```
 1: #include<iostream>
 2: #include<iomanip>
 3: #include<vector>
 4: #define N 13
 5:
 6: using namespace std;
 7:
 8: //W - Not Visited, G - Visited, but adjacent
    vertices needs to be visited, B - Completely
    visited, including the adjacent nodes
 9: enum MyColor {W,G,B};
10:
11: struct Vertex
12: {
13:     vector<int> AdjList; //To store the indices
    of vertex
14:     int Value; //For easy understanding, the
    index will be stored as vertex value
15:     MyColor Color; //Flag used to represent a
    vertex is visited or not
16:     int Parent; //Index of the parent vertex in
    the resultant BFS tree
17:     int st; //To record the starting visiting time
18:     int end; //To record the finishing visiting
    time
19: };
20:
21: class Graph
22: {
23:     Vertex *V;
```

```
24:     int size;
25:
26:     public:
27:
28:         //Initialize the size and the vertex'
    values
29:         Graph(int);
30:
31:         //Adding an edge between two vertices.
    index of 'from' and 'to' are given as input
32:         void AddEdge(int,int);
33:
34:         //To perform the DFS on input graph and
    to obtain the topological order.
35:         void DFS(vector<Vertex> &);
36:         void DFS_Visit(int,int&,vector<Vertex> &);
37:
38:         //To display the final graph
39:         void ShowGraph();
40:
41:         //To return the topological order
42:         vector<Vertex> TopologicalOrder();
43: };
44:
45: vector<Vertex> Graph::TopologicalOrder()
46: {
47:     vector<Vertex> TopOrder;
48:
49:     //Optain Topological order by Performing DFS
    on Graph
```

```
50:     DFS(TopOrder);
51:
52:     //We may view the graph content after DFS
53:     cout<<"\n\n\tGraph After DFS:";
54:     ShowGraph();
55:
56:     return TopOrder;
57:
58: }
59:
60: Graph::Graph(int n)
61: {
62:     size = n;
63:     V = new Vertex[n];
64:     for(int i=0;i<n;i++)
65:     {
66:         V[i].Value = i;
67:         V[i].Color = W;
68:         V[i].Parent = -1;
69:         V[i].st = V[i].end = 0;
70:     }
71:
72: }
73:
74: void Graph::ShowGraph()
75: {
76:
    cout<<"\n\t**********************************************
77:     cout<<"\n\tParent | Vertex Value | Color |
    Start | Finish";
```

```
 78:
    cout<<"\n\t**************************************************
 79:     for(int i=0;i<size;i++)
 80:     {
 81:         cout<<"\n\t";
 82:         if(V[i].Parent==-1)
 83:             cout<<setw(6)<<"NULL";
 84:         else
 85:             cout<<setw(6)<<V[i].Parent;
 86:
 87:         cout<<" | ";
 88:         cout<<setw(12)<<V[i].Value;
 89:
 90:         cout<<" | ";
 91:         cout<<setw(5)<<V[i].Color;
 92:
 93:         cout<<" | ";
 94:         cout<<setw(5)<<V[i].st;
 95:
 96:         cout<<" | ";
 97:         cout<<setw(6)<<V[i].end;
 98:
 99:     }
100:
    cout<<"\n\t**************************************************
101:
102: }
103: void Graph::DFS(vector<Vertex> &TopOrder)
104: {
105:
```

```
106:     int time=0;
107:     for(int i=0;i<size;i++)
108:     {
109:         if(V[i].Color==W)
110:         {
111:             V[i].Parent = -1;
112:             DFS_Visit(i,time,TopOrder);
113:         }
114:     }
115: }
116:
117: //Input is the starting vertex's index
118: void Graph::DFS_Visit(int i, int &time,
     vector<Vertex> &TopOrder)
119: {
120:
121:     int u;
122:
123:     //Visit the starting vertex
124:     V[i].Color = G;
125:     V[i].st = ++time;
126:
127:     for(int p=0;p<V[i].AdjList.size();p++)
128:     {
129:         u = V[i].AdjList.at(p);
130:         if(V[u].Color==W)
131:         {
132:             V[u].Parent = i;
133:             //cout<<"\n\t"<<V[u].Parent<<"--
     >"<<V[u].Value;
```

```
134:                 DFS_Visit(u,time,TopOrder);
135:             }
136:         }
137:
138:         V[i].end = ++time;
139:         V[i].Color = B;
140:         TopOrder.insert(TopOrder.begin(),V[i]);
141: }
142:
143:
144: //from and to are the indices of nodes
145: void Graph::AddEdge(int from, int to)
146: {
147:         V[from].AdjList.insert(V[from].AdjList.end(),
     to);
148: }
149:
150:
151:
152: int main()
153: {
154:
155:         //In this example, we considered a graph with
     5 vertices. (0,1,...4)
156:         //If you want to test for other graphs, need
     to give appropriate details.
157:         //The index are considered as the VALUE of
     the vertices.
158:
159:         //Test Input-1 (Size-5)
```

```
160:     //int a[] = {0,1,2,3,4};
161:
162:     Graph g(N);
163:
164:     /*int b[][N]    = {          {0,1,0,1,0,0},
165:                               {0,0,0,0,1,0},
166:                               {0,0,0,0,1,1},
167:                               {0,1,0,0,0,0},
168:                               {0,0,0,1,0,0},
169:                               {0,0,0,0,0,0} };
170:     */                //   0 1 2 3 4 5 6 7 8 9 0 1
    2
171:     int b[][N]  =   {   {0,1,0,0,0,1,1,0,0,0,0,0,
    0},
172:                         {0,0,0,0,0,0,0,0,0,0,0,0,
    0},
173:                         {1,0,0,1,0,0,0,0,0,0,0,0,
    0},
174:                         {0,0,0,0,0,1,0,0,0,0,0,0,
    0},
175:                         {0,0,0,0,0,0,0,0,0,0,0,0,
    0},
176:                         {0,0,0,0,1,0,0,0,0,0,0,0,
    0},
177:                         {0,0,0,0,1,0,0,0,0,1,0,0,
    0},
178:                         {0,0,0,0,0,0,1,0,0,0,0,0,
    0},
179:                         {0,0,0,0,0,0,0,1,0,0,0,0,
    0},
```

```
180:                             {0,0,0,0,0,0,0,0,0,0,1,1,
    1},
181:                             {0,0,0,0,0,0,0,0,0,0,0,0,
    0},
182:                             {0,0,0,0,0,0,0,0,0,0,0,0,
    1},
183:                             {0,0,0,0,0,0,0,0,0,0,0,0,
    0}};
184:
185:     for(int i=0;i<N;i++)
186:     {
187:         for(int j=0;j<N;j++)
188:         {
189:             if(b[i][j]!=0)
190:             {
191:                 g.AddEdge(i,j);
192:             }
193:         }
194:     }
195:     vector<Vertex> TopOrder;
196:
197:     TopOrder = g.TopologicalOrder();
198:
199:     int topsize = TopOrder.size();
200:     Vertex v;
201:
202:     cout<<"\n\n\tTopological Order of Vertices: ";
203:     for(int i=0;i<topsize;i++)
204:     {
205:         v = TopOrder.at(i);
```

```
206:             cout<<setw(5)<<v.Value;
207:         }
208:
209:         cout<<"\n\n";
210: }
```