/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Reference:--https://www.educative.io/edpresso/how-to-find-the-height-of-a-binary-tree

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include <iostream>

using namespace std;

/\*

Experiment No. 2 : Create binary search tree.Find height of the tree and print leaf nodes.

Find mirror image, print original and mirror image using

level-wise printing.

\*/

struct node

{ int data;

node \*left,\*right;

};

class tree

{public:

node \*root,\*temp;

int height1(node \*T);//recursive counterpart of height()

int print0(node \*T);//recursive counterpart of count\_leaf\_nodes()

node \* mirror1(node \*T);//recursive counterpart of mirror()

tree() { root=NULL;}

void create();

void insert(node \*,node \*);

int height(){return(height1(root));}

int print\_leaf\_nodes(){return(print0(root));}

void level\_wise();//level wise traversal

void preorder(node \*);

void min(node \*);

int count(node \*);

void search(node \*,int);

};

class Q

{

node \*data[30];

int R,F;

public:

Q(){ R=F=-1; }

void init()

{

R=F=-1;

}

int empty()

{

if(R==-1)

return 1;

return 0;

}

void insert(node \*p)

{

if(empty())

R=F=0;

else

R=R+1;

data[R]=p;

}

node \*Delete()

{

node \*p=data[F];

if(R==F)

R=F=-1;

else

F=F+1;

return(p);

}

};

int tree::height1(node \*T)

{

if(T==NULL)

return(0);

if(T->left==NULL && T->right==NULL)

return(0);

return(max(height1(T->left),height1(T->right))+1);

}

int tree::count(node \*T)

{

if(T==NULL)

return(0);

if(T->left==NULL && T->right==NULL)

return(1);

return(max(count(T->left),count(T->right))+1);

}

int tree::print0(node \*T)

{

if(T==NULL)

return(0);

if(T->left==NULL && T->right==NULL)

{

cout<<" "<<T->data;

return(1);

}

return(print0(T->left)+print0(T->right));

}

void tree::create()

{

root=NULL;

char ch;

do{

temp=new node;

cout<<" enter data";

cin>>temp->data;

temp->left=NULL;

temp->right=NULL;

if(root==NULL)

root=temp;

else

{

insert(root,temp);

}

cout<<"do u want to continue";

cin>>ch;

}while(ch=='y');

}

void tree::insert(node \*root,node \*temp)

{char ch1;

if(temp->data<root->data)

{if(root->left==NULL)

root->left=temp;

else

insert(root->left,temp);

}

else if(temp->data>root->data)

{if(root->right==NULL)

root->right=temp;

else

insert(root->right,temp);

}

}

node \* tree::mirror1(node \*T)

{

node \*temp;

if(T==NULL)

return NULL;

else

{

temp=T->left;

T->left=mirror1(T->right);

T->right=mirror1(temp);

return T;

}

}

void tree::level\_wise()

{

Q q1,q2;

node \*p1,\*p2;

node \*T=root;

if(T==NULL)

return;

q1.insert(T);

cout<<"\n "<<T->data;

while(!q1.empty())

{ /\*Replace all nodes of the queue 'q1' with the nodes at the

next level.Store nodes of next level in 'q2' \*/

cout<<"\n";

q2.init();

while(!q1.empty())

{

p1=q1.Delete();

if(p1->left !=NULL)

{

q2.insert(p1->left);

cout<<" "<<p1->left->data;

}

if(p1->right !=NULL)

{

q2.insert(p1->right);

cout<<" "<<p1->right->data;

}

}

q1=q2;

}

}

void tree::preorder(node \*root)

{

if(root!=NULL)

{

cout<<root->data;

preorder(root->left);

preorder(root->right);

}

}

void tree::min(node \*root)

{

while(root->left!=NULL)

root=root->left;

cout<<root->data;

}

void tree::search(node \* root,int x)

{

int flag=0;

while(root!=NULL)

{

if(x<root->data)

{

root=root->left;

}

else if(x>root->data)

{

root=root->right;

}

else if(x==root->data)

{

flag=1;

break;

}

}

if(flag==1)

cout<<"data found";

else

cout<<"not found";

}

int main()

{

tree t1;

int xx,op,x,c;

do

{

cout<<"\n\n1)Create\n2)Mirror\n3)Print leaf nodes";

cout<<"\n4)Height\n5)preorder display\n 6.minimum value\n 7 count\n8.Search";

cout <<"\nEnter Your Choice :";

cin>>op;

switch(op)

{

case 1: t1.create();break;

case 2: cout<<"\n level Wise traversal on original tree \n";

t1.level\_wise();

t1.root=t1.mirror1(t1.root);

cout<<"\n level Wise traversal on mirror tree \n";

t1.level\_wise();

break;

case 3: xx=t1.print\_leaf\_nodes();

cout<<"\nNo of leaf nodes= "<<xx;break;

case 4: cout<<"\nHeight = "<<t1.height();break;

case 5:

t1.preorder(t1.root);

break;

case 6: t1.min(t1.root);

break;

case 7: c=t1.count(t1.root);

cout<<"no of leaf nodes"<<c;

break;

case 8:

cout<<"enter element to search";

cin>>x;

t1.search(t1.root,x);

break;

}

}while(op!=9);

return 0;

}