//dsa 5 bst

#include<iostream>

using namespace std;

class BST

{

class Node

{ public:

int Data;

Node \*Left,\*Right;

Node(){ Left=Right=NULL; }

} \*Root;

public:

BST(){Root=NULL;}

int isEmpty(){ return Root==NULL;}

void Create();

void Insert(int); //Binary Search Tree Creation

int Search(int); //Search

void inOrder(Node\*); //Display

void preOrder(Node\*); //Dsiplay Leaf Nodes

int Depth(); //BFS

void mirror(); //Display mirror image

void Levelwise(); //Breath First Search Levelwise Display

void Delete(int); //Delete Node

Node\* return\_Root(){ return Root;}

};

//create tree

void BST::Create()

{

int Num;

char ans;

do

{

cout<<"Enter Number:";

cin>>Num;

Insert(Num);

Levelwise();

cout<<"\nAdd More...(Y/N)";

cin>>ans;

}while(ans=='y' || ans=='Y');

}

//Binary Search Tree Creation

void BST::Insert(int Num)

{

Node \*nNode,\*Temp;

nNode=new Node;

nNode->Data=Num;

if(Root==NULL)

Root=nNode;

else

{

Temp=Root;

while(1)

{

if(nNode->Data>Temp->Data)

{

if(Temp->Right==NULL)

{Temp->Right=nNode;break;}

else Temp=Temp->Right;

}

else if(nNode->Data<Temp->Data)

{

if(Temp->Left==NULL)

{Temp->Left=nNode;break;}

else Temp=Temp->Left;

}

else { cout<<"\nNumber already present";break;}

}//while

}

}

//Display

void BST::inOrder(Node \*Temp)

{

if(Temp!=NULL)

{

inOrder(Temp->Left);

cout<<" "<<Temp->Data;

inOrder(Temp->Right);

}

}

// Search

int BST::Search(int Num)

{

Node \*Tree=Root;

while(Tree!=NULL)

{

if(Num>Tree->Data)

Tree=Tree->Right;

else if(Num<Tree->Data)

Tree=Tree->Left;

else return 1;

}

return 0;

}

//Display Leaf Nodes

void BST::preOrder(Node \*Temp)

{

if(Temp!=NULL)

{

if(Temp->Left==NULL && Temp->Right==NULL)

cout<<" "<<Temp->Data;

preOrder(Temp->Left);

preOrder(Temp->Right);

}

}

//Find Depth of Tree

int BST ::Depth()

{ Node \*Queue[20],\*Temp=Root;

int Front=0,Rear=1,Level=0;

if(isEmpty()) return 0;

else {

Queue[0]=Temp; Queue[1]=NULL;

do {

Temp=Queue[Front++];

if(Temp==NULL)

{ Queue[++Rear]=NULL;

++Level; }

else

{

if(Temp->Left!=NULL)

Queue[++Rear]=Temp->Left;

if(Temp->Right!=NULL)

Queue[++Rear]=Temp->Right;

}

}while(Front<Rear);

}

return Level+1;

}

//Display Mirror image

void BST::mirror()

{

Node \*Queue[20],\*Temp=Root;

int Front=0,Rear=1;

if(isEmpty())cout<<"\nEmpty Tree";

else {

Queue[0]=Temp; Queue[1]=NULL;

do {

Temp=Queue[Front++];

if(Temp==NULL)

{ Queue[++Rear]=NULL;

cout<<"\n";

}

else

{cout<<" "<<Temp->Data;

if(Temp->Right!=NULL)

Queue[++Rear]=Temp->Right;

if(Temp->Left!=NULL)

Queue[++Rear]=Temp->Left;

}

}while(Front<Rear);

}

}

//Display Levelwise

void BST ::Levelwise()

{ Node \*Queue[20],\*Temp=Root;

int Front=0,Rear=1;

if(isEmpty())cout<<"\nEmpty Tree";

else {

Queue[0]=Temp; Queue[1]=NULL;

do {

Temp=Queue[Front++];

if(Temp==NULL)

{ Queue[++Rear]=NULL;

cout<<"\n";

}

else

{cout<<" "<<Temp->Data;

if(Temp->Left!=NULL)

Queue[++Rear]=Temp->Left;

if(Temp->Right!=NULL)

Queue[++Rear]=Temp->Right;

}

}while(Front<Rear);

}

}

//Delete Node

void BST::Delete(int Num)

{

Node \*Stack[10],\*pTemp=Root,\*Temp=Root,\*RTemp,\*pRTemp;

char Flag;

while(Temp->Data!=Num &&Temp!=NULL)

{

if(Num>Temp->Data)

{

pTemp=Temp;

Flag='R';

Temp=Temp->Right;

}

else

{

pTemp=Temp;

Flag='L';

Temp=Temp->Left;

}

}//while

if(Temp!=NULL)

{

if(Temp->Left==NULL && Temp->Right==NULL)//leaf node

{

if(Root==Temp) Root=NULL;

if(Flag=='R') pTemp->Right=NULL;

else pTemp->Left=NULL;

delete Temp;

}

else // delete Node with Left and Right children

if(Temp->Left!=NULL && Temp->Right!=NULL)

{

pRTemp=RTemp=Temp->Right;

while(RTemp->Left!=NULL)

{

pRTemp=RTemp;

RTemp=RTemp->Left;

}

if(pRTemp!=RTemp)

pRTemp->Left=RTemp->Right;

else Temp->Right= RTemp->Right;

Temp->Data=RTemp->Data;

delete RTemp;

} //if

else

{

if(Temp->Right!=NULL)//with Right child

{

if(Root==Temp) Root=Root->Right;

else if(Flag=='L') pTemp->Left=Temp->Right;

else pTemp->Right=Temp->Right;

}

else if(Temp->Left!=NULL) //with Left child

{

if(Root==Temp) Root=Root->Left;

else if(Flag=='L') pTemp->Left=Temp->Left;

else pTemp->Right=Temp->Left;

}

delete Temp;

}

}

else

cout<<"\nNode is not present";

}//Function end

int main()

{ BST B;

int D,ch,Num;

cout<<"\nCreate Tree: \n";

B.Create();

do{

cout<<"\n 1: Insert 2: Display Tree 3:Search 4:Display Leaf Nodes";

cout<<"\n5:Depth of Tree 6: Display Mirror image 7 :Display Levelwise 8: Delete Node 9:

Exit";

cin>>ch;

switch(ch)

{

case 1:

cout<<"Enter Number";

cin>>Num;

B.Insert(Num);

B.Levelwise();

break;

case 2:

cout<<"\n Inorder Display";

B.inOrder(B.return\_Root());

break;

case 3:

cout<<"\nEnter Data to search";

cin>>D;

if(B.Search(D)==1)

cout<<"\n Node Present";

else cout<<"\nNode Not Present ";

break;

case 4:

cout<<"\nLeaf Nodes";

B.preOrder(B.return\_Root());

break;

case 5:

D=B.Depth();

if(D==0)cout<<"\nEmpty Tree";

else cout<<"\n Depth of Tree "<<D;

break;

case 6:

cout<<"\nMirror\n";

B.mirror();

break;

case 7:

cout<<"\n Levelwise Display\n";

B.Levelwise();

break;

case 8:

cout<<"\nEnter Data to Delete";

cin>>D;

B.Delete(D);

cout<<"\n Inorder Display";

B.inOrder(B.return\_Root());

break;

}

}while(ch<9);

}

Output: