#include<iostream>

using namespace std;

class Node{

    public:

        int data;

        Node\* left;

        Node\* right;

        Node(int data){

            this->data=data;

            left=NULL;

            right=NULL;

        }

};

class BST{

    public:

        Node\* root;

        BST(){

            root=NULL;

        }

        Node\* SearchinBST(Node\* root,int val)

        {

            if(root ==NULL)

            {

                cout<<"tree is empty"<<endl;

                return NULL;

            }

            if(root->data==val)

            {

                cout<<"you can insert and operate with this tree"<<endl;

                return root;

            }

            if(root->data > val)//move to left side of tree

            {

                return SearchinBST(root->left, val);

            }

            else{//move to right side of tree

                return SearchinBST(root->right, val);

            }

        }

        Node\* InsertinBST(Node\* r, int val )

        {

            if (r==NULL)

            {

                //creating new node for inserting value

                Node\* t= new Node(val);

            if (r==root)

            {

                root= r=t;

            }

            else

            {

                r=t;

            }

            return r;

            }

            else if (r->data== val){

                cout<<"Duplicate values are: "<<val;

                return r;

            }

            else if (val < r->data)

                r->left = InsertinBST(r->left , val );

            else if (val > r->data)

                r->right= InsertinBST( r->right,val);

        }

        Node\* DeleteinBST(Node\* root, int data)

        {

            if(root==NULL)

            //no deletion because tree is empty

             return root;

            else if (data<root->data)

                root->left = DeleteinBST(root->left, data);

            else if (data> root->data)

                root->right = DeleteinBST(root->right, data);

            else

            {

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

                {

                    delete root;

                    root = NULL;

                    return root;

                }

                else if(root->right == NULL)

                {

                    Node\* temp = root;

                    root= root->left;

                    delete temp;

                }

                else if(root->left == NULL)

                {

                    Node\* temp = root;

                    root= root->right;

                    delete temp;

                }

                else

                {

                    Node\* temp = MaxinBST(root->left);

                    root->data = temp->data;

                    root->left = DeleteinBST(root->left, temp->data);

                }

            }

            return root;

        }

        void inOrderTraversal(Node\* r)

        {

            if (r==NULL)

              return ;

            inOrderTraversal(r->left);

            cout<<" "<< r->data <<" -> ";

            inOrderTraversal(r->right);

        }

        Node\* MaxinBST(Node\* r)//find maximum value

        {

            while(r->right!=NULL){

                r= r->right;

            }

            return r;

            cout<<"Maximum value is "<<r<<endl;

        }

};

int main (){

    BST tree;

    tree.InsertinBST(tree.root, 1);

    tree.InsertinBST(tree.root, 2);

    tree.InsertinBST(tree.root, 3);

    tree.InsertinBST(tree.root, 1);

    tree.InsertinBST(tree.root, 4);

    tree.InsertinBST(tree.root, 3);

    tree.InsertinBST(tree.root, 1);

    tree.InsertinBST(tree.root, 10);

    tree.InsertinBST(tree.root, 42);

    tree.InsertinBST(tree.root, 12);

    cout<<endl;

    if(tree.SearchinBST(tree.root, 1)==NULL)

    {

        cout<<"Value not found in tree";

     }

     else

     {

        cout<<"Value found successfuly";

     }

    tree.DeleteinBST(tree.root, 4);

    cout<<endl;

    tree.inOrderTraversal(tree.root);

    return 0;

}