```c
#include<stdio.h>
#include<stdlib.h>
#define size 100

struct graph* gr=NULL;
struct queue* q=NULL;

struct queue{
    int items[size];
    int front,rear;
};

struct node{
    int dest;
    struct node* next;
};

struct adj_list{
    struct node* head;

};

struct graph{
    int v;
    struct adj_list* array;
};

struct node* cnode(int dest){
    struct node* temp=(struct node*)malloc(sizeof(struct node));
    temp->dest=dest;
    temp->next=NULL;
    return(temp);
}

struct graph* cgraph(int v){
    struct graph* gr=(struct graph*)malloc(sizeof(struct graph));
    gr->v=v;
    gr->array=(struct adj_list*)malloc(v*sizeof(struct adj_list));
    int i=0;
    for(i=0;i<v;++i){
        gr->array[i].head=NULL;
    }
    return(gr);
}
void add_edge(struct graph* gr,int src,int dest){
    struct node* temp=cnode(dest);
    temp->next=gr->array[src].head;
    gr->array[src].head=temp;
    temp=NULL;
    free(temp);
}

struct queue* cqueue(){
    struct queue* q=(struct queue*)malloc(sizeof(queue));
```

```
 55:     q->front=-1;
 56:     q->rear=-1;
 57:     return(q);
 58: 
 59: }
 60: 
 61: int isEmpty(struct queue* q) {
 62:     if(q->rear == -1)
 63:         return 1;
 64:     else
 65:         return 0;
 66: }
 67: 
 68: void enqueue(struct queue* q, int value){
 69:     if(q->rear == size-1)
 70:         printf("\nQueue is Full!!");
 71:     else {
 72:         if(q->front == -1)
 73:             q->front = 0;
 74:         q->rear++;
 75:         q->items[q->rear] = value;
 76:     }
 77: }
 78: 
 79: int dequeue(struct queue* q){
 80:     int item;
 81:     if(isEmpty(q)){
 82:         printf("Queue is empty");
 83:         item = -1;
 84:     }
 85:     else{
 86:         item = q->items[q->front];
 87:         q->front++;
 88:         if(q->front > q->rear){
 89:             q->front = q->rear = -1;
 90:         }
 91:     }
 92:     return item;
 93: }
 94: 
 95: void t node* cnode(int dest){
 96:     int indegree[gr->v];
 97: 
 98:      for(int i=0;i<gr->v;++i)
 99:         indegree[i]=0;
100: 
101:     for(int i=0;i<gr->v;++i){
102:         struct node* temp=gr->array[i].head;
103:         while(temp!=NULL){
104:             int k=temp->dest;
105:             indegree[k]++;
106:             temp=temp->next;
107:         }
108:     }
```

```
109:     for(int v=1;v<gr->v;v++){
110:         if(indegree[v]==0){
111:             indegree[v]=-1;
112:             enqueue(q,v);
113: 
114:         while(isEmpty(q)==false){
115:             int u=dequeue(q);
116:             indegree[u]=-1;
117:             printf("%d --->",u);
118:             struct node* temp=gr->array[u].head;
119:             while(temp!=NULL){
120:                 int k=temp->dest;
121:                 indegree[k]=indegree[k]-1;
122:                 if(indegree[k]==0)
123:                     enqueue(q,k);
124: 
125:                 temp=temp->next;
126:                 }
127:             }
128:         }
129:     }
130: }
131: 
132: int main(){
133:      gr = cgraph(9);
134:     add_edge(gr, 1,5);
135:     add_edge(gr, 1,4);
136:     add_edge(gr, 1,3);
137:    add_edge(gr, 2,3);
138:     add_edge(gr, 2,8);
139:     add_edge(gr, 3,6);
140:    add_edge(gr, 4,6);
141:     add_edge(gr, 4,8);
142:     add_edge(gr,5,8);
143:     add_edge(gr, 6,7);
144:     add_edge(gr, 7,8);
145:     printf("Topological order of given graph is:\n");
146:     q=cqueue();
147:     topl_order();
148: return 0;
149: }
```