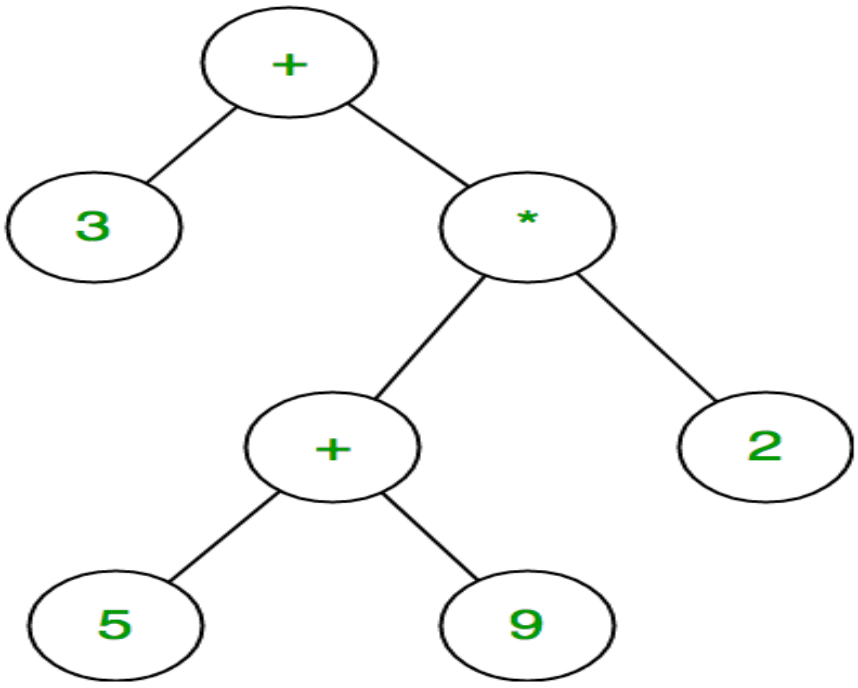
Expression Tree

What Does Expression Tree Mean?

An expression tree is a representation of expressions arranged in a tree-like data structure. In other words, it is a tree with leaves as operands of the expression and nodes contain the operators. Similar to other data structures, data interaction is also possible in an expression tree. Expression trees are mainly used for analyzing, evaluating and modifying expressions, especially complex expressions.

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 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 postorder traversal it gives postfix expression)

Evaluating the expression represented by an 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 an expression tree we use a stack. We loop through input expression and do the following for every character.

If a character is an operand push that into the stack

If a character is an operator pop two values from the stack make them its child and push the current node again.

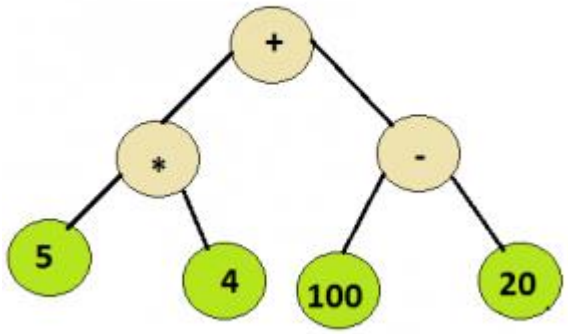
In the end, the only element of the stack will be the root of an expression tree.

Evaluation of Expression Tree

Given a simple expression tree, consisting of basic binary operators i.e., + , – ,* and / and some integers, evaluate the expression tree.

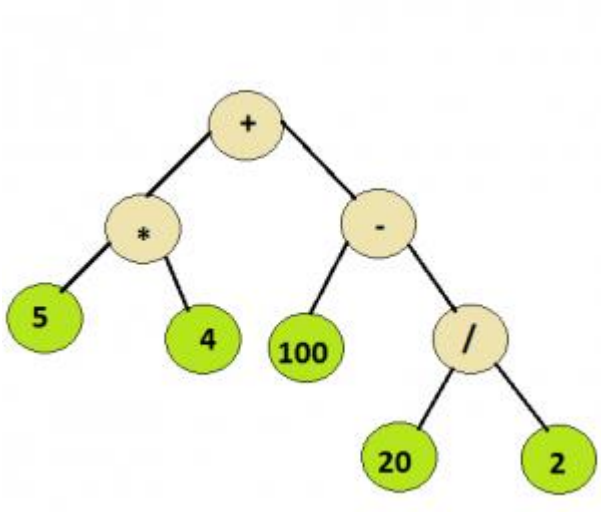
Examples:

Input: Root node of the below tree



Output:100

Input: Root node of the below tree



Output: 110

Approach: The approach to solve this problem is based on following observation:

As all the operators in the tree are binary, hence each node will have either 0 or 2 children. As it can be inferred from the examples above, all the integer values would appear at the leaf nodes, while the interior nodes represent the operators.

Therefore we can do inorder traversal of the binary tree and evaluate the expression as we move ahead.

To evaluate the syntax tree, a recursive approach can be followed.

Algorithm:

Let t be the syntax tree

If t is not null then

If t.info is operand then

Return t.info

Else

A = solve(t.left)

B = solve(t.right)

return A operator B, where operator is the info contained in t