Practical No.1

```
#include "iostream"
// #include "stack"

using namespace std;
struct Node {
  Node *left, *right;
  int data;
};
class Stack {
  public:
  int t = -1;
  Node *s[30];
  void push(Node *ele) {
     t += 1;
     s[t] = ele;
  }
  Node* top() {
     return s[t];
  }
  void pop() {
     t -= 1;
  }
  bool empty() {
     return bool(t == -1);
  }
};
class BinaryTree {
  public:
  Node *root = NULL;
  BinaryTree operator=(BinaryTree &BT){
     Node *current = BT.root;
     BinaryTree newTree;
     newTree.root = new Node;
     newTree.root->data = current->data;
     copyElementsOfTree(current, newTree.root);
     return newTree;
  }
  void copyElementsOfTree(Node *current, Node *newObjsCurrent){
     if(!current->left && !current->right) return;
     if(current->left){
        newObjsCurrent->left = new Node;
        newObjsCurrent->left->data = current->left->data;
        copyElementsOfTree(current->left, newObjsCurrent->left);
     }
     if(current->right){
        newObjsCurrent->right = new Node;
        newObjsCurrent->right->data = current->right->data;
        copyElementsOfTree(current->right, newObjsCurrent->right);
     }
  }
  void createNewCopy() {
     BinaryTree &newBT = *this;
     cout<<"\n=======================================";
     cout<<"\nNew Copy created";
     cout<<"\nInorder of new Elements: ";
     newBT.recursiveInorderPrint(newBT.root);
     cout<<"\nTransfer control over to copied Tree? (y/n): ";
```

```
    char ans;
    cin>>ans;
    if(ans == 'y'){
        newBT.exec();
    }
    cout<<"\nTRANSFERED CONTROL BACK TO INITIAL TREE";
    cout<<"\n=======================================";
}
void insertElement(int ele){
    Node *temp = root;
    char direction;
    cout<<"\nInsert To the l/r? :";
    cin>>direction;
    if(temp == NULL){
        root = new Node;
        root->data = ele;
        root->left = NULL;
        root->right = NULL;
        cout<<"\nElement added to root";
        return;
    }
    bool isDest = false;
    while(!isDest) {
        if(direction == 'r') {
            if(temp->right == NULL) {
                isDest = true;
                continue;
            }
            temp = temp->right;
            cout<<"\nInert at l/r of "<<temp->data<<"?";
            cin>>direction;
        }
        else if(direction == 'l') {
            if(temp->left == NULL) {
                isDest = true;
                continue;
            }
            temp = temp->left;
            cout<<"\nInert at l/r of "<<temp->data<<"?";
            cin>>direction;
        }
        else{
            cout<<"\nDiplicate Element!!!";
            return;
        }
    }
    struct Node *newEle = new Node;
    newEle->data = ele;
    newEle->left = NULL;
    newEle->right = NULL;
    if(direction == 'r') {
        temp->right = newEle;
    }
    else{
        temp->left = newEle;
    }
}

int getLeaf(Node *node) {
```

```
    if(!node->left && !node->right)
    return 1;
    int left=0, right=0;
    if(node->left){
        left = getLeaf(node->left);
    }
    if(node->right){
        right = getLeaf(node->right);
    }
    return left+right;
}

int getinter(Node *node){
    if(!node->left && !node->right)
        return 0;
    int left, right;
    if(node->left){
        left = getinter(node->left);
    }
    if(node->right){
        right = getinter(node->right);
    }
    return left+right+1;
}

void erase(Node *root){
    if (root == NULL) return;
    erase(root->left);
    erase(root->right);
    cout << "\n Deleting node: " << root->data;
    delete root;
}
void recursiveInorderPrint(Node *current) {
    if(current == NULL) return;
    recursiveInorderPrint(current->left);
    cout<<"\n"<<current->data;
    recursiveInorderPrint(current->right);
}

void recursivePreorderPrint(Node *current) {
    if(current == NULL) return;
    cout<<"\n"<<current->data;
    recursivePreorderPrint(current->left);
    recursivePreorderPrint(current->right);
}

void recursivePostorderPrint(Node *current) {
    if(current == NULL) return;
    recursivePostorderPrint(current->left);
    recursivePostorderPrint(current->right);
    cout<<"\n"<<current->data;
}

void changeRolesOfLnR(Node *current) {
    if(current == NULL) return;
    if(current->left && current->right){
        Node *temp = current->left;
        current->left = current->right;
        current->right = temp;
```

```
    }
    changeRolesOfLnR(current->right);
    changeRolesOfLnR(current->left);
}

int height(Node *current) {
    if(current == NULL) {
        return 0;
    }
    if(height(current->left) > height(current->right)) {
        return (1+height(current->left));
    }
    else{
        return (1+height(current->right));
    }
}

void iterativeInorderPrint() {
    Stack s;
    Node *current = root;

    while(current != NULL || !s.empty()){
        while(current != NULL) {
            s.push(current);
            current = current->left;
        }
        current = s.top();
        s.pop();
        cout<<"\n"<<current->data;
        current = current->right;
    }
}

void iterativePreorderPrint() {
    Stack s;
    Node *current = root;

    while(current != NULL || !s.empty()){
        while(current != NULL) {
            cout<<"\n"<<current->data;
            if(current->right){
                s.push(current->right);
            }
            current = current->left;
        }
        if(!s.empty()){
            current = s.top();
            s.pop();
        }
    }
}

void iterativePostorderPrint() {
    Stack s1, s2;
    Node *current = root;
    s1.push(current);
    while(!s1.empty()) {
        current = s1.top();
        s1.pop();
```

```
      s2.push(current);
      if(current->left) {
         s1.push(current->left);
      }
      if(current->right) {
         s1.push(current->right);
      }
   }
   while(!s2.empty()){
      current = s2.top();
      s2.pop();
      cout<<"\n"<<current->data;
   }
}

void printMenu(){
   cout<<"\nMENU";
   cout<<"\n1. Add Element";
   cout<<"\n2. recursive inorder traversal";
   cout<<"\n3. recursive preorder traversal";
   cout<<"\n4. recursive post traversal";
   cout<<"\n5. iterative inorder traversal";
   cout<<"\n6. iterative preorder traversal";
   cout<<"\n7. iterative postorder traversal";
   cout<<"\n8. Height of a tree";
   cout<<"\n9. change roles of left and right pointers";
   cout<<"\n10. Copy this tree to another";
   cout<<"\n11. Count number of leaves, internal nodes";
   cout<<"\n12. erase whole tree";
   cout<<"\n13. EXIT";
}

void exec() {
   cout<<"\nEnter number of elements you want to insert initially: ";
   int n;
   cin>>n;
   for(int i = 0; i < n; i++) {
      cout<<"\nEnter "<<i+1<<"th element: ";
      int ele;
      cin>>ele;
      insertElement(ele);
   }
   bool isExit = false;
   while(!isExit) {
      printMenu();
      cout<<"\nEnter your choice: ";
      int ch;
      cin>>ch;
      switch(ch) {
         case 1:{
            cout<<"\nEnter element: ";
            int ele;
            cin>>ele;
            insertElement(ele);
            cout<<"\nInserted element";
            break;
         }
         case 2:
            recursiveInorderPrint(root);
```

```
            break;
        case 3:
            recursivePreorderPrint(root);
            break;
        case 4:
            recursivePostorderPrint(root);
            break;
        case 5:
            iterativeInorderPrint();
            break;
        case 6:
            iterativePreorderPrint();
            break;
        case 7:
            iterativePostorderPrint();
            break;
        case 8:
            cout<<"\nHeight of tree is: "<<height(root);
            break;
        case 9:
            changeRolesOfLnR(root);
            recursiveInorderPrint(root);
            break;
        case 10:
            createNewCopy();
            break;
        case 11: {
            int lf = getLeaf(root);
            int intr = getinter(root);
            cout<<"\nLeafs: "<<lf<<"\nInternals: "<<intr;
            break;
        }
        case 12:
            erase(root);
            root = NULL;
            break;
        case 13:
            isExit = true;
            break;
      }
    }
  }
};

int main() {
  BinaryTree bst;
  bst.exec();
  return 0;
}
```

Output:
1.Recursive

Inorder Traversa

```
Enter number of elements you want to insert initially: 5

Enter 1th element: 50

Insert To the l/r? :l

Element added to root
Enter 2th element: 25

Insert To the l/r? :l

Enter 3th element: 100

Insert To the l/r? :r

Enter 4th element: 10

Insert To the l/r? :l

Insert at l/r of 25?l

Enter 5th element: 125

Insert To the l/r? :r

Insert at l/r of 100?r

**********Binary Tree Operations**********
----------MENU----------
1. Add Element to the tree
2. Perform Recursive Inorder Traversal
3. Perform Recursive Preorder Traversal
4. Perform Recursive Postorder Traversal
5. Perform Iterative Inorder Traversal
6. Perform Iterative Preorder Traversal
7. Perform Iterative Postorder Traversal
8. Exchange Node pointers
9. Get Height of the tree
10. Copy contents of tree
11. Print Leaf node count and Internal node count
12. Delete the tree
13. Exit
Enter your choice: 2

10
25
50
100
125
```

2. Postorder Traversal .Recursive Preorder and

```
**********Binary Tree Operations**********
----------MENU----------
1. Add Element to the tree
2. Perform Recursive Inorder Traversal
3. Perform Recursive Preorder Traversal
4. Perform Recursive Postorder Traversal
5. Perform Iterative Inorder Traversal
6. Perform Iterative Preorder Traversal
7. Perform Iterative Postorder Traversal
8. Exchange Node pointers
9. Get Height of the tree
10. Copy contents of tree
11. Print Leaf node count and Internal node count
12. Delete the tree
13. Exit
Enter your choice: 3

50
25
10
100
125
**********Binary Tree Operations**********
----------MENU----------
1. Add Element to the tree
2. Perform Recursive Inorder Traversal
3. Perform Recursive Preorder Traversal
4. Perform Recursive Postorder Traversal
5. Perform Iterative Inorder Traversal
6. Perform Iterative Preorder Traversal
7. Perform Iterative Postorder Traversal
8. Exchange Node pointers
9. Get Height of the tree
10. Copy contents of tree
11. Print Leaf node count and Internal node count
12. Delete the tree
13. Exit
Enter your choice: 4

10
25
125
100
50
```

3 .Iterative Inorder and Preorder

```
**********Binary Tree Operations**********
----------------------MENU----------------------
1. Add Element to the tree
2. Perform Recursive Inorder Traversal
3. Perform Recursive Preorder Traversal
4. Perform Recursive Postorder Traversal
5. Perform Iterative Inorder Traversal
6. Perform Iterative Preorder Traversal
7. Perform Iterative Postorder Traversal
8. Exchange Node pointers
9. Get Height of the tree
10. Copy contents of tree
11. Print Leaf node count and Internal node count
12. Delete the tree
13. Exit
Enter your choice: 5

10
25
50
100
125
**********Binary Tree Operations**********
----------------------MENU----------------------
1. Add Element to the tree
2. Perform Recursive Inorder Traversal
3. Perform Recursive Preorder Traversal
4. Perform Recursive Postorder Traversal
5. Perform Iterative Inorder Traversal
6. Perform Iterative Preorder Traversal
7. Perform Iterative Postorder Traversal
8. Exchange Node pointers
9. Get Height of the tree
10. Copy contents of tree
11. Print Leaf node count and Internal node count
12. Delete the tree
13. Exit
Enter your choice: 6

50
25
10
100
125
```

4 .Iterative Postorder Traversal

```
**********Binary Tree Operations**********
----------------------MENU----------------------
1. Add Element to the tree
2. Perform Recursive Inorder Traversal
3. Perform Recursive Preorder Traversal
4. Perform Recursive Postorder Traversal
5. Perform Iterative Inorder Traversal
6. Perform Iterative Preorder Traversal
7. Perform Iterative Postorder Traversal
8. Exchange Node pointers
9. Get Height of the tree
10. Copy contents of tree
11. Print Leaf node count and Internal node count
12. Delete the tree
13. Exit
Enter your choice: 7

10
25
125
100
50
```

5 .Exchange Node Pointers

```
**********Binary Tree Operations**********
----------------------MENU----------------------
1. Add Element to the tree
2. Perform Recursive Inorder Traversal
3. Perform Recursive Preorder Traversal
4. Perform Recursive Postorder Traversal
5. Perform Iterative Inorder Traversal
6. Perform Iterative Preorder Traversal
7. Perform Iterative Postorder Traversal
8. Exchange Node pointers
9. Get Height of the tree
10. Copy contents of tree
11. Print Leaf node count and Internal node count
12. Delete the tree
13. Exit
Enter your choice: 8

Inorder traversal of the mirror tree is as follows:
125
100
50
25
10
```

6.Height of Tree

```
**********Binary Tree Operations**********
----------------------MENU----------------------
1. Add Element to the tree
2. Perform Recursive Inorder Traversal
3. Perform Recursive Preorder Traversal
4. Perform Recursive Postorder Traversal
5. Perform Iterative Inorder Traversal
6. Perform Iterative Preorder Traversal
7. Perform Iterative Postorder Traversal
8. Exchange Node pointers
9. Get Height of the tree
10. Copy contents of tree
11. Print Leaf node count and Internal node count
12. Delete the tree
13. Exit
Enter your choice: 9

 Height of the tree is 3
```

7.Copy contents of tree

```
**********Binary Tree Operations**********
----------------------MENU----------------------
1. Add Element to the tree
2. Perform Recursive Inorder Traversal
3. Perform Recursive Preorder Traversal
4. Perform Recursive Postorder Traversal
5. Perform Iterative Inorder Traversal
6. Perform Iterative Preorder Traversal
7. Perform Iterative Postorder Traversal
8. Exchange Node pointers
9. Get Height of the tree
10. Copy contents of tree
11. Print Leaf node count and Internal node count
12. Delete the tree
13. Exit
Enter your choice: 10

new copy created...
Inorder traversal of new tree elements is:

125
100
50
25
10
```

```
**********Binary Tree Operations**********
----------------------MENU----------------------
1. Add Element to the tree
2. Perform Recursive Inorder Traversal
3. Perform Recursive Preorder Traversal
4. Perform Recursive Postorder Traversal
5. Perform Iterative Inorder Traversal
6. Perform Iterative Preorder Traversal
7. Perform Iterative Postorder Traversal
8. Exchange Node pointers
9. Get Height of the tree
10. Copy contents of tree
11. Print Leaf node count and Internal node count
12. Delete the tree
13. Exit
Enter your choice: 11

The leaf count of binary tree is: 2

Number of Internal Nodes are: 3

**********Binary Tree Operations**********
----------------------MENU----------------------
1. Add Element to the tree
2. Perform Recursive Inorder Traversal
3. Perform Recursive Preorder Traversal
4. Perform Recursive Postorder Traversal
5. Perform Iterative Inorder Traversal
6. Perform Iterative Preorder Traversal
7. Perform Iterative Postorder Traversal
8. Exchange Node pointers
9. Get Height of the tree
10. Copy contents of tree
11. Print Leaf node count and Internal node count
12. Delete the tree
13. Exit
Enter your choice: 12
Deleted node is: 125
Deleted node is: 100
Deleted node is: 10
Deleted node is: 25
Deleted node is: 50
Tree deleted...
```