# Expression Evaluation

# (. Stallings, W. Computer Organization & Architecture

Mathematical formulas are usually expressed in what is known as **infix** notation. In this form, a binary operation appears between the operands (e.g. a + b). For complex expressions, parentheses are used to determine the order of evaluation of expressions. For example, a + (b x c) will yield a different result than (a + b) x c. To minimize the use of parenthesis, operations have an implied precedence. Generally, multiplication takes precedence over addition, so that a + b x c is equivalent to a + (b x c).

An alternative technique is known as **reverse Polish**, or **postfix**, notation. In this notation, the operator follows its two operands. For example,

a + b becomes a b +

a + (b x c) becomes a b c x +

(a + b) x c becomes a b + c x

Note that, regardless of the complexity of an expression, no parentheses are required when using reverse polish.

**The advantage of postfix notation is that an expression in this form is easily evaluated using a stack:**

*An expression in postfix notation is scanned from left to right. For each element of the expression, the following rules are applied:*

1. *If the element is a variable or constant, push it onto the stack.*
2. *If the element is an operator, pop the top two items of the stack, perform the operation, and push the result.*

*After the entire expression has been scanned, the result is on the top of the stack.*

The simplicity of this algorithm makes it a convenient one for evaluating expressions. Accordingly, many compilers will take an expression in a high-level language, convert it to postfix notation, and then generate the machine instructions from that notation.

**For example, the postfix version of (a – b)/(c + d x e) is a b – c d e x + /.**

The sequence of operations for evaluating f = (a – b)/(c + d x e) using

a b – c d e x + / are:

1. Push a
2. Push b
3. Subtract
4. Push c
5. Push d
6. Push e
7. Multiply
8. Add
9. Divide
10. Pop f

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  | d |
|  |  | b |  |  |  | c |  | c |
| a |  | a |  | a-b |  | a-b |  | a-b |
| 1 |  | 2 |  | 3 |  | 4 |  | 5 |
|  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |
| e |  |  |  |  |  |  |  |  |
| d |  | d x e |  |  |  |  |  |  |
| c |  | c |  | d x e + c |  |  |  |  |
| a-b |  | a-b |  | a-b |  | (a-b)/d x e + c |  |  |
| 6 |  | 7 |  | 8 |  | 9 |  | 10 |
|  |  |  |  |  |  |  |  |  |

# Conversion of an Expression from Infix to Postfix Notation

The process of converting an infix expression to a postfix expression is itself most easily accomplished using a stack. The following algorithm is due to Dijkstra. The infix expression is scanned from left to right, and the postfix expression is developed and output during the scan. The steps are as follows:

1. Examine the next element in the input.
2. If it is an operand, output it.
3. If it is an opening parenthesis, push it onto the stack.
4. If it is an operator, then
   * If the top of the stack is on opening parenthesis, then push the operator.
   * If it has higher priority than the top of the stack (multiply and divide have higher priority than add and subtract), then push the operator.
   * Else, pop operation from stack to output, and repeat step 4.
5. If it is a closing parenthesis, pop operators to the output until an opening parenthesis is encountered. Pop and discard the opening parenthesis.
6. If there is more input, go to step 1.
7. If there is no more input, pop the remaining operators.

The following table illustrates the use of this algorithm.

|  |  |  |
| --- | --- | --- |
| **Input** | **Output** | **Stack** |
| A + B x C + (D + E) x F | empty | empty |
| + B x C + (D + E) x F | A | empty |
| B x C + (D + E) x F | A | + |
| x C + (D + E) x F | A B | + |
| C + (D + E) x F | A B | + x |
| + (D + E) x F | A B C | + x |
| (D + E) x F | A B C x + | + |
| D + E) x F | A B C x + | + ( |
| + E) x F | A B C x + D | + ( |
| E) x F | A B C x + D | + ( + |
| ) x F | A B C x + D E | + ( + |
| x F | A B C x + D E + | + |
| F | A B C x + D E + | + x |
| empty | A B C x + D E + F | + x |
| empty | A B C x + D E + F x + | empty |