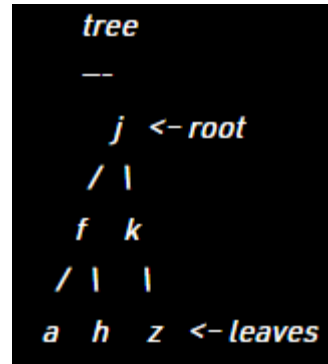


Name	Virinchi Sadashiv Shettigar
UID no.	2021300118
Experiment No.	6

AIM:	Implement a given problem statement using Expression Trees.
THEORY:	<p style="text-align: center;"><b>Tree</b></p> <p>A tree is a popular data structure that is non-linear in nature. Unlike other data structures like an array, stack, queue, and linked list which are linear in nature, a tree represents a hierarchical structure. The ordering information of a tree is not important. A tree contains nodes and 2 pointers. These two pointers are the left child and the right child of the parent node. Let us understand the terms of tree in detail.</p> <p><b><u>Root:</u></b> The root of a tree is the topmost node of the tree that has no parent node. There is only one root node in every tree.</p> <p><b><u>Edge:</u></b> Edge acts as a link between the parent node and the child node.</p> <p><b><u>Leaf:</u></b> A node that has no child is known as the leaf node. It is the last node of the tree. There can be multiple leaf nodes in a tree.</p> <p><b><u>Subtree:</u></b> The subtree of a node is the tree considering that particular node as the root node.</p> <p><b><u>Depth:</u></b> The depth of the node is the distance from the root node to that particular node.</p> <p><b><u>Height:</u></b> The height of the node is the distance from that node to the deepest node of that subtree.</p> <p><b><u>Height of tree:</u></b> The Height of the tree is the maximum height of any node. This is same as the height of root node.</p>



The types of Trees in the Data Structure.

1. Binary tree.
2. Binary Search Tree.
3. AVL Tree.
4. B-Tree.

## Expression Tree

An expression tree is one form of binary tree that is used to represent expressions. An expression tree for arithmetic, relational, or logical expression is a binary tree in which:

- The parentheses in the expression do not appear.
- The leaves are the variables or constants in the expression.
- The non-leaf nodes are the operators in the expression:
  - A node for a binary operator has two non-empty subtrees.
  - A node for a unary operator has one non-empty subtree.

The operators, constants, and variables are arranged in such a way that an in-order traversal of the tree produces the original expression without parentheses.

## Evaluation of Expression Tree

- You can easily form the algebraic expression using a binary expression tree by recursively calling the left subtree, then printing the root operator, and then recursively calling the right subtree. This strategy of calling the left subtree, the root node, and the right subtree is eventually called in-order traversal

method.

- Apart from this, you can also use the post-order traversal strategy where the left subtree is printed first, then the right subtree, and lastly the root node operator.
- As an alternative, you can also print the root node first and then recursively call the left and right subtree respectively. This strategy is called pre-order traversal.

The expression tree is a binary tree in which each external or leaf node corresponds to the operand and each internal or parent node corresponds to the operators so for example expression tree for  $7 + ((1+8)*3)$  would be:

## Use of Expression tree

1. The main objective of using the expression trees is to make complex expressions that'd be easily evaluated using these expression trees.
2. It is also used to determine each operator's associativity in the expression.

It is also used to solve the postfix, prefix, and infix expression evaluation.

## Implementation of an Expression tree

To implement the expression tree and write its program, we will be required to use a stack data structure.

As we know that the stack is based on the last in first out LIFO principle, the data element pushed recently into the stack has been popped out whenever required. For its implementation, the main two operations of the stack, push and pop, are used. Using the push

operation, we will push the data element into the stack, and by using the pop operation, we will remove the data element from the stack.

### **Main functions of the stack in the expression tree implementation:**

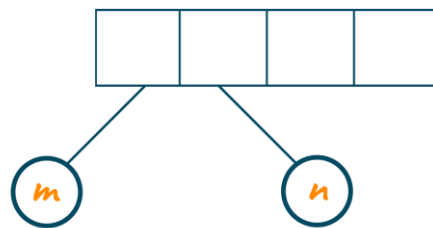
First of all, we will do scanning of the given expression into left to the right manner, then one by one check the identified character,

- If the scanned character is an operand, we will apply the push operation and push it into the stack.
- If a scanned character is an operator, we will apply the pop operation into it to remove the two values from the stack to make them its child, and after then we will push back the current parent node into the stack.

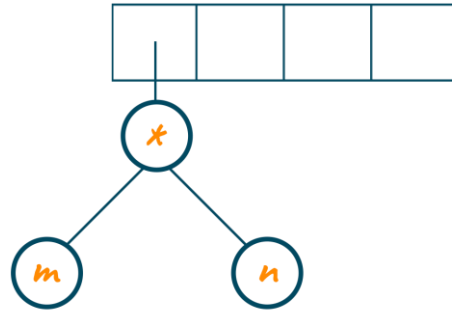
## **Example**

If the postfix notation is:  $m\ n\ *\ p\ q\ r\ +\ *\ +$

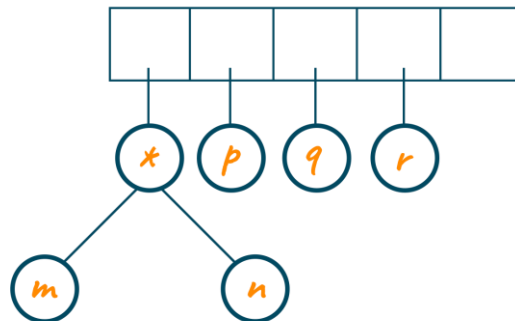
Here the first two symbols are operands, i.e.  $m$  and  $n$ . So, the one-node tree is created as shown in the below image, and the pointers of these operands are pushed into the stack.



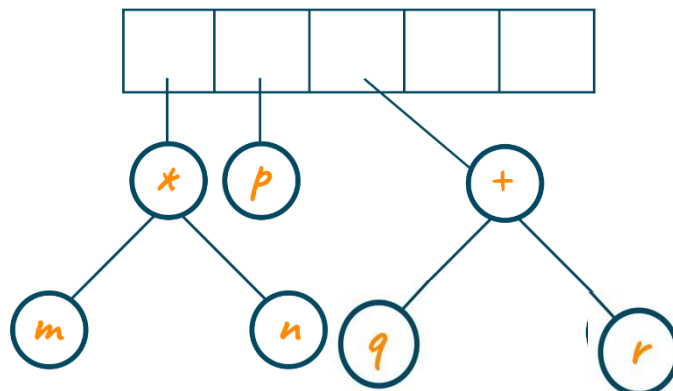
The next in the equation is the “\*” operator. Therefore, we will pop the operands pointers from the stack and form a new tree where the operator serves as root node and operands serves as left and right child. Later, the pointer to the tree is pushed into the stack as shown in the below example.



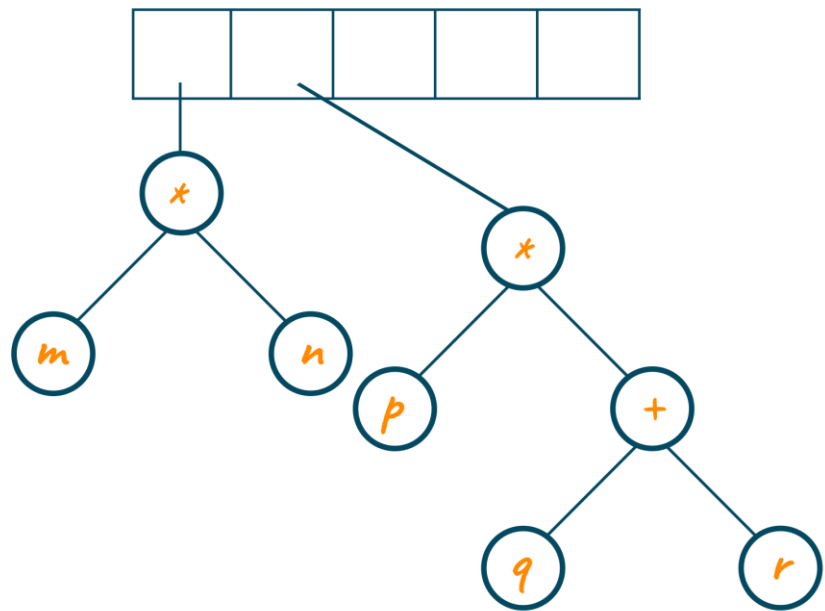
Now, the postfix expression traverse to “p”, “q”, and “r”. As they are operands, the one-node tree is formed and the pointer to each node is pushed into the stack.



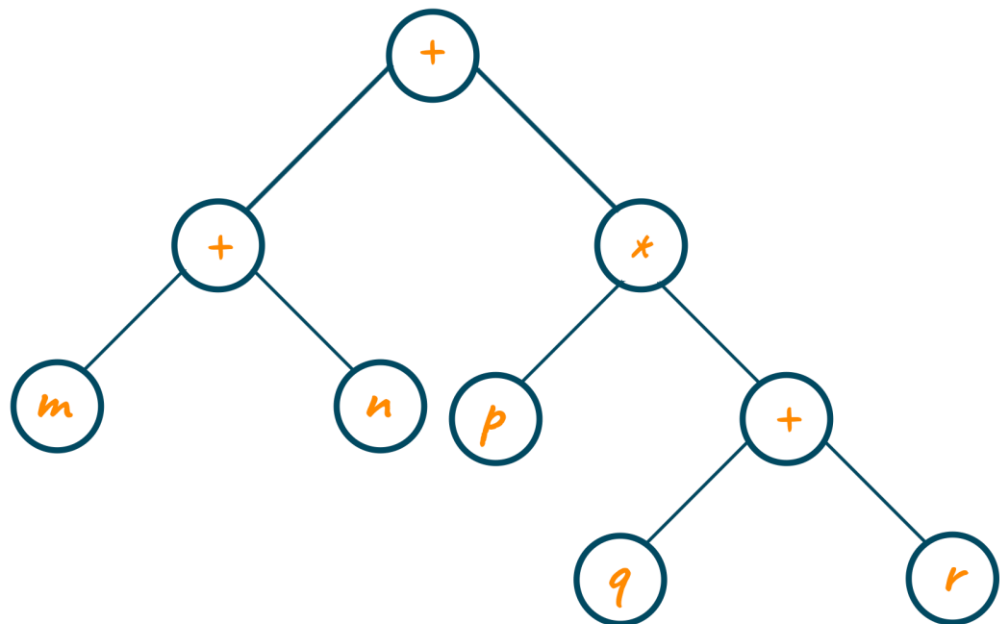
Later, the operator “+” is encountered and it serves as the root node to the last two one-node operands in the stack. The pointer to this new tree is stored in the stack as shown in the below image.



Now, again “\*” operator is read . The last two tree pointers are popped from the stack and a new tree is built with root node as “\*” operator as shown in the below image.



At last, the two individual trees are combined with the “+” operator, and the final expression tree is formed. The pointer to the new tree is stored in the stack as shown below.



**ALGORITHM:****Stack class**

1. Initialize top and maxsize
2. Also initialize the array of Node stack
3. Constructor stack(int x)
  - Assign maxsize equal to x
  - Assign top equals to -1
  - And assign array of stack to new Node of maxsize

**Void push(Node p)**

1. Increment top by 1 and assign stack[top]=p

**Node pop()**

1. Check if top equals to -1 of yes then return null
2. Else decrement top by 1 and return stack[top+1]

**ExpTree class****boolean isOperator(char c)**

1. If the character is an operator then return true
2. Else false

**Node expressiontree(String s)**

1. Assign stack object sa to stack(s,length())
2. Initialize operand and operator equal to 0
3. Run a for loop till length of the string
4. Check if s.charAt(i) is not an operator if yes then increment the operand by 1
5. Else increment the operator by 1 and check if operand not equals to operator+1 if yes then print Invalid Expression
6. Again, run a for loop till length of the string
7. Check if s.charAt(i) is not an operator if yes then assign p a newNode at character of string and push in the stack
8. Else create a Node t1,t2 and pop
9. Also create temporary Node temp and assign in to s.charAt(i)
10. Assign left of temp equal to t2 and right of temp equal to t1 and do sa. push(temp)
11. Return the root

**Void Inorder(Node root)**

1. Check if root is not equal to null if yes then call Inorder(root.left) print data of root and call Inorder(root.right)

Void preorder(Node root)

1. Check if root is not equal to null if yes then print data of root call preorder(root.left) and call preorder(root.right)

Void postorder(Node root)

1. Check if root is not equal to null if yes then call postorder(root.left) call postorder(root.right) and print data of root

Exp6 Class

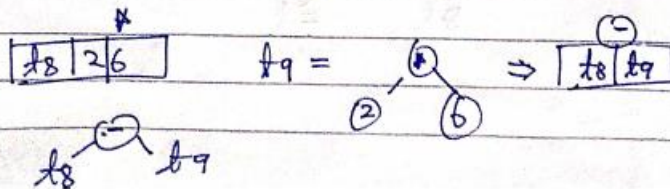
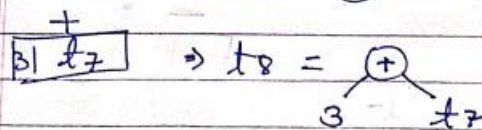
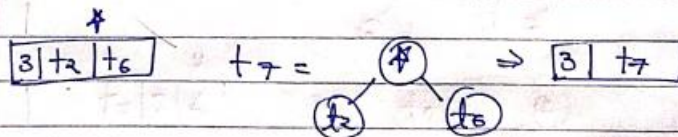
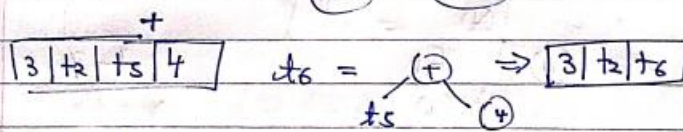
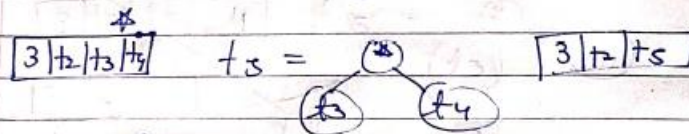
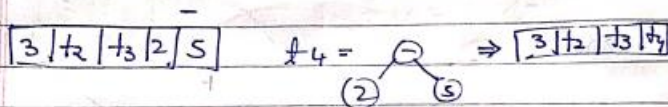
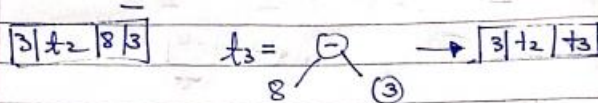
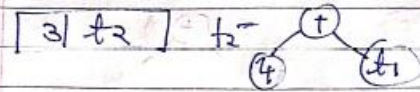
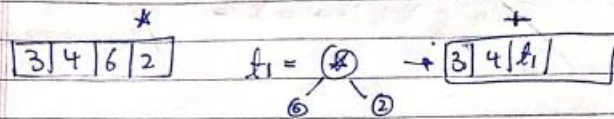
Main function

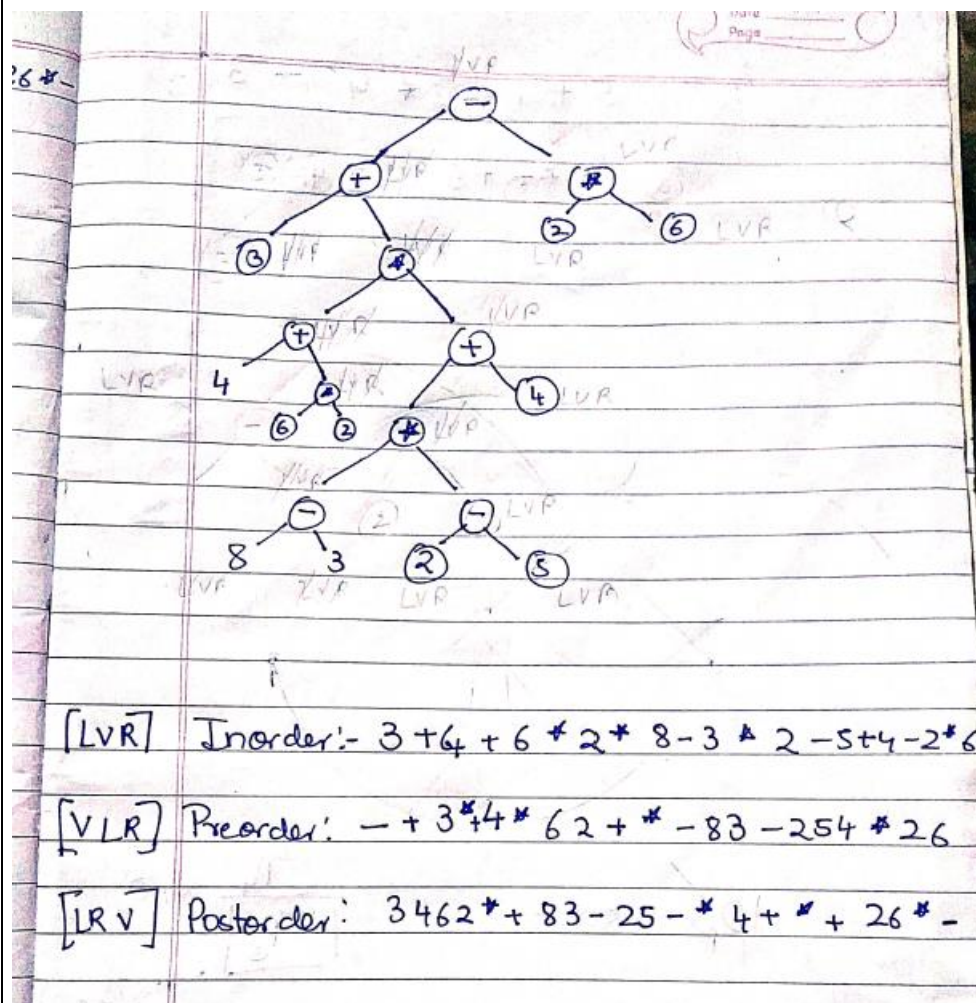
1. Create a ExpTree object t
2. Assign root to call class expressiontree with the string input expression
3. Input the postfix expression from user
4. Display menu driven program
5. Input the choice from the user
6. If choice 1, call inorder(root)
7. If choice 2, call preorder(root)
8. If choice 3, call postorder(root)
9. Repeat steps from 4 to 8 until user chose to exit the program.



**PROBLEM-SOLVING:**

Postfix expression: 3 4 6 2 \* + 8 3 - 2 5 - \* 4 + \* + 2 6 \*





**PROGRAM:**

```
import java.util.*;

class Node {
    char data;
    Node left;
    Node right;

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

class Exptree {
    public static boolean isOperator(char ch) {
        if (ch == '+' || ch == '-' || ch == '*' || ch == '/' || ch == '^')
            return true;
    }
}
```

```

        return false;
    }

    Node expressiontree(String s) {
        stack sa = new stack(s.length());
        int operator = 0;
        int operand = 0;
        for (int i = 0; i < s.length(); i++) {
            if (!isOperator(s.charAt(i))) {
                operand++;
            } else {
                operator++;
            }
        }
        if (operand != operator + 1) {
            System.out.print("Invalid");
            System.exit(0);
        }
        for (int i = 0; i < s.length(); i++) {
            if (!isOperator(s.charAt(i))) {
                Node temp = new Node(s.charAt(i));
                sa.push(temp);
            } else {
                Node t1 = sa.pop();
                Node t2 = sa.pop();
                Node temp = new Node(s.charAt(i));
                temp.left = t2;
                temp.right = t1;
                sa.push(temp);
            }
        }
        Node root = sa.pop();
        return root;
    }

    void Inorder(Node root) {
        if (root != null) {
            Inorder(root.left);
            System.out.print("" + root.data + "");
            Inorder(root.right);
        }
    }

    void preorder(Node root) {
        if (root != null) {
            System.out.print("" + root.data + "");

```

```

        preorder(root.left);
        preorder(root.right);
    }
}

void postorder(Node root) {
    if (root != null) {
        postorder(root.left);
        postorder(root.right);
        System.out.print("" + root.data + "");
    }
}
}

class stack {
    Node stack[];
    int top, maxsize;

    stack(int y) {
        maxsize = y;
        stack = new Node[maxsize];
        top = -1;
    }

    void push(Node p) {
        stack[++top] = p;
    }

    Node pop() {
        if (top == -1)
            return null;
        return stack[top--];
    }
}

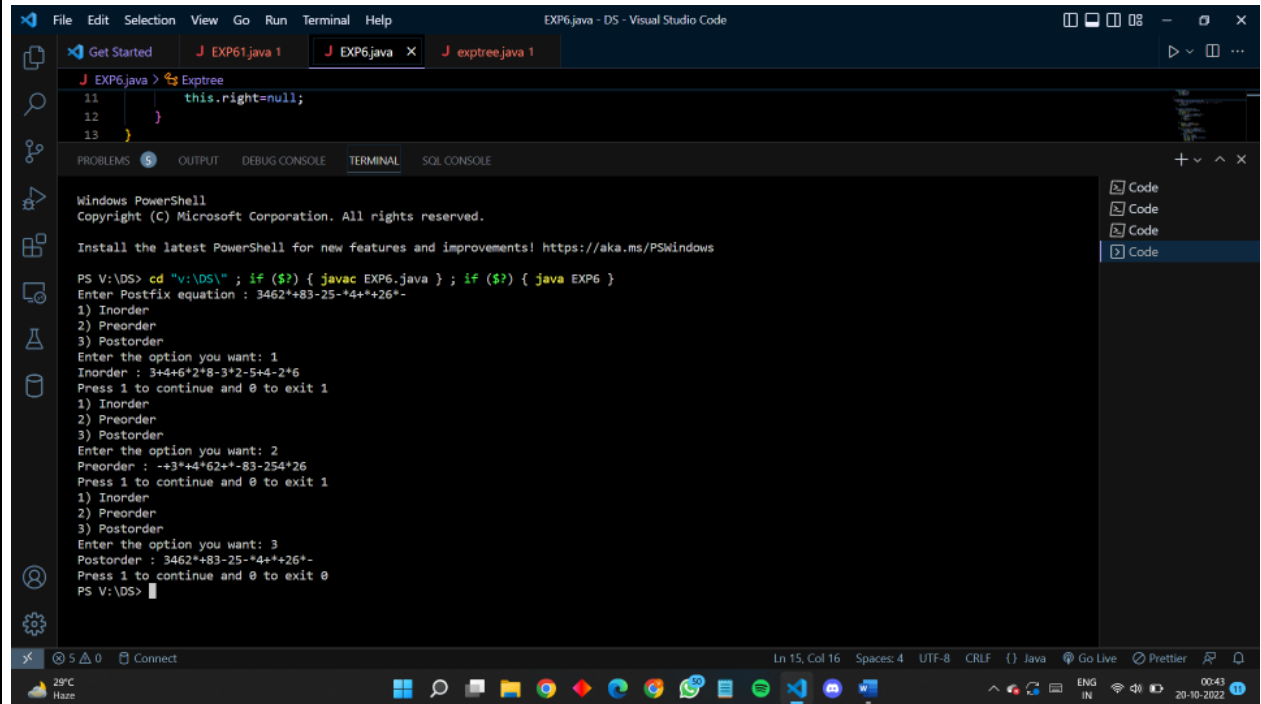
class EXP6 {
    public static void main(String[] args) {
        Exptree t = new Exptree();
        Node root;
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter Postfix equation : ");
        String s = sc.nextLine();
        root = t.expressiontree(s);
        int flag=1;
        while (flag==1) {
            System.out.print("1) Inorder \n2) Preorder \n3) Postorder\n");
            System.out.print("Enter the option you want: ");

```

```
int choice=sc.nextInt();
switch (choice) {
    case 1:
        System.out.print("Inorder : ");
        t.Inorder(root);
        break;
    case 2:
        System.out.print("Preorder : ");
        t.preorder(root);
        break;
    case 3:
        System.out.print("Postorder : ");
        t.postorder(root);
        break;

    default:
        break;
}
System.out.print("\nPress 1 to continue and 0 to exit ");
flag=sc.nextInt();
if (flag==0) {
    break;
}
}
sc.close();
}
```

## OUTPUT:



The screenshot shows the Visual Studio Code editor with a Java file named `EXP6.java` containing the following code:

```
11     this.right=null;
12 }
13 }
```

The terminal window shows the execution of the program in a Windows PowerShell environment. The user runs `javac EXP6.java` and `java EXP6`. The program prompts the user to enter a postfix equation: `3462*+83-25-*4*+26*-`. It then asks the user to choose an option (1 for Inorder, 2 for Preorder, 3 for Postorder). The user selects 1, and the program outputs the inorder traversal: `3+4+6*2*8-3*2-5+4-2*6`. The user then selects 2, and the program outputs the preorder traversal: `++3*+4*62*+-83-254*26`. Finally, the user selects 3, and the program outputs the postorder traversal: `3462*+83-25-*4*+26*-`.

## CONCLUSION:

In this experiment, I learned about expression trees and learned about its uses. Then I implemented an expression tree program by taking postfix expressions as input and converting them into inOrder, preOrder, and postOrder.