Name: Siddhant Kumar Sahu

Batch: E3, 57

PRN: 202301070159

CODE

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

struct Tree {
    int data;
    Tree *left, *right;
    bool leftThread, rightThread; // Flags for threading
};

Tree* createTree(int data) {
    Tree* newTree = new Tree();
    newTree->data = data;
    newTree->left = newTree->right = NULL;
    newTree->leftThread = newTree->rightThread = true;
    return newTree;
}

Tree* Insert(Tree* Root, int key) {
    if (!Root) return createTree(key);
    Tree* current = Root;
    Tree* parent = NULL;
    while (current) {
        if (key == current->data) {
            cout << "Duplicate keys are not allowed." << endl;
            return Root;
        }
        parent = current;
        if (key < current->data) {
            if (current->leftThread) break;
            current = current->left;
        } else {
            if (current->rightThread) break;
            current = current->right;
        }
    }
    Tree* newTree = createTree(key);
    if (key < parent->data) {
        newTree->left = parent->left;
        newTree->right = parent;
        parent->leftThread = false;
        parent->left = newTree;
    } else {
        newTree->right = parent->right;
        newTree->left = parent;
        parent->rightThread = false;
        parent->right = newTree;
    }
    return Root;
}

Tree* leftmost(Tree* node) {
    while (node && !node->leftThread) node = node->left;
    return node;
}

void inOrderNonRecursive(Tree* Root) {
    Tree* current = leftmost(Root);
    while (current) {
        cout << current->data << " ";
        if (current->rightThread) {
            current = current->right;
        } else {
            current = leftmost(current->right);
        }
    }
    cout << endl;
}

void preOrderNonRecursive(Tree* Root) {
    Tree* current = Root;
    while (current) {
        cout << current->data << " ";
        if (!current->leftThread) {
            current = current->left;
        } else if (!current->rightThread) {
            current = current->right;
        } else {
            while (current && current->rightThread) current = current->right;
            if (current) current = current->right;
        }
    }
    cout << endl;
}

void inOrderRecursive(Tree* Root) {
    if (!Root) return;
    if (!Root->leftThread) inOrderRecursive(Root->left);
    cout << Root->data << " ";
    if (!Root->rightThread) inOrderRecursive(Root->right);
}

void preOrderRecursive(Tree* Root) {
    if (!Root) return;
    cout << Root->data << " ";
    if (!Root->leftThread) preOrderRecursive(Root->left);
    if (!Root->rightThread) preOrderRecursive(Root->right);
}

void postOrderRecursive(Tree* Root) {
    if (!Root) return;
    if (!Root->leftThread) postOrderRecursive(Root->left);
    if (!Root->rightThread) postOrderRecursive(Root->right);
    cout << Root->data << " ";
}

void postOrderNonRecursive(Tree* Root) {
    stack<Tree*> s1, s2;
    if (!Root) return;
    s1.push(Root);
    while (!s1.empty()) {
        Tree* current = s1.top(); s1.pop();
        s2.push(current);
        if (!current->leftThread && current->left) s1.push(current->left);
        if (!current->rightThread && current->right) s1.push(current->right);
    }
    while (!s2.empty()) {
        cout << s2.top()->data << " ";
        s2.pop();
    }
    cout << endl;
}

Tree* search(Tree* Root, int key) {
    Tree* current = Root;
    while (current) {
        if (key == current->data) return current;
        if (key < current->data) {
            if (current->leftThread) break;
            current = current->left;
        } else {
            if (current->rightThread) break;
            current = current->right;
        }
    }
    return NULL;
}
```

```cpp
Tree* deleteNode(Tree* root, int key) {
    Tree* parent = NULL, *current = root;
    while (current && current->data != key) {
        parent = current;
        if (key < current->data) {
            if (current->leftThread) return root;
            current = current->left;
        } else {
            if (current->rightThread) return root;
            current = current->right;
        }
    }
    if (!current) return root;
    if (current->leftThread && current->rightThread) {
        if (!parent) return NULL;
        if (parent->left == current) {
            parent->left = current->left;
            parent->leftThread = true;
        } else {
            parent->right = current->right;
            parent->rightThread = true;
        }
        delete current;
    } else {
        Tree* child = (!current->leftThread) ? current->left : current->right;
        if (!parent) return child;
        if (parent->left == current) parent->left = child;
        else parent->right = child;
        delete current;
    }
    return root;
}

int main() {
    Tree* 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\n";
        cout << "2. In-Order Traversal (Recursive)\n";
        cout << "3. Pre-Order Traversal (Recursive)\n";
        cout << "4. Post-Order Traversal (Recursive)\n";
        cout << "5. In-Order Traversal (Non-Recursive)\n";
        cout << "6. Pre-Order Traversal (Non-Recursive)\n";
        cout << "7. Post-Order Traversal (Non-Recursive)\n";
        cout << "8. Search Node\n";
        cout << "9. Delete Node\n";
        cout << "9. 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 (Recursive): ";
                inOrderRecursive(Root);
                cout << endl;
                break;
            case 3:
                cout << "Pre-Order (Recursive): ";
                preOrderRecursive(Root);
                cout << endl;
                break;
            case 4:
                cout << "Post-Order (Recursive): ";
                postOrderRecursive(Root);
                cout << endl;
                break;
            case 5:
                cout << "In-Order (Non-Recursive): ";
                inOrderNonRecursive(Root);
                cout << endl;
                break;
            case 6:
                cout << "Pre-Order (Non-Recursive): ";
                preOrderNonRecursive(Root);
                cout << endl;
                break;
            case 7:
                cout << "Post-Order (Non-Recursive): ";
                postOrderNonRecursive(Root);
                cout << endl;
                break;
            case 8:
                cout << "Enter value to search: ";
                cin >> value;
                if (search(Root, value))
                    cout << "Node found!" << endl;
                else
                    cout << "Node not found." << endl;
                break;
            case 9:
                cout << "Enter key to delete: ";
                cin >> value;
                Root = deleteNode(Root, value);
                cout << "Key deleted if found.\n";
                break;
            case 10:
                cout << "Exiting Program";
                break;
            default:
                cout << "Invalid choice. Please try again." << endl;
                break;
        }
    } while (choice != 9);
    return 0;
}
```

OUTPUT

```
PS D:\College Assignments\Sem - 4 SY-Btech\Advanced Data Structure(ADS)> cd "d:\College Assignments\Sem - 4 SY-Btech\Advanced Data Structure(ADS)\Practical - 2\" ; if ($?) { g++ ThreadedBinaryTree.cpp -o ThreadedBinaryTree } ; if ($?) { .\ThreadedBinaryTree }


Menu:

1. Insert Node

2. In-Order Traversal (Recursive)

3. Pre-Order Traversal (Recursive)

4. Post-Order Traversal (Recursive)

5. In-Order Traversal (Non-Recursive)

6. Pre-Order Traversal (Non-Recursive)

7. Post-Order Traversal (Non-Recursive)

8. Search Node

9. Delete Node
```

```
9. Exit
Enter your choice: 1
Enter value to insert: 57

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

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

```
Enter your choice: 3

Pre-Order (Recursive): 45 10 5 30 50 48 55 60 57


Menu:

1. Insert Node

2. In-Order Traversal (Recursive)

3. Pre-Order Traversal (Recursive)

4. Post-Order Traversal (Recursive)

5. In-Order Traversal (Non-Recursive)

6. Pre-Order Traversal (Non-Recursive)

7. Post-Order Traversal (Non-Recursive)

8. Search Node

9. Delete Node

9. Exit

Enter your choice: 4

Post-Order (Recursive): 5 30 10 48 57 60 55 50 45


Menu:

1. Insert Node

2. In-Order Traversal (Recursive)

3. Pre-Order Traversal (Recursive)

4. Post-Order Traversal (Recursive)

5. In-Order Traversal (Non-Recursive)

6. Pre-Order Traversal (Non-Recursive)

7. Post-Order Traversal (Non-Recursive)

8. Search Node

9. Delete Node

9. Exit

Enter your choice: 5
```

```
In-Order (Non-Recursive): 5 10 30 45 48 50 55 57 60


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


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

```
Enter your choice: 7

Post-Order (Non-Recursive): 5 30 10 48 57 60 55 50 45



Menu:

1. Insert Node

2. In-Order Traversal (Recursive)

3. Pre-Order Traversal (Recursive)

4. Post-Order Traversal (Recursive)

5. In-Order Traversal (Non-Recursive)

6. Pre-Order Traversal (Non-Recursive)

7. Post-Order Traversal (Non-Recursive)

8. Search Node

9. Delete Node

9. Exit

Enter your choice: 8

Enter value to search: 45

Node found!


Menu:

1. Insert Node

2. In-Order Traversal (Recursive)

3. Pre-Order Traversal (Recursive)

4. Post-Order Traversal (Recursive)

5. In-Order Traversal (Non-Recursive)

6. Pre-Order Traversal (Non-Recursive)

7. Post-Order Traversal (Non-Recursive)

8. Search Node

9. Delete Node
```

```
9. Exit

Enter your choice: 9

Enter key to delete: 45

Key deleted if found.

PS D:\College Assignments\Sem - 4 SY-Btech\Advanced Data Structure(ADS)\Practical - 2>
```