**Assignment No : 04**

**Title: Construct Expression Tree**

**Course Outcome:** **CO1(C214447.1):** Analyze algorithms and to determine algorithm correctness and time efficiency class.

**CO2(C214447.2):** Implement abstract data type (ADT) and data structures for given application.

**CO3(C214447.3):** Design algorithms based on techniques like brute -force, divide and conquer, greedy, etc).

**CO5(C24447.5):** Analyze of algorithms with respect to time and space complexity.

**Date of Completion: 28/11/2021**

**Assessment Grade / Marks:**

**Assessor’s Sign with Date:**

**Assignment No: 04**

**Title:** Construct an Expression Tree

**Aim:** Construct an Expression Tree from Postfix and Prefix expression and perform Inorder , Preorder and Postorder Traversal

**Objective:** Construct an Expression Tree from Postfix and Prefix expression and perform Inorder ,Preorder and Postorder Traversal

**Problem Statement:** Construct an Expression Tree from postfix and prefix

expression.

**Perform :-**

**• Recursive traversals**

In-order

Pre-order

Post-order

**• Non-recursive traversals**

In-order,

Pre-order

Post-order

**Course Outcome:** CO Number: Applicable CO : Blooms Taxonomy Category

**Requirements:** (Components / Digital Kits / Platform / Software / Hardware)

**Platform :-** Online GDB Compiler

**Theory / Procedure / Diagrams / Circuits:**

**Expression Tree:-**

Expression trees are those in which the leaf nodes have the values to be operated, and internal nodes contain the operator on which the leaf node will be performed.To construct an Expression Tree for the given expression, we generally use Stack Data Structure.

Initially we Iterate over the given postfix expression and follow the steps as given below :-

1. If we get an operand in the given expression, then push it in the stack. It will become the root of the expression Tree.

2. If an operator gets two values in the expression, then add in the expression tree as its child, and push them in the current node.

3. Repeat Step-1 and Step-2 until we do not complete over the given expression.

4. Now check if every root node contains nothing but operands and every child node contains only values.

**InOrder Traversal:-**

An inorder traversal is a traversal technique that follows the policy, i.e., Left Root Right. Here, Left Root Right means that the left subtree of the root node is traversed first, then the root node, and then the right subtree of the root node is traversed. Here, inorder name itself suggests that the root node comes in between the left and the right subtrees.

**PreOrder Traversal:-**

A preorder traversal is a traversal technique that follows the policy, i.e., Root Left Right. Here, Root Left Right means root node of the tree is traversed first, then the left subtree and finally the right subtree is traversed. Here, the Preorder name itself suggests that the root node would be traversed first.

**PostOrder Traversal:-**

A Postorder traversal is a traversal technique that follows the policy, i.e., Left Right Root. Here, Left Right Root means the left subtree of the root node is traversed first, then the right subtree, and finally, the root node is traversed. Here, the Postorder name itself suggests that the root node of the tree would be traversed at the last.

**Algorithm / Methods / Steps: (if applicable):**

MAIN FUNCTION()

STEP 1: Create a 'Node' structure with three members data,left and right.

STEP 2: Create a ‘Stack’ structure with two members Root and next.

PUSH FUNCTION()

STEP 1: Create a newNode

STEP 2: Set newNode-->Root = Root

STEP 3: Set newNode-->next = \*Top

STEP 4: Set \*Top = newNode

POP FUNCTION()

STEP 1: Define a Node pointer 'temp' and set it to 'Top'

STEP 2: Set \*Top = (\*Top)->next;

STEP 3: Return temp

POSTFIX\_EXPRESSION\_TREE FUNCTION()

STEP 1: Set ‘Top’ pointer to’NULL’

STEP 2: FOR (int i = 0; i < Postfix.length (); i++)

STEP 3: IF (!is\_Operator (Postfix[i]))

STEP 4: Set temp->data = Postfix[i]

STEP 5: Set temp->left = temp->right = NULL

STEP 6: Push (temp, &Top)

STEP 7: END IF

STEP 8: ELSE

STEP 9: Set right = Pop (&Top)

STEP 10: Set left = Pop (&Top)

STEP 11: Set temp->data = Postfix[i]

STEP 12: Set temp->left = left

STEP 13: Push (temp, &Top)

STEP 14: END FOR

STEP 15: Return Pop (&Top)

PREFIX\_EXPRESSION\_TREE FUNCTION()

STEP 1: Set ‘Top’ pointer to’NULL’

STEP 2: Set string Postfix = reverse (Prefix)

STEP 3: FOR (int i = 0; i < Postfix.length (); i++)

STEP 4: IF (!is\_Operator (Postfix[i]))

STEP 5: Set temp->data = Postfix[i];

STEP 6: Set temp->left = temp->right = NULL

STEP 7: Push (temp, &Top)

STEP 8: END IF

STEP 9: ELSE

STEP 10: Set temp->data = Postfix[i]

STEP 11: Set temp->left = Pop (&Top)

STEP 12: Set temp->right = Pop (&Top)

STEP 13: Push (temp, &Top)

STEP 14: END FOR

STEP 15: Return Pop (&Top)

INORDER\_TRAVERSAL\_RECURSIVE FUNCTION()

STEP 1: IF (Root != NULL)

STEP 2: Get Inorder\_Traversal\_Recursion (Root->left)

STEP 3: Get Root->data

STEP 4: Get Inorder\_Traversal\_Recursion (Root->right)

STEP 5: END IF

PREORDER\_TRAVERSAL\_RECURSIVE FUNCTION()

STEP 1: IF (Root != NULL)

STEP 2: Get Root->data

STEP 3: Get Preorder\_Traversal\_Recursion (Root->left)

STEP 4: Get Preorder\_Traversal\_Recursion (Root->right)

STEP 5: END IF

POSTORDER\_TRAVERSAL\_RECURSIVE FUNCTION()

STEP 1: IF (Root != NULL)

STEP 2: Get Postorder\_Traversal\_Recursion (Root->left);

STEP 3: Get Postorder\_Traversal\_Recursion (Root->right);

STEP 4: Get Root->data

STEP 5: END IF

INORDER\_TRAVERSAL\_NON\_RECURSIVE FUNCTION()

STEP 1: IF (Root != NULL)

STEP 2: Set ‘Top’ pointer to’NULL’

STEP 3: Define ‘temp’ Pointer

STEP 4: Define ‘current Node’ and Set it = Root

STEP 5: WHILE (current\_Node != NULL || Top != NULL)

STEP 6: WHILE (current\_Node != NULL)

STEP 7: Push (current\_Node, &Top)

STEP 8: Set current\_Node = current\_Node->left

STEP 9: END WHILE

STEP 10: IF (current\_Node == NULL && Top != NULL)

STEP 11: Set temp = Pop (&Top)

STEP 12: Get temp->data

STEP 13: Set current\_Node = temp->right

STEP 14: END WHILE

STEP 15: END IF

PREORDER\_TRAVERSAL\_NON\_RECURSIVE FUNCTION()

STEP 1: IF (Root != NULL)

STEP 2: Set ‘Top’ pointer to’NULL’

STEP 3: Define ‘temp’ Pointer

STEP 4: Push (Root, &Top)

STEP 5: WHILE (Top != NULL)

STEP 6: Get temp->data

STEP 7: IF (temp->right != NULL)

STEP 8: Push (temp->right, &Top)

STEP 9: END IF

STEP 10: IF (temp->left != NULL)

STEP 11: Push (temp->left, &Top)

STEP 12: END IF

STEP 13: END WHILE

STEP 14: END IF

POSTORDER\_TRAVERSAL\_NON\_RECURSIVE FUNCTION()

STEP 1: IF (Root != NULL)

STEP 2: Define ‘temp’ Pointer

STEP 3: Define ‘Num 1’ and ‘Num 2’ Pointer AND Set them = NULL

STEP 4: Push (Root, &Num1)

STEP 5: WHILE (Num1 != NULL)

STEP 6: Set temp = Pop (&Num1)

STEP 7: Push (temp, &Num2)

STEP 8: IF (temp->left != NULL)

STEP 9: Push (temp->left, &Num1)

STEP 10: END IF

STEP 11: IF (temp->right != NULL)

STEP 12: Push (temp->right, &Num1);

STEP 13: END IF

STEP 14: END WHILE

STEP 15: WHILE (Num2 != NULL)

STEP 16: Get Pop (&Num2)->data

STEP 17: END WHILE

STEP 18: END IF

**Input: (Test Cases / Data sets / Database Links)**

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

From which expression you want to construct binary expression tree?

1) Prefix Expression.

2) Postfix Expression.

3) Exit.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Enter Your Choice: 1

...............................................

Enter Prefix Expression: +/\*+ab-cdef

In which manner you want to traverse the binary expression tree?

1) Recursively.

2) Non-Recursively.

3) Exit.

Enter Your Choice: 1

Recursive preorder is: + / \* + a b - c d e f

Recursive inorder is: a + b \* c - d / e + f

Recursive postorder is: a b + c d - \* e / f +

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

From which expression you want to construct binary expression tree?

1) Prefix Expression.

2) Postfix Expression.

3) Exit.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Enter Your Choice: 2

...............................................

Enter Postfix Expression: ab+cd-\*e/f+

In which manner you want to traverse the binary expression tree?1) Recursively.

2) Non-Recursively.

3) Exit.

Enter Your Choice: 2

Non-Recursive preorder is: + / \* + a b - c d e f

Non-Recursive inorder is: a+b\*c-d/e+f

Non-Recursive postorder is: a b + c d - \* e / f +

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

From which expression you want to construct binary expression tree?

1) Prefix Expression.

2) Postfix Expression.

3) Exit.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Enter Your Choice:

**Inference:** Hence we have studied the construction of Expression Tree and InOrder , PreOrder and PostOrder operations using Recursive and Non-Recursive traversal