Notes on Jlang

wycd.net

January, 2014

Abstract

This is a collection of non-comprehensive notes by a programmer learning the J programming language who was previously familiar with C and C++. These notes are meant to provide clarification for language features with examples. If desired, please email any errors or ambiguities to `mail@wycd.net`.

`j64-701`

Contents

1 Syntax

1.1 Arithmetic

Operation	Expression	J Expression
Addition	$(x + y)$	`x + y`
Multiplication	$(x * y)$	`x * y`
Subtraction	$(x - y)$	`x - y`
Division	$(\frac{x}{y})$	`x % y`
Modulo	$(x \ mod \ y)$	`y | x`
Reciprocal	(x^{-1})	`% x`
Negation	$(-x)$	`- x`
Power	(x^y)	`x ^ y`
Squared	(x^2)	`*: x`
Square Root	$(\sqrt{x})$	`%: x`
Double	$(2x)$	`+: x`
Halve	$(\frac{1}{2}x)$	`-: x`
Signum	$(sgn(x))$	`* x`
Exponential	(e^x)	`^ x`
Natural Log	$(ln(x))$	`^. x`
Log	$(log_y(x))$	`y ^. x`
Pi Times	(πx)	`o. x`
Factorial	$(x!)$	`! x`
Floor	$(\lfloor x \rfloor)$	`<. x`
Ceiling	$(\lceil x \rceil)$	`>. x`
Sine	$(sin(x))$	`1 o. x`
Cosine	$(cos(x))$	`2 o. x`
Tangent	$(tan(x))$	`3 o. x`
Complex (Dyadic)	$(Complex(\{x, y\}) \implies x + yi)$	`x j. y`
Complex (Monadic)	$(Complex(x) \implies xi)$	`j. x`
Magnitude, Angle	$(\{\|x\|, \angle x\})$	`*. x`
Real, Imaginary	$(\{Re\{x\}, Im\{x\}\})$	`+. x`
Conjugate	(x^*)	`+ x`
Magnitude	$(\|x\|)$	`| x`

Negative numbers are represented with a leading underscore: e.g. `_42` This allows for discrimination between the negative sign of a literal number and the operator `-`.

Complex numbers are expressed in the form `xjy`, where x is the real part and y is the imaginary in $x + yi$. The j notation was most likely chosen as not to conflict with the `i.` integer operator, but it is convenient that this is the same notation used in electrical engineering. $3 + 5i$ may be expressed as `3j5`.

1.2 Evaluation Order

Unless parentheses are used, J always evaluates from **right to left**. The following are the same:

```
2 * 8 + 4 % 3 - 2
2 * (8 + (4 % (3 - 2)))
```

1.3 Monads and Dyads

The same symbol may have different meanings depending on the context. For example, the symbol `-` may be used for subtraction if placed between two numbers (`4 - 2`), but it may also be used for negation if there is only one number (`-4`). A function with a single argument on its right is called a monadic function (monad) while a function taking one argument on each side is called a dyadic function (dyad). The monad `- x` has different functionality than the dyad `x - y`.

1.4 Types

Thre are four primitive types in J: **number, character, box, and symbol**. There are six subtypes of number two subtypes of character plus there are sparse versions of some of the types. Boxes are general-purpose containers while symbols consist of the operators in the J language.

The full list can be found at `http://www.jsoftware.com/docs/help701/dictionary/dx003.htm`

1.5 Parts of Speech

Functions in J are known as **verbs** because they operate on expressions which are known as **nouns**. For example, in the expression `2 + 3`, the verb `+` dyadically operates on the nouns `2` and `3` to yield the noun `5`. In `*: 8`, the square verb `*:` operates on the single noun `8` to yield the noun texttt64.

`Adverbs` are operators that act upon and modify the behavior of functions just as adverbs in English affect verbs. For example, the Insert adverb `/` causes a verb to be inserted between all elements in a list. These are the same:

```
+/ 1 2 3 4 5
1 + 2 + 3 + 4 + 5
```

The new verb `+/` is formed by the verb and adverb, and might be labeled `sum =: +/` for convenience. Adverbs always

1.6 Notation

Throughout the J literature,

- Noun arguments to a verb are `x` and `y` (e.g. `x + y`).
- Verb arguments to a conjunction are `u` and `v` (e.g. `u @ v`).
- Verbs can also be `f`, `g`, and `h` (e.g. `x (f g h) y`).
- Noun arguments to a conjunction are `m` and `n` (e.g. `m " n`).

1.7 Arrays and Operations

The name `array` in J refers to a data object with a number of dimensions. Scalars (0-dimensional), lists (1-dimensional) and tables (2-dimensional) are all considered arrays. Scalars can be expressed by single terms such as `5`, `3j5`, or `1.25`. Lists can be constructed from adjacent scalars such as `1 2 3 4 5`, `1.1 1.25 3j3 1 3 9 7`, or `'abcde'`.

The Shape verb `$` can be used to manipulate