4. Expression Tree

TITLE: Construct an Expression Tree from postfix and prefix expression. Perform recursive and non- recursive In-order, pre-order and post-order traversals.

OBJECTIVE:

1. Understand the concept of expression tree and binary tree.
2. Understand the different type of traversals (recursive & non-recursive).

THEORY:

1. Definition of an expression tree with diagram:

Algebraic expressions such as: a/b + (c-d) e

The terminal nodes (leaves) of an expression tree are the variables or constants in the expression (a, b, c, d, and e). The non-terminal nodes of an expression tree are the operators (+, -, ×, and ÷). Notice that the parentheses which appear in Equation do not appear in the tree. Nevertheless, the tree representation has captured the intent of the parentheses since the subtraction is lower in the tree than the multiplication.

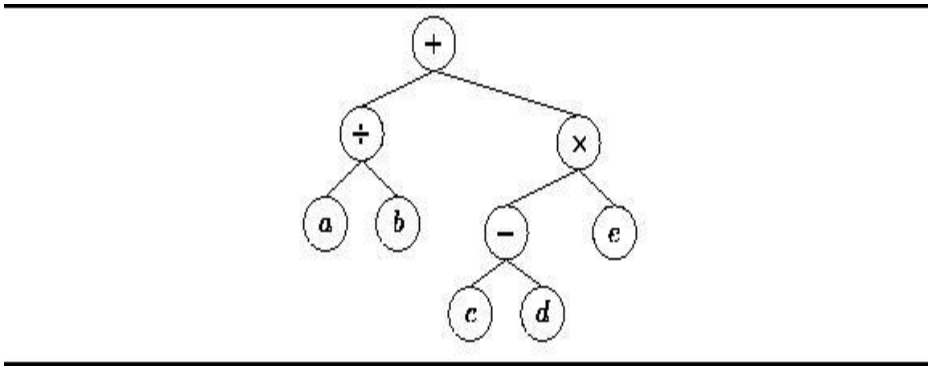


Figure: Tree representing the expression a/b+(c-d)e.

2. **Different type of traversals with example:**

To traverse a non-empty binary tree in **preorder**,

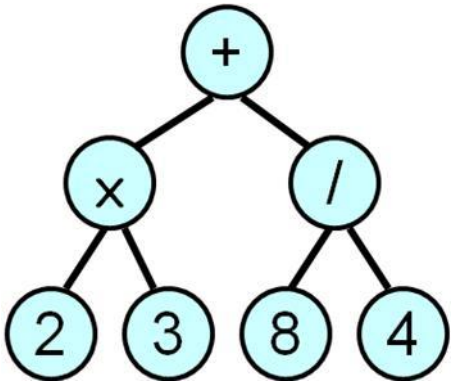
1. Visit the root.
2. Traverse the left subtree.
3. Traverse the right subtree.

To traverse a non-empty binary tree in **inorder**:

1. Traverse the left subtree.
2. Visit the root.
3. Traverse the right subtree.

To traverse a non-empty binary tree in **postorder**,

1. Traverse the left subtree.
2. Traverse the right subtree.
3. Visit the root.



- **Pre-order (prefix)**
  + * 2 3 / 8 4
- **In-order (infix)**
  2* 3 + 8 / 4
- **Post-order (postfix)**
  2 3 * 8 4 / +

ALGORITHM:

Define structure for Binary Tree (Information, Left Pointer & Right Pointer)

Create Expression Tree:

CreateTree()

Root& Node pointer variable of type structure. Stack is an pointer array of type structure. String is character array which contains postfix expression. Top & I are variables of type integer.

Step 1: Top = -1 , I = 0;
Step 2: Do Steps 3,4,5,6 While String[I] != NULL
Step 3: Create Node of type of structure
Step 4: Node->Data=String[I];
Step 5: If isalnum(String[I])
Then Stack[Top++] = Node;
Else
Node->Right = Stack [--Top];
Node->Left = Stack[--Top];
Stack[Top++] = Node;
Step 6: Increment I;
Step 7: Root = Stack[0];
Step 8: Return Root

Inorder Traversal Recursive :

Tree is pointer of type structure.
InorderR(Tree)
Step 1: If Tree != NULL

Step 2: InorderR(Tree->Left)
Step 3: Print Tree->Data
Step 4: InorderR(Tree->Right)

Postorder Traversal Recursive:

Tree is pointer of type structure.
PostorderR(Tree)
Step 1: If Tree != NULL
Step 2: PostorderR(Tree->Left)
Step 3: PostorderR(Tree->Right)
Step 4: Print Tree->Data

Preorder Traversal Recursive:

Tree is pointer of type structure.
PreorderR(Tree)
Step 1: If Tree != NULL
Step 2: Print Tree->Data
Step 3: PreorderR(Tree->Left)
Step 4: PreorderR(Tree->Right)

Postorder Traversal Nonrecursive :

NonR_Postorder(Tree)
Tree, Temp is pointer of type structure. Stack is pointer array of type structure.Top variable of type integer.
Step 1: Temp = Tree // current pointer pointing to root
Step 2: if Temp != NULL then push current pointer along with its initial flag value _L‘ on to the stack and traverse on the left side.
Step 3: Otherwise if the stack is not empty then pop an address from stack along with its flag value.
Step 4: For the current pointer if the flag value is _L‘ then change it to _R‘ and push current pointer along with its flag value _R‘ on to the stack and traverse on the right side.
Step 5: otherwise, For the current pointer if the flag value is _R‘ then display the element and make the current pointer NULL (i.e.temp=NULL)
Step 6: repeat steps 2, 3, 4, 5 until the stack becomes empty.

Preorder Traversal Nonrecursive :

NonR_Preorder(Tree)
Tree, Temp is pointer of type structure. Stack is pointer array of type structure. Top variable of type integer.
Step 1: Temp = Tree
Step 2: Do Steps3,4,5,6,7,& 8 While Temp != NULL And Stack is not Empty
Step 3: Do Steps 4,5&6 While Temp != NULL
Step 4: Print Temp->Data
Step 5: Stack[++ Top] = Temp //Push Element
Step 6: Temp = Temp->Left
Step 7: Temp = Stack [Top --] //Pop Element
Step 8: Temp = Temp->Right

Inorder Traversal Nonrecursive :

NonR_Inorder(Tree)
Tree, Temp is pointer of type structure. Stack is pointer array of type structure. Top variable of type integer.
Step 1: Temp = Tree
Step 2: Do Steps3,4,5,6,7,&8 While Temp != NULL And Stack is not Empty
Step 3: Do Steps 4,5 While Temp != NULL
Step 4: Stack[++ Top] = Temp;//Push Element
Step 5: Temp = Temp->Left
Step 6: Temp = Stack[Top --]//Pop Element
Step 7: Print Temp->Data
Step 8: Temp = Temp->Right

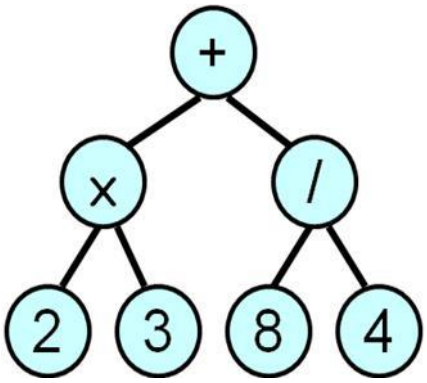
INPUT:

Postfix Expression: **2 3 * 8 4 / +**

OUTPUT:

Display result of each operation with error checking.

Expression tree



CONCLUSION: Thus, we have studied expression tree from prefix/postfix expression and perform recursive and non-recursive in-order, post-order, pre-order traversals.

FAQ:

1. What is tree?
2. What is Expression tree?
3. What is pre-order, post-order, in-order traversal?
4. Difference between recursive & Non-recursive traversal?
5. What is Binary tree, Binary search tree, Expression tree & General tree?