Topological Sorting

Write a C program to create a graph and display the ordering of vertices.

PROGRAM:

```
#include <stdio.h>

#include <stdlib.h>


// Define the structure for the adjacency list node

struct AdjListNode {

   int dest;

   struct AdjListNode* next;

};


// Define the structure for the adjacency list

struct AdjList {

   struct AdjListNode* head;

};


// Define the structure for the graph

struct Graph {

   int V;

   struct AdjList* array;

};


// Create a new adjacency list node

struct AdjListNode* newAdjListNode(int dest) {

   struct AdjListNode* newNode = (struct AdjListNode*)malloc(sizeof(struct AdjListNode));

   newNode->dest = dest;

   newNode->next = NULL;

   return newNode;

}


// Create a graph with V vertices

struct Graph* createGraph(int V) {
```

```
    struct Graph* graph = (struct Graph*)malloc(sizeof(struct Graph));
    graph->V = V;
    graph->array = (struct AdjList*)malloc(V * sizeof(struct AdjList));
    for (int i = 0; i < V; ++i)
        graph->array[i].head = NULL;
    return graph;
}

// Add an edge to the graph
void addEdge(struct Graph* graph, int src, int dest) {
    struct AdjListNode* newNode = newAdjListNode(dest);
    newNode->next = graph->array[src].head;
    graph->array[src].head = newNode;
}

// Function to perform topological sort using Kahn's algorithm
void topologicalSort(struct Graph* graph) {
    int V = graph->V;
    int* inDegree = (int*)malloc(V * sizeof(int));
    for (int i = 0; i < V; i++)
        inDegree[i] = 0;

    // Calculate in-degree of each vertex
    for (int u = 0; u < V; u++) {
        struct AdjListNode* node = graph->array[u].head;
        while (node != NULL) {
            inDegree[node->dest]++;
            node = node->next;
        }
    }

    // Create a queue and enqueue all vertices with in-degree 0
    int* queue = (int*)malloc(V * sizeof(int));
    int front = 0, rear = 0;
```

```
for (int i = 0; i < V; i++) {
    if (inDegree[i] == 0)
        queue[rear++] = i;
}

// Initialize count of visited vertices
int count = 0;

// Create an array to store the result (topological ordering)
int* topOrder = (int*)malloc(V * sizeof(int));

// Process vertices in queue
while (front != rear) {
    int u = queue[front++];

    // Add u to topological order
    topOrder[count++] = u;

    // Decrease in-degree of all adjacent vertices
    struct AdjListNode* node = graph->array[u].head;
    while (node != NULL) {
        if (--inDegree[node->dest] == 0)
            queue[rear++] = node->dest;
        node = node->next;
    }
}

// Check if there was a cycle
if (count != V) {
    printf("There exists a cycle in the graph\n");
} else {
    printf("Topological Sort: ");
    for (int i = 0; i < count; i++)
        printf("%d ", topOrder[i]);
```

```
        printf("\n");
    }

    // Clean up
    free(inDegree);
    free(queue);
    free(topOrder);
}

// Main function to test the above functions
int main() {
    int V = 6; // Number of vertices in the graph
    struct Graph* graph = createGraph(V);
    addEdge(graph, 5, 2);
    addEdge(graph, 5, 0);
    addEdge(graph, 4, 0);
    addEdge(graph, 4, 1);
    addEdge(graph, 2, 3);
    addEdge(graph, 3, 1);

    printf("Created graph:\n");
    for (int i = 0; i < V; i++) {
        printf("Adjacency list of vertex %d\n head ", i);
        struct AdjListNode* node = graph->array[i].head;
        while (node) {
            printf("-> %d", node->dest);
            node = node->next;
        }
        printf("\n");
    }

    printf("\nPerforming topological sort:\n");
    topologicalSort(graph);
```

```
    return 0;
}
```

OUTPUT:

```
Created graph:
Adjacency list of vertex 0
 head
Adjacency list of vertex 1
 head
Adjacency list of vertex 2
 head -> 3
Adjacency list of vertex 3
 head -> 1
Adjacency list of vertex 4
 head -> 1 -> 0
Adjacency list of vertex 5
 head -> 0 -> 2

Performing topological sort:
Topological Sort: 5 4 2 3 1 0
```