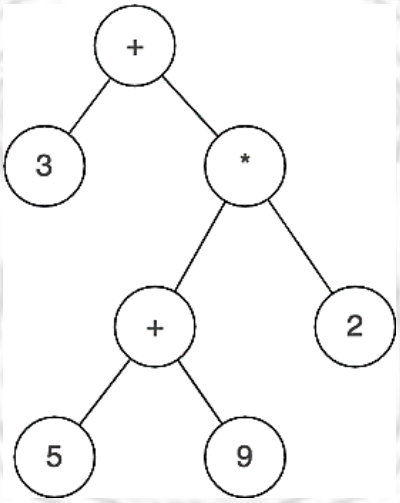
EXPRESSION TREES

- The expression tree is a tree used to represent the various expressions. The tree data structure is used to represent the expressional statements. In this tree, the internal node always denotes the operators. The leaf nodes always denote the operands.
- For example, expression tree for 3 + ((5+9)*2) would be:



Properties of an Expression tree

- In this tree, the internal node always denotes the operators.
- The leaf nodes always denote the operands.
- The operations are always performed on these operands.
- The operator present in the depth of the tree is always at the highest priority.
- The operator, which is not much at the depth in the tree, is always at the lowest priority compared to the operators lying at the depth.
- The operand will always present at a depth of the tree; hence it is considered the highest priority among all the operators.

Construction of Expression Tree

- Let us consider a postfix expression is given as an input for constructing an expression tree. Following are the step to construct an expression tree:
  - Read one symbol at a time from the postfix expression.
  - Check if the symbol is an operand or operator.

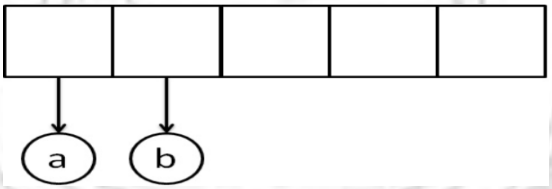
- ❖ If the symbol is an operand, create a one node tree and push a pointer onto a stack
- ❖ If the symbol is an operator, pop two pointers from the stack namely T1 & T2 and form a new tree with root as the operator, T1 & T2 as a left and right child
- ❖ A pointer to this new tree is pushed onto the stack

➢ Thus, an expression is created or constructed by reading the symbols or numbers from the left. If operand, create a node. If operator, create a tree with operator as root and two pointers to left and right subtree

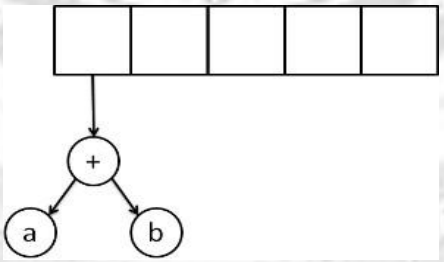
Example - Postfix Expression Construction

➢ **The input is: a b + c ***

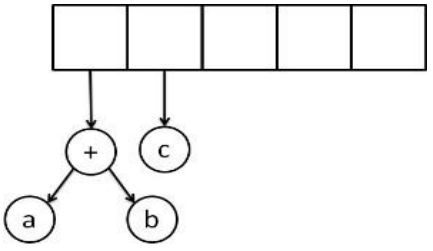
- o The first two symbols are operands, we create one-node tree and push a pointer to them onto the stack.



- o Next, read a'+' symbol, so two pointers to tree are popped, a new tree is formed and push a pointer to it onto the stack.

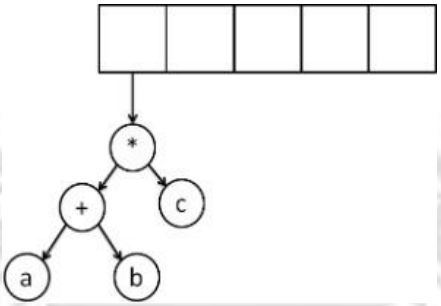


- o Next, 'c' is read, we create one node tree and push a pointer to it onto the stack.



- o Finally, the last symbol is read ' * ', we pop two tree pointers and form a new tree with a, ' * ' as root, and a pointer to the final tree remains on

thestack.



Implementation of Expression tree in C Programming language

// C program for expression tree implementation

```
#include <stdio.h>
#include <stdlib.h>
/* The below structure node is defined as a node of a binary tree consists
of left child and the right child, along with the pointer next which points to the
next node */
struct node
{
   char info ;
   struct node* l ;
   struct node* r ;
   struct node* nxt ;
};
struct node *head=NULL;
/* Helper function that allocates a new node with
thegiven data and NULL left and right pointers. */
struct node* newnode(char data)
{
   struct node* node = (struct node*) malloc ( sizeof ( struct node ) )
   ;node->info = data ;
   node->l = NULL ;
```

```
    node->r = NULL ;
    node->nxt = NULL
    ;return ( node ) ;
}
void Inorder(struct node* node)
{
    if ( node == NULL)
        return ;
    else
    {
    /* first recur on left child */
    Inorder ( node->l ) ;
    /* then print the data of node */
    printf ( "%c " , node->info ) ;
    /* now recur on right child */
    Inorder ( node->r ) ;
    }
}
void push ( struct node* x )
{
    if ( head == NULL )
    head = x ;
    else
    {
        ( x )->nxt = head ;
        head = x ;
    }
    // struct node* temp ;
    // while ( temp != NULL )
```

```
    // {
    //  printf ( " %c " , temp->info ) ;
    //  temp = temp->nxt ;
    // }
}
struct node* pop()
{
    // Poping out the top most [pointed with head] element
    struct node* n = head ;
    head = head->nxt ;
    return n ;
}
int main()
{
    char t[] = { 'X' , 'Y' , 'Z' , '*' , '+' , 'W' , '/' } ;
    int n = sizeof(t) / sizeof(t[0]) ;
    int i ;
    struct node *p , *q , *s ;
    for ( i = 0 ; i < n ; i++ )
    {
        // if read character is operator then popping two
        // other elements from stack and making a binary
        // tree
        if ( t[i] == '+' || t[i] == '-' || t[i] == '*' || t[i] == '/' || t[i] == '^' )
        {
            s = newnode ( t [ i ] ) ;
            p = pop() ;
            q = pop()
            ;s->l = q ;
```

```
            s->r = p;
            push(s);
        }
        else {
            s = newnode ( t [ i ] ) ;
            push ( s ) ;
        }
    }
    printf ( " The Inorder Traversal of Expression Tree: " ) ;
    Inorder ( s ) ;
    return 0 ;
}
```

The output of the above program is

X + Y * Z / W

Use of Expression tree

- The main objective of using the expression trees is to make complex expressions and can easily be evaluated using these expression trees.
- It is also used to find out the associativity of each operator in the expression.
- It is also used to solve the postfix, prefix, and infix expression evaluation.