CODE:

#include <iostream>

using namespace std;

class bst

{

public :

int id;

bst \*right;

bst \*left;

bst()

{

}

bst(int value)

{

id = value;

left = right = NULL;

}

bst \* Insert(bst\* root, int value)

{

bst \*j=NULL;

j=new bst;

j->right=NULL;

j->left=NULL;

j->id=value;

if(root==NULL)

{

root=j;

}

else if (value > root->id)

{

if(root->right==NULL)

{

root->right=j;

return root;

}

else

root->right = Insert(root->right, value);

}

else

{

if(root->left==NULL)

{

root->left=j;

return root;

}

root->left = Insert(root->left, value);

}

return root;

}

bst\* search(bst\* root, int key)

{

// Base Cases: root is null or key is present at root

if (root == NULL || root->id == key)

return root;

// Key is greater than root's key

if (root->id < key)

return search(root->right, key);

// Key is smaller than root's key

return search(root->left, key);

}

void display(bst\* root)

{

if (root != NULL)

{

display(root->left);

cout<<root->id<<" ";

display(root->right);

}

}

int maxDepth(bst\* node)

{

if (node == NULL)

return 0;

else

{

/\* compute the depth of each subtree \*/

int lDepth = maxDepth(node->left);

int rDepth = maxDepth(node->right);

/\* use the larger one \*/

if (lDepth > rDepth)

return(lDepth + 1);

else return(rDepth + 1);

}

}

void mirror(bst\* node)

{

if (node == NULL)

return;

else

{

bst\* temp;

/\* do the subtrees \*/

mirror(node->left);

mirror(node->right);

/\* swap the pointers in this node \*/

temp = node->left;

node->left = node->right;

node->right = temp;

}

}

bst\* deleteNode(bst\* root, int key) {

if(root == NULL || root->id == key) return deleteRoot(root);

bst\* curr = root;

while(1) {

int x = curr->id;

if(key < x){

if(curr->left == NULL || curr->left->id == key){

curr->left = deleteRoot(curr->left);

break;

}

curr = curr->left;

} else {

if(curr->right == NULL || curr->right->id == key){

curr->right = deleteRoot(curr->right);

break;

}

curr = curr->right;

}

}

return root;

}

bst\* deleteRoot(bst\* root){

if(!root || root->id == 0)return NULL;

if(root->right == NULL) return root->left;

bst\* x = root->right;

while(x->left)x = x->left;

x->left = root->left;

return root->right;

}

};

int main(int argc, char\*\* argv) {

bst \*p,\*q,\*r,\*s,\*t,\*u;

bst a;

p=a.Insert(NULL,9);

q=a.Insert(p,4);

r=a.Insert(p,11);

s=a.Insert(p,2);

t=a.Insert(p,7);

u=a.Insert(p,10);

// 9

// / \

// / \ tree will be like this

// 4 11

// / \ /

// 2 7 10

int w,m;

do

{

// cout<<"Press 1) Insert,2) Display, 3)Depth of the tree, 4) Search a node, 5) Find its mirror image 6) Delete 7) Mirror imagelevel wise."<<endl;

cout<<"1. Insert 2. Delete 3.Depth of the tree 4. Search a node 5. Find its mirror image 6. Display 7. Mirror imagelevel wise.";

cin>>m;

switch(m)

{

case 1:

cout<<"Enter element :"<<endl;

int r;

cin>>r;

if(a.search(p,r)==NULL)

{

a.Insert(t,r);

cout<<endl;

cout<<"Insert succesfully"<<endl;

}

else{

cout<<"Node is already present"<<endl;

}

break;

case 2:

int o;

cout<<"Enter node to delete"<<endl;

cin>>o;

if(a.search(t,o)!=NULL)

{

a.deleteNode(t,o);

cout<<endl;

cout<<"After delete"<<endl;

a.display(p);

cout<<endl;

}

else{

cout<<"Node is not present in given Binary search tree"<<endl;

}

break;

case 3:

cout<<"Depth :"<<a.maxDepth(t)<<endl;

break;

case 4:

int w;

cout<<"Enter node to search"<<endl;

cin>>w;

a.search(p,w);

break;

case 5:

a.mirror(p);

cout<<"After mirror image"<<endl;

a.display(p);

cout<<endl;

break;

case 6:

a.display(p);

break;

case 7:

cout<<"After mirror image level wise :"<<endl;

a.mirror(p);

cout<<endl;

break;

}

cout<<"Press 1 for continue......"<<endl;

cin>>w;

}while(w==1);

return 0;

}

Output:

