to obtain the Topological ordering of vertices in a given digraph.

#include<stdio.h>

#include<stdlib.h>

#define MAX\_VERTICES 100

// Structure for representing a node in the adjacency list

struct Node {

int vertex;

struct Node\* next;

};

// Structure for representing the graph

struct Graph {

int numVertices;

struct Node\* adjList[MAX\_VERTICES];

};

// Function to create a new node

struct Node\* createNode(int v) {

struct Node\* newNode = (struct Node\*)malloc(sizeof(struct Node));

newNode->vertex = v;

newNode->next = NULL;

return newNode;

}

// Function to create a graph with a given number of vertices

struct Graph\* createGraph(int vertices) {

struct Graph\* graph = (struct Graph\*)malloc(sizeof(struct Graph));

graph->numVertices = vertices;

for (int i = 0; i < vertices; ++i)

graph->adjList[i] = NULL;

return graph;

}

// Function to add an edge to the graph

void addEdge(struct Graph\* graph, int src, int dest) {

struct Node\* newNode = createNode(dest);

newNode->next = graph->adjList[src];

graph->adjList[src] = newNode;

}

// Function to obtain the topological ordering of vertices

void topologicalSort(struct Graph\* graph) {

// Array to store in-degree of vertices

int indegree[MAX\_VERTICES] = {0};

// Calculate in-degree of each vertex

for (int v = 0; v < graph->numVertices; ++v) {

struct Node\* temp = graph->adjList[v];

while (temp != NULL) {

indegree[temp->vertex]++;

temp = temp->next;

}

}

// Queue for vertices with in-degree 0

int queue[MAX\_VERTICES];

int front = 0, rear = -1;

// Enqueue vertices with in-degree 0

for (int i = 0; i < graph->numVertices; ++i) {

if (indegree[i] == 0)

queue[++rear] = i;

}

// Initialize count of visited vertices

int count = 0;

// Array to store topological order

int topOrder[MAX\_VERTICES];

// Perform topological sorting

while (front <= rear) {

// Extract front of queue and add it to topological order

int u = queue[front++];

topOrder[count++] = u;

// Decrease in-degree of adjacent vertices

struct Node\* temp = graph->adjList[u];

while (temp != NULL) {

indegree[temp->vertex]--;

// If in-degree becomes 0, enqueue the vertex

if (indegree[temp->vertex] == 0)

queue[++rear] = temp->vertex;

temp = temp->next;

}

}

// Print topological order

printf("Topological ordering of vertices: ");

for (int i = 0; i < count; ++i)

printf("%d ", topOrder[i]);

}

int main() {

// Example graph

int vertices = 6;

struct Graph\* graph = createGraph(vertices);

addEdge(graph, 5, 2);

addEdge(graph, 5, 0);

addEdge(graph, 4, 0);

addEdge(graph, 4, 1);

addEdge(graph, 2, 3);

addEdge(graph, 3, 1);

// Perform topological sorting

topologicalSort(graph);

return 0;

}

Output:Topological ordering of vertices: 4 5 0 2 3 1