CS 2002D PROGRAM DESIGN

Tree Traversal
Expression Tree, Evaluation

SALEENA N
CSED, NIT CALICUT
06.11.2020

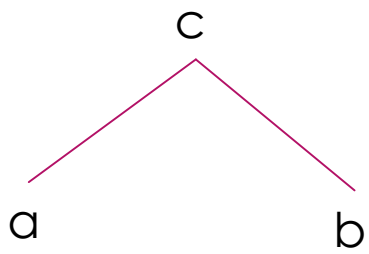
Overview

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

Tree Traversals

- Tree Traversal (Tree Walk in CLRS)
- Preorder
  - Visit Root, Left subtree in preorder, Right subtree in preorder
- Inorder
  - Left subtree in inorder, Visit Root, Right subtree in inorder
- Postorder
  - Left subtree in postorder, Right subtree in postorder, visit Root

Tree Traversals

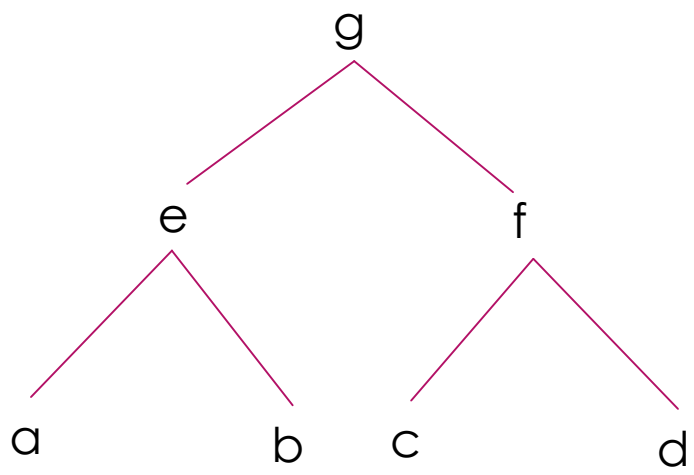


Preorder: c a b

Inorder: a c b

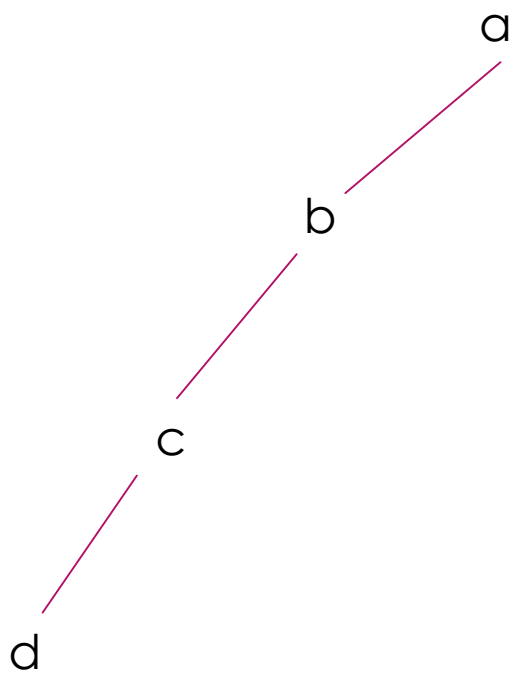
Postorder: a b c

Tree Traversals



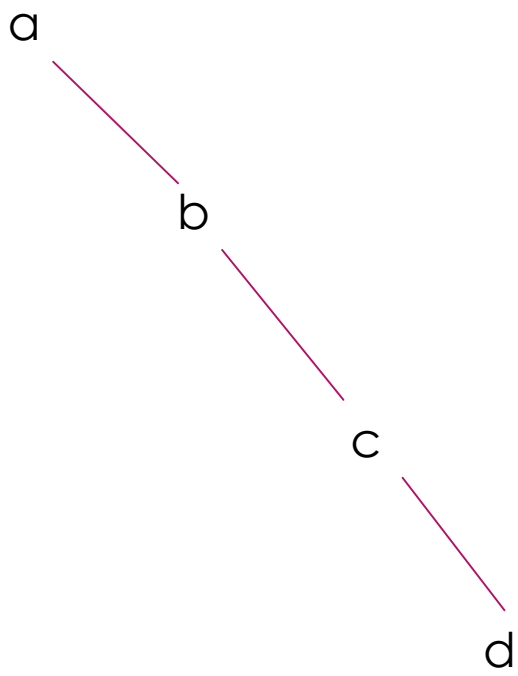
List the nodes in preorder, inorder, postorder

Tree Traversals



List the nodes in preorder, inorder, postorder

Tree Traversals



List the nodes in preorder, inorder, postorder

Skewed Tree (Right skewed)

Inorder Tree Walk - Algorithm

```
INORDER-TREE-WALK (x)
    if x ≠ NIL
        INORDER-TREE-WALK (x.left)
        print x.data
        INORDER-TREE-WALK (x.right)
```

Inorder Tree Walk - Algorithm

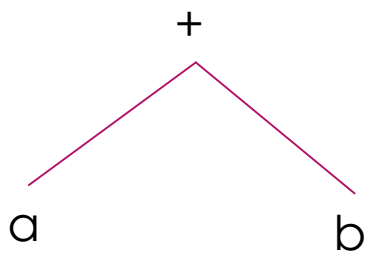
```
INORDER-TREE-WALK (x)
    if x ≠ NIL
        INORDER-TREE-WALK (x.left)
        print x.data
        INORDER-TREE-WALK (x.right)
```

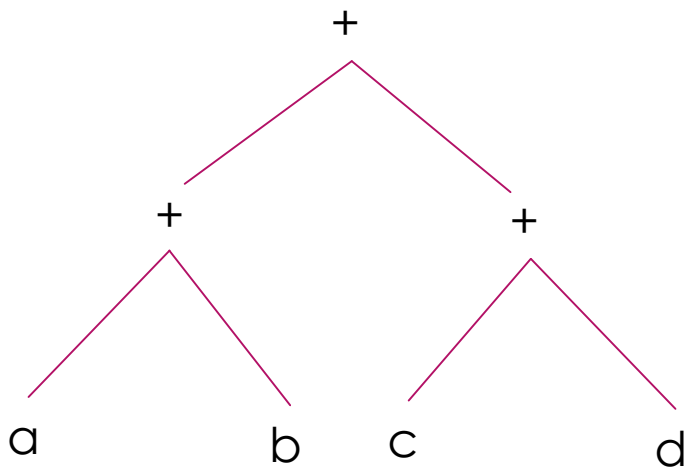
- x is a node in the tree
- To traverse the entire tree T, invoke as INORDER-TREE-WALK (T.root)

Tree Walk - Algorithms

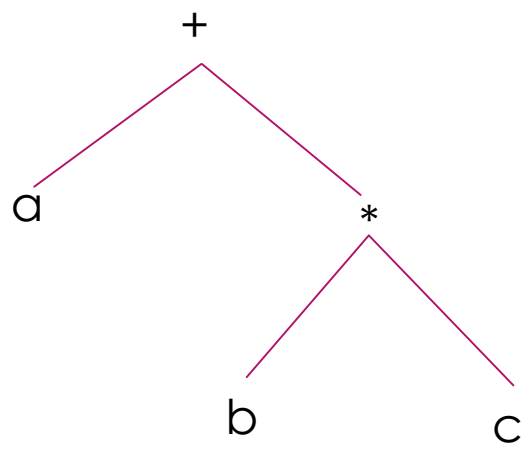
- **Write recursive algorithms for**
  - `PREORDER-TREE-WALK()`
  - `POSTORDER-TREE-WALK()`

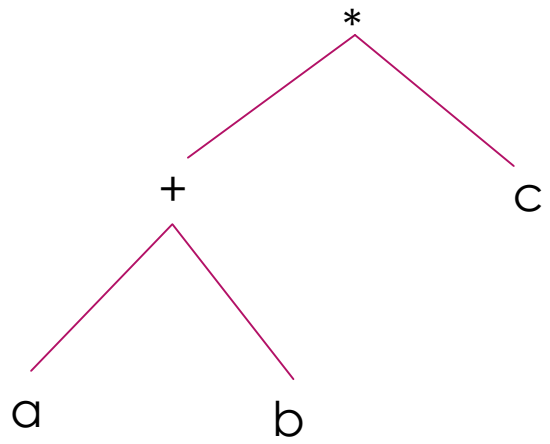
Expression Tree - Traversals



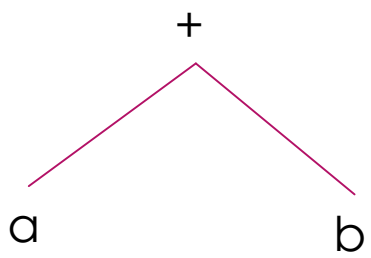


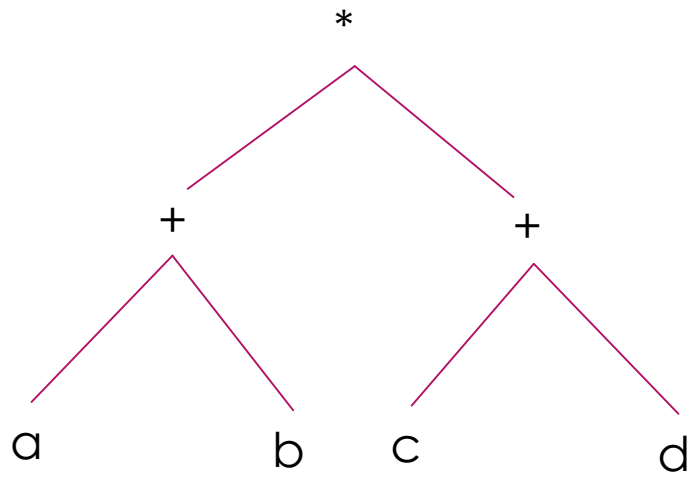
Expression Tree - Traversal





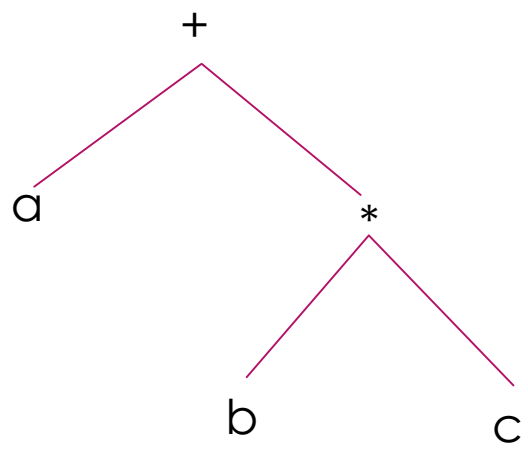
Expression Tree - Evaluation

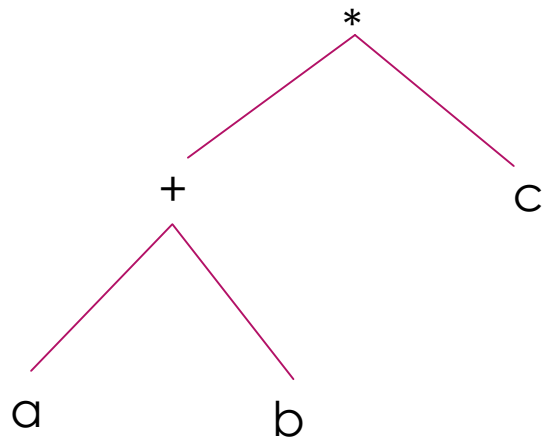




Order of evaluation : Evaluate the subtrees first

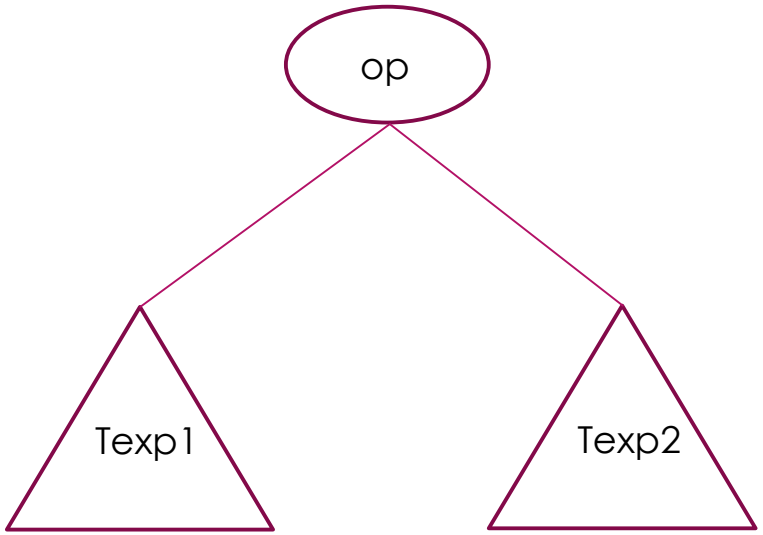
Expression Tree





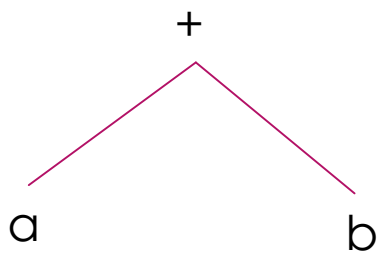

```
Evaluate   t1 = b*c
Evaluate   a + t1
```

Expression Tree - Evaluation




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

Expression Tree – Traversals

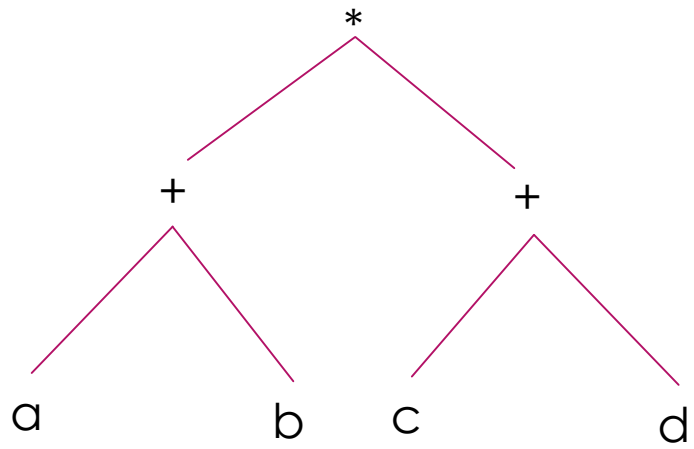


+ab **Preoder**

a+b **inorder**

ab+ **postorder**

Expressions- Infix, Prefix, Postfix



***+ab+cd prefix form**

a+b*c+d infix form

ab+cd+* postfix form

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 a, Pop out b**
    - **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)
```

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**