Q6

```
#include <iostream>
using namespace std;
typedef struct mlnode *mlptr;
struct queue
{
   int size;
   int front;
   int rear;
   mlptr elements[50];
};
void enqueue(struct queue &Q, mlptr x)
{
   if(((Q.rear+1)%Q.size)==Q.rear)
   {
      cout<<"queue is full"<<endl;
   }
   else
   {
      if(Q.front==-1)
      {
         Q.rear = 0, Q.front=0;
      }
      else
      {
         Q.rear = (Q.rear+1)%Q.size;
      }
      Q.elements[Q.rear]=x;

   }

}

mlptr dequeue(struct queue &Q)
{
   mlptr t;
   if(Q.front==-1)
   {
      cout<<"Queue is empty"<<endl;
      return t;
   }
   else
   {
      if(Q.front==Q.rear)
      {
```

```
            t = Q.elements[Q.front];
            Q.front=-1, Q.rear=-1;
        }
        else
        {
            t = Q.elements[Q.front];
            Q.front = (Q.front+1)%Q.size;
        }
        return t;
    }

}
struct mlnode
{
    int mldata;
    struct mlnode * dlink;
    struct mlnode *mlnext;
};

void addEndMultiLL(mlptr &MainML, int n)
{

    mlptr TML, ML = MainML;
    TML = new(mlnode);
    TML->mldata = n, TML->dlink = NULL, TML->mlnext = NULL;
    if(MainML==NULL)
    MainML = TML;
    else
    {
        while (ML->mlnext!=NULL)
        ML = ML->mlnext;
        ML->mlnext = TML;
    }
}
void join_multi_ll(mlptr ML, mlptr TML)
{
    while(ML->mlnext!=NULL) ML = ML->mlnext;
    ML->dlink = TML;
}
mlptr construct_multi_level_ll()
{
    mlptr ML = NULL, TempML;
    int n;
    cin>>n;
    while(n!=-1)
```

```
    {
        if(n==1)
        {
            TempML =  construct_multi_level_ll();
            join_multi_ll(ML, TempML);

        }
        else if(n==0)
        {

        }
        else
        {
            addEndMultiLL(ML, n);
        }
        cin>>n;
    }
    return ML;
}

void depthWise_printing(mlptr ML)
{
    if(ML!=NULL)
    {
        cout<<ML->mldata<<" ";
        if(ML->dlink!=NULL) depthWise_printing(ML->dlink);
        depthWise_printing(ML->mlnext);
    }
}
int main()
{
    mlptr ML, TempML;
    struct queue Q;
    Q.size = 30;
    Q.rear = -1;
    Q.front = -1;
    ML = construct_multi_level_ll();
    enqueue(Q, ML);
    while(Q.front!=-1)
    {
        TempML = dequeue(Q);
        while (TempML!=NULL)
        {
            if(TempML->dlink!=NULL) enqueue(Q, TempML->dlink);
            cout<<TempML->mldata<<" ";
```

```
            TempML = TempML->mlnext;
        }
    }
    cout<<endl;
    depthWise_printing(ML);

}
```