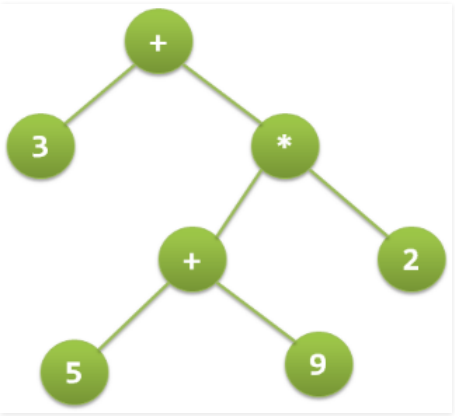
Expression Tree

Expression tree is a binary tree in which each internal node corresponds to operator and each leaf node corresponds to operand so for example expression tree for 3 + ((5+9)*2) would be:



Inorder traversal of expression tree produces infix version of given postfix expression (same with preorder traversal it gives prefix expression)

Evaluating the expression represented by expression tree:

```
Let t be the expression tree
If  t is not null then
      If t.value is operand then
                Return  t.value
      A = solve(t.left)
      B = solve(t.right)

      // calculate applies operator 't.value'
      // on A and B, and returns value
      Return calculate(A, B, t.value)
```

Construction of Expression Tree:

Now For constructing expression tree we use a stack. We loop through input expression and do following for every character.

1) If character is operand push that into stack

2) If character is operator pop two values from stack make them its child and push current node again.

At the end only element of stack will be root of expression tree.

Recommended: Please solve it on "*PRACTICE*" first, before moving on to the solution.

Below is the implementation :

C/C++ Java Python

```python
# Python program for expression tree

# An expression tree node
class Et:

    # Constructor to create a node
    def __init__(self , value):
        self.value = value
        self.left = None
        self.right = None

# A utility function to check if 'c'
# is an operator
def isOperator(c):
    if (c == '+' or c == '-' or c == '*'
        or c == '/' or c == '^'):
        return True
    else:
        return False

# A utility function to do inorder travers
def inorder(t):
    if t is not None:
        inorder(t.left)
        print t.value,
        inorder(t.right)

# Returns root of constructed tree for
# given postfix expression
def constructTree(postfix):
    stack = []

    # Traverse through every character of
```

```
    for char in postfix :

        # if operand, simply push into sta
        if not isOperator(char):
            t = Et(char)
            stack.append(t)

        # Operator
        else:

            # Pop two top nodes
            t = Et(char)
            t1 = stack.pop()
            t2 = stack.pop()

            # make them children
            t.right = t1
            t.left = t2

            # Add this subexpression to st
            stack.append(t)

    # Only element  will be the root of ex
    t = stack.pop()

    return t

# Driver program to test above
postfix = "ab+ef*g*-"
r = constructTree(postfix)
print "Infix expression is"
inorder(r)
```

Run on IDE

Output:

```
infix expression is
a + b - e * f * g
```

This article is contributed by **Utkarsh Trivedi**. Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.