Deccan Education Society's (DES)
Pune University, Pune
School of Engineering and Technology
Department of Computer Engineering and Technology
Program: B. Tech in Computer Science and Engineering

Academic Year: 2024-25	**Year:** Third Year	**Term:** II
PRN No.: 1012412079	**Name: Ratnajeet Patil**	
Subject: Python Programming		
Assignment No.: 1		
Date:		

Lab Assignment: 01

Title: Create a binary tree and perform inorder , preorder and postorder traversals.

```java
import java.util.Scanner;
import java.util.Stack;

class Node {
    int data;
    Node left, right;

    public Node(int data) {
        this.data = data;
        left = right = null;
    }
}

class BinaryTree {
    Node root;
    Scanner scanner = new Scanner(System.in);

    public void insert(int data) {
        if (root == null) {
            root = new Node(data);
            System.out.println("Inserted as root node.");
            return;
        }

        Node current = root;
        while (true) {
            System.out.print("Enter direction (L/R) from " + current.data + ": ");
            String direction = scanner.next();
            if (direction.equals("L")) {
                if (current.left == null) {
                    current.left = new Node(data);
                    System.out.println("Inserted at left of " + current.data);
                    break;
                } else {
                    current = current.left;
                }
            } else if (direction.equals("R")) {
                if (current.right == null) {
                    current.right = new Node(data);
                    System.out.println("Inserted at right of " + current.data);
                    break;
                } else {
```

```
                    current = current.right;
                }
            } else {
                System.out.println("Invalid direction! Enter either 'left' or 'right'.");
            }
        }
    }

    public void inOrderTraversal() {
        if (root == null)
            return;

        Stack<Node> stack = new Stack<>();
        Node current = root;

        while (current != null || !stack.isEmpty()) {
            // Reach leftmost node of current node
            while (current != null) {
                stack.push(current);
                current = current.left;
            }

            // Current is null at this point
            current = stack.pop();
            System.out.print(current.data + " ");

            // Move to the right subtree
            current = current.right;
        }
    }

    public void preOrderTraversal() {
        if (root == null)
            return;

        Stack<Node> stack = new Stack<>();
        stack.push(root);

        while (!stack.isEmpty()) {
            // Pop the top node and print it
            Node current = stack.pop();
            System.out.print(current.data + " ");

            // Push right child first so that left child is processed first
            if (current.right != null) {
                stack.push(current.right);
            }
            if (current.left != null) {
                stack.push(current.left);
            }
        }
    }

    public void postOrderTraversal() {
        if (root == null)
            return;

        Stack<Node> stack1 = new Stack<>();
        Stack<Node> stack2 = new Stack<>();

        stack1.push(root);
```

```
104.
105.         // First stack to get nodes in reverse post-order
106.         while (!stack1.isEmpty()) {
107.             Node current = stack1.pop();
108.             stack2.push(current);
109.
110.             if (current.left != null) {
111.                 stack1.push(current.left);
112.             }
113.             if (current.right != null) {
114.                 stack1.push(current.right);
115.             }
116.         }
117.
118.         // Second stack has nodes in post-order
119.         while (!stack2.isEmpty()) {
120.             Node current = stack2.pop();
121.             System.out.print(current.data + " ");
122.         }
123.     }
124.
125.     // Public methods to call the traversal methods
126.     public void inOrder() {
127.         System.out.println("In-order Traversal of the Tree (Iterative):");
128.         inOrderTraversal();
129.         System.out.println();
130.     }
131.
132.     public void preOrder() {
133.         System.out.println("Pre-order Traversal of the Tree (Iterative):");
134.         preOrderTraversal();
135.         System.out.println();
136.     }
137.
138.     public void postOrder() {
139.         System.out.println("Post-order Traversal of the Tree (Iterative):");
140.         postOrderTraversal();
141.         System.out.println();
142.     }
143. }
144.
145. public class BinaryTreeUserInput {
146.     public static void main(String[] args) {
147.         Scanner scanner = new Scanner(System.in);
148.         BinaryTree tree = new BinaryTree();
149.         int choice;
150.         boolean exit = false;
151.
152.         System.out.print("Enter number of nodes for the initial tree: ");
153.         int n = scanner.nextInt();
154.
155.         for (int i = 0; i < n; i++) {
156.             System.out.print("Enter value for node: ");
157.             int value = scanner.nextInt();
158.             tree.insert(value);
159.         }
160.
161.         while (!exit) {
162.             System.out.println("\nBinary Tree Operations:");
163.             System.out.println("1. Insert a node");
164.             System.out.println("2. Pre-order traversal");
```

```
165.             System.out.println("3. In-order traversal");
166.             System.out.println("4. Post-order traversal");
167.             System.out.println("5. Exit");
168.             System.out.print("Enter your choice: ");
169.
170.             choice = scanner.nextInt();
171.
172.             switch (choice) {
173.                 case 1:
174.                     System.out.print("Enter value for new node: ");
175.                     int value = scanner.nextInt();
176.                     tree.insert(value);
177.                     break;
178.                 case 2:
179.                     tree.preOrder();
180.                     break;
181.                 case 3:
182.                     tree.inOrder();
183.                     break;
184.                 case 4:
185.                     tree.postOrder();
186.                     break;
187.                 case 5:
188.                     exit = true;
189.                     System.out.println("Exiting program...");
190.                     break;
191.                 default:
192.                     System.out.println("Invalid choice! Please try again.");
193.             }
194.         }
195.         scanner.close();
196.     }
197. }
198.
199.
```

```
Binary Tree Operations:
1. Insert a node
2. Pre-order traversal
3. In-order traversal
4. Post-order traversal
5. Exit
Enter your choice: 2
Pre-order Traversal of the Tree:
10 8 7 5 5 2 6 11 12 8

Binary Tree Operations:
1. Insert a node
2. Pre-order traversal
3. In-order traversal
4. Post-order traversal
5. Exit
Enter your choice: 3
In-order Traversal of the Tree:
2 5 5 6 7 8 10 11 8 12

Binary Tree Operations:
1. Insert a node
2. Pre-order traversal
3. In-order traversal
4. Post-order traversal
5. Exit
Enter your choice: 4
Post-order Traversal of the Tree:
2 5 6 5 7 8 8 12 11 10
```