Conversion of Infix to Postfix Expression

Overview

- Conversion of Infix to Postfix Expression

- Tree Traversals - Exercise

Expression Evaluation

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

Infix to Postfix Conversion

<u>Example:</u>

a / b – c + d * e - a * c

((((a / b) - c) + (d * e)) - (a * c))

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**

Infix to Postfix Conversion - Algorithm

- Using stack (assuming only + and *)
  - Scan expression left to right
  - Operand - Print
  - Operator – Pop and print/ Push ?

a + b * c	a b c * +	Incoming opr priority > stack top opr priority
a * b + c	a b * c +	Stack top op priority > Incoming opr priority

Infix to Postfix Conversion

a + b * c

print a, push +, print b, push * (higher priority than +), print c, pop,

print *, pop,print +

a * b + c

print a, push *, print b, pop * (higher priority than +), push +, print c, pop, print +

Infix to Postfix Conversion - Algorithm

- **Using stack**
  - **Scan expression left to right**
  - **Operand - Print**
  - **Operator – Pop out until an operator with a lower priority is on top of stack**

Infix to Postfix Conversion

- **Parenthesized expressions a * (b + c)**

- **For) pop out all operators till the last (**
- **Do not print (or)**
- **Parenthesis can be treated as operators**

- **print a, push *, push (, print b, push +, print c, pop, print +, pop, pop, print ***

Infix to Postfix Conversion - Algorithm

- **Scan expression left to right**
  - **Operand - Print**
  - **Operator – Pop out until an operator with a lower priority is on top of stack**
  - **(- Push**
  - **)- Pop out until the last (**
  - **Priority values to be assigned to (and)**

Infix to Postfix Conversion - Algorithm

- **Priority values for operators**
  - **In stack priority (*isp*)**
  - **Incoming priority (*icp*)**
  - **Appropriately define *isp* and *icp* for each operator**
  - **Compare *icp* of incoming operator with *isp* of top operator**
  - **Include other operators (unary -, exponentiation ^)**

Tree Traversal Exercise

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

Construct Tree from Inorder and Preorder traversals

Inorder sequence: D B E A F C
Preorder sequence: A B D E C F

```
                                              A
                                           /        \
              A                           /          \
          /       \                      B            C
                                        / \          /
        /           \                  /   \        /
      D B E      F C                  D     E      F
```

Construct Tree from Inorder and Postorder traversals

Inorder sequence: 4, 8, 2, 5, 1, 6, 3, 7
Postorder sequence: 8, 4, 5, 2, 6, 7, 3, 1

```
             1
       /           \
[4, 8, 2, 5]   [6, 3, 7]
```

```
         1
       /   \
     2       3
   /   \    /  \
  4     5  6    7
   \
    8
```

Tree Traversal Exercise

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

References

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