**🔷 THEORY**

**📌 Expression Tree:**

An **expression tree** is a binary tree representing expressions. Each internal node corresponds to an operator and each leaf node to an operand (variables or constants).

* Example (Prefix): +--a\*bc/def
* Equivalent Infix: (((a - (b \* c)) + ((d / e) - f)))

**📌 Tree Construction (Prefix Expression):**

* Prefix notation (Polish notation) places operators before operands.
* For prefix, tree is built **right-to-left**.
* Operands become **leaf nodes**, and operators become **internal nodes** with two children.

**📌 Post-order Traversal:**

* Post-order (Left → Right → Root) processes the entire expression.
* Used for evaluation of the tree or converting to postfix.

**📌 Deletion of Tree:**

* Post-order deletion ensures that children are deleted before their parent, avoiding memory leaks.

**🔷 ALGORITHMS**

**🔧 1. Construct Expression Tree from Prefix**

**Input:** Prefix expression string  
**Output:** Root of expression tree

text

CopyEdit

Algorithm constructExpressionTree(prefix):

1. Initialize an empty stack.

2. Traverse prefix from right to left:

a. If character is operand:

- Create a new node and push to stack.

b. If character is operator:

- Pop two nodes from stack.

- Create a new node with operator.

- Set popped nodes as left and right children.

- Push new node to stack.

3. Return the top of stack (root of expression tree).

**🔍 2. Non-Recursive Post-order Traversal**

**Input:** Root of expression tree  
**Output:** Print post-order sequence

text

CopyEdit

Algorithm postOrderTraversal(root):

1. Initialize stack, current = root, lastVisited = NULL

2. While stack not empty or current not NULL:

a. If current != NULL:

- Push current to stack

- Move to left child

b. Else:

- Peek top of stack

- If right child exists and not lastVisited:

- Move to right child

- Else:

- Print data, pop, update lastVisited

**🧹 3. Delete the Tree (Recursive)**

**Input:** Root of expression tree  
**Output:** Memory freed

text

CopyEdit

Algorithm deleteTree(root):

1. If root is NULL, return

2. deleteTree(root->left)

3. deleteTree(root->right)

4. delete root

