CS 2002D PROGRAM DESIGN

Infix to Postfix

SALEENA N
CSED, NIT CALICUT

09.11.2020

Overview

- Infix Expression
  - Priority of operators
- Postfix form
- Conversion from Infix to Postfix

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**

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**

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**

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 * +
a * b + c	a b * c +

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, 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**

Tree Traversal Exercise

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

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**