```
/*Implement binary search tree and perform following operations:
a) Insert (Handle insertion of duplicate entry)
b) Delete
c) Search
d) Display tree (Traversal)
e) Display - Depth of tree
f) Display - Mirror image
g) Create a copy
h) Display all parent nodes with their child nodes
i) Display leaf nodes
j) Display tree level wise
*/
#include <iostream>
#include<stdlib.h>
using namespace std;
struct node
{
int data;
struct node *left;
struct node *right;
};
node *insert(node *root, int val){
if (root == NULL){
node *temp; //new node temp
temp=new node;
temp->data=val;
temp->left=temp->right=NULL; //left and right is NULL bcz only one
```

```
node create;
return temp; // return single node
}
if (val < root->data){
root->left = insert(root->left, val);
}
else{
//val>root->data
root->right = insert(root->right, val);
}
return root;
}
void inorder(node *root)
{
if (root == NULL){
return;
}
inorder(root->left);
cout << root->data << " ";
inorder(root->right);
}
node* inorderSucc(node* root){
node* curr=root;
while(curr && curr->left!=NULL){
curr=curr->left;
}
return curr;
}
node *delet(node *root, int key){
```

```
if(key<root->data){
root->left=delet(root->left, key);
}
else if(key>root->data){
root->right=delet(root->right,key);
}
//if key==root->data
else{
if(root->left==NULL){
node* temp=root->right;
free(root);
return temp;
}
else if(root->right==NULL){
node* temp=root->left;
free(root);
return temp;
}
node* temp=inorderSucc(root->right);
root->data=temp->data;
root->right=delet(root->right, temp->data);
}
return root;
}
node *search(node* root, int val){
if(root==NULL){
return NULL;
}
if(val>root->data){
```

```
return search(root->right, val);
}
else if(val<root->data){
return search(root->left, val);
}
else{
return root;
}
}
void mirrorImg(node* root){
if(root==NULL){
return;
}
else{
struct node *temp;
mirrorImg(root->left);
mirrorImg(root->right);
swap(root->left,root->right);
}
}
node *copy(node *root){
node *temp=NULL;
if(root!=NULL){
temp=new node();
temp->data=root->data;
temp->left=copy(root->left);
temp->right=copy(root->right);
}
return temp;
```

```
}
void leafNodes(node* root){
if(root==NULL){
return;
}
if(!root->left && !root->right){
cout<<root->data<<" ";
return;
}
if(root->right)
leafNodes(root->right);
if(root->left)
leafNodes(root->left);
}
int calHeight(node* root){
if(root==NULL){
return 0;
}
int lheight=calHeight(root->left);
int rheight=calHeight(root->right);
return max(lheight,rheight)+1;
}
node *findMin(node *root){
if(root==NULL){
return NULL;
}
if(root->left)
return findMin(root->left);
else
```

```
return root;
}
node *findMax(node *root){
if(root==NULL){
return NULL;
}
if(root->right)
return findMax(root->right);
else
return root;
}
int main()
{
node *root=NULL, *temp; //initially tree is NULL
int ch;
while (1){
cout<<"\n\n\t1)Insert" << endl;
cout<<"\t2)Delete" << endl;
cout<<"\t3)Search" << endl;
cout<<"\t4)Create the copy "<<endl;
cout<<"\t5)Display leaf nodes "<<endl;
cout<<"\t6)Height of the tree"<<endl;
cout<<"\t7)Find the minimum"<<endl;
cout<<"\t8)Find the maximum"<<endl;
cout<<"\t9)Mirror image"<<endl;
cout<<"\t10)Exit"<<endl;
cout<<"\nEnter your choice: ";
cin>>ch;
switch (ch){
```

```
case 1:
cout << "Enter the element to be insert: ";
cin >> ch;
root= insert(root, ch);
cout << "*****Elements in BST are*****: ";
inorder(root);
break;
case 2:
cout<<"Enter the element to be deleted: ";
cin>>ch;
root=delet(root, ch);
cout<<"Element deleted successfully !!";
cout<<"\n*****After deletion the elements in the BST are*****: ";
inorder(root);
break;
case 3:
cout<<"Enter the element to be searched: ";
cin>>ch;
temp=search(root, ch);
if(temp==NULL){
cout<<"*****Element is not found*****";
}
else{
cout<<"*****Element is found*****";
}
break;
case 4:
cout<<"The copy of the tree is: ";
root=copy(root);
```

```
inorder(root);
break;
case 5:
cout<<"The leaf nodes are: ";
leafNodes(root);
break;
case 6:
cout<<"Height of the binary search tree is: "<<calHeight(root);
break;
case 7:
temp=findMin(root);
cout<<"\nMinimum element is : "<<temp->data;
break;
case 8:
temp=findMax(root);
cout<<"\nMaximum element is: "<<temp->data;
break;
case 9:
cout<<" inorder tree: ";
inorder(root);
cout<<endl;
mirrorImg(root);
cout<<"mirror image is: ";
inorder(root);
break;
case 10:
return 0;
default:
cout<<"\nInvalid choice !! Please enter your choice again";
```

```
}
}
return 0;
}
```