A "merge" operator for Dyalog

Motivation

Tool-of-thought:

Array Languages

let us think about aggregates rather than items.

Functional Languages

let us think about expressions rather than state.

Currently, Dyalog lacks an expression for the "triadic" operation:

THAT array but with THESE items at THOSE positions.

Instead, it employs a 3-step procedure:

```
TMP←...  ⍝ name the array to form a variable
TMP[X]←Y ⍝ mutate the variable
...TMP   ⍝ dereference the name to extract the
         ⍝ new value
```

WIBNI we could embed some <<mechanism>> within an expression:

... <<Y X>> ...

(both J and K have such constructs)

and, while we're at it, replace "modified indexed assignment":

```
TMP←... ⋄ TMP[X] F← Y ⋄ ...TMP
```

with:

```
... <<Y F X>> ...
```

See also "mesh" and "mask" (KEI 1962)

```
      x ≡ p (b mesh) q ↔ p≡(~b)≠x ⋄ q≡b≠x
      x ≡ p (b mask) q ↔ (b≠x)≡b≠q ⋄ ((~b)≠x)≡(~b)≠q

     'sek' (0 1 0 1 0 mesh) 'ta'
steak

     'abcde' (0 1 0 1 0 mask) 'ABCDE'
aBcDe
```

(see also select function in dfns.dws: Google[dyalog select])

Nomenclature

Nouns make better names for functions than do transitive verbs: sqrt, succ,

merge(n)

amend(vt) - J, K

mask(n) - KEI

fuse[ion] - Olympus bar FP session, Monday night.

Design Considerations

- The most frequent cases should have the simplest expression
- Don't overload one operator with too many cases but avoid using separate glyphs for strongly related operations
- Minimise the requirement for (especially adjacent) parentheses

John'S 2p

- "Fuse" is a dyadic operator, as opposed to, say, special new syntax.
- The selector is on the right, to reduce parentheses.
- The selector is a Boolean value or function, as opposed to an index.

- The selection operates on major cells (along the leading axis)

Glyph

How about ⍸ ?

- Quiet
- Easy to type
- No confusion with branching / suspensionclearing

Examples

spec: Deal of an ⍵-deck (?⍨⍵), with alternate items zapped to 0.

eg: ⎕io←1 ⋄ ⍵←5 : 2 3 4 1 5 : 0 3 0 1 0

```
f ← {2|⍳⍵}
```

bool-returning function: "alternate"

```
b ← f ⍳⍵
```

pre-computed Boolean selection vector

```
T←?⍨⍵ ⋄ (b/T)←0 ⋄ T
```

selective assignment using vector b ⍸ cf: |

```
0 ⍸b ?⍨5
```

fuse operator using vector b

70%

```
0 ⍸b ⊢5?5
```

⊢ to prevent binding of b with 5

```
T←?⍨⍵ ⋄ ((f T)/T)←0 ⋄ T  ⍝ selection function
```

cf:

```
0 ⍸f ?⍨5
```

selection function

```
0 ⍸f 5?5
```

no need for ⊢

A static analysis of a customer's application showed that 70% of selective assignments were simple Boolean selection, as above.

Deal of an ω-deck, with alternate items incremented.

```
T←5?5 ⋄ (b/T)+←1 ⋄ T
```

modified, selective assignment"

```
T←5?5 ⋄ ((f T)/T)+←1 ⋄ T
```

cf:

```
1 +→b ⊢5?5
1 +→f  5?5
```

Model

```
{A←⍵
   2=⎕nc'⍵⍵': A⊣(⍵⍵ ≠A)⍺⍺⍨←⍺    ⍝ ⍵⍵ is bool vec
   A⊣((⍵⍵ ⍵)≠A)⍺⍺⍨←⍺           ⍝ ⍵⍵ is bool fn
}
```

What next?

- Hoping for some discussion in the Dyalog forum or email to john@dyalog.com
- JS to explore opportunities for → in "real" application code.