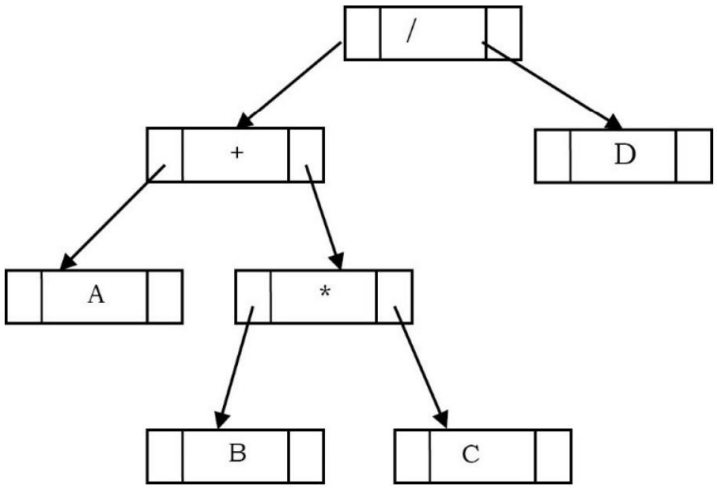
Name	**Hatim Yusuf Sawai**
UID no.	**2021300108**
Experiment No.	**6**

AIM:	**Program on Expression Tree**
Program 1	
PROBLEM STATEMENT:	Write a program to store a **postfix** expression into an expression tree, convert it to **infix/prefix** form & evaluate it.
THEORY:	**Expression Tree:** A tree representing an expression is called an expression tree. In expression trees, leaf nodes are operands and non-leaf nodes are operators. That means an expression tree is a binary tree where internal nodes are operators and leaves are operands. An expression tree consists of a binary expression. But for a unary operator, one subtree will be empty. The figure below shows a simple expression tree for **(A+B*C)/D:** 

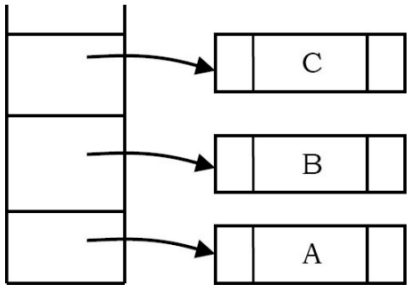
How to insert an expression into the tree:

Assume that 1 symbol is read at a time. If the symbol is an operand, we create a tree node and push a pointer to it onto a stack. If the symbol is an operator, pop pointers to two trees T1 and T2 from the stack (T1 is popped first) and forms a new tree whose root is the operator and whose left and right children point to T2 and T1 respectively. A pointer to this new tree is then pushed onto the stack.

As an example, assume the input is **ABC*+D/**

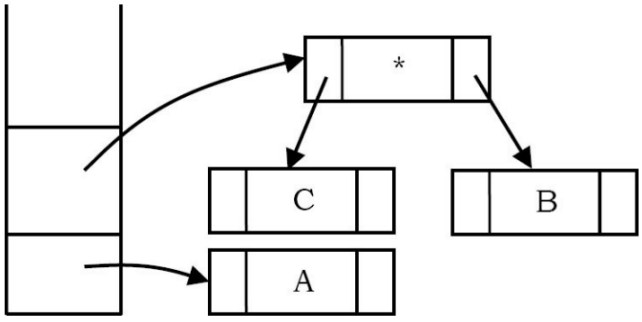
Step-1:

First 3 symbols are operands -> push to stack



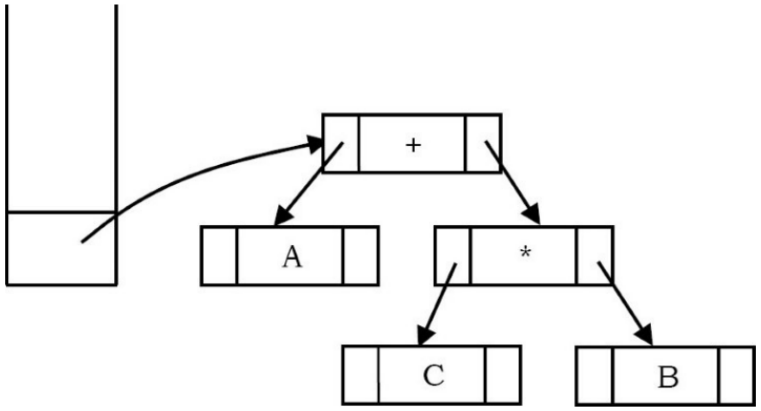
Step-2:

Next, operator * is read -> 2 pointers are popped, a new tree is formed and a pointer to it is pushed onto the stack.



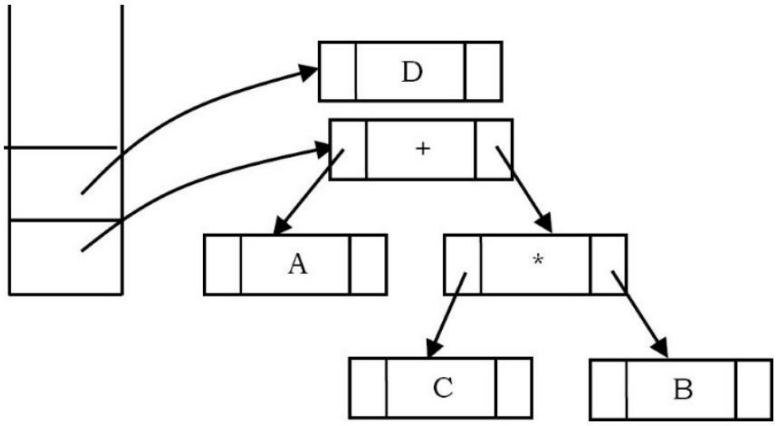
Step-3:

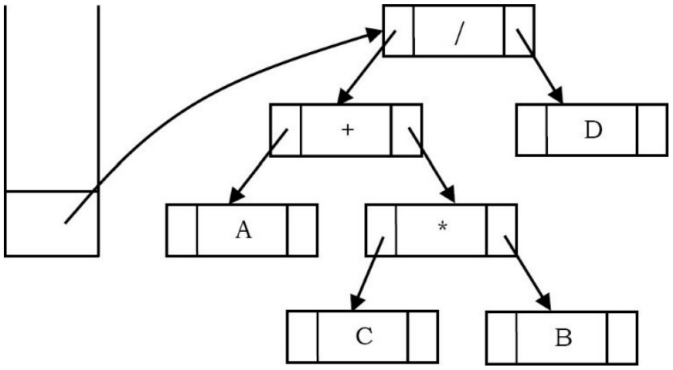
Next, operator + is read -> 2 pointers are popped, a new tree is formed and a pointer to it is pushed onto the stack.



Step-4:

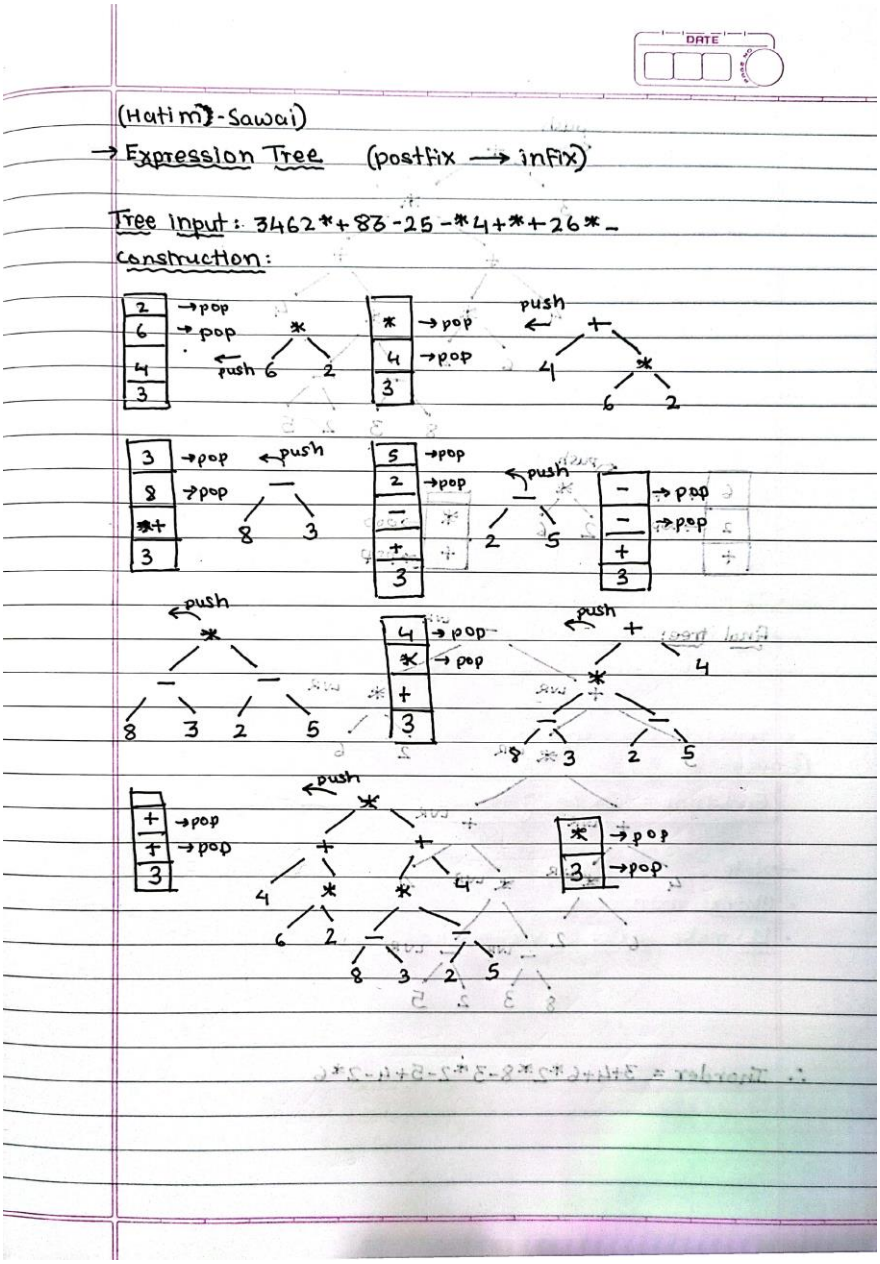
Next symbol is an operand -> push to stack

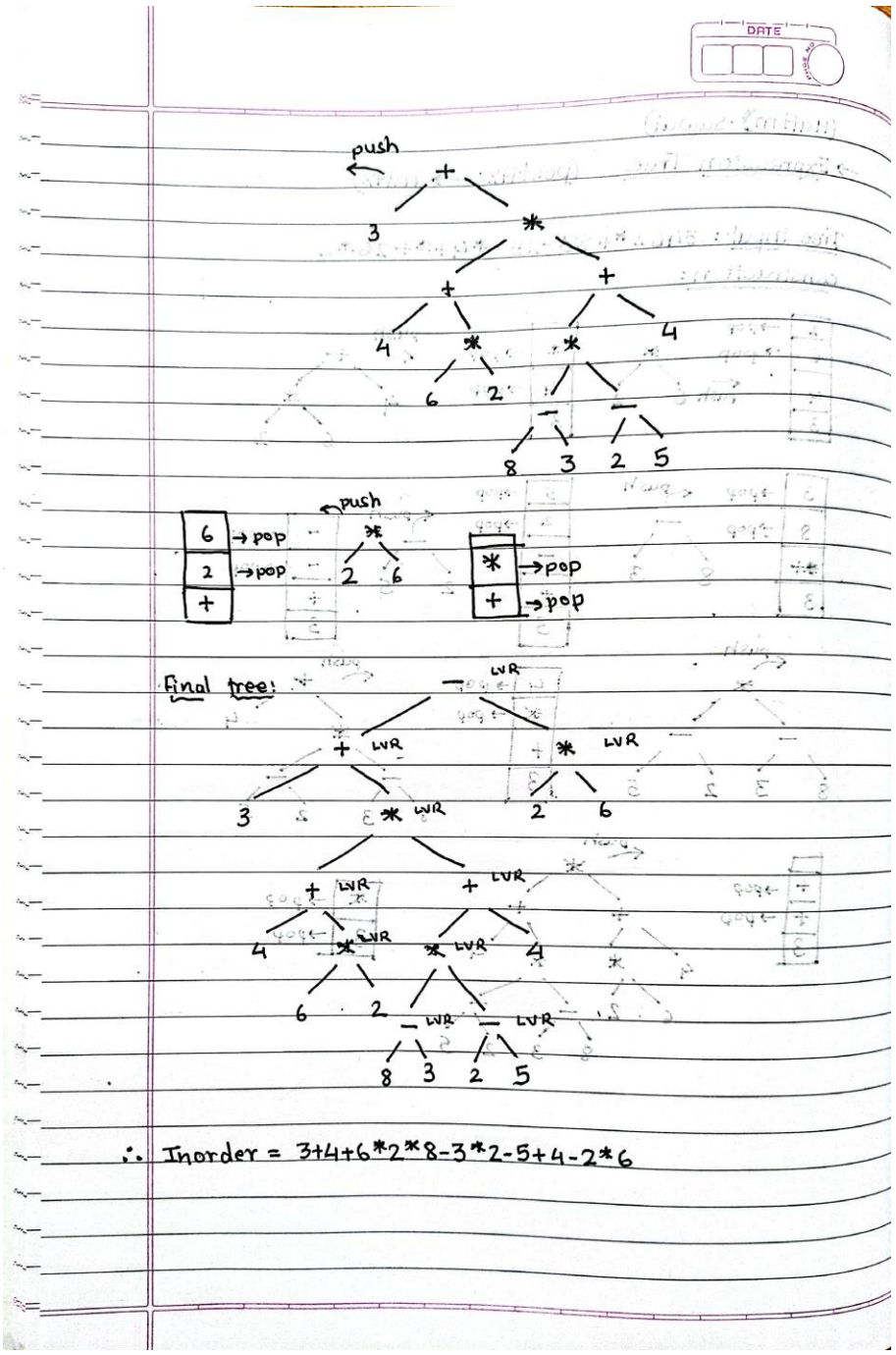


	Step-5: **Finally, symbol / is read, two trees are merged and a pointer to the final tree is left on the stack.** 
ALGORITHM:	1. Create Node Class common for both stack & tree classes 2. Create 2 ref vars: right, left & char data 3. Create LLStack Class & ExpTree Class **LLStack Class:** 1. Initialise top = -1, int size 2. Initialise Node array of size 20 3. Create a constructor for initialising the Node array with ' ' **Push Method:** 1. node[++top] = x 2. increment size by 1 **Pop Method:** 1. decrement size by 1 2. return node[--top] **ExpTree Class:** 1. Initialize root node to null 2. Create methods for diff operations for ExpTree **Bool isOperator Method:** 1. if char is equal to +,-,*,/ or ^ return true 2. else return false

	buildExpr Method: 1. pass char[] postfix & size to recurring function: 2. root = buildExprTree(postfix, size) **buildExprTree Method:** 1. Initialise new LLStack 'stack' 2. Initialise 3 nodes: t, t1,t2 3. for loop from i=0 to i<size to traverse the postfix array: a) if !isOperator(postfix[i]) is true: b) initialise t = postfix[i] & stack.push(t) c) Else: d) t = postfix[i] e) pop 2 nodes from stack and store in t1,t2 f) set right & left of t to t1 & t2 respectively 4. pop the final tree root stored as t 5. return t **PreOrder Method:** 1. if root is not null 2. print root.data 3. recurr: PreOrder(root.left) 4. recurr: PreOrder(root.right) **InOrder Method:** 1. if root is not null 2. recurr: PreOrder(root.left) 3. print root.data 4. recurr: PreOrder(root.right) **PostOrder Method:** 1. if root is not null 2. recurr: PreOrder(root.left) 3. recurr: PreOrder(root.right) 4. print root.data

PROBLEM SOLVING:




push
+
3
*
+
+
4
*
*
4
6
2
8
3
2
5
push
6
2
+
→pop
→pop
*
2
6
*
+
→pop
→pop
Final tree:
LVR
+
*
3
2
6
4
8
3
2
5
∴ Inorder = 3+4+6*2*8-3*2-5+4-2*6

PROGRAM:

ETCheck.java:

```
import java.util.Scanner;
import exptreeds.ExpTree;
public class ETCheck {
   public static void main(String[] args) {
      Scanner sc = new Scanner(System.in);
      ExpTree et = new ExpTree();
      System.out.print("Enter the postfix expression: ");
      String postfix = sc.nextLine();
      char[] expr = postfix.toCharArray();
      int size = expr.length;
      et.buildExpr(expr,size);
      System.out.println("infix expression is");
      et.inorder(et.root);
      System.out.println();
      System.out.println("prefix expression is");
      et.preorder(et.root);
      System.out.println();
      System.out.println("postfix expression is");
      et.postorder(et.root);
      System.out.println();
      System.out.println("value of expression is " + et.eval(et.root));
      sc.close();
   }
}
```

ExpTree.java:

```
package exptreeds;
class Node {
   char data;
   Node left, right;
   Node(char data) {
      this.data = data;
      left = right = null;
   }
}
```

```
class LLStack {
   int top = -1;
   public int size;
   Node node[] = new Node[20];
   LLStack() {
      for (int i = 0; i < node.length; i++) {
         node[i] = new Node(' ');
      }
   }
   void push(Node x) {
      node[++top] = x;
      size++;
   }
   Node pop() {
      size--;
      return node[top--];
   }
}
public class ExpTree {
   public Node root;
   public void buildExpr(char[] postfix,int size) {
      root = buildExprTree(postfix,size);
   }
   public boolean isOperator(char c) {
      if (c == '+' || c == '-' || c == '*' || c == '/' || c == '^') {
         return true;
      }
      return false;
   }
   private Node buildExprTree(char[] postfix,int size) {
      LLStack stack = new LLStack();
      Node t, t1, t2;
      for (int i = 0; i < size; i++) {
         if (!isOperator(postfix[i])) {
            t = new Node(postfix[i]);
            stack.push(t);
```

```
        } else {
            t = new Node(postfix[i]);
            t1 = stack.pop();
            t2 = stack.pop();
            t.right = t1;
            t.left = t2;
            stack.push(t);
        }
    }
    t = stack.pop();
    return t;
}
public int eval(Node root) {
    if (root == null) {
        return 0;
    }
    if (root.left == null && root.right == null) {
        return root.data-'0';
    }
    int l_val = eval(root.left);
    int r_val = eval(root.right);
    if (root.data == '+') {
        return l_val + r_val;
    }
    if (root.data == '-') {
        return l_val - r_val;
    }
    if (root.data == '*') {
        return l_val * r_val;
    }
    return l_val / r_val;
}
public void inorder(Node t) {
    if (t != null) {
        inorder(t.left);
        System.out.print(t.data);
```

```
            inorder(t.right);
        }
    }
    public void preorder(Node t) {
        if (t != null) {
            System.out.print(t.data);
            preorder(t.left);
            preorder(t.right);
        }
    }
    public void postorder(Node t) {
        if (t != null) {
            postorder(t.left);
            postorder(t.right);
            System.out.print(t.data);
        }
    }
}
```

OUTPUT:

```
Enter the postfix expression: 3462*+83-25-*4+*+26*-
infix expression is
3+4+6*2*8-3*2-5+4-2*6
prefix expression is
-+3*+4*62+*-83-254*26
postfix expression is
3462*+83-25-*4+*+26*-
value of expression is -185
PS D:\Data Structures\Exp6>
```

CONCLUSION: In this experiment, we learned how to read and store a postfix expression in an expression tree (variation of a binary tree) and traverse the tree using preorder & in order to convert the expression into prefix & infix expressions.