Name: Shubham Patgavkar
Div: C Batch: C3
Roll No: 233074
PRN No: 22320100

Practical No: 1

Title: Construct an expression tree from postfix/prefix expression and perform recursive inorder, preorder and post order traversals.

Recursive approach ⟶

```java
import java.util.*;

class TreeNode {
    char data;
    TreeNode left, right;

    public TreeNode(char item) {
        data = item;
        left = right = null;
    }
}

class Practical1 {
    public static TreeNode constructTreeFromPostfix(String postfix) {
        Stack<TreeNode> stack = new Stack<>();

        for (char c : postfix.toCharArray()) {
            if (Character.isLetterOrDigit(c)) {
                stack.push(new TreeNode(c));
            } else {
                TreeNode operand2 = stack.pop();
                TreeNode operand1 = stack.pop();
                TreeNode operator = new TreeNode(c);
                operator.left = operand1;
                operator.right = operand2;
                stack.push(operator);
            }
        }

        return stack.pop();
    }
```

```
    public static void inorderTraversal(TreeNode root) {
        if (root != null) {
            inorderTraversal(root.left);
            System.out.print(root.data + " ");
            inorderTraversal(root.right);
        }
    }

    public static void preorderTraversal(TreeNode root) {
        if (root != null) {
            System.out.print(root.data + " ");
            preorderTraversal(root.left);
            preorderTraversal(root.right);
        }
    }

    public static void postorderTraversal(TreeNode root) {
        if (root != null) {
            postorderTraversal(root.left);
            postorderTraversal(root.right);
            System.out.print(root.data + " ");
        }
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.println("Enter postfix expression: ");
        String postfixExpression = sc.nextLine();
        TreeNode root = constructTreeFromPostfix(postfixExpression);

        System.out.println("Inorder Traversal:");
        inorderTraversal(root);

        System.out.println("\nPreorder Traversal:");
        preorderTraversal(root);

        System.out.println("\nPostorder Traversal:");
        postorderTraversal(root);
    }
}
```

Output:

Enter postfix expression:
456789+-
Inorder Traversal:

7 - 8 + 9
Preorder Traversal:
- 7 + 8 9
Postorder Traversal:
7 8 9 + -

For expression tree, perform non-recursive inorder, preorder and post order traversals.

Non-Recursive approach →

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

class TreeNode {
    char data;
    TreeNode left;
    TreeNode right;

    TreeNode(char val) {
        data = val;
        left = null;
        right = null;
    }
}

public class PR1_2 {

    static boolean isOperand(char c) {
        return (c >= '0' && c <= '9') || (c >= 'a' && c <= 'z') || (c >=
'A' && c <= 'Z');
    }

    // Function to construct expression tree from postfix expression
    static TreeNode constructExpressionTreePostfix(String postfix) {
        Stack<TreeNode> st = new Stack<>();
        for (char c : postfix.toCharArray()) {
            if (isOperand(c)) {
                st.push(new TreeNode(c));
            } else {
                TreeNode operand2 = st.pop();
                TreeNode operand1 = st.pop();
                TreeNode newNode = new TreeNode(c);
                newNode.left = operand1;
                newNode.right = operand2;
                st.push(newNode);
            }
```

```
        }
        return st.pop();
    }

    static void nonRecursiveInorder(TreeNode root) {
        Stack<TreeNode> st = new Stack<>();
        while (root != null || !st.isEmpty()) {
            while (root != null) {
                st.push(root);
                root = root.left;
            }
            root = st.pop();
            System.out.print(root.data + " ");
            root = root.right;
        }
    }

    static void nonRecursivePreorder(TreeNode root) {
        Stack<TreeNode> st = new Stack<>();
        st.push(root);
        while (!st.isEmpty()) {
            root = st.pop();
            System.out.print(root.data + " ");
            if (root.right != null)
                st.push(root.right);
            if (root.left != null)
                st.push(root.left);
        }
    }

    static void nonRecursivePostorder(TreeNode root) {
        Stack<TreeNode> st1 = new Stack<>();
        Stack<TreeNode> st2 = new Stack<>();
        st1.push(root);
        while (!st1.isEmpty()) {
            root = st1.pop();
            st2.push(root);
            if (root.left != null)
                st1.push(root.left);
            if (root.right != null)
                st1.push(root.right);
        }

        while (!st2.isEmpty()) {
            System.out.print(st2.pop().data + " ");
        }
    }
```

```
    public static void main(String[] args) {
        String postfixExpression = "ab+ef*g*-";

        TreeNode postfixRoot =
constructExpressionTreePostfix(postfixExpression);

        // Non-recursive traversals
        System.out.println("Non-Recursive Inorder (Postfix): ");
        nonRecursiveInorder(postfixRoot);
        System.out.println("\nNon-Recursive Preorder (Postfix): ");
        nonRecursivePreorder(postfixRoot);
        System.out.println("\nNon-Recursive Postorder (Postfix): ");
        nonRecursivePostorder(postfixRoot);
    }
}
```

Output:

Non-Recursive Inorder (Postfix):
a + b - e * f * g
Non-Recursive Preorder (Postfix):
- + a b * * e f g
Non-Recursive Postorder (Postfix):
a b + e f * g * -