04_Operators

September 22, 2019

1 4. Operators

- Operators are_
  - also called verbs in q
  - are built-in functions
  - can beused with in-fix or prefix notation
  - can have purely symbolic names
  - atomic functions act recursively on data structures

1.0.1 4.0.2 Primitives, verbs and functional notation

- Primitive operators: basic arithmetic, relation and comparison operators
  - they can have names: simple ASCII symbol, compound operator, names
- Extension of atomic functions to items in a list

1.1 4.1. Operator precedence

- q has no rule for operator precedence
- q expressions are evaluated from left of right (right to left)
  - -> no ambibuity from the compiler's perspective
- having no operator precedence renders analyzing all components of an expression before evaluation moot

1.2 4.2. Match ~

- ~ is the identity evaluator operator
- if two q entities are identical (x ~ y = 1b), then they have the same:
  - shape
  - type
  - value, but
  - they may occupy separate storage locations
  - -> clones are considered identical in q

1.3 4.3. Equality and relational operators

- relational operators
  - are atomic functions
  - the types of operands do not have to match

[ ]:
```
1 2 3<10 20 30
```

1.3.1 4.3.1. Equality = and disequality <>

- The equality operator is atomic in both of its operands (the match operator is not): it tests its operands atom-wise
- Numeric, temporal and char types are all compatible for equality comparison
  - When comparing a numeric type with a character type, their underlying bit pattern is compared
- Symbols only compatible with symbols when it comes to comparison
- Disequality can be tested in two ways:
  - a<>b
  - not a=b
- When comparing floating point numbers, multiplicative tolerance vor non-zero values is used: 1=(1%3)+(1%3)+(1%3)

[ ]:
```
42="*"
`a<=`a
```

[ ]:
```
r:1%3
2=r+r+r+r+r+r
```

- not is applicable not only to boolean values but to numberic temporal and character types
- not generalizes the reversal of true and false to any entity having an underlying numeric value

[ ]:
```
not 0.00001
not 0xff
"i"$0xff
not " "
not "\000"
```

1.3.2 4.3.3. Order: <;<=;>;>=

- Same rules apply as in case of equality comparison
- Symbol comparison is based on lexicographical order

[ ]:
```
 / Entire ASCII collation sequence
16 16#"c"$til 256
```

1.4 4.4 Basic arithmetic: + - * %

- Arithmetic operators are defined for all numeric and temporal types
- the result of division is always a float
- when a floating point type occurs in an expression, the result is a float
- Type promotion rules:
  - bynary types are promoted to int
  - the result type of an operation is the narrowest type that will accomodate both operands
- the minus sign can only be used with numeric values, but not on variables. use neg instead
- arithmetic operators and their promotion are performed atom-wise on lists

1.5 4.5. Minimum, maximum, | and &

- These atomic dyadic operators follow the same type promotion and compatibility rules as arithmetic operators.
- They are defined for all values with underlying numeric values but are not defined for symbols and GUIDs.
- reduces to OR (notation: or) and & reduces to AND (notation: and) for binary data types as operands

1.6 4.6. Amend

- An overload of : which assigns in place
- +: is the same as += in C or Java (increment)
- you can amend lists with indexing: list[x]+:1
- idiom: appent to list in place: list,:item_n, item_n+1

[ ]:
```
aaa:5
aaa
aaa+:1
aaa
LL:1 2 3 4
LL,:5 6
LL
```

1.7 4.7. Exponential primitives: sqrt, exp, xexp, log, xlog

- exp num: base e raised to the power of num
- num1 xexp num2: num1 raised to the power of num2
- log x: returns the natural logarithm of x
- x xexp y: returns the x base logarithm of y

[ ]:
```
-2 xexp 0.5
```

1.8 4.8. More numeric primitives

- x div y: integer division
- x mod y: modulo, remainder division
- signum x: returns the sign of any numeric and temporal type input: 1i - positive, -1i - negative, 0i - zero
- reciprocal x : 1.0 % x (returns float type)
- floor x: largest long that is less than or equal to x (domain: numeric types)
- ceiling x: smallest long that is greater than or equal to x
  - floor and ceiling do not apply to short type
- abs x: absolute value

1.9 4.9. Operations on temporal values

- there is no concept of time zones in q

1.9.1 4.9.1. Temporal comparison

- To compare temporal values of different types, q converts to the most granular type and then does a straight comparison of the underlying values.

1.9.2 4.9.2. Temporal arithmetic

1.10 4.10. Operations on infinties and null

- An **infinity value** equals or matches only itself.
- All **nulls** are equal (they represent missing data)
- Different type **nulls** do not match (type matters).
- In contrast to some languages, such as C, separate instances of NaN are equal.
- The not operator returns 0b for all infinities and nulls since they all fail the test of equality with 0.
- The neg operator reverses the sign of infinities but does nothing to nulls since sign is meaningless for missing data.
- for any numeric type
  - null < negative infinity < normal value < positive infinity
  - Nulls of different type, while equal, are not otherwise comparable – i.e., any relational comparison results in 0b.
  - Infinities of different type are ordered by their width. For positive infinities:
    * short < int < long < real < float
  - For negative infinities
    * -float < -real < -long < -int < -short

[ ]:
```
0Wi&0W
```

1.11 4.11. Alias ::

- the :: alias is a variable that is an expression itself
- the alias can be used to defer the evaluation of an expression
- 0N! is the same as show? use it to inspect the inner workings of an in-flight evaluation

1.11.1 4.11.2. Alias vs function

- Two key differentces:
  - To evaluate an expression wrapped in a function you explicitly provide the arguments and apply the function all in one step. With an alias you set the variables at any point in the program and **the expression is evaluated when, and only when, the alias variable is referenced**.
  - **The function does not memoize its result**, so it recalculates on every application, even if the arguments do not change.

1.11.2 4.11.3. Dependencies

- A list of all dependencies of an alias is maintained in the system dictionary (.z.b)

1.11.3 4.11.4. Views

- Aliases are commonly used to provide a database view by specifying a query as the expression

[ ]:
```
t:([]c1:`a`b`c`a;c2:20 15 10 20;c3:99.5 99.45 99.42 99.4)
v::select sym:c1,px:c3 from t where c1=`a
v
update c3:42.0 from `t where c1=`a
v
```

[31]:
```
.z.b / dependencies are maintained in the system dictionary
```

[31]:
```
t| v
```