```
#include <iostream>
#include<stdio.h>
using namespace std;


class Node
{
public
    Node *lchild;
    int data;
    Node *rchild;
};

class Queue {
private
    int front;
    int rear;
    int size;
    Node **Q;
public
    Queue(){front=rear=-1;size=10;Q=new Node*[size];}
    Queue(int size){front=rear=-1;this->size=size;;Q=new
Node*[size];}
    void enqueue(Node *x);
    Node *dequeue();
    int isEmpty(){ return front==rear;}
};
void Queue::enqueue(Node *x)
{
    if(rear==size-1)
        printf "Queue Full\n"
    else
    {
        rear++;
        Q[rear]=x;
    }
}
Node *Queue::dequeue()
{
```

```
    Node *x=NULL;
    if front    rear
        printf "Queue is Empty\n"
    else
    {

    x=Q[front+1];
    front
    }
    return
}

class Tree
{
    Node *root;
public
    Tree(){root=NULL;}
    void CreateTree();
    void Preorder(){Preorder(root);}
    void Preorder(Node *p);
    void Postorder(){Postorder(root);}
    void Postorder(Node *p);
    void Inorder(){Inorder(root);}
    void Inorder(Node *p);
    void Levelorder(){Levelorder(root);}
    void Levelorder(Node *p);
    int Height(){return Height(root);}
    int Height(Node *root);
};
void Tree::CreateTree()
{
    Node *p,*t=NULL;
    int x;
    Queue    100
    printf "Eneter root value "
    scanf("%d",&x);
    root=new Node;
    root->data=x;
    root   lchild  root   rchild  NULL
```

```
      enqueue root
    while     isEmpty
    {
        p=q.dequeue();
        printf "eneter left child of %d "       data
        scanf("%d",&x);
        if(x!=-1)
        {
            t=new Node;
            t->data=x;
            t->lchild=t->rchild=NULL;
            p->lchild=t;
            q.enqueue(t);
        }
        printf "eneter right child of %d "       data
        scanf("%d",&x);
        if(x!=-1)
        {
            t=new Node;
            t->data=x;
            t->lchild=t->rchild=NULL;
            p->rchild=t;
            q.enqueue(t);

        }
    }
}
void Tree::Preorder(struct Node *p)
{
    if(p)
    {

    printf "%d "       data
    Preorder       lchild
    Preorder       rchild

    }
}
void Tree::Inorder(struct Node *p)
```

```
{
    if(p)
    {

            Inorder(p->lchild);
            printf("%d ",p->data);
            Inorder(p->rchild);
    }
}

void Tree::Postorder(struct Node *p)
{
    if(p)
    {
        Postorder       lchild
        Postorder       rchild
        printf("%d ",p->data);

    }
}

    void Tree::Levelorder(struct Node *p)
    {

    Queue    100
    printf "%d " root    data
       enqueue root
    while     isEmpty
    {
        root=q.dequeue();
        if(root->lchild)
        {
            printf("%d ",root->lchild->data);
            q.enqueue(root->lchild);
        }
        if(root->rchild)
        {
        }
        printf("%d ",root->rchild->data);
        q.enqueue(root->rchild);
```

```
    }
}
int Tree::Height(struct Node *root)
{

    int x=0,y=0;
    if(root==0)
        return 0;
    x=Height(root->lchild); y=Height(root->rchild);
    if(x>y)
        return x+1;
    else
            return y+1;
}
int main() {
    Tree t;
      CreateTree
    cout    "Preorder "
      Preorder
    cout<<endl;
    cout    "Inorder "
      Inorder
    cout<<endl<<endl;
    return 0
}
```