Name: Siddhant Kumar Sahu

Batch: E3, 57

PRN: 202301070159

CODE

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

struct Tree{
    int data; // to store data of the tree
    Tree *left, *right; //Pointers for Tree
};

Tree* Create(int a){
    Tree *Root = new(Tree);

    Root->data = a;
    Root->left = NULL;
    Root->right = NULL;

    return Root;
}

Tree* Insert(Tree* Root, int a){
    if(Root == NULL){
        return Create(a); //if root is empty we create a node
    }
    if(a > Root->data){
        Root->right = Insert(Root->right , a); // if data is greater than node
data we go to right
    }
    if(a < Root->data){
        Root->left = Insert(Root->left , a); // if data is lesser than node
data we go to left
    }

    return Root;
}

Tree* Search(Tree* Root, int key){
    if(Root == NULL || Root->data == key){
        return Root;
    }
    if(key < Root->data){
        return Search(Root->left, key);
```

```
    }
    return Search(Root->right, key);
}

void InOrder(Tree* Root){
    if(Root == NULL){
        return;
    }

    InOrder(Root->left);
    cout << Root->data << " ";
    InOrder(Root->right);
}

void PreOrder(Tree* Root){
    if(Root == NULL){
        return;
    }

    cout << Root->data << " ";
    PreOrder(Root->left);
    PreOrder(Root->right);
}

void PostOrder(Tree* Root){
    if(Root == NULL){
        return;
    }

    PostOrder(Root->left);
    PostOrder(Root->right);
    cout << Root->data << " ";
}

void InOrderNR(Tree* Root){
    Tree *curr = Root;
    stack<Tree*> s;
    while(curr != NULL || s.empty() == false){
        while(curr != NULL){
            s.push(curr);
            curr = curr->left;
        }
        curr = s.top();
        s.pop();

        cout << curr->data << " ";
        curr = curr->right;
    }
```

```
}

void PreOrderNR(Tree* Root){
    Tree *curr = Root;
    stack<Tree*> s;
    while(curr != NULL){
        cout << curr->data << " ";
        s.push(curr);
        curr = curr->left;
    }
    while(!s.empty()){
        curr = s.top();
        s.pop();
        curr = curr->right;
        while(curr != NULL){
            cout << curr->data << " ";
            s.push(curr);
            curr = curr->left;
        }
    }
}

void PostOrderNR(Tree* Root){
    if(Root == NULL) return;
    stack<Tree*> s1, s2;
    s1.push(Root);
    while(!s1.empty()){
        Tree* curr = s1.top();
        s1.pop();
        s2.push(curr);
        if(curr->left != NULL) s1.push(curr->left);
        if(curr->right != NULL) s1.push(curr->right);
    }
    while(!s2.empty()){
        cout << s2.top()->data << " ";
        s2.pop();
    }
}
int tree_height(Tree* Root){
    if(!Root){
        return 0;
    }

    else{
        int left_height = tree_height(Root->left);
        int right_height = tree_height(Root->right);

        if(left_height >= right_height){
```

```
            return left_height+1;
        }else{
            return right_height+1;
        }
    }
}

void printLevel(Tree* Root, int level_no){
    if(!Root){
        return;
    }
    if(level_no == 0){
        cout << Root->data << " ";
    }else{
        printLevel(Root->left, level_no-1);
        printLevel(Root->right, level_no-1);
    }
}

void print_tree_level_order(Tree* Root){
    if(!Root){
        return;
    }
    int height = tree_height(Root);
    for(int i=0; i<height; i++){
        cout << "Level " << i << " : ";
        printLevel(Root, i);
        cout << endl;
    }
    cout << endl;

    cout << "\nComplete Level Wise Traversal\n\n";
    for(int i = 0; i<height; i++){
        printLevel(Root, i);
    }
    cout << endl;
}

Tree* findMin(Tree* Root){
    while(Root->left != NULL){
        Root = Root->left;
    }
    return Root;
}

Tree* DeleteNode(Tree* Root, int key){
    if(Root == NULL){
        return Root;
```

```
    }

    if(key < Root->data){
        Root->left = DeleteNode(Root->left, key);
    }else if(key > Root->data){
        Root->right = DeleteNode(Root->right, key);
    }else{
        if(Root->left == NULL){
            Tree* temp = Root->right;
            delete Root;
            return temp;
        } else if(Root->right == NULL){
            Tree* temp = Root->left;
            delete Root;
            return temp;
        }
        Tree* temp = findMin(Root->right);
        Root->data = temp->data;
        Root->right = DeleteNode(Root->right, temp->data);
    }
    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. Search Node\n";
        cout << "3. In-Order Traversal (Recursive)\n";
        cout << "4. Pre-Order Traversal (Recursive)\n";
        cout << "5. Post-Order Traversal (Recursive)\n";
        cout << "6. In-Order Traversal (Non-Recursive)\n";
        cout << "7. Pre-Order Traversal (Non-Recursive)\n";
        cout << "8. Post-Order Traversal (Non-Recursive)\n";
        cout << "9. Delete Node\n";
        cout << "10. 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 << "Enter value to search: ";
                cin >> value;
                if(Search(Root, value))
                    cout << "Node found!\n";
                else
                    cout << "Node not found!\n";
                break;
            case 3:
                cout << "Pre-Order Traversal (Recursive): ";
                PreOrder(Root);
                cout << endl;
                break;

            case 4:
                cout << "Post-Order Traversal (Recursive): ";
                PostOrder(Root);
                cout << endl;
                break;

            case 5:
                cout << "In-Order Traversal (Non-Recursive): ";
                InOrderNR(Root);
                cout << endl;
                break;

            case 6:
                cout << "Pre-Order Traversal (Non-Recursive): ";
                PreOrderNR(Root);
                cout << endl;
                break;

            case 7:
                cout << "Post-Order Traversal (Non-Recursive): ";
                PostOrderNR(Root);
                cout << endl;
                break;

            case 8:
                cout << "Level-Wise Traversal/Printing: \n";
```

```
                print_tree_level_order(Root);
                break;

            case 9:
                cout << "Enter value to delete: ";
                cin >> value;
                Root = DeleteNode(Root, value);
                break;
            case 10:
                cout << "Exiting program." << endl;
                break;
            default:
                cout << "Invalid choice. Please try again." << endl;
                break;
        }
    } while(choice != 10);

    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 - 1\" ; if ($?) { g++ BinarySearchTree.cpp -o BinarySearchTree } ; if ($?) { .\BinarySearchTree }

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. Level-Order Traversal

9. Delete Node

10. Exit

Enter your choice: 2

In-Order Traversal (Recursive): 5 10 30 45 48 50 55 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. Level-Order Traversal
9. Delete Node
10. Exit

Enter your choice: 3

Pre-Order Traversal (Recursive): 45 10 5 30 50 48 55 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. Level-Order Traversal
9. Delete Node
10. Exit

Enter your choice: 4

Post-Order Traversal (Recursive): 5 30 10 48 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. Level-Order Traversal

9. Delete Node

10. Exit

Enter your choice: 5

In-Order Traversal (Non-Recursive): 5 10 30 45 48 50 55 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. Level-Order Traversal

9. Delete Node

10. Exit

Enter your choice: 6

Pre-Order Traversal (Non-Recursive): 45 10 5 30 50 48 55 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. Level-Order Traversal

9. Delete Node

10. Exit

Enter your choice: 7

Post-Order Traversal (Non-Recursive): 5 30 10 48 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. Level-Order Traversal

9. Delete Node

10. Exit

Enter your choice: 8

Level-Wise Traversal/Printing:

Level 0 : 45

Level 1 : 10 50

Level 2 : 5 30 48 55

Level 3 : 60

Complete Level Wise Traversal

45 10 50 5 30 48 55 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. Level-Order Traversal
9. Delete Node
10. Exit

Enter your choice: 9

Enter value to delete: 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. Level-Order Traversal

9. Delete Node

10. Exit

Enter your choice: 8

Level-Wise Traversal/Printing:

Level 0 : 48

Level 1 : 10 50

Level 2 : 5 30 55

Level 3 : 60

Complete Level Wise Traversal

48 10 50 5 30 55 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. Level-Order Traversal

9. Delete Node

10. Exit

Enter your choice: 10

```
Exiting program.

PS D:\College Assignments\Sem - 4 SY-Btech\Advanced Data Structure(ADS)\Practical - 1>
```