##### **Experiment No. 05**

**Aim:** To illustrate the concept of Binary tree for expression tree.

**Problem Statement :**

Construct an expression tree from the given prefix expression eg. +--a\*bc/def and traverse it using post order traversal (non recursive) and then delete the entire tree.

**Learning Objectives:**

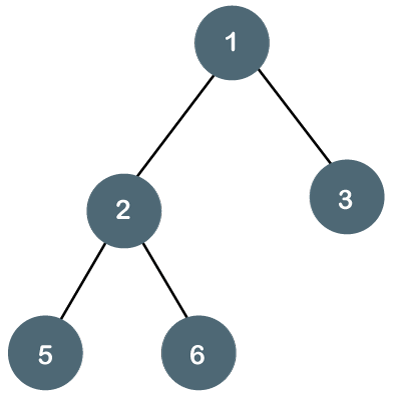
* To understand directed and undirected graph.
* To implement program to represent graph using adjacency matrix and list.

**Learning Outcome:**

* To implement the basic concept of Binary Search Tree to store a numbers in it.
* To perform basic Operation Insert, Delete and search, Traverse in tree in Data structure.
* Apply and analyze non linear data structures to solve real world complex problems.
* Student able to implement program for Binary tree representation.

**Theory:**

Binary Tree: a binary tree is **a tree data structure in which each node has at most two children**, which are referred to as the left child and the right child.



**Traversal of Binary Trees:**

Traversal is a process to visit all the nodes of a tree and may print their values too. Because, all nodes are connected via edges (links) we always start from the root (head) node. That is, we cannot randomly access a node in a tree. There are three ways which we use to traverse a tree –

* In-order Traversal
* Pre-order Traversal
* Post-order Traversal

## **Pre-order Traversal:** In this traversal method, the root node is visited first, then the left subtree and finally the right subtree.

## **Post-order Traversal**

In this traversal method, the root node is visited last, hence the name. First we traverse the left subtree, then the right subtree and finally the root node.

## **In-order Traversal**

In this traversal method, the left subtree is visited first, then the root and later the right sub-tree. We should always remember that every node may represent a subtree itself.

For Above Binary Tree the different travel sequences are as follow;

In order: 5, 2, 6, 1, 3

Pre-order; 1, 2, 5, 6, 3

Post-Order: 5, 6,2, 3, 1

**Expression** **Tree**: The expression tree is a binary tree in which each internal node corresponds to the operator and each leaf node corresponds to the operand.

For example, the postfix notation a b + c d e + \* \* results in the following expression tree. The corresponding infix notation is (a+b)\*(c\*(d+e)) which can be produced by traversing the expression tree in an inorder fashion. However, an opening and closing parenthesis must be added at the beginning and end of each expression (every subtree represents a subexpression).

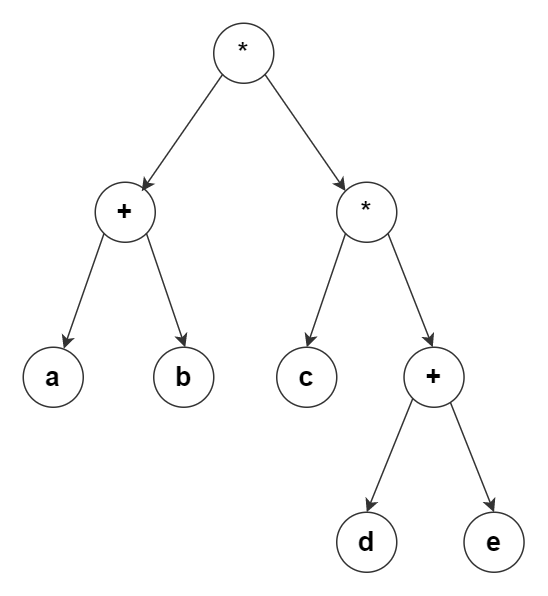


Figure : Expression Tree

The construction of the expression tree takes place by reading the postfix expression one symbol at a time. If the symbol is an operand, a new binary tree node is created, and its pointer is pushed onto a stack. If the symbol is an operator, the pointers to two trees, x and y, are popped from the stack, and a new tree whose root is the operator and whose left and right children point to y and x, respectively is formed. A pointer to this new tree is then pushed to the stack. In the end, a pointer to the full expression tree remains on the stack

**How to construct an expression tree?**

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.
   1. A finite set of vertices also called as nodes.
   2. A finite set of ordered pair of the form (u, v) called as edge. The pair is ordered because (u, v) is not same as (v, u) in case of directed graph(di-graph). The pair of form (u, v) indicates that there is an edge from vertex u to vertex v. The edges may contain weight/value/cost.

**Conclusion:** Binary tree can be used efficiently to implement Expression tree.

Questions

1. Explain Binary tree?
2. Define Expression tree.
3. What are the methods to visit binary tree.