Syntax and Semantics of Expressions

Loosely speaking:

- ***Syntax*** means “form”.
- ***Semantics*** means “meaning”.

In the context of programming languages:

- The Java expressions `a - 3 - b` and `u - 7 - v` have the *same syntax*, but *different semantics*.
- The Java expressions `a - 3 - b` and `((a - 3) - b)` have the *same semantics*, but *different syntax*.
- The Lisp expression `(- (- 2 3) 4)` and the Java expression `2 - 3 - 4` have the *same semantics* but have *different syntax*.

Operators, Arities, and Operands

An ***operator of arity*** *k*, also called a ***k-ary operator***, is a symbol that represents a function of *k* arguments.

- ***Binary*** means 2-ary.
- ***Unary*** means 1-ary.
- ***Ternary*** means 3-ary.

When a *k*-ary operator appears in an expression, each of the *k* subexpressions whose values are the *k* arguments of the function it represents is called an ***operand*** of the operator.

Examples

In a Java or C++ expression x – y + z

- + is a binary operator whose operands are x – y and z.
- – is a binary operator whose operands are x and y.

In the Lisp expression (+ (– x y) 3 (* z z))

- + is a ternary operator whose operands are (– x y), 3, and (* z z).

Overview of Some Expression Notations

We will say that two expressions are ***equivalent*** if they have the same semantics.

Here is a Java expression: `f(g(h(1,2), f(3,4)), 5)`
We will now write equivalent expressions in:

- **Infix notation, not making use of precedence classes**
- **Infix notation, making use of precedence classes**
- **Lisp notation** (also called **Cambridge Polish Notation**)
- **Prefix notation**
- **Postfix notation** (also called **Reverse Polish Notation**)

Infix Notation

In ***infix*** notation, we write binary operators ***between*** their operands--e.g., `f(3,4)` would be written `3 f 4`.

So the above Java expression is equivalent to the following infix expression: `((1 h 2) g (3 f 4)) f 5`

The designer of an infix notation will usually give ***precedence and associativity rules*** for the operators, to reduce the need for parentheses.

Suppose the notation designer specifies that:

- f and g belong to the ***same*** precedence class, and that precedence class is ***left-associative***.
- h belongs to a ***higher*** precedence class than f and g.

Now the above infix expression `((1 h 2) g (3 f 4)) f 5`
is equivalent to: `1 h 2 g (3 f 4) f 5`

An Analogous Example

In ordinary infix arithmetic notation:

- + and - belong to the same precedence class, and that precedence class is left-associative.
- * belongs to a higher precedence class than + and –.

Thus the infix expression `((1 * 2) – (3 + 4)) + 5`
is equivalent to: `1 * 2 – (3 + 4) + 5`

Lisp Notation and Prefix Notation

The above Java expression f(g(h(1,2), f(3,4)), 5)
is equivalent to the
following Lisp expression: (f (g (h 1 2) (f 3 4)) 5)

Prefix notation is *Lisp notation* ***without*** *parentheses*.
Thus the above Java expression is equivalent to the following prefix expression: f g h 1 2 f 3 4 5

Prefix notation is not ambiguous ***if we know the arity of every operator***: Then we can "put parentheses back" to produce an equivalent Lisp expression.

If we do not know the arities of operators, then prefix notation can be ambiguous.

For example, if + and – are binary then
+ 1 – 2 3 is equivalent to (+ 1 (– 2 3)),
but if + is ternary and – is unary then
+ 1 – 2 3 is equivalent to (+ 1 (– 2) 3).

"RpnLisp" Notation and Postfix Notation

In Lisp, a function call is written as a list whose ***first*** element is the function name.

Now consider a notation we'll call ***rpnLisp*** that's the same as Lisp except in that a function call is written as a list whose ***last*** element is the function name.

Thus the Lisp expression `(+ (- 1 2) (* 3 4 5) 6)`
is equivalent to the
following rpnLisp expression: `((1 2 -) (3 4 5 *) 6 +)`

Just as ***prefix*** notation is "Lisp notation without parentheses", ***postfix*** *notation is "rpnLisp notation* ***without*** *parentheses"*.

So the above expressions are equivalent to this ***postfix*** expression: `1 2 - 3 4 5 * 6 +`

In ***rpnLisp***, ***rpn*** stands for "**r**everse **p**olish **n**otation".

The above Java expression f(g(h(1,2), f(3,4)), 5)
is equivalent to the
Lisp expression (f (g (h 1 2) (f 3 4)) 5)
and this rpnLisp expression: ((1 2 h) (3 4 f) g) 5 f)

Thus the expression is equivalent
to this ***postfix*** expression: 1 2 h 3 4 f g 5 f

As is the case with prefix notation, postfix notation isn't ambiguous ***if we know the arity of each operator***:

If we do not know the arities of operators, then postfix notation can be ambiguous.

For example:

- If + and – are binary, the postfix expression 1 2 3 + –
  is equivalent to the rpnLisp expression (1 (2 3 +) –)
  and equivalent to the Lisp expression (– 1 (+ 2 3)) => –4.
- If + is ternary and – is unary, the expression 1 2 3 + –
  is equivalent to the rpnLisp expression ((1 2 3 +) –)
  and equivalent to the Lisp expression (– (+ 1 2 3)) => –6.