Name: Siddhant Kumar Sahu

Batch: E3, 57

PRN: 202301070159

CODE

```
#include <bits/stdc++.h>
using namespace std;

struct Node {
    int key, height;
    Node* left;
    Node* right;
    Node(int value) : key(value), height(1), left(NULL), right(NULL) {}
};

int getHeight(Node* node) {
    return node ? node->height : 0;
}

int getBalanceFactor(Node* node) {
    return node ? getHeight(node->left) - getHeight(node->right) : 0;
}

Node* rotateRight(Node* y) {
    Node* x = y->left;
    Node* T2 = x->right;
    x->right = y;
    y->left = T2;
    y->height = max(getHeight(y->left), getHeight(y->right)) + 1;
    x->height = max(getHeight(x->left), getHeight(x->right)) + 1;
    return x;
}

Node* rotateLeft(Node* x) {
    Node* y = x->right;
    Node* T2 = y->left;
    y->left = x;
    x->right = T2;
    x->height = max(getHeight(x->left), getHeight(x->right)) + 1;
    y->height = max(getHeight(y->left), getHeight(y->right)) + 1;
    return y;
}

Node* Insert(Node* node, int key) {
    if (!node) return new Node(key);
    if (key < node->key) node->left = Insert(node->left, key);
    else if (key > node->key) node->right = Insert(node->right, key);
    else return node;

    node->height = max(getHeight(node->left), getHeight(node->right)) + 1;

    int balance = getBalanceFactor(node);

    if (balance > 1 && key < node->left->key) return rotateRight(node);
    if (balance < -1 && key > node->right->key) return rotateLeft(node);
    if (balance > 1 && key > node->left->key) {
        node->left = rotateLeft(node->left);
        return rotateRight(node);
    }
    if (balance < -1 && key < node->right->key) {
        node->right = rotateRight(node->right);
        return rotateLeft(node);
    }
    return node;
}

void inOrder(Node* Root) {
    if (Root) {
        inOrder(Root->left);
        cout << Root->key << " ";
        inOrder(Root->right);
    }
}

void preOrder(Node* Root) {
    if (Root) {
        cout << Root->key << " ";
        preOrder(Root->left);
        preOrder(Root->right);
    }
}

void postOrder(Node* Root) {
    if (Root) {
        postOrder(Root->left);
        postOrder(Root->right);
        cout << Root->key << " ";
    }
}

Node* search(Node* Root, int key) {
    if (!Root || Root->key == key) return Root;
    if (key < Root->key) return search(Root->left, key);
    return search(Root->right, key);
}

Node* minValueNode(Node* node) {
    Node* current = node;
    while (current->left) current = current->left;
    return current;
}

Node* deleteNode(Node* Root, int key) {
    if (!Root) 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 || !Root->right) {
            Node* temp = Root->left ? Root->left : Root->right;
            if (!temp) {
                temp = Root;
                Root = NULL;
            } else *Root = *temp;
            delete temp;
        } else {
            Node* temp = minValueNode(Root->right);
            Root->key = temp->key;
            Root->right = deleteNode(Root->right, temp->key);
        }
    }

    if (!Root) return Root;

    Root->height = max(getHeight(Root->left), getHeight(Root->right)) + 1;

    int balance = getBalanceFactor(Root);

    if (balance > 1 && getBalanceFactor(Root->left) >= 0) return
rotateRight(Root);
    if (balance > 1 && getBalanceFactor(Root->left) < 0) {
        Root->left = rotateLeft(Root->left);
        return rotateRight(Root);
    }
    if (balance < -1 && getBalanceFactor(Root->right) <= 0) return
rotateLeft(Root);
    if (balance < -1 && getBalanceFactor(Root->right) > 0) {
        Root->right = rotateRight(Root->right);
        return rotateLeft(Root);
    }
```

```cpp
    return Root;
}

int main() {
    Node* Root = NULL;
    Root = Insert(Root,45);
    Root = Insert(Root,50);
    Root = Insert(Root,10);
    Root = Insert(Root,30);
    Root = Insert(Root,5);
    Root = Insert(Root,55);
    Root = Insert(Root,48);
    Root = Insert(Root,60);
    int choice, value;
    do {
        cout << "\nMenu:\n";
        cout << "1. Insert Node\n2. In-Order (Recursive)\n3. Pre-Order
(Recursive)\n4. Post-Order (Recursive)\n5. Search Node\n6. Delete Node\n7.
Exit\n";
        cout << "Enter your choice: ";
        cin >> choice;
        switch (choice) {
            case 1:
                cout << "Enter value to insert: ";
                cin >> value;
                Root = Insert(Root, value);
                break;
            case 2:
                cout << "In-Order Traversal: ";
                inOrder(Root);
                cout << endl;
                break;
            case 3:
                cout << "Pre-Order Traversal: ";
                preOrder(Root);
                cout << endl;
                break;
            case 4:
                cout << "Post-Order Traversal: ";
                postOrder(Root);
                cout << endl;
                break;
            case 5:
                cout << "Enter value to search: ";
                cin >> value;
                if (search(Root, value))
                    cout << "Node found!\n";
                else
                    cout << "Node not found.\n";
                break;
            case 6:
                cout << "Enter value to delete: ";
                cin >> value;
                Root = deleteNode(Root, value);
                cout << "Node deleted if found.\n";
                break;
            case 7:
                cout << "Exiting Program.\n";
                break;
            default:
                cout << "Invalid choice. Try again.\n";
        }
    } while (choice != 7);
    return 0;
}
```

OUTPUT

```
PS D:\College Assignments\Sem - 4 SY-Btech\Advanced Data Structure(ADS)\Practical - 3> cd "d:\College Assignments\Sem - 4 SY-Btech\Advanced Data Structure(ADS)\Practical - 3\" ; if ($?) { g++ AVLTree.cpp -o AVLTree } ; if ($?) { .\AVLTree }


Menu:

1. Insert Node

2. In-Order (Recursive)

3. Pre-Order (Recursive)

4. Post-Order (Recursive)

5. Search Node

6. Delete Node

7. Exit

Enter your choice: 1

Enter value to insert: 57


Menu:
```

```
1. Insert Node
2. In-Order (Recursive)
3. Pre-Order (Recursive)
4. Post-Order (Recursive)
5. Search Node
6. Delete Node
7. Exit
Enter your choice: 2
In-Order Traversal: 5 10 30 45 48 50 55 57 60

Menu:
1. Insert Node
2. In-Order (Recursive)
3. Pre-Order (Recursive)
4. Post-Order (Recursive)
5. Search Node
6. Delete Node
7. Exit
Enter your choice: 3
Pre-Order Traversal: 45 10 5 30 50 48 57 55 60

Menu:
1. Insert Node
2. In-Order (Recursive)
3. Pre-Order (Recursive)
4. Post-Order (Recursive)
5. Search Node
6. Delete Node
7. Exit
```

```
Enter your choice: 4
Post-Order Traversal: 5 30 10 48 55 60 57 50 45

Menu:
1. Insert Node
2. In-Order (Recursive)
3. Pre-Order (Recursive)
4. Post-Order (Recursive)
5. Search Node
6. Delete Node
7. Exit
Enter your choice: 5
Enter value to search: 45
Node found!

Menu:
1. Insert Node
2. In-Order (Recursive)
3. Pre-Order (Recursive)
4. Post-Order (Recursive)
5. Search Node
6. Delete Node
7. Exit
Enter your choice: 5
Enter value to search: 57
Node found!

Menu:
1. Insert Node
```

```
2. In-Order (Recursive)
3. Pre-Order (Recursive)
4. Post-Order (Recursive)
5. Search Node
6. Delete Node
7. Exit
Enter your choice: 6
Enter value to delete: 57
Node deleted if found.

Menu:
1. Insert Node
2. In-Order (Recursive)
3. Pre-Order (Recursive)
4. Post-Order (Recursive)
5. Search Node
6. Delete Node
7. Exit
Enter your choice: 7
Exiting Program.
PS D:\College Assignments\Sem - 4 SY-Btech\Advanced Data Structure(ADS)\Practical - 3>
```