```
#include<stdio.h>
#define SIZE 20

void read_graph(int *nv, int adj[][SIZE])
{
        int i, j;

        printf("\nEnter the number of vertices : ");
        scanf("%d", nv);


        printf("\nEnter the adjecency matrix (order %d x %d) :\n", *nv, *nv);
        for( i = 0; i < *nv; i++ )
            for( j = 0; j < *nv; j++ )
                scanf("%d", &adj[i][j]);
}


int indegree(int v, int *nv, int adj[][SIZE])
{
        int i,id = 0;

        for( i = 0; i < *nv; i++ )
                if( adj[i][v] == 1 )
                        id++;
        return id;
}

int delete_queue( int queue[], int *front, int *rear )
{
        int del_item;
        if ( *front == -1 || *front > *rear )
        {
                printf("\nQueue underflow\n");
                return 0;
        }
        else
        {
                del_item = queue[ *front ];
                *front = *front + 1;
                return del_item;
        }
}


void insert_queue( int vertex, int queue[], int *front, int *rear )
{
        if ( *rear == SIZE - 1 )
                printf("\nQueue overflow\n");
        else
        {
                if ( *front == -1 )
                        *front = 0;
                *rear = *rear + 1;
                queue[ *rear ] = vertex ;
        }
}


int isEmpty_queue( int *front, int *rear )
{
        if( *front == -1 || *front > *rear )
                return 1;
        else
                return 0;
}


void topo_sort( int *nv, int adj[][SIZE], int topo_order[], int *flag )
{
    int i, v;
    int count = 0;
    int indeg[SIZE];

    int queue[ SIZE ], front, rear;
        front = rear = -1;

    *flag = 1;

    if( !*nv )
    {
        printf("\nPlease read a graph \n");
        return;
    }

    for( i = 0; i < *nv; i++ )
    {
        indeg[i] = indegree( i, nv, adj );
        if( indeg[i] == 0 )
            insert_queue( i, queue, &front, &rear );
    }

    while(  !isEmpty_queue( &front, &rear ) && count < *nv )
    {
        v = delete_queue( queue, &front, &rear );
        topo_order[ ++count ] = v + 1;
        for( i = 0; i < *nv; i++ )
        {
            if( adj[v][i] == 1 )
            {
                adj[v][i] = 0;
                indeg[i] = indeg[i] - 1;
                if(indeg[i] == 0)
                    insert_queue( i, queue, &front, &rear );
            }
        }
    }

    if( count < *nv )
    {
            printf("\nNo topological ordering possible, graph contains cycle\n");
            *flag = 0;
            return;
    }

    printf("\nTopological ordering of vertices successfully conducted\n");

}


void display( int *nv, int adj[][SIZE], int topo_order[], int *flag )
{
   int i, j;
   if( *nv )
   {
       printf("\nThe given adjecency matrix (order %d x %d) is :\n", *nv, *nv);
       for( i = 0; i < *nv; i++ )
       {
           for( j = 0; j < *nv; j++ )
               printf("%d ", adj[i][j]);
           printf("\n");
       }
        if( *flag )
        {
            printf("\nVertices in topological order are :\n");
            for( i = 1; i <= *nv; i++ )
                printf( "%d ", topo_order[i] );
            printf("\n");
        }
   }
   else
   {
       printf("\nPlease read a graph \n");
       return;
   }
}

int main()
{
    int adj[SIZE][SIZE], topo_order[SIZE];
    int nv = 0;
    int flag = 0;

        int ele = 1, ch;

        while( ele )
        {
                printf( "\n--------------MENU--------------\n" );
                printf( "\n\t1. Read Graph\n\t2. Topological Sort\n\t3.
Display\n\t4. Exit\n" );
                printf( "\n--------------------------------\n" );
                printf( "\n Enter your choice:" );
                scanf( "%d", &ch );

                switch( ch )
                {
                        case 1 : read_graph( &nv, adj );
                                 break;
                        case 2 : topo_sort( &nv, adj, topo_order, &flag );
                                 break;
                        case 3 : display( &nv, adj, topo_order, &flag );
                             break;
                        case 4 : ele = 0;
                             printf("\nExit from the program\n");
                                 break;
                        default: printf( "\n Invalid choice. Please enter a valid
choice... \n" );
                }

        }
        return 0;
}
```