Topological Sorting

```
#include<stdio.h>
#include<stdlib.h>

#define MAX 100

int n;    /*Number of vertices in the graph*/
int adj[MAX][MAX]; /*Adjacency Matrix*/
void create_graph();

int queue[MAX], front = -1,rear = -1;
void insert_queue(int v);
int delete_queue();
int isEmpty_queue();

int indegree(int v);

int main()
{
int i,v,count,topo_order[MAX],indeg[MAX];

create_graph();

      /*Find the indegree of each vertex*/
for(i=0;i<n;i++)
      {
indeg[i] = indegree(i);
if(indeg[i] == 0 )
insert_queue(i);
```

```
        }

count = 0;

while(  !isEmpty_queue( ) && count < n )
        {
                v = delete_queue();
topo_order[++count] = v; /*Add vertex v to topo_order array*/
                /*Delete all edges going fron vertex v */
for(i=0; i<n; i++)
                {
if(adj[v][i] == 1)
                        {
adj[v][i] = 0;
indeg[i] = indeg[i]-1;
if(indeg[i] == 0)
insert_queue(i);
                        }
                }
        }
if( count < n )
        {
printf("\nNo topological ordering possible, graph contains cycle\n");
exit(1);
        }
printf("\nVertices in topological order are :\n");
for(i=1; i<=count; i++)
printf( "%d ",topo_order[i] );
printf("\n");
```

```
return 0;
}/*End of main()*/

void insert_queue(int vertex)
{
if (rear == MAX-1)
printf("\nQueue Overflow\n");
else
        {
if (front == -1)  /*If queue is initially empty */
front = 0;
rear = rear+1;
queue[rear] = vertex ;
        }
}/*End of insert_queue()*/
int isEmpty_queue()
{
if(front == -1 || front > rear )
return 1;
else
return 0;
}/*End of isEmpty_queue()*/
int delete_queue()
{
int del_item;
if (front == -1 || front > rear)
        {
printf("\nQueue Underflow\n");
```

```
exit(1);
        }
else
        {
del_item = queue[front];
front = front+1;
return del_item;
        }
}/*End of delete_queue() */
int indegree(int v)
{
int i,in_deg = 0;
for(i=0; i<n; i++)
if(adj[i][v] == 1)
in_deg++;
return in_deg;
}/*End of indegree() */
void create_graph()
{
int i,max_edges,origin,destin;
printf("\nEnter number of vertices : ");
scanf("%d",&n);
max_edges = n*(n-1);
for(i=1; i<=max_edges; i++)
        {
printf("\nEnter edge %d(-1 -1 to quit): ",i);
scanf("%d %d",&origin,&destin);
if((origin == -1) && (destin == -1))
break;
```

```
if( origin >= n || destin>= n || origin<0 || destin<0)
            {
printf("\nInvalid edge!\n");
i--;
            }
else
adj[origin][destin] = 1;
      }
}
```

OUTPUT:

```
Enter number of vertices : 6
Enter edge 1(-1 -1 to quit): 0 1
Enter edge 2(-1 -1 to quit): 0 2
Enter edge 3(-1 -1 to quit): 0 3
Enter edge 4(-1 -1 to quit): 1 3
Enter edge 5(-1 -1 to quit): 2 4
Enter edge 6(-1 -1 to quit): 3 2 5
Enter edge 7(-1 -1 to quit): 3 4
Enter edge 8(-1 -1 to quit): 3 5
Enter edge 9(-1 -1 to quit): 4 5
Enter edge 10(-1 -1 to quit): 1 5
Enter edge 11(-1 -1 to quit): -1 -1
Vertices in topological order are :
0 1 2 3 4 5
```