```
#include<iostream>
#include<iomanip>
#include<vector>
#define N 13

using namespace std;

//W - Not Visited, G - Visited, but adjacent vertices needs to be visited, B - Completely visited, including the adjacent nodes
enum MyColor {W,G,B};

struct Vertex
{
    vector<int> AdjList; //To store the indices of vertex
    int Value; //For easy understanding, the index will be stored as vertex value
    MyColor Color; //Flag used to represent a vertex is visited or not
    int Parent; //Index of the parent vertex in the resultant BFS tree
    int st; //To record the starting visiting time
    int end; //To record the finishing visiting time
};

class Graph
{
    Vertex *V;
    int size;

    public:

        //Initialize the size and the vertex' values
        Graph(int);

        //Adding an edge between two vertices. index of 'from' and 'to' are given as input
        void AddEdge(int,int);

        //To perform the DFS on input graph and to obtain the topological order.
        void DFS(vector<Vertex> &);
        void DFS_Visit(int,int&,vector<Vertex> &);

        //To display the final graph
        void ShowGraph();

        //To return the topological order
        vector<Vertex> TopologicalOrder();
};

vector<Vertex> Graph::TopologicalOrder()
{
    vector<Vertex> TopOrder;

    //Optain Topological order by Performing DFS on Graph
    DFS(TopOrder);

    //We may view the graph content after DFS
    cout<<"\n\n\tGraph After DFS:";
    ShowGraph();

    return TopOrder;

}

Graph::Graph(int n)
{
    size = n;
    V = new Vertex[n];
    for(int i=0;i<n;i++)
    {
        V[i].Value = i;
        V[i].Color = W;
        V[i].Parent = -1;
        V[i].st = V[i].end = 0;
    }

}

void Graph::ShowGraph()
{

    cout<<"\n\t********************************************
    cout<<"\n\tParent | Vertex Value | Color | Start | Finish";
```

```
cout<<"\n\t********************************************
    for(int i=0;i<size;i++)
    {
        cout<<"\n\t";
        if(V[i].Parent==-1)
            cout<<setw(6)<<"NULL";
        else
            cout<<setw(6)<<V[i].Parent;

        cout<<" | ";
        cout<<setw(12)<<V[i].Value;

        cout<<" | ";
        cout<<setw(5)<<V[i].Color;

        cout<<" | ";
        cout<<setw(5)<<V[i].st;

        cout<<" | ";
        cout<<setw(6)<<V[i].end;

    }
cout<<"\n\t********************************************

}
void Graph::DFS(vector<Vertex> &TopOrder)
{

    int time=0;
    for(int i=0;i<size;i++)
    {
        if(V[i].Color==W)
        {
            V[i].Parent = -1;
            DFS_Visit(i,time,TopOrder);
        }
    }
}

//Input is the starting vertex's index
void Graph::DFS_Visit(int i, int &time, vector<Vertex> &TopOrder)
{

    int u;

    //Visit the starting vertex
    V[i].Color = G;
    V[i].st = ++time;

    for(int p=0;p<V[i].AdjList.size();p++)
    {
        u = V[i].AdjList.at(p);
        if(V[u].Color==W)
        {
            V[u].Parent = i;
            //cout<<"\n\t"<<V[u].Parent<<"-->"<<V[u].Value;
            DFS_Visit(u,time,TopOrder);
        }
    }

    V[i].end = ++time;
    V[i].Color = B;
    TopOrder.insert(TopOrder.begin(),V[i]);
}


//from and to are the indices of nodes
void Graph::AddEdge(int from, int to)
{
    V[from].AdjList.insert(V[from].AdjList.end(), to);
}



int main()
{

    //In this example, we considered a graph with 5 vertices. (0,1,...4)
    //If you want to test for other graphs, need to give appropriate details.
    //The index are considered as the VALUE of the vertices.

    //Test Input-1 (Size-5)
```

```
//int a[] = {0,1,2,3,4};

Graph g(N);

/*int b[][N]     = {         {0,1,0,1,0,0},
                        {0,0,0,0,1,0},
                        {0,0,0,0,1,1},
                        {0,1,0,0,0,0},
                        {0,0,0,1,0,0},
                        {0,0,0,0,0,0} };
*/              //   0 1 2 3 4 5 6 7 8 9 0 1 2
int b[][N]  =   {   {0,1,0,0,0,1,1,0,0,0,0,0,0},
                    {0,0,0,0,0,0,0,0,0,0,0,0,0},
                    {1,0,0,1,0,0,0,0,0,0,0,0,0},
                    {0,0,0,0,0,1,0,0,0,0,0,0,0},
                    {0,0,0,0,0,0,0,0,0,0,0,0,0},
                    {0,0,0,0,1,0,0,0,0,0,0,0,0},
                    {0,0,0,0,1,0,0,0,0,1,0,0,0},
                    {0,0,0,0,0,0,1,0,0,0,0,0,0},
                    {0,0,0,0,0,0,0,1,0,0,0,0,0},
                        {0,0,0,0,0,0,0,0,0,0,1,1,1},
                        {0,0,0,0,0,0,0,0,0,0,0,0,0},
                        {0,0,0,0,0,0,0,0,0,0,0,0,1},
                        {0,0,0,0,0,0,0,0,0,0,0,0,0}};

    for(int i=0;i<N;i++)
    {
        for(int j=0;j<N;j++)
        {
            if(b[i][j]!=0)
            {
                g.AddEdge(i,j);
            }
        }
    }
    vector<Vertex> TopOrder;

    TopOrder = g.TopologicalOrder();

    int topsize = TopOrder.size();
    Vertex v;

    cout<<"\n\n\tTopological Order of Vertices: ";
    for(int i=0;i<topsize;i++)
    {
        v = TopOrder.at(i);
        cout<<setw(5)<<v.Value;
    }

    cout<<"\n\n";
}
```