3.3 EXPRESSION TREES

- Expression Tree is a binary tree in which the leaf nodes are operands and the interior nodes are operators.
- Like binary tree, expression tree can also be traversed by inorder, preorder and postorder traversal.

Constructing an Expression Tree

Let us consider postfix expression given as an input for constructing an expression tree

1. Read one symbol at a time from the postfix expression.

2. Check whether the symbol is an operand or operator.

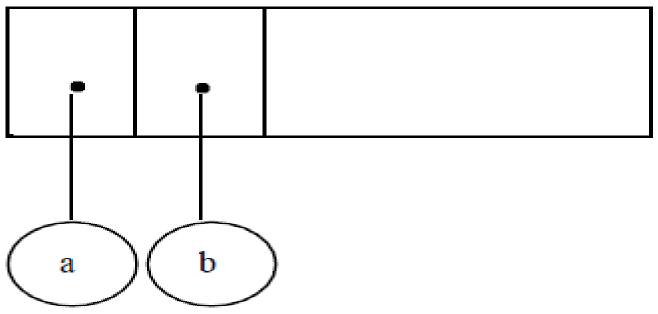
(a) If the symbol is an operand, create a one - node tree and push a pointer on to the stack.

(b) If the symbol is an operator pop two pointers from the stack namely T1 and T2 and form a new tree with root as the operator and T2 as a left child and T1 as a right child. A pointer to this new tree is then pushed onto the stack.

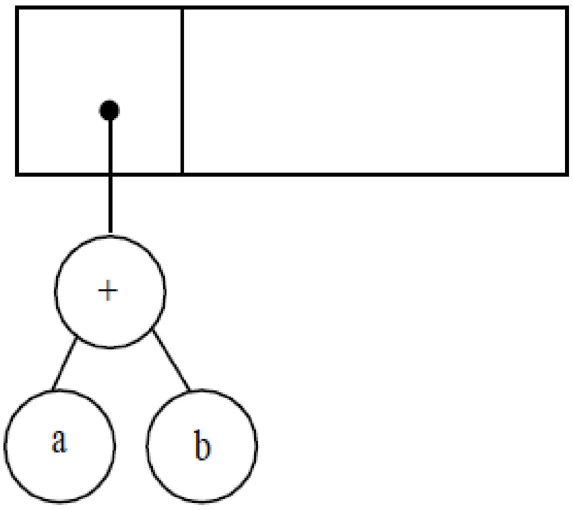
Example: -

ab + c *

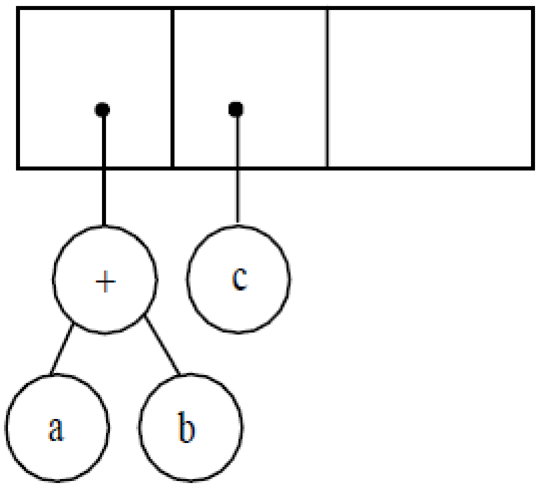
The first two symbols are operand, so create a one node tree and push the pointer on to the stack.



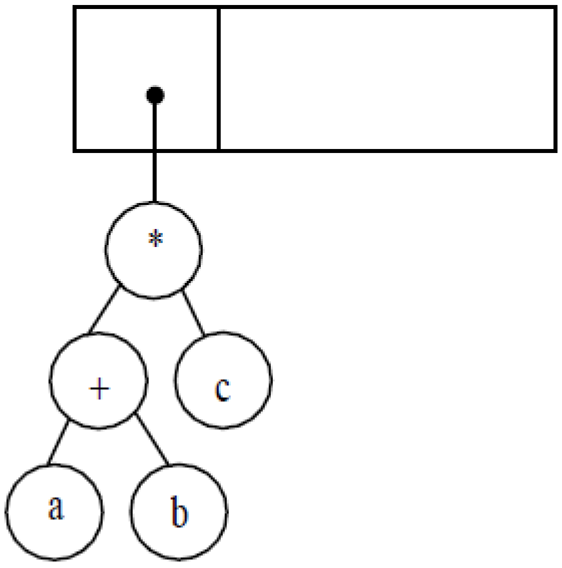
Next „+" symbol is read, so two pointers are popped, a new tree is formed and a pointer to this is pushed on to the stack.



Next the operand C is read, so a one node tree is created and the pointer to it is pushed onto the stack.



Now „*" is read, so two trees are merged and the pointer to the final tree is pushed onto the stack



Advantages of Expression Trees

1. Clearly represents **arithmetic expressions** in tree form
2. Simplifies **expression evaluation**
3. Useful for **converting expressions** (infix, prefix, postfix)
4. Helps in **syntax analysis** in compilers
5. Supports easy **optimization of expressions**

Disadvantages of Expression Trees

1. **Complex to construct** compared to simple expressions
2. Requires **more memory** due to node pointers
3. Traversal logic can be **difficult for beginners**
4. Not suitable for **very simple calculations**

Applications of Expression Trees

1. **Compilers and interpreters** for expression evaluation

2. **Arithmetic expression parsing**
3. **Syntax trees** in programming languages
4. **Mathematical computation systems**
5. **Expression optimization** in code generation