**Name:** Rohan Arun Nalawade

**PRN:** 22310407

**Roll No:** 231012

**Assignment 1**

**Binary Search Tree**

**Code:**

#include <iostream>

#include <vector>

#include <queue>

using namespace std;

struct Node

{

    int data;

    Node\* left;

    Node\* right;

    Node(int value): data(value), left(nullptr), right(nullptr) {}

};

class BST

{

private:

    Node\* root;

    // Insert a value into the tree

    Node\* insert(Node\* node, int value)

    {

        if (!node) return new Node(value);

        if (value < node->data)

            node->left = insert(node->left, value);

        else if (value > node->data)

            node->right = insert(node->right, value);

        return node;

    }

    void inOrderTraversal(Node\* node)

    {

        if (node)

        {

            inOrderTraversal(node->left);

            cout << node->data << " ";

            inOrderTraversal(node->right);

        }

    }

    void preOrderTraversal(Node\* node)

    {

        if (node)

        {

            cout << node->data << " ";

            preOrderTraversal(node->left);

            preOrderTraversal(node->right);

        }

    }

    void postOrderTraversal(Node\* node)

    {

        if (node)

        {

            postOrderTraversal(node->left);

            postOrderTraversal(node->right);

            cout << node->data << " ";

        }

    }

    Node\* deleteNode(Node\* node, int value)

    {

        if (!node) return nullptr;

        if (value < node->data)

            node->left = deleteNode(node->left, value);

        else if (value > node->data)

            node->right = deleteNode(node->right, value);

        else

        {

            if (!node->left)

            {

                Node\* temp = node->right;

                delete node;

                return temp;

            }

            else if (!node->right)

            {

                Node\* temp = node->left;

                delete node;

                return temp;

            }

            Node\* temp = findMin(node->right);

            node->data = temp->data;

            node->right = deleteNode(node->right, temp->data);

        }

        return node;

    }

    Node\* findMin(Node\* node)

    {

        while (node && node->left)

            node = node->left;

        return node;

    }

    bool search(Node\* node, int value)

    {

        if (!node) return false;

        if (value == node->data) return true;

        if (value < node->data)

            return search(node->left, value);

        return search(node->right, value);

    }

    int height(Node\* node)

    {

        if (!node) return -1;

        int leftHeight = height(node->left);

        int rightHeight = height(node->right);

        return max(leftHeight, rightHeight) + 1;

    }

    void levelWiseDisplay(Node\* node)

    {

        if (!node) return;

        queue<Node\*> q;

        q.push(node);

        while (!q.empty())

        {

            Node\* current = q.front();

            q.pop();

            cout << current->data << " ";

            if (current->left) q.push(current->left);

            if (current->right) q.push(current->right);

        }

    }

    void mirror(Node\* node)

    {

        if (node)

        {

            swap(node->left, node->right);

            mirror(node->left);

            mirror(node->right);

        }

    }

public:

    BST(): root(nullptr) {}

    void insertValues(const vector<int>& values)

    {

        for (int value : values)

        {

            root = insert(root, value);

        }

    }

    void deleteValues(const vector<int>& values)

    {

        for (int value : values)

        {

            root = deleteNode(root, value);

        }

    }

    bool search(int value)

    {

        return search(root, value);

    }

    void inOrder() { inOrderTraversal(root); cout << endl; }

    void preOrder() { preOrderTraversal(root); cout << endl; }

    void postOrder() { postOrderTraversal(root); cout << endl; }

    void displayHeight()

    {

        cout << "Height of Tree: " << height(root) << endl;

    }

    void displayLevelWise()

    {

        cout << "Level-wise Display: ";

        levelWiseDisplay(root);

        cout << endl;

    }

    void mirrorTree()

    {

        mirror(root);

    }

};

int main()

{

    BST tree;

    int choice;

    do

    {

        cout << "\nBinary Search Tree Operations:\n";

        cout << "1. Insert Values\n2. Delete Values\n3. Search\n4. In-order Traversal\n5. Pre-order Traversal\n6. Post-order Traversal\n";

        cout << "7. Display Height\n8. Level-wise Display\n9. Mirror Tree\n10. Exit\n";

        cout << "Enter your choice: ";

        cin >> choice;

        switch (choice)

        {

            case 1:

            {

                int n;

                cout << "Enter the number of values to insert: ";

                cin >> n;

                vector<int> values(n);

                cout << "Enter the values: ";

                for (int i = 0; i < n; ++i)

                {

                    cin >> values[i];

                }

                tree.insertValues(values);

                break;

            }

            case 2:

            {

                int n;

                cout << "Enter the number of values to delete: ";

                cin >> n;

                vector<int> values(n);

                cout << "Enter the values: ";

                for (int i = 0; i < n; ++i)

                {

                    cin >> values[i];

                }

                tree.deleteValues(values);

                break;

            }

            case 3:

            {

                int value;

                cout << "Enter value to search: ";

                cin >> value;

                if (tree.search(value))

                    cout << "Value found in the tree.\n";

                else

                    cout << "Value not found in the tree.\n";

                break;

            }

            case 4:

                cout << "In-order Traversal: ";

                tree.inOrder();

                break;

            case 5:

                cout << "Pre-order Traversal: ";

                tree.preOrder();

                break;

            case 6:

                cout << "Post-order Traversal: ";

                tree.postOrder();

                break;

            case 7:

                tree.displayHeight();

                break;

            case 8:

                tree.displayLevelWise();

                break;

            case 9:

                tree.mirrorTree();

                cout << "Tree mirrored successfully.\n";

                break;

            case 10:

                cout << "Exiting...\n";

                break;

            default:

                cout << "Invalid choice. Please try again.\n";

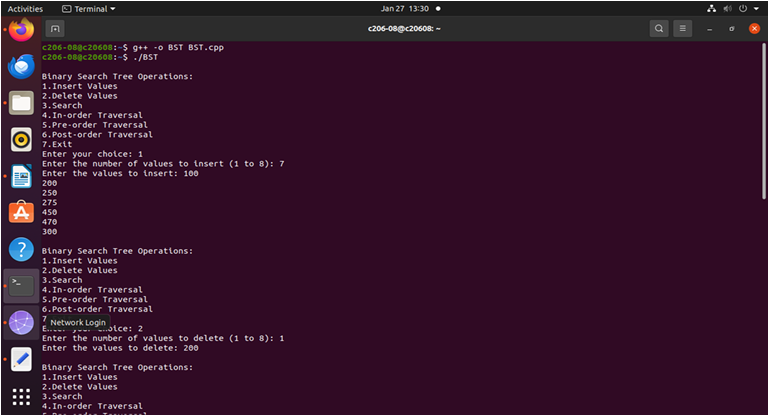
        }

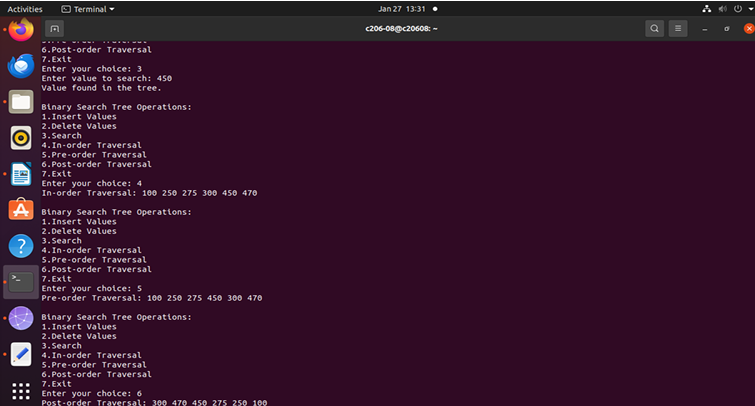
    } while (choice != 10);

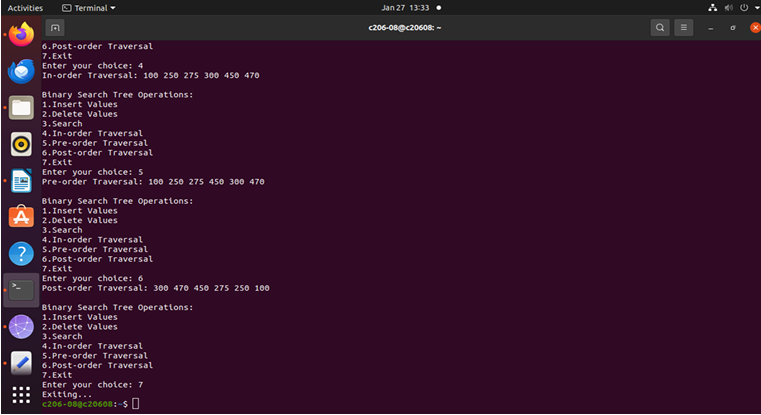
    return 0;

}

**Output:**

****

****

****