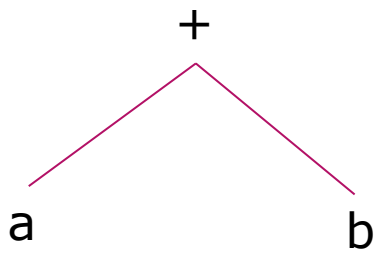
Evaluation of Postfix Expression and Conversion of Postfix Expression to Expression Tree

Overview

- **Tree Traversal Algorithms**
  - **Preorder**
  - **Inorder**
  - **Postorder**
- **Postfix Expression**
  - **Evaluation**
  - **Conversion to Expression Tree**

Expression Tree – Traversals

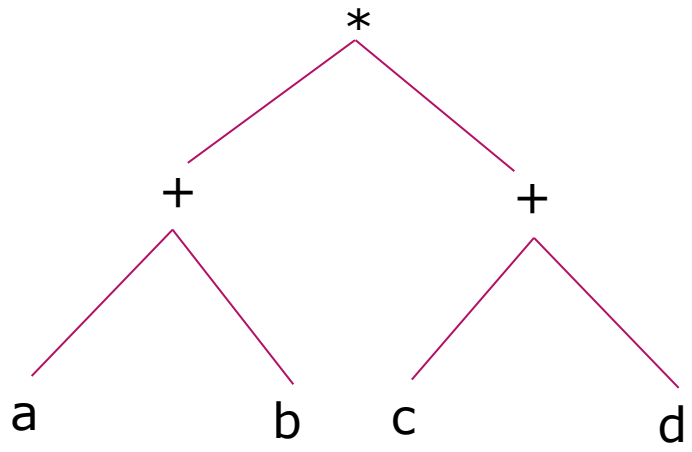


+ab **Preoder**

a+b **inorder**

ab+ **postorder**

Expressions- Infix, Prefix, Postfix



***+ab+cd prefix form**

a+b*c+d infix form

ab+cd+* postfix form

Tree Traversal Exercise

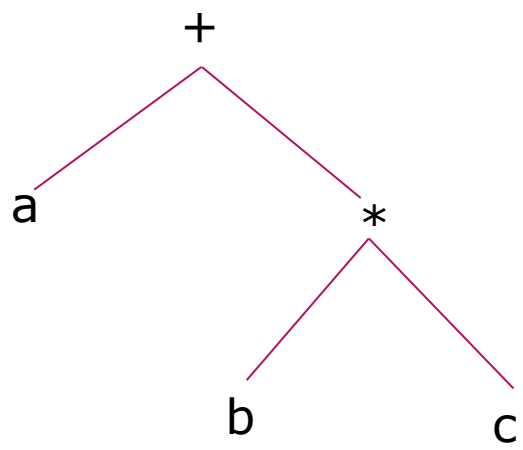
- **Given two traversals for a binary tree, can you construct the tree?**
  - **Inorder, Preorder**
  - **Inorder, Postorder**
  - **Preorder, Postorder**

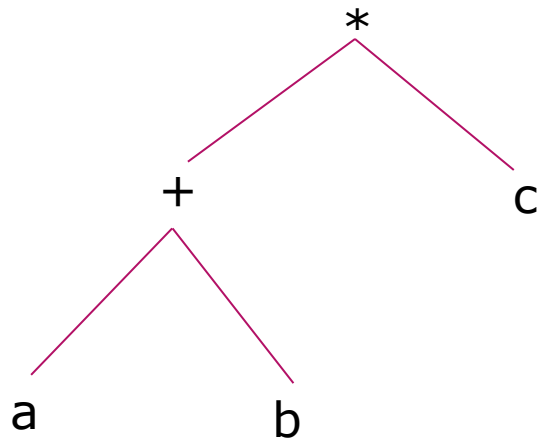
Tree Traversal Exercise

- **Iterative algorithm for tree traversal**
  - **Using Stack**
- **Level order traversal**

Expression Tree - Evaluation

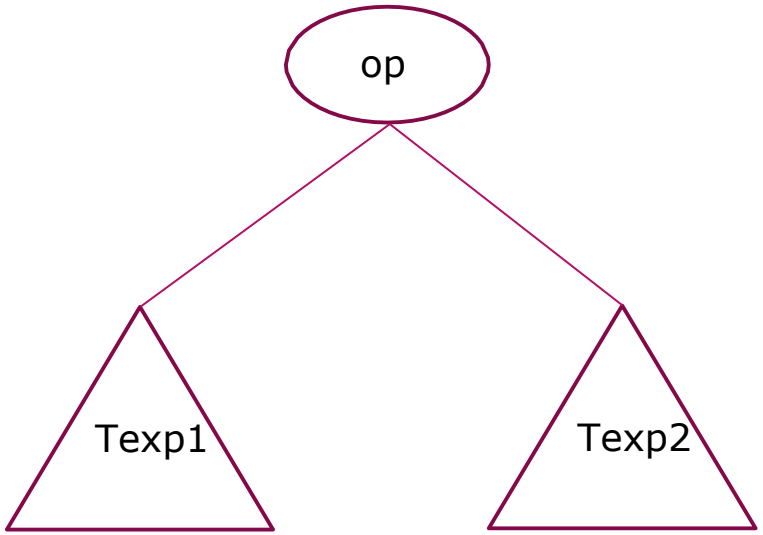
Order of evaluation : Evaluate the subtrees first






```
Evaluate  t1 = b*c
Evaluate  a + t1
```

Expression Tree - Evaluation




```
t1 = evaluate(Texp1)
t2 = evaluate(Texp2)
Result = t1 op t2
```

Expression Evaluation

- **Convert from Infix to Postfix**
  1. **Evaluate postfix**
  2. **Postfix to expression tree, and then evaluate expression tree**

Postfix Expressions

- **Easy evaluation of expressions**
- **Parentheses free**
- **Priority of operators is not relevant**
- **Evaluation by a single left to right scan**
  - **stacking operands**
  - **evaluating operators by popping out the required number of operands**
  - **finally placing result in the stack**

Evaluation of Postfix Expressions

- **Evaluate a b + c d + * (left to right scan, using a stack)**
  - **Push a**
  - **Push b**
  - **Upon getting +**
    - **pop out b, Pop out a**
    - **Evaluate t1 = a+b**
    - **Push t1**
  - **Push c**
  - ……

Evaluation of Postfix Expressions

- **Eval(Expression e)**
  - **evaluates the expression e in postfix form**
  - **e is terminated by #**
- **getNextToken(e)**
  - **returns the next token from e**
  - **Token can be either operand or operator**
- **Stack S to store tokens**
- **Upon termination, the value of e will be in S**

Evaluation of Postfix Expressions

```
Eval( Expression e)
    for(x = getNextToken(e); x!='#'; x= getNextToken(e))
        if (x is an operand)  PUSH(S, x)

        else   //x is an operator
            POP out the required number of operands for x from S
            Perform the operation x and PUSH the result to S
```

Postfix Expressions to Expression Tree

```
PostfixToExpressionTree( Expression e)
    for(x = getNextToken(e); x!='#'; x= getNextToken(e))
        if (x is an operand)
            node = createTreeNode(x, NIL, NIL)
            PUSH(S, node)
        else   //x is an operator
            rchild = POP(S); lchild = POP(S); // assuming binary operator
            node = createTreeNode(x, lchild, rchild)
            PUSH(S, node)
```

Infix Expressions

- **a + b * c / d**
- **x * 100 + y / n + (b * c – 6.5)**
- **p && q || r && s || !t**
- **(x <= y) && (a <= b)**

Infix Expressions – order of evaluation

- **a + b * c / d**
- **x * 100 + y / n + (b * c – 6.5)**
- **p && q || r && s || !t**
- **(x <= y) && (a <= b)**

Operators

- **Arithmetic**
  - **+ - * / % unary minus**
- **Logical**
  - **&& || !**
- **Relational**
  - **< <= > >= = =!=**

Expression Semantics

- **Semantics or meaning of an expression**
  - **a + b * c / d**
- **Order of evaluation of operators (subexpressions)**
  - **As per the language specification**

Operator Priority (sample)

1. **unary minus !**
2. *** / %**
3. **+ -**
4. **< <= >= >**
5. **== !=**
6. **&&**
7. **||**

Operators with same priority

- **Associativity rules**
  - **Left associative / Right associative**
  - **a + b + c + d**
- **Parenthesise to override**
  - **(a + b) + (c + d)**

Expression Evaluation

- **Convert from Infix to Postfix**
  1. **Evaluate postfix**
  2. **Postfix to expression tree, and then evaluate expression tree**

Infix to Postfix Conversion

a / b – c + d * e - a * c

Infix to Postfix Conversion

a / b – c + d * e - a * c

1. **Fully parenthesize**
2. **Move each operator to its corresponding right parenthesis**
3. **Delete all parenthesis**

Reference

1. **T H Cormen, C E Leiserson, R L Rivest, C Stein *Introduction to Algorithms,* 3rd ed., PHI, 2010**
2. **E. Horowitz, E. Sahni, D. Mehta *Fundamentals of Data Structures in C++*, 2nd ed., Universities Press, 2007**