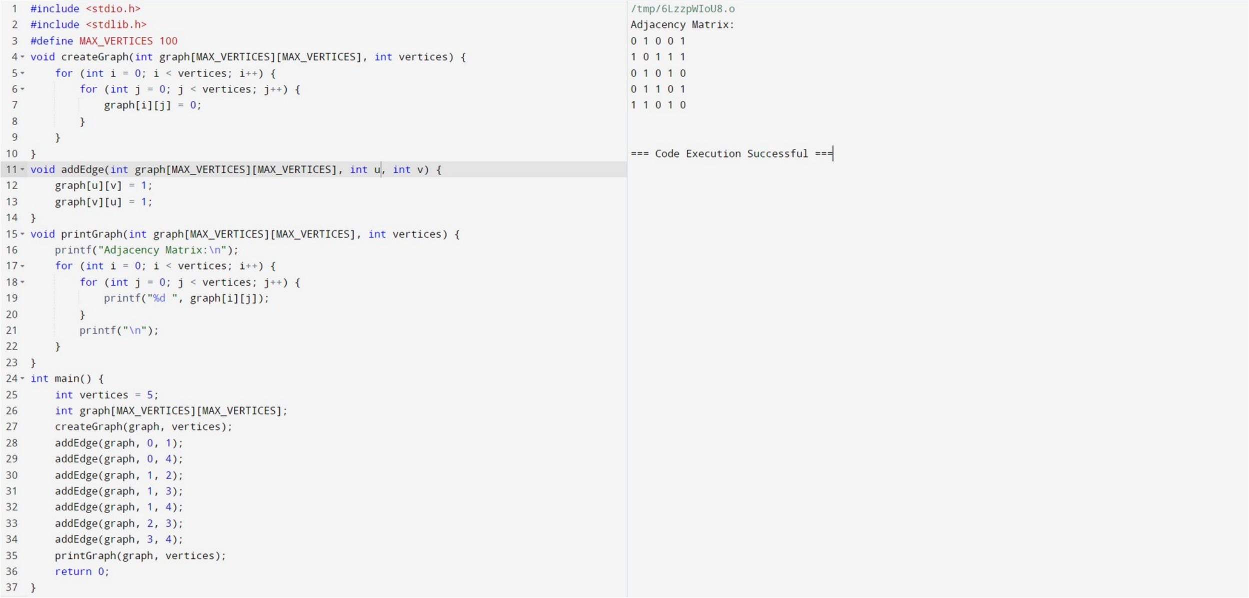
1. Write a C program for GRAPH

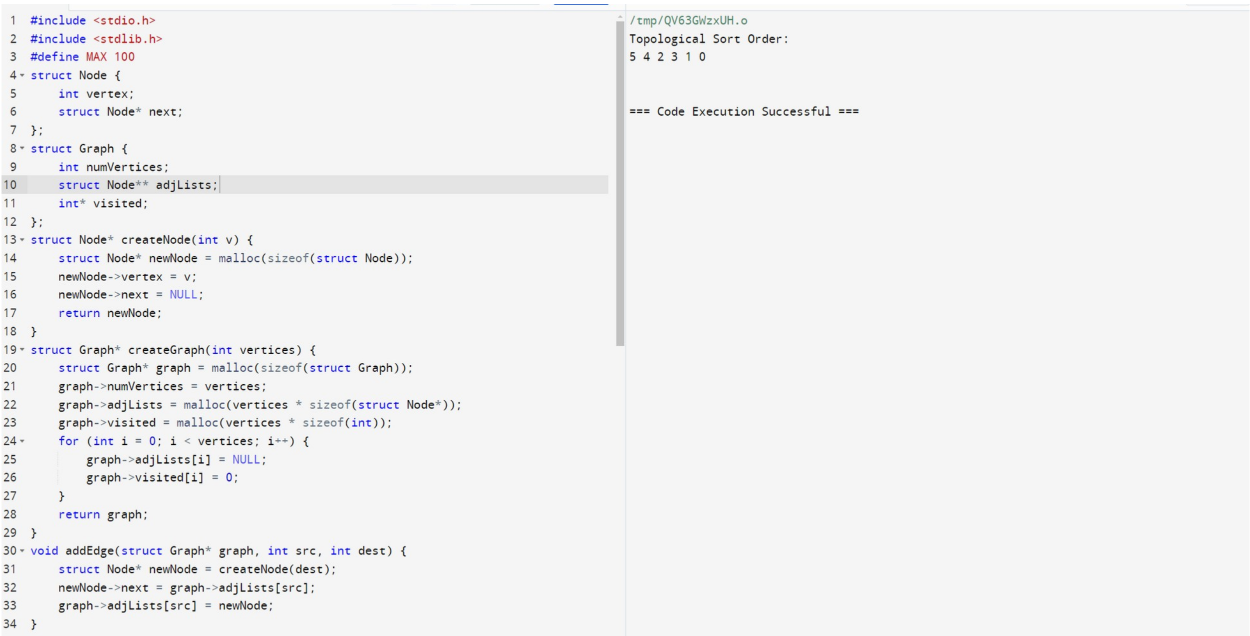


```c
#include <stdio.h>
#include <stdlib.h>
#define MAX_VERTICES 100
void createGraph(int graph[MAX_VERTICES][MAX_VERTICES], int vertices) {
    for (int i = 0; i < vertices; i++) {
        for (int j = 0; j < vertices; j++) {
            graph[i][j] = 0;
        }
    }
}
void addEdge(int graph[MAX_VERTICES][MAX_VERTICES], int u, int v) {
    graph[u][v] = 1;
    graph[v][u] = 1;
}
void printGraph(int graph[MAX_VERTICES][MAX_VERTICES], int vertices) {
    printf("Adjacency Matrix:\n");
    for (int i = 0; i < vertices; i++) {
        for (int j = 0; j < vertices; j++) {
            printf("%d ", graph[i][j]);
        }
        printf("\n");
    }
}
int main() {
    int vertices = 5;
    int graph[MAX_VERTICES][MAX_VERTICES];
    createGraph(graph, vertices);
    addEdge(graph, 0, 1);
    addEdge(graph, 0, 4);
    addEdge(graph, 1, 2);
    addEdge(graph, 1, 3);
    addEdge(graph, 1, 4);
    addEdge(graph, 2, 3);
    addEdge(graph, 3, 4);
    printGraph(graph, vertices);
    return 0;
}
```

```
/tmp/6LzzpWIoU8.o
Adjacency Matrix:
0 1 0 0 1
1 0 1 1 1
0 1 0 1 0
0 1 1 0 1
1 1 0 1 0


=== Code Execution Successful ===
```

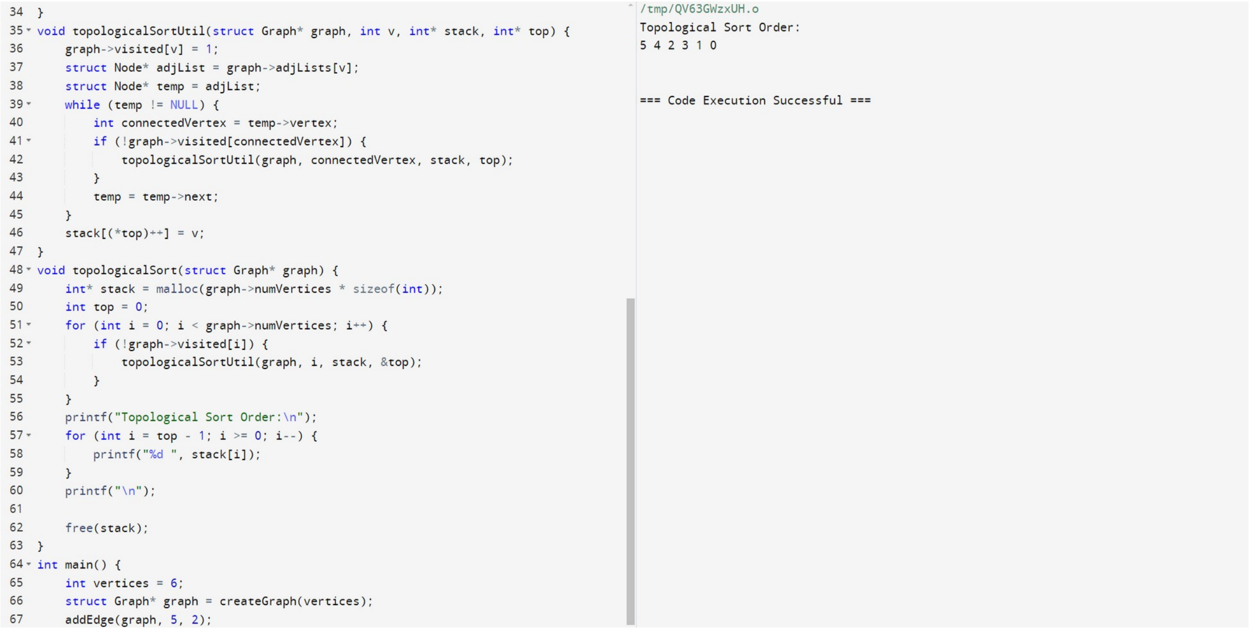
2.Write a C program for TOPOLOGICAL GRAPH.



```c
#include <stdio.h>
#include <stdlib.h>
#define MAX 100
struct Node {
    int vertex;
    struct Node* next;
};
struct Graph {
    int numVertices;
    struct Node** adjLists;
    int* visited;
};
struct Node* createNode(int v) {
    struct Node* newNode = malloc(sizeof(struct Node));
    newNode->vertex = v;
    newNode->next = NULL;
    return newNode;
}
struct Graph* createGraph(int vertices) {
    struct Graph* graph = malloc(sizeof(struct Graph));
    graph->numVertices = vertices;
    graph->adjLists = malloc(vertices * sizeof(struct Node*));
    graph->visited = malloc(vertices * sizeof(int));
    for (int i = 0; i < vertices; i++) {
        graph->adjLists[i] = NULL;
        graph->visited[i] = 0;
    }
    return graph;
}
void addEdge(struct Graph* graph, int src, int dest) {
    struct Node* newNode = createNode(dest);
    newNode->next = graph->adjLists[src];
    graph->adjLists[src] = newNode;
}
```

```
/tmp/QV63GWzxUH.o
Topological Sort Order:
5 4 2 3 1 0


=== Code Execution Successful ===
```

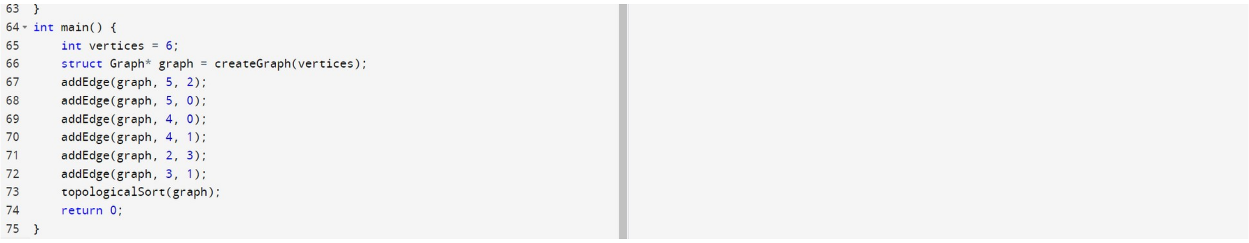


```c
}
void topologicalSortUtil(struct Graph* graph, int v, int* stack, int* top) {
    graph->visited[v] = 1;
    struct Node* adjList = graph->adjLists[v];
    struct Node* temp = adjList;
    while (temp != NULL) {
        int connectedVertex = temp->vertex;
        if (!graph->visited[connectedVertex]) {
            topologicalSortUtil(graph, connectedVertex, stack, top);
        }
        temp = temp->next;
    }
    stack[(*top)++] = v;
}
void topologicalSort(struct Graph* graph) {
    int* stack = malloc(graph->numVertices * sizeof(int));
    int top = 0;
    for (int i = 0; i < graph->numVertices; i++) {
        if (!graph->visited[i]) {
            topologicalSortUtil(graph, i, stack, &top);
        }
    }
    printf("Topological Sort Order:\n");
    for (int i = top - 1; i >= 0; i--) {
        printf("%d ", stack[i]);
    }
    printf("\n");

    free(stack);
}
int main() {
    int vertices = 6;
    struct Graph* graph = createGraph(vertices);
    addEdge(graph, 5, 2);
```

```
/tmp/QV63GWzxUH.o
Topological Sort Order:
5 4 2 3 1 0


=== Code Execution Successful ===
```



```c
}
int main() {
    int vertices = 6;
    struct Graph* graph = createGraph(vertices);
    addEdge(graph, 5, 2);
```

```c
    addEdge(graph, 5, 0);
    addEdge(graph, 4, 0);
    addEdge(graph, 4, 1);
    addEdge(graph, 2, 3);
    addEdge(graph, 3, 1);
    topologicalSort(graph);
    return 0;
}
```