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

struct node
{
	int key;
	struct node *left, *right;
};

struct node *newNode(int item)
{
	struct node *temp = (struct node *)malloc(sizeof(struct node));
	temp->key = item;
	temp->left = temp->right = NULL;
	return temp;
}
void inorder(struct node *root)
{
	if(root != NULL)
	{
		inorder(root->left);
		cout << root->key << " -> ";
		inorder(root->right);
	}
}
void preorder(struct node *root)
{
	if(root != NULL)
	{
		cout << root->key << " -> ";
		preorder(root->left);
		preorder(root->right);
	}
}
void postorder(struct node *root)
{
	if(root!=NULL)
	{
		postorder(root->left);
		postorder(root->right);
		cout << root->key << " -> ";
	}
}

void display1(struct node *root)
{
	if(root != NULL)
	{
		display1(root->left);
```

```
                display1(root->right);
                if(root->left==NULL&&root->right==NULL)
                cout << root->key << " -> ";
        }
}
struct node *insert(struct node *node, int key)
{
        if (node == NULL)
                return newNode(key);
        if(key < node->key)
                node->left = insert(node->left, key);
        else
                node->right = insert(node->right, key);
        return node;
}
void search(struct node *root)
{
        int data;
        node *temp = new node;
        temp = root;
        cout<<"\nEnter the elements to be searched: ";
        cin>>data;
        while(temp != NULL)
        {
                if(temp->key == data)
                {
                        cout<<"\ndata found";
                        return;
                }
                else if(temp->key > data)
                        temp=temp->left;
                else
                        temp=temp->right;
        }
        cout<<"\nData not found";
        return;
}
struct node* minValueNode(struct node* node)
{
        struct node* current = node;
        while (current && current->left != NULL)
        current = current ->left;
        return current;
}
struct node* deleteNode (struct node* root , int key)
{
        if(root == NULL)
        return root;
        if(key < root-> key)
                root ->left = deleteNode(root ->left , key);
```

```
        else if (key >root -> key )
                root -> right = deleteNode(root ->right, key);
        else
        {
                if(root ->left==NULL and root ->right==NULL)
                        return NULL;
                else if (root ->left == NULL)
                {
                        struct node* temp = root->right;free(root );
                        return temp;
                }
                else if (root ->right ==NULL)
                {
                        struct node* temp = root -> left;
                        free (root);
                        return temp;
                }
                struct node* temp = minValueNode(root->right);
                root-> key = temp->key;
                root ->right = deleteNode(root ->right, temp ->key);
        }
        return root;
}
void mirror(struct node* node)
{
        if(node == NULL)
                return;
        else
        {
                struct node* temp;
                mirror(node -> left);
                mirror(node ->right);
                temp= node ->left;
                node->left = node ->right;
                node ->right= temp;
        }
}

int maxDepth(node* node )
{
        if (node ==NULL)
                return 0;
        else
        {
                int lDepth = maxDepth(node -> left);
                int rDepth = maxDepth(node -> right );
                if (lDepth > rDepth)
                        return (lDepth+1);
                else
                        return(rDepth+1);
```

```
        }
}
int main()
{
        struct node *root = NULL;
        root =insert(root,50);
        root =insert(root,30);
        root =insert(root,20);
        root =insert(root,40);
        root =insert(root,70);
        root =insert(root,60);
        root =insert(root,80);

        cout << "\n Inorder traversal:- ";
        inorder(root);
        cout << "\n Preorder traversal:- ";
        preorder(root);
        cout << "\n Postorder traversal:- ";
        postorder(root);
        cout <<"\n The leaves are:\n ";
        display1(root);
        cout <<"\n searching node::\n " ;
        search(root);
        cout << "\nDelete 20\n";
        root = deleteNode(root, 20);
        cout << "Inoder traversal of the modified tree \n";
        inorder(root);

        cout << "\nDelete 30\n";
        root = deleteNode(root, 30);
        cout << "Inorder traversal of the modified tree \n";
        inorder(root);
        cout << "\nDelete 50\n";
        root = deleteNode(root, 50);
        cout << "Inorder traversal of the modified tree \n";
        inorder(root);
        mirror(root);
        cout << "\nInorder traversal of the mirror tree" << " is \n";
        inorder(root);
        cout << "\n Height of tree is:- " << maxDepth(root);
}
```

.....

Output:
Inorder traversal:- 20 -> 30 -> 40 -> 50 -> 60 -> 70 -> 80 ->
Preorder traversal:- 50 -> 30 -> 20 -> 40 -> 70 -> 60 -> 80 ->
Postorder traversal:- 20 -> 40 -> 30 -> 60 -> 80 -> 70 -> 50 ->
The leaves are:
20 -> 40 -> 60 -> 80 ->
searching node::

Enter the elements to be searched: 60

data found
Delete 20
Inoder traversal of the modified tree
30 -> 40 -> 50 -> 60 -> 70 -> 80 ->
Delete 30
Inorder traversal of the modified tree
40 -> 50 -> 60 -> 70 -> 80 ->
Delete 50
Inorder traversal of the modified tree
40 -> 60 -> 70 -> 80 ->
Inorder traversal of the mirror tree is
80 -> 70 -> 60 -> 40 ->
Height of tree is:- 3