Dyalog APL

P. Kantek

27 de julho de 2020

Sumário

 ©75-20, Pedro Kantek

Capítulo 1

APL

O APL é uma linguagem à frente do seu tempo por excelência. Hoje, graças às interfaces orientadas a gráfico, rompeu-se o grande represor antigo que era a dificuldade de representar os caracteres APL. (Ajuda também do UNICODE). Prova de estar à frente de seu tempo: O APL é de 1964, O Windows de 1988 e o Unicode de 1995. Hoje, em 2020, temos as seguintes alternativas:

APL2 da IBM. Legítimo sucessor do APL que começou quando o Ken Inverson juntou-se a uma turma boa (Adin Falkoff entre outros) na IBM para gerar a primeira versão de APL operacional. Teve como sucessor o APL2, que ainda hoje é comercializado pela IBM. Grande software: estável e com enorme base instalada. A versão que uso é para 32 bits, obtida em 2004 através de acordo universitário. A versão 64 bits existe, mas precisa ser comprada.

NARS2000 Uma tentativa de escrever um APL freeware. Disponível em `http://www.nars2000.org/`. Parece bem completa, tenho instalada no meu computador, mas nunca fui muito longe com ela.

APLX Uma iniciativa da empresa inglesa microapl (`http://microapl.com/APL`). Era um excelente APL, cheguei a usá-lo por alguns anos. Comprei uma licença pagando 99 libras em 2009. Anos mais tarde, a empresa decidiu descontinuar o produto. A notícia boa é que ao fazer isso disponibilizou ao mundo a versão que até então era paga. Já adaptada para vigorar em 32 ou 64 bits. Seu sítio migrou para o sítio do Dyalog APL. Gera excelentes produtos compilados. Para usar a versão liberada: *Enter a Licence number of 70702153 and a Serial number of W7CBS4CX and click OK.*

Dyalog É esta que vai ser objeto de estudo aqui. Não é freeware, mas inteligentemente pode ser baixada e usada enquanto prova de conceito. Não pode gerar software comercial. Quando isto ocorrer, as licenças precisam ser compradas. Veja mais em `https://www.dyalog.com/`.

outras Tem algumas outras versões/opções. Lembro de cabeça: `VSAPL` , `APLPLUS` e `Sharp APL` entre outras. Não vão ser tratadas aqui, tem cheiro de mofo. Por exemplo a APLPLUS, que era muito boa, mas só conseguia acessar 640KB de memória, já que operava em uma janela DOS. Para o APL isso é um traque apenas.

Então, daqui para a frente, está-se tratando apenas do Dyalog APL, também tratado aqui como DAPL2. O 2 no nome indica ser versão compatível com arrays aninhados, a grande mudança que a IBM fez ao passar do APL → APL2.

Capítulo 2

Dyalog APL

2.1 Qualidades e novidades

Eis uma lista incompleta e simples das novidades que ela apresenta:

- 100 % de compatibilidade reversa. Isso é importante porque permite reinflar software feito em APL em anos passados. A compatibilidade só se perde quando o software antigo usava características nativas do ambiente antigo. Por exemplo, ao usar variáveis e funções de sistema. Sendo mais específico: ao converter software feito em APL2 e que – lastimavelmente – usava a função `⎕FMT` daquele ambiente.
- Novos operadores. Lembrando que o APL tinha 4 ou 5 operadores (produtos interno e externo, redução e scan além de eixo). Agora são muitos, e precisam ser estudados. Tem um capítulo inteiro, dedicado a eles, a seguir.
- Muitas variáveis e sobretudo funções de sistema novas. Algumas francamente inúteis, mas deve haver o que se salva.
- Suporte pleno a Unicode. Se o Unicode já é importante no software em geral, aqui no APL ele é fundamental sob pena de não haver comunicação com nada e ninguém.
- Suporte à programação estruturada. Este realmente era o calcanhar de Aquiles do APL e igualmente do APL2. Embora o APL já menosvalorizasse o conceito de laço pela sua natural vocação a substituí-los, ainda assim, é muito difícil programar sem estas ferramentas.
- Comandos de usuário. Uma sacada genial, que permite escrever código APL para rodar standalone (isto é, sem estar fisicamente no ws onde é executado). Outra sacada genial é identificar estes comandos por um `]`, numa óbvia analogia com os comandos de sistema. Aparece a seguir no capítulo SALT.
- DFN. Uma iniciativa bem interessante. Equivalem às funções *lambda* de LISP. Permitem definir funções de maneira muito simples e rápida para uso instantâneo.

2.2 Problemas

Fica meio chato falar isso, mas vamos lá:

- Documentação. Embora tenha uma quantidade enorme de manuais, eles ficam a desejar bastante. Há falta de boas referências e os manuais são um tanto crípticos, o que em se tratando de APL é um defeito e tanto.

Capítulo 3

Primitivas

A idéia aqui é destacar as principais mudanças em relação ao APL *old fashioned*, já que se o objetivo fosse descrever completamente o APL este texto deveria ter centenas de páginas. Menos, menos.

3.1 Axis Operator - Operador Eixo

Usado em inúmeras situações, sendo algumas novas. Acompanhe

```
      1 0 1/[1]3 2⍴⍳6
1 2
5 6
      1 2 3+[2]2 3⍴10 20 30
11 22 33
11 22 33
```

Um número fracionário indica um novo eixo.

```
      'NOMES',[0.5]'-'
NOMES
-----
```

Obviamente este operador depende do valor de ⎕IO.

3.2 Abort - Cancelamento

A flecha à direita sozinha aborta uma interrupção e limpa o state indicator.

3.3 Adição

Operação de adição convencional. A possível novidade fica por conta dos complexos

```
      1J3 20+3J3
4J6 23J3
```

3.4 And / MMC - And / Mínimo Múltiplo Comum

Se ambos operandos são booleanos, a função é AND (∧). Se um dos operandos não é booleano, a função retorna o mínimo múltiplo comum.

```
      1 0 1 ∧ 0 0 1
0 0 1
      22 33 44 ∧ 4 5 6
44 165 132
```

3.5 Assignement - Assinalamento

Cria ou altera uma ou mais variáveis

```
      a←4.5
      a
4.5
      b←2 3 4
      b
2 3 4
      (a b)←2
      a,b
2 2
      (a x y)←'alfa' 4 (4 5 6)
      a
alfa
      x
4
      y
4 5 6
      a ← b ← c ← 0
```

A novidade é o assinalamento de funções. Note que não há aspas.

```
      somatorio←+/
      somatorio 1 2 3 4
10
      media←{(+/⍵)÷⍴⍵}
      media 1 2
1.5
```

Pode-se assinalar uma referência a um namespace em uma variável, que terá classe 9. Um array contendo referências terá classe 2. Um nome já existente pode receber um novo valor desde que não se altere a sua classe. Veja à pág 15 para mais detalhes.

O assinalamento pode ser indexado, com 3 possibilidades. i. Assinalamento com índices simples

```
      a←⍳5
      a[2 3]←50
      a
1 50 50 4 5
```

Em um array, índices podem ser omitidos, significando a extensão total. Veja

```
      a←3 4⍴⍳12
      a[2;]←50
      a
 1  2  3  4
50 50 50 50
 9 10 11 12
```

ii. Assinalamento a índices escolhidos

```
      c
11 12 13 14
21 22 23 24
      c[(1 2) ( 2 3)]←102 103
      c
11 102  13 14
21  22 103 24
```

iii. Assinalamento para índice encontrado: Quando a especificação do índice I não é um array de inteiros simples

```
      E←('gato' 'cachorro' 'cavalo')
      E
 gato  cachorro  cavalo
      E[⊂2 1]←'X'
      E
 gato  Xachorro  cavalo
```

Note que para qualquer array A, a quantidade `A[⊂⍬]` representa uma quantidade escalar, então

```
      a←⍳6
      a[⊂⍬]←1
      a
1
```

3.6 Selective Assignement - Assinalamento seletivo

Tem como formato

```
      (exp X) ← Y
```

A expressão `exp` seleciona elementos de X. A expressão Y é atribuída aos elementos selecionados de X por *exp*. A seguir uma lista de funções para assinalamento seletivo

```
↑ (take), ↓ (drop), , (ravel), ⍪ (tabela), ⌽ ⊖ (reverso, rotação), ⍴ (reformatação)
⊃ (disclose, pick), ⍉ (transposta monádica e diádica), / ⌿ (replicação),
\ ⍀ (expansão), ⌷ (indexação), ∈ (enlist)
```

Eis alguns exemplos

```
      x←'Curitiba'
      ((x∈'aeiou')/x)←'#'
      x
C#r#t#b#
      y←3 4⍴⍳12
      (7↑,y)←111
      y
111 111 111 111
111 111 111   8
  9  10  11  12
     mat←3 3 ⍴⍳9
      1 1⍉mat
1 5 9
      (1 1⍉mat)←0
      mat
0 2 3
4 0 6
7 8 0
```

Tudo isto pode ser usado junto com o operador each (`¨`)

```
       x←'olha' 'a' 'pandemia'
      (2↑¨x)←'*'
      x
 **ha * **ndemia
      a←'hello world'
      ((a='o')/a)←'*'
      a
hell* w*rld
```

3.7 Binomial

Em `R←X!Y`, X e Y podem ser quaisquer números. A exceção é que se Y é negativo, X deve ser inteiro. Para ambos inteiros `X!Y=(!Y)÷((!X)×(!Y-X))`. Para outros argumentos é a função beta `beta(X,Y)=÷Y×(X-1)!X+Y-1`. Para argumentos inteiros, o resultado é o número de seleções de X tomados de Y.

3.8 Branch - Desvio

É a flecha à direita, com alguma expressão após. Tem inúmeras formas, mas vamos ficar com as duas consagradas: `→label` significa desviar incondicionalmente para o label. Já na expressão `→(condição)/label` só haverá desvio se a condição for verdadeira ou igual a 1.

 ©75-20, Pedro Kantek

3.9 Catenate/Laminate - Concatenação e Laminação

Concatenar significa tomar junto e laminar tomar em uma nova dimensão. Lembrar que quando não há especificação de eixo, vale sempre o último. Já o símbolo ⍪ é usado para o primeiro eixo. Veja

```
      (⍳3),⍳2
1 2 3 1 2
       (2 3⍴⍳6),0
1 2 3 0
4 5 6 0
       (2 3⍴⍳6)⍪0
1 2 3
4 5 6
0 0 0
      (2 3⍴⍳6),[0.5]0
1 2 3
4 5 6

0 0 0
0 0 0
      (2 3⍴⍳6),[1.5]0
1 2 3
0 0 0

4 5 6
0 0 0
```

3.10 Ceiling - Teto

No formato `R←⌈Y`. Y deve ser numérico, e se for complexo, a operação será aplicada às partes real e imaginária do número.

```
       ⌈1 1.2 1.9 ¯3.4
1 2 2 ¯3
       ⌈1.2J3.9
2J4
```

3.11 Circular

No formato `R←X○Y`, onde Y deve ser numérico e X estar entre 12 positivo e negativo (`¯12 ≤ X ≤ 12`). Veja-se a tabela, na qual a e b são `a+bi` e θ é a fase.

X	X negativo: `(-X)○Y`	X positivo: `X○Y`
0	igual à direita (não existe -0)	`(1-Y*2)*.5`
1	`arcsen Y`	`sen Y`
2	`arccos Y`	`cos Y`
3	`arctan Y`	`tan Y`
4	`Y=¯1:0` `Y≠¯1:(Y+1)×((Y-1)÷Y+1)*0.5`	`(1+Y*2)*.5`
5	`arcsenh Y`	`senh Y`
6	`arccosh Y`	`cosh Y`
7	`arctanh Y`	`tanh Y`
8	`-8○Y`	`(-1+Y*2)*.5`
9	`Y`	`a`
10	`+Y`	`|Y`
11	`Y×0J1`	`b`
12	`*Y×0J1`	θ

3.12 Conjugate - Conjugado

Em `R←+Y`, se Y é real ou não numérico devolve Y sem alteração, como ⊣, mas este é mais rápido. Se Y é complexo, nega a parte imaginária

```
     a←'alfa'
      +a
alfa
      +3J4
3J¯4
```

3.13 Deal - Aleatórios sem repetição

Em `R←X?Y`, Y deve ser escalar não negativo, idem para X, porém `X≤Y`. R é um vetor de seleções randômicas de `⍳Y` sem repetição.

```
      5?8
8 6 5 3 4
```

3.14 Decode - Decodificação

No formato `R←X⊥Y`, onde X e Y devem ser arrays numéricos simples. R resulta no número obtido pela avaliação do número Y na base de numeração X

```
      60 60⊥ 2 27
147
      2⊥1 0 1 1
11
```

Este comando também permite avaliar polinômios como em $y_1.x^2 + y_2.x + y_3$ devendo-se neste caso escrever `x⊥y₁,y₂,y₃` como em `2⊥1 2 3 4 = 26` que corresponde ao polinômio $y_1.x^3 + y_2.x^2 + y_3.x + y_4 =$ o que é $1.2^3 + 2.2^2 + 3.2 + 4 = 8 + 8 + 6 + 4 = 26$

3.15 Depth - Profundidade

Cujo formato é `R←≡Y` e Y pode ser qualquer array. R é o número máximo de aninhamento de Y. Alguém escalar tem depth de 0. Qualquer array simples, sem aninhamento tem depth de 1. Qualquer array cujos elementos não são simples é aninhado e seu depth é uma unidade maior do que o elemento mais aninhado que o formar. Y tem depth uniforme se todos os seus elementos têm o mesmo depth uniforme. Se Y não tem depth uniforme a resposta é negativa.

```
        ≡55
0
      ≡2 3⍴⍳8
1
      ≡⊂1 2 3
2
      ≡( 1 2 (⊂1 2))
¯3
```

3.16 Direction (Signal) - Direção (sinal)

No formato `R←×Y`, Y deve ser um array numérico. Para Y real, informa o sinal de Y, a saber: 0 se Y é zero, 1 se Y é positivo e -1 se Y é negativo. Se Y é complexo a resposta é um complexo com a mesma fase e com magnitude igual a 1.

```
     ×10 0 ¯11
1 0 ¯1
      ×3J4
0.6J0.8
```

3.17 Disclose - Desempacotamento

No formato `R←⊃Y` ou `R←↑Y`. Aqui há um problema: 2 símbolos a depender do nível de migração. Pode ser `⊃` ou `↑`. Fiquemos com o primeiro. Devolve o primeiro elemento do ravel de Y ou se este é vazio devolve o protótipo de Y.

```
      ⊃2 3⍴5+⍳5
6
      ⊃⍬
0
```

 ©75-20, Pedro Kantek

3.18 Divide - Divisão

No formato `R←X÷Y`, aqui R é a divisão de X por Y. Se `⎕DVI` é 0 e Y é 0 então se X é 0 a divisão é 1. Se X é diferente de 0 a divisão dá um erro de domínio. Se `⎕DVI` é 1 e Y é 0, a divisão dá 0 para qualquer valor de X.

```
      ⎕DIV
0
      1 2 3 ÷ 2 3 4
0.5 0.6666666667 0.75
      4J6÷ 2 3J1
2J3 1.8J1.4
```

3.19 Drop

No formato `R←X↓Y` elimina as extremidades selecionadas

```
       3↓⍳10
4 5 6 7 8 9 10
      ¯2↓⍳10
1 2 3 4 5 6 7 8
      1 ¯1↓3 3⍴⍳9
4 5
7 8
```

3.20 Enclose - Empacotamento

Cujo formato é `R←⊂Y`, onde Y pode ser qualquer array. R é um array escalar formado por Y. Se Y é um escalar, R fica escalar e não muda. Caso contrário a profundidade de R é uma unidade maior do que a profundidade de Y

```
      ≡⊂4
0
      ≡4
0
      ⊂4 2⍴⍳8
 1 2
 3 4
 5 6
 7 8
      ⍴⊂4 2⍴⍳8

      ≡⍴⊂4 2⍴⍳8
1
```

O enclosed pode ser feito com índices k. Este é um vetor de zero ou mais eixos em Y.

3.21 Encode - Codificação

No formato `R←X⊤Y`, devolve a representação de Y no sistema numérico definido por X

```
     2 2 2 2 ⊤7
0 1 1 1
      60 60⊤193
3 13
```

3.22 Enlist - Pertence à lista

No formaro `R←∈Y` devolve os elementos como um simples vetor. Veja

```
      M←1 (2 2⍴2 3 4 5) (6(7 8))
      M
1  2 3  6  7 8
   4 5
      ,M
```

```
1  2 3  6  7 8
   4 5
      ⍴,M
3
      ∈M
1 2 3 4 5 6 7 8
      ⍴∈M
8
```

3.23 Equal - Igual

No formato `R←X=Y`, retorna 1 se os operandos forem iguais e 0 senão. Depende de `⎕CT`. Não tem versão monádica.

```
      'aqui'=⍳4
0 0 0 0
           1=1.0000000000001
0
           1=1.00000000000001
1
         ⎕CT
1E¯14
```

3.24 Exclude - Subtração de conjunto

No formato `R←X~Y`, devolve um vetor com os elementos de X que não estão em Y na ordem em que eles ocorrem em X. Na linguagem de conjuntos, $a \sim b \Leftrightarrow a - b$

```
         'Curitiba'~'Parana'
Cuitib
      3 5 7 7 9~⍳6
7 7 9
```

3.25 Execute

No formato `R←{X}⍎Y`, Y deve ser um escalar ou vetor de caracteres contendo uma expressão válida de APL. A expressão pode conter subexpressões separadas por `⋄`. X se presente deve ser um namespace no qual a expressão seja executada.

Tudo ocorre como se a expressão tivesse sido recém digitada: ela é executada

```
      ⍎'2+5'
7
```

3.26 Expand - Expansão

No formato `R←X\[K]Y`. O eixo K é opcional e se ausenta implica o último eixo de Y

```
       1 ¯2 3 ¯4 5\1
1 0 0 1 1 1 0 0 0 0 1 1 1 1 1
      1 2 3\4 5 6
4 5 5 6 6 6
      1 0 1 1 0 1\'faca'
f ac a
```

No primeiro eixo, pode-se usar o símbolo `⍀`.

3.27 Exponential - Exponencial

No formato `R←*Y`, Y deve ser numérico. R é numérico e corresponde à $Y - sima$ potência de e, a base dos logaritmos naturais

```
      *¯1 0 1 2
0.3678794412 1 2.718281828 7.389056099
```

Atentemos para a brilhante fórmula de Euler, aquela que engloba os 5 principais símbolos da matemática ($e^{\pi i} + 1 = 0$)

```
      1+*○0j1
0
```

3.28 Factorial - Fatorial

R←!Y. Quando Y é inteiro. Quando não é, equivale à função gamma de $Y + 1$.

```
      !1 2 3 4 5
1 2 6 24 120
```

3.29 Find

No formato R←X⍷Y, o resultado é um booleano com o formato de Y que identifica ocorrências de X em Y. Note que ao contrário de ∈ a ocorrência deve ser do X completo. Acompanhe

```
      3 4 5 ⍷ ⍳10
1 1 1
      3 4 5 ∈ ⍳10
0 0 1 0 0 0 0 0 0 0
      3 4 6 ⍷ ⍳10
0 0 0 0 0 0 0 0 0 0
```

3.30 Floor - Piso

No formato R←⌊Y devolve o maior inteiro menor ou igual a Y. Para complexos, o resultado depende do relacionamento entre as partes reais e imaginárias do número

```
     ⌊ 1J0.1 2.9J¯1.1 3.3J3.3
1 2J¯1 3J3
     ⌊ 1 1.1 1.9 ¯1.1 ¯1.9 2
1 1 1 ¯2 ¯2 2
```

3.31 Format - Formatação

No formato monádico R←⍕Y transforma Y de numérico para caracter. Números flutuantes dependem de ⎕PP. A representação é igual à da tela.

Já o format diádico no formato R←X⍕Y permite controlar (via X) a apresentação da formatação de Y. Se qualquer elemento de Y é complexo, a função retorna erro de domínio.

```
      ⍕1234.567
1234.567
      ⍴⍕1234.567
8
      2 ⍕1234.567
#1234.57      ⍝ represento espaço por #
      12 1⍕1234.567
######1234.6
      12 ¯1⍕1234.567
########1.E3
```

3.32 Grade Down - Índices decrescentes

No formato R←⍒Y, retorna os índices que ordenam Y em ordem decrescente. No formato diádico, R←X⍒Y, a ordenação base é aquela dada por X. Veja

```
      a←3 5 12 88 7 ¯2
      ⍒a
4 3 5 2 1 6
      a[⍒a]
88 12 7 5 3 ¯2
     b←'agora Vai MESMO'
      ⍒b
```

```
4 3 9 2 1 5 8 7 13 15 11 14 12 6 10
      b[⍒b]
roigaaaVSOMME
      'ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz'⍒b
6 10 4 3 9 2 1 5 8 7 13 15 11 14 12
      'abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ'⍒b
6 10 7 13 15 11 14 12 4 3 9 2 1 5 8
```

3.33 Grade Up - Índices crescentes

Não vale a pena gastar muita pena aqui, mas é a mesma coisa do anterior só que agora em ordem crescente.

3.34 Greater - Maior

No formato `R←X>Y` retorna 1 se $X > Y$ e 0 senão. Não tem versão monádica.

```
      1 5 77 > 2
0 1 1
      6>2 4⍴⍳8
1 1 1 1
1 0 0 0
```

3.35 Greater or equal - Maior ou igual

No formato `R←X≥Y` retorna 1 se $X \geq Y$ e 0 senão. Não tem versão monádica.

```
      1 5 77 ≥ 2
0 1 1
      6≥2 4⍴⍳8
1 1 1 1
1 1 0 0
```

3.36 Index - Índice

Esta é uma função mandrake no sentido de que ganhou símbolo apenas para coerência com o resto do APL. Na versão clássica era indicada por elementos entre colchetes e aqui o pecado mortal: usar 2 símbolos para uma única função, fugindo ao esquema geral da linguagem. O formato é `R←X⌷Y` e isto deve ser assim interpretado

```
(I J ... ⌷ Y) ≡ Y[I;J;...]
```

Note que este índice pode ser usado em especificações seletivas. Depende de `⎕IO`. Pode ser usado com o operador eixo

```
     a←'CuritibaParaná'
      a[2 5 8]
uta
      2 5 8⌷a   ⍝ dará erro: veja a definição acima
LENGTH ERROR
      2 5 8⌷a
           ^
      (⊂2 5 8)⌷a
uta
      a←3 4⍴⍳12
      a
1  2  3  4
5  6  7  8
9 10 11 12
      2⌷a
5 6 7 8
      1⌷[1]a
1 2 3 4
      1⌷[2]a
1 5 9
```

 ©75-20, Pedro Kantek

```
      2⌷[1]a
5 6 7 8
      2⌷[2]a
2 6 10
```

3.37 Index Generator - Gerador de índices

No formato `R←⍳Y`. Y deve ser escalar ou vetor numérico inteiro positivo. Se Y é escalar a resposta é o vetor que começa em `⎕IO` e tem X elementos, com incremento de 1.

Quando Y é um vetor, a resposta é um array numérico composto pelo conjunto de todas as coordenadas possíveis em um array de forma Y. Veja

```
      ⎕IO←0
      ⍳5
0 1 2 3 4
      ⎕IO←1
      ⍳5
1 2 3 4 5
      ⍳2 3
 1 1  1 2  1 3
 2 1  2 2  2 3
```

3.38 Index of - Índice de

No formato `R←X⍳Y`. Retorna a primeira ocorrência de Y em X. Se o elemento não pode ser achado, a resposta é `⎕IO+⊃⍴X`. O formato da resposta é o de Y.

```
      'uxih' ⍳ 'Curitiba'
5 1 5 3 5 3 5 5
      'Curitiba' ⍳ 'uxih'
2 9 4 9
      a←3 4 ⍴ ⍳ 12
      a
1  2  3  4
5  6  7  8
9 10 11 12
      a ⍳ 1 2 3 4
1
      a ⍳ 4 5 6 7
4
      a ⍳ 5 6 7 8
2
```

3.39 Indexing - Indexação

No formato `R←X[Y]`, usa 2 caracteres para uma única função. Paciência. Índices são separados por `;`. Depende de `⎕IO`.

```
      a←10 20 30
      a[2 2⍴⍳3]
10 20
30 10
      'um' 'dois' 'tres' [2 1]
 dois  um
      c←3 4⍴⍳12
      c[1;3]
3
      c[;3]
3 7 11
```

Há um tipo de indexação chamado *choose indexing* no qual Y é um array não simples. Cada elemento de Y identifica um elemento de X por um conjunto de índices com um elemento por eixo de X na ordem principal.

```
      c←3 4⍴⍳12
      c[⊂1 3]
3
      c[(2 1) (2 2) (1 2)]
5 6 2
      s←5
      s[4⍴⊂⍳0]  ⍝ um escalar é indexado por vetor vazio empacotado (⊂⍳0)
5 5 5 5
```

3.40 Intersection - Intersecção

No formato `R←X∩Y`. Ambos devem ser escalares ou vetores. Se forem escalares serão tratados como vetores unitários. A resposta é o conjunto de elementos que aparecem em X e Y na ordem determinada por X. Se o elemento é repetido em X e ocorre em Y aparecerá repetido na saída

```
      ¯2 11 3 66 7 9 11∩⍳12
11 3 7 9 11
      1 2  'mais' 'agora' ∩ 1 2 3 4
1 2
```

3.41 Interval Index - Intervalo de índices

No formato `R←X⍸Y`. X é um array não escalar ordenado que representa conjuntos de intervalos. O resultado é um array inteiro que identifica em qual intervalo o correspondente valor de Y cai. Neste exemplo

```
      1 10 100 1000 ⍸ 5 50 50000
1 2 4
```

Neste exemplo, pode-se criar a seguinte tabela de intervalos

0	`<1`
1	`≥1 ∧ <10`
2	`≥10 ∧ <100`
3	`≥100 ∧ <1000`
4	`≥1000`

Se X não estiver em ordem crescente, há um erro de domínio.

```
      'aeiou'⍸'Curitiba'
0 5 4 3 4 3 1 1
      'aeiou'⍸'Curitiba' 1 2
0 5 5
```

3.42 Left - Esquerda

No formato `R←X⊣Y` devolve X

```
      2⊣3
2
```

3.43 Less - Menor

No formato `R←X<Y` retorna 1 se $X < Y$ e 0 senão. Não tem versão monádica.

```
      1 5 77 < 2
1 0 0
      6<2 4⍴⍳8
0 0 0 0
0 1 1 1
```

 ©75-20, Pedro Kantek

3.44 Less or equal - Menor ou igual

No formato `R←X≤Y` retorna 1 se $X \leq Y$ e 0 senão. Não tem versão monádica.

```
      1 5 77 ≤ 2
1 0 0
      6≤2 4⍴⍳8
0 0 0 0
0 1 1 1
```

3.45 Logarithm - Logaritmo

No formato `R←X⍟Y` devolve o logaritmo de Y na base X

```
      2 ⍟ 10 200
3.321928095 7.64385619
      10 ⍟ 1 10 200
0 1 2.301029996
      10 ⍟ 2J3
0.5569716762J0.4268218909
```

Note que o logaritmo diádico é definido em termos do logaritmo monádico como segue $\log_x Y \Leftrightarrow \frac{\log_e Y}{\log_e X}$ veja-se o exemplo: $\log_{10} 2 = 0.3010 = \frac{\log_e 2}{\log_e 10} = \frac{0.6931}{2.3025} = 0.3020$ a menos de erros de arredondamento. Em APL, portanto, `X⍟Y ≡(⍟Y)÷⍟X`.

3.46 Magnitude

No formato `R←|Y`, se Y é real devolve o valor absoluto de Y e se é complexo, $(a + bi) = \sqrt{a^2 + b^2}$

```
      |1 ¯2 3 4
1 2 3 4
      |2J2
2.828427125
```

3.47 Match

No formato `R←X≡Y` devolve 1 se X e Y são idênticos, 0 senão. Dois arrays não vazios são idênticos se têm a mesma estrutura e os mesmos elementos nos lugares equivalentes. Arrays nulos só são iguais se têm o mesmo protótipo.

```
      ⍬≡⍳0
1
      ⍬≡''
0
      (2 4⍴⍳8)≡'alfabeto'
0
```

3.48 Materialize

No formato `R←⍎Y`, onde Y é uma referência a uma instância de classe com propriedade default. Se não for, retorna Y. Se for, retorna as propriedades da classe.

3.49 Matrix Divide - Divisão Matricial

No formato `R←X⌹Y`, onde X e Y devem ser arrays numéricos simples de rank menor ou igual a 2. Y deve ser não singular, isto é, ter determinante diferente de 0. Um escalar é tratado como matriz unitária. Se é um vetor, é tratado como matriz coluna. O número de linhas de ambos deve ser igual. R é a divisão matricial de X por Y ou seja, `Y+.×R` é X. R é calculado de forma a minimizar `(X-Y+.×R)*2`

Dada a característica meio exotérica desta função, vamos a um exemplo um pouco mais elaborado. Seja o sistema

$$\begin{array}{rrrcl} 3x & +4y & +2z & = & 25 \\ 9x & -y & -z & = & 1 \\ 5x & & +z & = & 10 \end{array}$$

que pode ser resolvido em APL com facilidade fazendo

```
      a←3 3⍴3 4 2 9 ¯1 ¯1 5 0 1
      25 1 10⌹a
1 3 5
      a+.×1 3 5
25 1 10
```

a interpretação é que $x = 1$, $y = 3$ e $z = 5$ é a solução do sistema.

Se o número de linhas é maior do que as colunas, a resposta usa o método dos mínimos quadrados para minimizar a solução. Se, no exemplo acima acrescentarmos outra equação, por exemplo $6x + 2y + z = 20$ a resposta agora é

```
      a←4 3⍴3 4 2 9 ¯1 ¯1 5 0 1  6 2 1
      a
3  4  2
9 ¯1 ¯1
5  0  1
6  2  1
      25 1 10 20⌹a
1.097888676 3.282149712 4.873320537
```

Obviamente, esta função aceita e devolve complexos, quando for o caso.

3.50 Matrix Inverse - Matriz Inversa

No formato `R←⌹Y` devolve a matriz inversa de Y desde que esta seja não singular ($det(Y) \neq 0$). No mundo do cálculo numérico, a inversa tem inúmeras aplicações, que ficam a cargo do leitor. Este texto é sobre APL, não sobre cálculo numérico.

```
      a←3 3⍴3 4 2 9 ¯1 ¯1 5 0 1
      b←⌹a
      b
 0.02040816327  0.08163265306  0.04081632653
 0.2857142857   0.1428571429  ¯0.4285714286
¯0.1020408163  ¯0.4081632653   0.7959183673
      b+.×25 1 10   ⍝ cuidado que esta multiplicação NÃO é comutativa
1 3 5
```

3.51 Maximum - Máximo

No formato `R←X⌈Y`, retorna o maior dos dois valores X ou Y

```
     1 2 3 4 ⌈ 4 3 2 1
4 3 3 4
     4⌈⍳8
4 4 4 4 5 6 7 8
```

3.52 Membership - Membro

No formato `R←X∈Y` devolve um booleano no formato de X informando 1 se o elemento de X pertence a Y e 0 senão.

```
      1 6 10∈⍳7
1 1 0
      'Curitiba' ∈ 'Paraná'
0 0 1 0 0 0 0 1
      'Curitiba' ∈ ⍳10
0 0 0 0 0 0 0 0
```

3.53 Minimum - Mínimo

No formato `R←X⌊Y`, retorna o menor dos dois valores X ou Y

```
      1 2 3 4 ⌊ 4 3 2 1
1 2 2 1
      4⌊⍳8
1 2 3 4 4 4 4 4
```

 ©75-20, Pedro Kantek

3.54 Minus - Menos

Em `R←X-Y` devolve a subtração de X menos Y. Obviamente obedece à regra dos sinais. Veja

```
      1 2 ¯3 4 5 6 - 3
¯2 ¯1 ¯6 1 2 3
      3 - 1 2 ¯3 4 5 6
2 1 6 ¯1 ¯2 ¯3
      1 5 10 - 3 2 1
¯2 3 9
```

3.55 Mix

Nos formatos `R←↑[K]Y` ou `R←⊃[K]Y`. Aqui neste Dyalog APL cujo `⎕ML` é 1, vale o primeiro formato. Então em `R←↑[K]Y`

```
      ⎕ML
1
      ↑(1 2)(3 4)(5 6)
1 2
3 4
5 6
      ↑(1 2)(3 4 5)(6 7)
1 2 0
3 4 5
6 7 0
```

Uma aplicação importante desta função é na leitura de um aquivo nativo via `⎕NGET`. O retorno é um array de vetores (cada linha é um vetor). Para transformar isso em uma matriz de caracteres (como é mais usual operar), usa-se função `↑`, como pode-se ver em

```
[0] r←{a}FMD arq;LF;aux;fi;ncol;nlin
[1] LF←⎕UCS 10
[2] →(0=⎕NC'a')/leitura
[3] gravacao:(,a,LF)⎕NPUT(arq)1
[4] r←a
[5] →fim
[6] leitura:aux←⎕NGET(arq)1
[7] r←↑⊃aux[1]
[8] fim:
```

Nesta função, a chamada monádica implica leitura e a diádica gravação. Note o uso de mix na linha 7 logo após a leitura.

```
      ↑'juan' 'alois' 'maria jose'
juan
alois
maria jose
      ⍴↑'juan' 'alois' 'maria jose'
3 10
```

3.56 Multiply - Multiplicação

Em `R←X×Y`, o resultado é a multiplicação de X por Y.

```
      2 3 4 × 1 ¯2 5
2 ¯6 20
      3J5 × 0J1 1 3J6
¯5J3 3J5 ¯21J33
```

3.57 Nand - Not and

Com formato `R←X⍲Y`, devolve a resposta conforme a tabela verdade

X	Y	R
0	0	1
0	1	1
1	0	1
1	1	0

Possivelmente uma das mais inúteis e menos usadas funções do APL, até porque basta negar a função and para obter este resultado.

3.58 Natural Logarith - Logaritmo natural

Em `R←⍟Y` devolve o logaritmo de Y na base natural (número $e = 2.718281828$).

```
      ⍟1 2 3 4
0 0.6931471806 1.098612289 1.386294361
      ⍟1J1 2J2 3J3 4J0
0.3465735903J0.7853981634   1.039720771J0.7853981634   1.445185879J0.7853981634
 1.386294361            ⍝   quebrei em 2 para ficar mais fácil de ler...
```

3.59 Negative - Negativo

Em `R←-Y` devolve o valor negativo de Y. Para complexos nega-se a parte real e a parte imaginária.

```
      - 1 2 ¯3 10 20
¯1 ¯2 3 ¯10 ¯20
      -1J1
¯1J¯1
      -1J¯1
¯1J1
```

3.60 Nest - Aninha

cujo formato é `R←⊆Y`, se Y é simples, R é um escalar com o Y aninhado. Se não é simples, R é o Y inalterado.

```
      ]DISPLAY 1 2 3
┌→─────┐
│1 2 3│
└~─────┘
      ]DISPLAY ⊆ 1 2 3
┌→─────┐
│┌→─────┐│
│└~─────┘│
└∈─────┘
      ]DISPLAY ⊆⊆ 1 2 3
┌→─────┐
│┌→─────┐│
│└~─────┘│
└∈─────┘
```

3.61 Nor - Not or

Com formato `R←X⍱Y`, devolve a resposta conforme a tabela verdade

X	Y	R
0	0	1
0	1	0
1	0	0
1	1	0

Vale o mesmo comentário do nand, vide acima.

3.62 Not - Negação

No formato `R←~Y`, exige que Y seja booleano, respondendo 1 se Y é 0, e 0 senão.

 ©75-20, Pedro Kantek

```
      ~0 1
1 0
      ~~0
0
```

3.63 Not Equal - Diferente

No formato `R←X≠Y`, devolve 1 se X é diferente de Y e 0 senão.

```
a←2 4 ⍴1 1 1 0 1
      a
1 1 1 0
1 1 1 1
      ~a
0 0 0 1
0 0 0 0
```

3.64 Not Match - Não coincidência

No formato `R←X≢Y`. X e Y são quaisquer. R é 1 se X é idêntico a Y, 0 senão.

```
      (⍳5) ≢ 3 3⍴⍳6
1
      33 ≢ 'aa'
1
```

3.65 Or / mdc - Ou / Máximo divisor comum

No formato `R←X∨Y`. Aqui se X e Y são booleanos, R é a função lógica OU ($\vee$) cuja tabela verdade é

X	Y	R
0	0	0
0	1	1
1	0	1
1	1	1

Se um dos operandos X ou Y não são booleanos, a função é o máximo divisor comum.

```
      1 0 0 1 ∨ 1 1 0 0
1 1 0 1
      22 35 40 ∨ 50 50 50
2 5 10
```

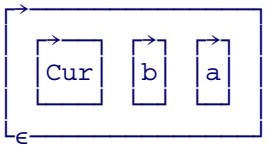
Uma dica para relembrar o mmc e o mdc de acordo com os símbolos. O de máximo divisor comum – mdc, é o de um número menor que os operandos. Daí o seu símbolo aponta para baixo ($\vee$). Já o mínimo múltiplo comum – mmc, é um número maior que X e Y. O seu símbolo aponta para cima ($\wedge$).

3.66 Partition - Partição

No formato `R←X⊆[K]Y`, devolve os elementos de Y particionados de acordo com X. X deve ser um vetor de inteiros.

```
      ]display 1 1 1 2 2 3 3 3 ⊆ 'Curitiba'
┌→──────────────┐
│ ┌→─┐ ┌→┐ ┌→──┐ │
│ │Cur│ │it│ │iba│ │
│ └──┘ └──┘ └───┘ │
└∈──────────────┘
      ]display 1 1 1 0 0 0 1 1    ⊆ 'Curitiba'
┌→──────┐
│ ┌→─┐ ┌→┐ │
│ │Cur│ │ba│ │
│ └──┘ └──┘ │
└∈──────┘
```

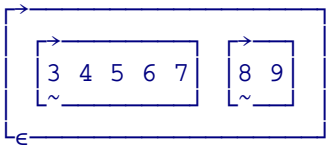
```
      ]display 1 1 1 0 0 0 5 6     ⊆ 'Curitiba'
```



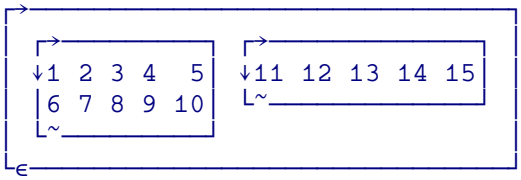
3.67 Partitioned Enclose - Partição anexada

No formato `R←X⊂[K]Y`. X deve ser um vetor de inteiros. R é um vetor de itens selecionados de Y pela inserção de 0 ou mais divisores, especificados por X entre as células principais.

```
      ]display 0 0 1 0 0 0 0 1 0 0 ⊂ ⍳9
```




```
      ]display 1 0 1⊂[1] 3 5⍴⍳15
```



3.68 Pi Times - Vezes Pi

No formato `R←○Y`, devolve o valor de Y multiplicado por $\pi = 3.141592654$.

```
      180÷○1  ⍝ quantos graus vale 1 radiano ?
57.29577951
```

3.69 Pick - Escolha

No formato `R←X⊃Y`, Y pode ser qualquer e X é um vetor de índices de Y. R é um item selecionado da estrutura de Y de acordo com X.

3.70 Plus - Adição

Cujo formato é `R←X+Y` é a operação de adição, sem maiores comentários.

3.71 Power - Potência

O formato é `R←X*Y` e o valor de R é X elevado à potência Y. Se Y é 0, R é 1. Se X é 0, Y deve ser não negativo. Se X é negativo, o resultado R provavelmente é complexo.

```
      1 2 0 ¯2 ¯3 4 * 0 ¯1 2 ¯3 1.8 ¯5
1  0.5  0  ¯0.125  5.84488409J¯4.246556863  0.0009765625
```

3.72 Ravel - Vetorização

No formato `R←,Y` ou `R←,[K]Y` devolve um vetor no primeiro caso e um array no segundo com os elementos de Y

```
      m←3 4⍴⍳12
      ,m
1 2 3 4 5 6 7 8 9 10 11 12
      ⍴,[0.5]m
1 3 4
      ⍴,[1.5]m
3 1 4
```

 ©75-20, Pedro Kantek

```
      ρ,[2.5]m
3 4 1
```

3.73 Recíprocal - Recíproco

No formato `R←÷Y`. R é o resultado de `1÷R`. Depende de `⎕DIV`

```
      ÷ 4 5 2
0.25 0.2 0.5
```

3.74 Replicate - Replicação

No formato `R←X/[K]Y`, Y pode ser qualquer. X é um vetor.

```
      1 0 1/'ABC'
AC
      1 ¯2 3 4/⍳4
1 0 0 3 3 3 4 4 4 4
      m←3 4⍴⍳12
      1 0 1/[1]m
1  2  3  4
9 10 11 12
      1 0 0 1/[2]m
1  4
5  8
9 12
```

3.75 Reshape - Reformatação

No formato `R←X⍴Y`, R tem os elementos do ravel de Y com a forma de X. Os elementos de Y são tomados ciclicamente ou desprezados, conforme o caso.

```
      2 4⍴⍳5
1 2 3 4
5 1 2 3
      2 4⍴⍳20
1 2 3 4
5 6 7 8
      2 3⍴0
0 0 0
0 0 0
```

3.76 Residue - Resto

No formato `R←X|Y`. Para argumentos positivos, R é o resto quando Y é dividido por X. Se $X = 0$, R é Y.

```
     4 4 ¯4 1 | 10 17 10 13
2 1 ¯2 0
     1j2|5j3
¯1J1
```

3.77 Reverse - Reverso

No formato `R←⌽[K]Y`, Y pode ser qualquer. O eixo é opcional e se ausente, vale o último. Para o primeiro eixo, o símbolo pode ser `⊖`.

```
      ⌽⍳5
5 4 3 2 1
      m ← 3 4 ⍴ ⍳ 12
      ⌽[1]m
9 10 11 12
5  6  7  8
```

```
1  2  3  4
      ⌽m
 4  3  2 1
 8  7  6 5
12 11 10 9
      ⊖m
9 10 11 12
5  6  7  8
1  2  3  4
```

3.78 Right - Direito

No formato `R←X⊢Y`. O resultado R é Y.

```
      1 2 3 ⊢ 'alfa' 123
 alfa  123
```

3.79 Roll - Rolo

No formato `R←?Y`. Gera um aleatório dentro do `⍳` do elemento correspondente de Y. A forma de R é a mesma de Y. Há um caso particular quando Y vale 0. Neste caso a resposta é um real entre 0 e 1 uniformemente distribuído.

```
      ?3 4 5
3 1 2
      ?3 3 ⍴50
 8 28 39
43 31 41
11  1 18
      ?0
0.3787086806
      ?4⍴0
0.3128095179 0.8912442253 0.5662684266 0.8561391733
```

3.80 Rotate - Rotação

No formato `R←X⌽[K]Y`. Y pode ser qualquer array. X deve ser um array simples de inteiros. R tem os elementos rotacionados.

```
      3⌽⍳10
4 5 6 7 8 9 10 1 2 3
      ¯2⌽⍳10
9 10 1 2 3 4 5 6 7 8
      m←3 4⍴⍳12
      1 2 3⌽m   ⍝ equivale a 1 2 3⌽[2]m
 2 3  4  1
 7 8  5  6
12 9 10 11
      1 2 3 4⌽[1]m
5 10  3  8
9  2  7 12
1  6 11  4
```

3.81 Same - Mesmo

Nos formatos `R←⊣Y` ou `R←⊢Y`, não faz nada e devolve Y

```
      ⊣1 2 3
1 2 3
      ⊢'abc'
abc
```

 ©75-20, Pedro Kantek

3.82 Shape - Forma

No formato `R←⍴Y` devolve a forma de R que é um vetor de inteiros positivos. Cada elemento de R informa uma dimensão de Y. Se Y é um escalar, R é um vetor vazio. O rank de Y é dado por `⍴⍴Y`.

```
      m←3 4 ⍴ ⍳12
      ⍴m
3 4
      ⍴⍴m
2
```

3.83 Split - Cisão

No formato `R←↓[K]Y`, Y pode ser qualquer. K é opcional e se ausente implica no último. Os ítens de R são os sub-arrays de Y. O rank diminuir uma unidade

```
      ]display a← 3 4⍴⍳12
┌→──────────┐
↓1  2  3  4 │
│5  6  7  8 │
│9 10 11 12 │
└~──────────┘
      ↓a
 1 2 3 4  5 6 7 8  9 10 11 12
      ⍴↓a
3
      ]display ↓a
┌→──────────────────────────────────┐
│ ┌→────────┐ ┌→────────┐ ┌→──────────┐ │
│ │1 2 3 4 │ │5 6 7 8 │ │9 10 11 12 │ │
│ └~────────┘ └~────────┘ └~──────────┘ │
└∊──────────────────────────────────┘
    ]display ↓[1]a
┌→──────────────────────────────────┐
│ ┌→──────┐ ┌→──────┐ ┌→──────┐ ┌→──────┐ │
│ │1 5 9 │ │2 6 10 │ │3 7 11 │ │4 8 12 │ │
│ └~──────┘ └~──────┘ └~──────┘ └~──────┘ │
└∊──────────────────────────────────┘
```

3.84 Subtract - Subtração

No formato `R←X-Y`, R é a diferença entre X e Y.

3.85 Table - Tabela

No formato `R←⍪Y`, R é matriz de 2 dimensões, com os elementos de Y tomados na ordem maior (a primeira), preservando as formas da primeira dimensão de Y se existir

```
      a ← 3 4 ⍴ ⍳ 12
      a⍪⍳4
1  2  3  4
5  6  7  8
9 10 11 12
1  2  3  4
      a⍪[2]⍳3
1  2  3  4 1
5  6  7  8 2
9 10 11 12 3
```

3.86 Take - Tomar

No formato `R←X↑Y` toma pela frente (se X positivo) ou por trás (se X negativo) os elementos de Y. Pode ter uma especificação de eixo (pág. 132)

```
      3↑⍳8
1 2 3
      ¯2↑⍳8
7 8
      5↑⍳3
1 2 3 0 0
      ¯5↑⍳3
0 0 1 2 3
     2 3↑3 4 ⍴ ⍳ 12
1 2 3
5 6 7
```

3.87 Tally - Contador

No formato `R←≠Y` retorna o número de células maiores em Y

```
    ≠3 4⍴⍳12
3
      ≠⍳8
8
```

3.88 Times - Multiplicação

No formato `R←X×Y` devolve a multiplicação entre X e Y.

3.89 Transpose - Transposta

No formato `R←⍉Y` devolve a transposta de Y

```
     ⍉3 4⍴⍳12
1 5  9
2 6 10
3 7 11
4 8 12
     ⍴ ⍉3 4⍴⍳12
4 3
```

Tem também a versão diádica no formato `R←X⍉Y` cuja descrição é bastante complicada. Veja na pág 135 e 136.

3.90 Type - Tipo

No formato `R←∈Y` devolve um array de mesmo formato de Y onde valores numéricos são trocados por 0 e valores alfa por ' '.

3.91 Union - União

No formato `R←X∪Y` exige que X e Y sejam vetores. Devolve os elementos de X conactenados com os elementos de Y que não estão em X

```
      (⍳6)∪1 5 10 20
1 2 3 4 5 6 10 20
      'Curitiba'∪'Paraná'
CuritibaPná
```

3.92 Unique - Único

No formato `R←∪Y` retorna os elementos principais únicos de Y. Elementos no caso de vetor, linhas no caso de matriz e assim por diante, na ordem em que eles aparecem.

 ©75-20, Pedro Kantek

```
      ∪ 1 3 5 6 3 2 4 5 6 4
1 3 5 6 2 4
      ∪ 3 4⍴1 2 3 7 0 0 9 8 1 2 3 7
1 2 3 7
0 0 9 8
```

3.93 Unique Mask - Máscara Única

No formato `R←≠Y` retorna um vetor booleano cujo comprimento equivale ao número de células majoritárias (elementos principais) de *R*. O valor é 1 na sua primeira ocorrência e 0 nas demais.

```
      ≠⍳5
1 1 1 1 1
      ≠1 2 1 2 3 1 2 3 4
1 1 0 0 1 0 0 0 1
      ≠ 3 4⍴1 2 3 7 0 0 9 8 1 2 3 7
1 1 0
```

3.94 Where - Onde

No formato `R←⍸Y`. *Y* deve ser array numérico de inteiros não negativos e a resposta é `{(,⍵)/,⍳⍴⍵}`. Simples assim

```
      ⍸3 4 5 6 7
1 1 1 2 2 2 2 3 3 3 3 3 4 4 4 4 4 4 5 5 5 5 5 5 5
      ⍸1 0 1 0
1 3
      ⍸3 1
1 1 1 2
```

3.95 Resumo das funções NOVAS em Dyalog APL

Veja a seguir

função	o que faz	exemplo
`a ∧ b`	Devolve o mínimo múltiplo comum, quando a e b não booleanos	`   22 33 44 ∧ 4 5 6` `44 165 132`

`∈b`	devolve elementos de b como vetor simples	`   M←1 (2 2⍴2 3 4 5) (6(7 8))`
		`   ∈M`
		`1 2 3 4 5 6 7 8`

`a~b`	subtração de conjuntos	`'   Curitiba'~'Parana'` `Cuitib`

`⍳a`	a como vetor	`   ⍳2 3`
		`1 1   1 2   1 3`
		`2 1   2 2   2 3`

`a\b`	a não booleana	`   1 2 3 \ 4 5 6` `4 5 5 6 6 6`
`a⍷b`	ocorrência completa de a em b	`   3 4 5⍷⍳10` `0 0 1 0 0 0 0 0 0 0`
`a∩b`	ambos vetores: conjunto união	
`a⍸b`	intervalor de índices	`   1 10 100 1000 ⍸ 5 50 50000` `1 2 4`
`a⊣b`	esquerdo: retorna a	

`↑a`	devolve matriz de vetor de vetores	`   ↑(1 2)(3 4 5)`
		`1 2 0`
		`3 4 5`

`⊆a`	se a é simples, retorna `⊂a`. Senão, não faz nada	
`a∨b`	devolve o máximo divisor comum, quando a e b não booleanos	`   22 35 40 ∨ 50 50 50` `2 5 10`
`a⊆b`	partição	`   1 1 1 2 2 3 3 3 ⊆ 'curitiba'` ` cur  it  iba`
`a⊂b`	partição anexada	`   1 0 0 1 0 1 0 0 ⊂ 'curitiba'` ` cur  it  iba`
`a/b`	replicação	`   1 0 ¯2 3/⍳4` `1 0 0 4 4 4`
`a⊢b`	direito: retorna b	
`?0`	retorna real entre 0 e 1	
`↓a`	split: diminui o rank uma unidade	`   a←3 4⍴⍳12 ⋄ ↓a` `1 2 3 4   5 6 7 8   9 10 11 12`
`a∪b`	união: exige ambos vetores.	
`∪a`	únicos: retorna elementos principais únicos	
`≠a`	máscara única: retorna booleano com 1 na primeira ocorrência e 0 nas demais	
`⍸a`	where> `{(,⍵)/,⍳⍴⍵}`	

 ©75-20, Pedro Kantek

Capítulo 4

Operadores novos

4.1 Each

Este operador monádico evita a necessidade de processar os itens de um array, um após o outro em um loop explícito. Traduzido por "cada", e representado pelo sinal dieresis (¨). Pode ser usado com funções de maneira monádica ou diádica.

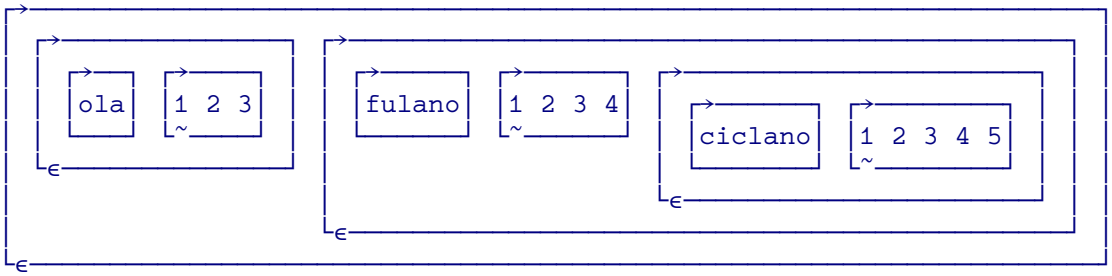
4.1.1 Each com operando monádico

Seu formato é

```
{r}←f¨Y
```

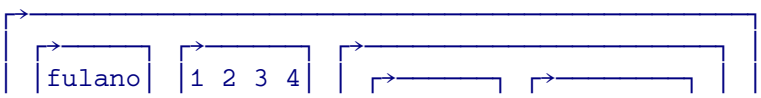
`f` pode ser qualquer função monádica. `Y` pode ser qualquer array. Cada um dos itens de Y recebe a ação da função f. Veja alguns exemplos

```
      ⍳5
1 2 3 4 5
      ⍳¨⍳5
 1  1 2  1 2 3  1 2 3 4  1 2 3 4 5
      G←('ola' (⍳3)) ('fulano' (⍳4)('ciclano'(⍳5)))
      G
  ola  1 2 3    fulano  1 2 3 4   ciclano  1 2 3 4 5
      ]display G
```




```
      ⍴G
2
      ⍴¨G
 2  3
      ⍴¨¨G
  3  3    6  4  2

      z← ¨G
      z
 fulano  1 2 3 4   ciclano  1 2 3 4 5
      ⍴z
3
      ]display z
```



```
│ └──────┘ └~──────┘ │ │ciclano│ │1 2 3 4 5│ │ │
│                    │ └───────┘ └~────────┘ │ │
│                    └∊──────────────────────┘ │
└∊─────────────────────────────────────────────┘
      w←¨¨G
      w
 ciclano  1 2 3 4 5
      ⍴w
2
      ]display w
┌→────────────────────────┐
│ ┌→──────┐ ┌→────────┐   │
│ │ciclano│ │1 2 3 4 5│   │
│ └───────┘ └~────────┘   │
└∊────────────────────────┘
```

4.1.2 Each com operando diádico

Seu formato é

`{r}←Xf¨Y`

`f` pode ser qualquer função diádica. `X` e `Y` são quaisquer arrays conformáveis. A função é aplicada aos pares formados entre X e Y. Veja alguns exemplos:

```
      1 2 3 4÷3 4 5 6
0.3333333333 0.5 0.6 0.6666666667
      1 2 3 4 ÷¨3 4 5 6
0.3333333333 0.5 0.6 0.6666666667
      'ABC','xyz'
ABCxyz
      'ABC',¨'xyz'
 Ax  By  Cz
      G←(1 (2 3))(4 (5 6))(8 9) 10
      G
 1  2 3   4  5 6   8 9  10
      1⌽G
 4  5 6   8 9  10  1  2 3
      1⌽¨G
  2 3  1    5 6  4   9 8  10
      1⌽¨¨G
 1  3 2   4  6 5   8 9  10
      1 2 3 4↑¨G
 1  4  5 6   8 9 0  10 0 0 0
```

4.2 At

Substitui iens selecionados em Y por novos valores ou aplica uma função para modificar itens selecionados em Y. Seu formato

`r←{X}(f@g)Y`

O operando g identifica quais itens do array Y são substituídos ou modificados. O g é, de duas, uma

- um array que especifica um conjunto de índices em Y. Se ele for um escalar ou um vetor especifica células majoritárias (linhas em matrizes, p. exemplo) em Y.
- ou uma função que quando aplicada a Y retorna um array booleano de mesmo formato de Y (uma máscara) na qual um valor 1 indica que o elemento vai ser substituído ou modificado. Note que o ravel da máscara indica algo sobre o ravel de Y.

O operando esquerdo f, de duas uma

- Um array que contém os valores a substituir os itens em Y identificados por g
- ou uma função que se aplica a tais itens, cujo resultado os substitui. Se a função é diádica o argumento esquerdo é X

O resultado r é o mesmo Y com os itens especificados em g substituidos ou modificados por f. Exemplos

```
      (10 20@2 4)⍳5 ⍝ substitui o seg. e quarto em ⍳5 por 10 e 20
1 10 3 20 5
       10 20@2 4⍳5 ⍝ o parênteses não é necessário
1 10 3 20 5
      (2 3⍴10 11)(@2 4)4 3⍴⍳12
 1  2  3   ⍝ substitua as linhas (células
10 11 10   ⍝ majoritárias da matriz 4 3
 7  8  9   ⍝ por linhas (circulares) contendo
11 10 11   ⍝ 10 e 11
      (0@(1 1)(4 3))4 3⍴⍳12
 0  2 3   ⍝ substitua o primeiro (1 1)
 4  5 6   ⍝ e o último (4 3) elementos
 7  8 9   ⍝ na matriz 4x3 por zero
10 11 0
      ÷@2 4⍳5 ⍝ substitua o 2. e o 4. em ⍳5
1 0.5 3 0.25 5  ⍝ pelos seus inversos
      10×@2 4⍳5
1 20 3 40 5 ⍝ multiplique por 10 o 2. e o 4. em ⍳5
      0@(2∘|)⍳5 ⍝ troque os impares por 0
0 2 0 4 0
      ÷@(2∘|)⍳5 ⍝ troque os impares pelo recíproco
1 2 0.3333333333 4 0.2
```

4.3 Atop

Seu símbolo é ⍥ e seu formato

```
{r}←{X}f⍥gY
```

A função derivada é equivalente a `fgY` ou `fXgY` e não precisa retornar resultado. Acompanhe

```
    3 1 4 1 5 ~∈1 2 3
4 5
    3 1 4 1 5 ~⍥∈1 2 3
0 0 1 0 1
```

4.4 Beside

Cuja tradução é *ao lado*. Usa o símbolo ∘ e tem formato

```
{r}←{X}f∘gY
```

g pode ser qualquer função monádica que retorne resultado. Y deve ser apropriada para a função g, com gY adequada ao argumento direito de f.

Se X for omitido, f deve ser monádico. A função derivada é equivalente a fgY ou $XfgY$ e não precisa retornar.

```
      ⍳¨2 4 6
 1 2  1 2 3 4  1 2 3 4 5 6
      +/∘⍳¨2 4 6
3 10 21
      33,⍳¨⍳3
33  1  1 2  1 2 3
      33,∘⍳¨⍳3
33 1  33 1 2  33 1 2 3
```

4.5 Bind

Cuja tradução é *ligar, ligação*. Seu formato é

```
{r}←A∘fY ou {r}←(f∘B)Y
```

 ©75-20, Pedro Kantek

Ele conecta um array A ou B a uma função dinâmica f ou como seu argumento esquerdo ou direito respectivamente. O primeiro pode ser descrito como carregamento à esquerda e o segundo como carregamento à direita. A, B e Y podem ser arrays quaisquer cujos itens sejam apropriados à função f. No caso de B ser usado como argumento direito de f os parênteses são necessários para distinguir entre B e Y. Este exemplo usa ambas as formas de bind para listar as funções do workspace

```
      □∘←∘□VR¨↓□NL 3
...
      (*∘0.5)4 16 25
2 4 5
```

4.6 Commute

Tem formato

```
{r}←{X}f⍨Y
```

f pode ser qualquer função diádica. X e Y quaiquer arrays que sejam apropriados para f. A função derivada é equivalente a YfX. Se X é omitido, o Y é duplicado e ocupa seu lugar. MEIO INÚTIL, NÃO ?????

```
      N
3 2 5 4 6 1 3
      N/⍨2|N     ↔   (2|N)/N
3 5 1 3
      ⍴⍨3   ↔  3⍴3
3 3 3
```

4.7 Produto Interno

É uma generalização da multiplicação matricial. Esta, tem como formato

```
r←X+.×Y  →  generalização  r←Xf.gY
```

A generalização acima sugere que ao inves de usar + e × pode-se usar quaisquer funções diádicas. Veja-se por exemplo

```
      (⍳10)+.×⍳10
385
      nomes←↑'pedro' 'paula' 'felipe' 'ana'
pedro
paula
felipe
ana
      ⍴nomes
4 6
      nomes∧.=6↑'pedro'
1 0 0 0
```

4.8 Key

Traduzido por chave. Funciona de maneira similar à cláusula GROUP BY do SQL. Em Dyalog chama-se com CTRL-SHIFT-K. Seu formato.

```
r←{X}f⌸Y
```

Aplica a função f a cada chave única em X e as células majoritárias de Y que tem tal chave. Se X é omitido, Y é um array cujas células majoritárias representam chaves. Neste caso f é aplicado a cada chave única e aos elementos que têm tal chave. Os elementos em r aparecem na ordem que primeiro aparecem em Y. Veja o exemplo

```
      car←'2' 'J' 'A' '4' '6'
      nai←'paus' 'espadas' 'paus' 'ouros' 'copas'
      nai,[1.5]car
 paus    2
 espadas J
 paus    A
 ouros   4
```

```
 copas   6
      nai {α':'⍵} car
  paus  espadas  paus  ouros  copas  : 2JA46
      nai {α':'⍵} ⌸ car
  paus     : 2A
  espadas  : J
  ouros    : 4
  copas    : 6
      {α ⍵} ⌸ nai
  paus      1 3    ⍝ índices dos naipes
  espadas   2
  ouros     4
  copas     5
      {α,≠⍵}⌸nai
 paus     2   ⍝ quantas cartas de cada naipe
 espadas  1
 ouros    1
 copas    1
```

4.9 Produto Externo

Tem o formato

```
{r}←X∘.gY
```

g é aplicada a cada par de elementos de X e Y. Veja

```
      (⍳4)∘.×⍳4
1 2  3  4
2 4  6  8
3 6  9 12
4 8 12 16
      1 2 3∘.⍴'ABC'
 A    B    C
 AA   BB   CC
 AAA  BBB  CCC
```

4.10 Over - Sobre

No formato `{R}←{X}f⍤gY`. Gera a função `fgY` ou `(gX)f(gY)`. Este operador permite que as operações sejam coladas para criar funções mais complexas.

```
      ]display 2 3,⊂'texto'
┌→──────────┐
│           │
│  2 3 ┌→─┐ │
│      │texto│ │
│      └──┘ │
└∊──────────┘
      ]display 2 3,⍤⊂ 'texto'
┌→──────────┐
│           │
│  ┌→─┐ ┌→─┐ │
│  │2 3│ │texto│ │
│  └~─┘ └──┘ │
└∊──────────┘
```

4.11 Power - Potência

No formato `{R}←{X}(f⍣g)Y`. Um valor de g negativo aplica a inversa `⍣¯1` vezes.

```
      f←(32∘+)∘(×∘1.8)
      f 0 100
32 212
      c←f⍣¯1
```

```
      c 32 212
0 100
      invs←{(⍺⍺⍣¯1)⍵} ⍝ operador inverso
      +\invs 1 3 6 10
1 2 3 4
      2 ∘⊥invs 9
1 0 0 1
```

4.12 Rank

No formato `R←{X}(f⍤B)Y`. Aplica f sucessivamente aos sub arrays de Y ou a operação diádica F entre os sub arrays de X e Y. Sub-arrays são selecionados pelo operador B.

```
      y←2 3 4⍴24?24
      y
19 18  3  2
22  7 10 16
14  8 21 17

15 24  1  4
20 13 11 12
 5 23  6  9
      ]display ⊂⍤2⊢y
┌→──────────────────────────────┐
│ ┌→────────────┐ ┌→────────────┐ │
│ ↓19 18  3  2  │ ↓15 24  1  4  │ │
│ │22  7 10 16  │ │20 13 11 12  │ │
│ │14  8 21 17  │ │ 5 23  6  9  │ │
│ └~────────────┘ └~────────────┘ │
└∊──────────────────────────────┘
      ]display ⊂⍤1⊢y
┌→──────────────────────────────────────────┐
↓ ┌→──────────┐ ┌→──────────┐ ┌→──────────┐ │
│ │19 18 3 2  │ │22 7 10 16 │ │14 8 21 17 │ │
│ └~──────────┘ └~──────────┘ └~──────────┘ │
│ ┌→──────────┐ ┌→──────────┐ ┌→────────┐   │
│ │15 24 1 4  │ │20 13 11 12│ │5 23 6 9 │   │
│ └~──────────┘ └~──────────┘ └~────────┘   │
└∊──────────────────────────────────────────┘
```

4.13 Reduce - Redução

No formato `R←f/[K]Y`, é a redução tradicional do APL

```
      +/1 2 3 4
10
      z←3 4⍴⍳12
      +/[1]z
15 18 21 24
      +/[2]z
10 26 42
```

4.14 Reduce N-wise

No formato `R←Xf/[K]Y`. R é o array formado pela aplicação da função F entre itens de sub vetores de comprimento X de Y.

```
      3+/⍳4 ⍝ (1+2+3) (2+3+4)
6 9
      2+/⍳4 ⍝ (1+2) (2+3) (3+4)
3 5 7
      1+/⍳4 ⍝ (1) (2) (3) (4)
1 2 3 4
```

```
      0+/⍳4 ⍝ elemento identidade para +
0 0 0 0
      0×/⍳4 ⍝ elemento identidade para ×
1 1 1 1
      2,/⍳4 ⍝ (1,2) (2,3) (3,4)
1 2   2 3   3 4
      ¯2,/⍳4 ⍝ (2,1) (3,2) (4,3)
2 1   3 2   4 3
```

4.15 Scan

No formato `R←f\[K]Y`. Versão tradicional do scan

```
      +\⍳4
1 3 6 10
      +\3 4⍴⍳12
1  3  6 10
5 11 18 26
9 19 30 42
      +\[1]3 4⍴⍳12
 1  2  3  4
 6  8 10 12
15 18 21 24
      ,\'abcde'
a ab  abc  abcd  abcde
```

4.16 Spawn - Desovar

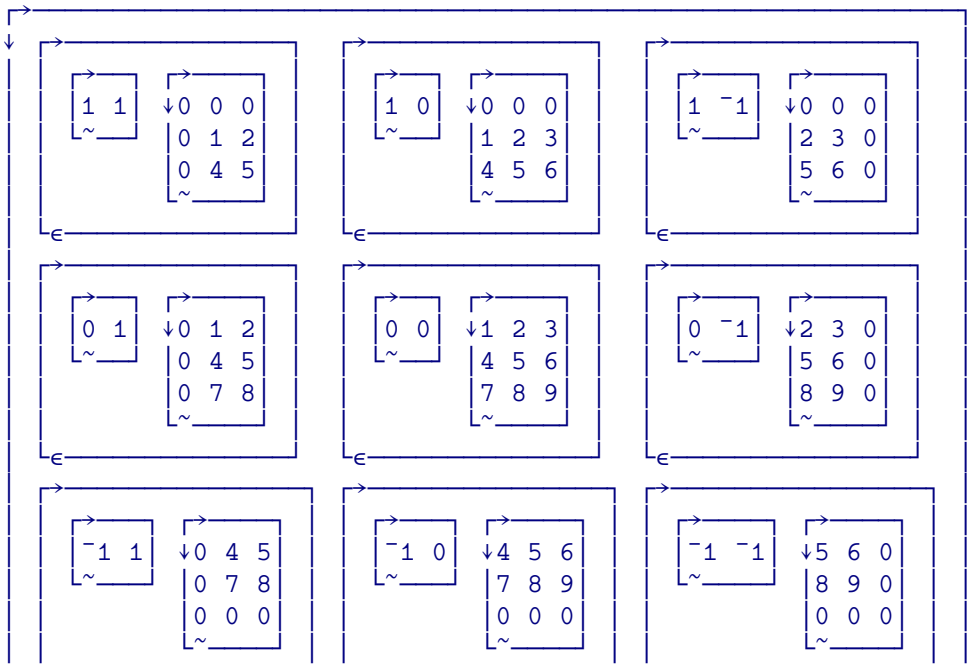
No formato `{R}←{X}f&Y`. Este operador gera uma nova thread na qual f é aplicada a Y ou entre os argumentos X e Y. O resultado é o número da thread.

```
      ⎕←÷&4
1
0.25
```

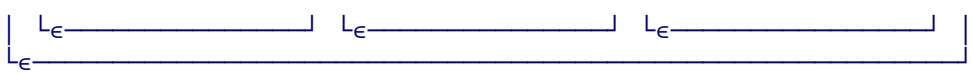
4.17 Stencil - Estêncil

No formato `R←(f⌺g)Y`. Conhecido como movimento de janela. Aplica f a uma série de retângulos (possivelmente overpostos) em Y.

```
      ]display {⊂⍺ ⍵}⌺3 3⊢3 3⍴⍳12
```



 ©75-20, Pedro Kantek



4.18 Variant - Variante

Não consegui decobrir como gerar o quad com 2 pontos dentro.

4.19 Um resumo das novidades no capítulo operadores

Veja a seguir

operador	o que faz	exemplo
`{|s}f/Y`	redução: f entre todos os itens de Y em grupos de `|s` no último eixo	`   2+/⍳5` `3 5 7 9`
`f¨Y`	each monádico	`   ⍳¨1 2 3 4 5` `1  1 2  1 2 3  1 2 3 4  1 2 3 4 5`

`Xf¨Y`	each diádico	`   'ABC','xyz'`
		`'ABCxyz'`
		`   'ABC',¨'xyz'`
		`Ax By Cz`

`{X}(f@g)Y`	At: itens em g modificados por f	`   ÷@2 3⍳5`
		`1 0.5 0.3333333333 4 5`
		`   10×@2 4⍳5`
		`1 20 3 40 5`

`{X}f⍤gY`	Atop: `fgy` ou `fXgY`	`   1 5 9-⍤+2 4 6` `¯3 ¯9 ¯15`

`Xf[K]Y`	eixo com operação diádica	`   mat←10×2 3⍴⍳6`
		`   mat+[1]1 2`
		`11 21 31`
		`42 52 62`

`{X}f∘gY`	compose: `fgy` ou `XfgY` (beside)	
`X∘gY`	compose: g entre X e Y ou seja, `XgY`	
`(f∘Y2)Y1`	compose: f entre Y_1 e Y_2 ou seja `Y1 f Y2`	

`{X}f⌸Y`	Key:f sobre os itens de Y agrupados por valores únicos de Y	`   val←5 1 10 20 3 ◇ dia←2 3 2 4 3`
		`   dia{(+/⍵)÷⍴⍵}⌸val`
		`7.5 2 20 ⍝ medias diarias`

`{X}f⍨Y`	Commute: o mesmo que `YfX` ou `YfY` (se monádico)	`   N←3 2 5 4 6 1 3`
		`   N/⍨2|N ↔ (2|N)/N`
		`3 5 1 3`
		`   ⍴⍨3 ↔ 3⍴3`
		`3 3 3`

`{X}f&Y`	spawn: f em Y em uma nova thread	
`{X}(f⍤B)Y`	rank: f sobre ou entre subarrays de rank B	

`(f⌺g)Y`	Stencil: movimento de janela
	No exemplo ao lado, cria-se uma janela 2x2 que passeia pela matriz somando-se os elementos de cada sub-matriz
		`   a←3 3⍴⍳12`
		`   ({+/,⍵}⌺ 2 2)a`
		`12 16`
		`24 28`

`{X}(f⍣g)Y`	power: itera `F` (ou `X∘f`) sobre Y até a condição $YgfY$ (ou $YgXfY$) ser verdadeira	
`{X}(f⍣Js)Y`	power: `f (ou X∘f)` sobre Y, Js vezes	

Capítulo 5

Operador I-beam

Existe um capítulo imenso de operações mistas (funções, operadores, funções de sistema, uma miscelânea) todos eles sob operação do operador I-beam (⌶).

5.1 Índice em tabela invertida

```
r ← X (8⌶) Y
```

Retorna os índices de Y em X, sendo que ambos são estruturas compatíveis. Veja no manual APL Language, páginas 198 e 199.

5.2 Execute

```
r ← X (85⌶) Y
```

Executa a expressão Y como a primitiva "execute" (⍎) mas maneja os resultados tímidos da expressão de maneira diferente. X pode ser 0 ou 1. Se for 1 e a expressão em Y retorna resultado explicito, ele vai para r. Se Y não retorna resultado ou retorna resultado tímido, a função sinaliza erro 85.

5.3 Reescreve áreas livres

```
r ← 127 ⌶ Y
```

Normalmente o workspace mantém dados prévios, mesmo após a eliminação de variáveis. Em aplicações sensíveis, a execução desta expressão garante que as áreas livres serão reescritas. Y é um array vazio, preferivelmente zilde (⍬). r é sempre 1.

5.4 Representação canônica

```
r ← 180 ⌶ Y
```

Funciona como ⎕CR, exceto que pode ser usada para obter a representação canônica de métodos em classes. É usada pelo comando]PROFILE.

5.5 Tipo do array

```
r ← 181 ⌶ Y
```

Funciona como o ⎕DR monádico e retorna o tipo de Y. Veja o manual APL Language, pág. 202.

5.6 Coloreando a síntaxe

```
r ← 200 ⌶ Y
```

Este sujeito retorna uma especificação estrutural de uma função. Y é um vetor de caracteres contendo a ⎕NR de uma função. Veja

```
r←isnum x
:If (83=⎕DR x) ⋄ r←1
    →0
:Else ⋄
    r←0
:EndIf
```

Agora o `⎕NR`

```
      ⎕NR'isnum'
 r←isnum x   :If (83=⎕DR x) ⋄ r←1       →0   :Else ⋄       r←0   :EndIf
```

E finalmente o `200 ⍪ Y`

```
      200⍪⎕NR'isnum'
 3 34 19 21 21 21 21 21 3 34  3 119 119 119 3 19 5 5 19 13 13 13 3
 34 19 3 25 3 34 19 5  3 3 3 3 3 19 5  3 123 123 123 123
 123 3 25  3 3 3 3 3 34 19 5  3 124 124 124 124 124 124
```

Aqui, neste exemplo, cada número vale:

21	constante de caracter
19	primitiva
3	espaço em branco
34	nome local
7	nome global
23	idioma
119	:If
...	...

5.7 Tokens da coloração de síntaxe

```
r ← 201 ⍪ Y
```

Obtém os valores completos da tabela acima. Y é zilde. r é uma matriz de 4 colunas, a saber: tipo do token, valor numérico dele, nome interno e a quarta coluna é usada para terminais tipo tty. Vale a pena olhar.

5.8 Compressão de vetor de inteiros short

```
r ← X (219 ⍪) Y
```

O vetor de inteiros deve ser formado por valores `¯128` a `127` (tipados como 83 pelo `⎕DR`. (Veja mais à pag. 354 do APL Language). Em muitos casos, esta funcionalidade é usada junto com a `220 ⍪` (serializar / desserializar arrays).

X especifica a operação, a biblioteca e algum parâmetro opcional. Y contém os dados a serem operados.

5.8.1 Compressão

Y deve ser um vetor de inteiros short. r é um vetor de 2 ítens. ambos são short ints. $r[1]$ descreve a compressão e $r[2]$ contém os dados resultados da aplicação da compressão sobre Y. Eis os valores de X

`X[1]`	`X[2]`	Biblioteca
1	não se aplica	LZ4
2	0..9	zlib
3	0..9	gzip

Se usado LZ4, X deve ser um escalar ou um vetor de 1 elemento. De outra forma, se X[2] é presente ele indica o nível de compressão. Números maiores indicam compressão maior, mas mais demorada.

5.8.2 Descompressão

r é um vetor de inteiros short, contendo o resultado da descompressão aos dados Y.

Os valores de X são

`X[1]`	Biblioteca
`¯1`	LZ4
`¯2`	zlib
`¯3`	gzip

Vamos ver exemplos

```
      oba ← {⍵-256×⍵>127}
```

A função dinâmica `oba` acerta o vetor de inteiros para o intervalo `¯128..127`

```
      utf8←'UTF-8' ∘ ⎕UCS
      utf8 'Pedro'
80 101 100 114 111
       ⎕AV[81]  ⍝ nao esqueça que aqui ⎕IO=0
P
```

O operador utf8, converte números em caracteres UNICODE e vice-versa. Se quiser olhar a definição de `⎕UCS`, veja a pág. 624 do APL Language (vol 2).

```
      nome←'Curitiba Paraná Pedro ⍰'
      ⊣ v ← oba utf8 nome      ⍝ o ⊣ mostra o resultado
67 117 114 105 116 105 98 97 32 80 97 114 97 110 ¯61 ¯95 32 80 101 100 114
111 32 ¯30 ¯116 ¯71
      ⍴v
26
      ⍴nome
23
```

E agora, a mágica

```
       ⊣comp←1 (219⌶) v
 8 ¯55 1 0 0 0 0 26  ¯16 11 67 117 114 105 116 105 98 97 32 80 97 114 97
110 ¯61 ¯95 32 80 101 100 114 111 32 ¯30 ¯116 ¯71
      utf8 256|0 (219⌶) comp
Curitiba Paraná Pedro ⍰
       ⍴comp
2
      ⍴⊃comp[1]
8
      ⍴⊃comp[2]
28
      ⍴nome
23
      comp[1]
 8 ¯55 1 0 0 0 0 26
      comp[2]
 ¯16 11 67 117 114 105 116 105 98 97 32 80 97 114 97 110 ¯61 ¯95 32 80 101
100 114 111 32 ¯30 ¯116 ¯71
```

5.9 Serialização/desserialização de vetor

```
r ← X (220 ⌶) Y
```

usado em conjunto com a funcionalidade anterior. X deve ser 0 (desserializa) ou 1 (serializa). Veja pág. 207 de APL Language.

5.10 Controle do compilador

```
r ← {X} (400 ⌶) Y
```

Veja o *Compiler User Guide*. A compilação em Dyalog serve apenas para acelerar a execução de funções (e não para distribuir software como em outras encarnações de APL). Uma função em Dyalog APL gasta 2 fatias de tempo para ser executada:

- A interpretação do comando da função
- Sua aplicação sobre os dados

A compilação opera apenas sobre a primeira dessas parcelas. E, só tem sentido quando aplicada a escalares ou arrays pequenos. Quando um comando se aplica a um array enorme, a parcela da interpretação é muito pequena, e portanto otimizá-la não tem grande significado. O código compilado é traduzido para um byte code otimizado e guardado junto com a função original no workspace. Ele é manipulado junto quando a função é copiada entre workspaces. Veja como a coisa funciona

```
⎕FX 'r←funcao y' ... ⍝ define a funcao
funcao 100           ⍝ executa interpretadamente
2(400⌶)'funcao'      ⍝ compila a função
funcao 100           ⍝ executa a versão compilada
```

Para perguntar se a função foi compilada adequadamente faça

```
1(400⌶)'funcao'
```

Responde 1 se houve sucesso na compilação.

Para compilar uma função faça

```
2(400⌶)'funcao'
```

Esta função devolve uma matriz de diagnósticos. Se a matriz tiver 0 linhas, deu tudo certo. Senão cada linha descreve um problema havido na compilação. Os itens da linha: o erro APL; o número da linha; o número da coluna (presentemente é 0) e a mensagem de erro. Eis aqui um exemplo Seja a função

```
       ∇media[⎕]∇
[0]   r←media x;a
[1]   a←+/x
[2]   r←a÷⍴,x

        2(400⌶)'media'
       1(400⌶)'media'
1
       4(400⌶)'media'
Dump of bytecode for media:
  0000: 00000012 // version 18
  0001: 00000000 // localised system variables: none
  0002: 00000201 // 2 slots
  0003: 00000002 // 0 uslots
  0004: 00009BC5 cpy PFUNCTION, rawlst[4]
  0005: 00000F46 cpy slot[0], Rarg
  0006: 00000024 eval
  0007: 00002F11 tokoff 002F
  0008: 00000E45 mov Rarg, slot[0]
  0009: 00002E66 mov slot[1], Rslt
  000A: 00002124 eval 0x21 // ,
  000B: 00003711 tokoff 0037
  000C: 00006045 mov Rarg, Rslt
  000D: 00002024 eval 0x20 // ⍴
  000E: 00003611 tokoff 0036
  000F: 00002E25 mov Larg, slot[1]
  0010: 00006045 mov Rarg, Rslt
  0011: 00000524 eval 0x05 // ÷
  0012: 00003511 tokoff 0035
  0013: 00000003 ret
```

Veja mais detalhes no APL Language Reference, vol-1, pág 208 e seguintes.

5.11 Controle de trap

```
r ← 600 ⌶ Y
```

Usada para temporariamente desabilitar os mecanismos de traping de erro usados por `:Trap` e por `⎕TRAP`. Isto pode ser importante em depuração de aplicações. O Y determina o que fazer: 0=habilita todos os traps; 1=desabilita todos os traps e 2=desabilita em funções suspensas.

5.12 Conversão de caixa

```
r ← {X} (819 ⌶) Y
```

Converte os caracters de Y para maiúscula ou minúscula. Esta função foi depreciada (deprecate) e substituída pela função de sistema `⎕C`. Se X é 0, converte para mnúscula e se x é 1 converte para maíuscula.

5.13 Chamada monádica

Identifica como a função corrente foi chamada.

```
r ← 900 ⌶ Y
```

R é booleano. Se a última função na pilha foi monádica, r é 1. Se não há função na pilha ou se ela não foi monádica r é 0.

5.14 Diretório temporário

retorna o nome do diretório temporário para arquivos de usuário. Y é 0.

```
      739⌶0
C:/Users/Pedro/AppData/Local/Temp
```

5.15 Bibliotecas carregadas

```
r ← 950 ⌶ Y
```

reporta o nome da dll que esta correntemente carregada como resultado da execução de ⎕NA (name association). Veja página 214 e depois páginas 436 do APL reference guide.

5.16 Número de threads

Especifica o número de threads na execução paralela.

```
r ← 1111 ⌶ Y
```

Se Y vale ⍬ r contém o número de threads. De outra forma, Y indica o número desejado de threads. O limite é 64. Aqui em casa, eu fiz

```
      1111⌶⍬
12
```

5.17 Limite para execução em paralelo

```
r ← 1112 ⌶ Y
```

Y é um inteiro que estabelece o tamanho limite para arrays. Se uma função que permite processamento em paralelo é chamada e o número de elementos do array é maior do que este limite, o processamento ocorre em paralelo. Se o valor não é alterado seu *default* é 32.768. r é o valor prévio.

5.18 Atualização do carimbo de tempo

```
{R} ← X ( 1159 ⌶) Y
```

Y é um array de nomes de funções como o argumento direito de ⎕AT. X é um array de atributos no mesmo formato da saída de ⎕AT. r informa se a troca foi feita ou não: 0=não foi feita, o nome não é uma função ou ela está trancada (locked). 1=o carimbo de tempo e de usuário foi atualizado.

5.19 Formato de data-tempo

```
r ← X ( 1200 ⌶ ) Y
```

Y é um array de qualquer forma, onde cada elemento contém uma data Dyalog entre 1 de janeiro de 0001 e 28 de fevereiro de 4000 no calendário Proléptico Gregoriano. X é um escalar ou vetor de caracteres especificando o padrão pelo qual Y deve ser formatado. Tem páginas e páginas de explicação (veja pags. 217 a 225 do APL reference (vol 1)). Um exemplo

```
      t←1 ⎕DT ⊂ 2020 7 17 11 10 0
      t
44028.46528
      'Dddd DDoo Mmmm YYYY; hh:mm:ss' (1200⌶) t
 Friday 17th July 2020; 11:10:00
```

 ©75-20, Pedro Kantek

5.20 Array de hash

Cria um array de hash, retorna uma cópia *unhashed* de um array ou relata o estado de hash de um array.

```
r ← {X} 1500 ⌶ T
```

Y pode ser qualquer array. Se X é omitido, r é uma cópia de Y invisivelmente marcada como hash. r funciona como Y em qualquer aspecto. A única diferença é que `⍳` diádica e relacionados funciona mais rápido em r do que em Y. Se X é 1, r informa: $r = 0$ Y não foi marcada para hashing; $r = 1$ Y foi marcada para hashing, mas a tabela hash ainda não foi criada. $r = 2$ informa que Y é uma tabela hash.

5.21 Estatísticas de gerenciamento de memória

```
r ← {X} ( 2000 ⌶ ) Y
```

Retorna informações sobre alocação de memória. Veja na pág. 228. Eis um exemplo

```
      2000⌶0   ⍝ pergunta o ⎕WA
208252184
      2000⌶1   ⍝ pergunta quanto já foi usado no ws
1463016
```

5.22 Especifica o espaço disponível

```
r ← 2002 ⌶ Y
```

Funciona como `⎕WA`, mas é o mescanismo para especificar a quantidade de memória *committed*. Veja a pág. 228 e 231.

5.23 Desabilitar gatilhos globais

```
r ← 2007 ⌶ Y
```

Temporariamente desabilita e re-habilita os gatilhos globais. Se $Y = 0$, desabilita e se $Y = 1$ habilita-os.

5.24 Demais I-beam codes

No manual tem uma série deles, mas acho meio irrelevante aqui. A maioria é para código .NET, veja-os lá.

Capítulo 6

Funções do usuário

Existem pelo menos 5 maneiras de criar uma função do usuário dentro de um workspace DAPL, a saber:

1. Estabelecendo ⎕FX de uma matriz de caracteres que seja adequada para conter uma função, possivelmente vindo de uma chamada a ⎕CR.

2. Usando o editor de caracteres. Este uso está bem descontinuado e só aparece aqui por compatibilidade com o passado, já que usa um protocolo caractere a caractere, herança dos antigos terminais de teletipo. Lembrando, durante muitos anos esta foi a única maneira de criar e alterar programas. Ainda hoje, às vezes é usada (talvez por saudosismo) para fazer uma pequena alteração em um comando. Entra-se (e saí-se) nesta modalidade teclando ∇.

3. Método MINI: Define-se a função escrevendo as operações sem nenhum operando na ordem em que elas vão aparecer. Veja

```
      media2←(+/)÷⍴
      media2 1 2 3 4
2.5
```

4. Método MIDI: Este método recebe o nome de **dfn**=função dinâmica dyalog. É uma expressão APL entre chaves podendo ou não ter nome e possivelmente usando ⍺ como argumento esquerdo e ⍵ como argumento direito. veja no primeiro caso sem nome e no segundo com nome (=media2):

```
      {(+/⍵)÷⍴⍵} 1 2 3 4
2.5
      media2←{(+/⍵)÷⍴⍵}
      media2 1 2 3 4
2.5
```

   Uma dfn pode ser definida:

   - para execução imediata: no meio da sessão escreve-se abre e fecha chave e segue o baile
   - dentro de uma função ou operador
   - dentro do operador each (¨)
   - dentro de outra dfn, o que permite funções locais aninhadas.

5. Método MAXI: o jeito padrão por excelência. Chamando o editor de funções e definindo nele a função completa e por extenso. A chamada se dá por)ed nome ou clicando duas vezes no nome da função, ou ainda chamando ⎕ed 'nome'. Veja o mesmo exemplo

```
    ∇ r←media x
[1] r←(+/x)÷⍴x ∇
```

6.1 Mais sobre dfn

Uma função direta (dfn, pronunciada como *dee fun*) é uma maneira alternativa de definir uma função. A mesma idéia se aplica à definição de um operador, aqui chamado de dop (pronunciado como *dee op*). Foram inventados por John Scholes em 1996.

São funções escritas entre chaves ({ e }) e separadas por nova-linha ou eventualmente por ⋄ (diamante). Nesta especificação alfa (⍺), indica o operando esquerdo da dfn enquanto ômega (⍵), indica o operando direito da dfn. Auto-referências, necessárias para chamar a si próprio em casos de recursividade usam o símbolo del (∇) como o nome desta dfn. Veja-se um exemplo interessante de dfn

```
   PT ←  { ( + / ⍵ * 2 ) = 2 × ( ⌈ / ⍵ ) * 2 }
   PT  3  4  5
1
   x
 4   5   3
 3  11   6
 5  13  12
17  16   8
11  12   4
17  15   8
   PT  x
1  0  1  0  0  1
```

Esta dfn acima verifica se 3 números fazem parte de um trio pitagórico (lados de um triângulo retângulo). A lógica é meio óbvia, mas por seu sincretismo vale alguma explicação. O código começa elevando os 3 números ao quadrado e somando este resultado, isto tudo dentro do parêntesis. A seguir, a função determina o maior valor (supostamente a hipotenusa) e eleva-a ao quadrado. Como este valor foi igualmente somado ao primeiro termo, aqui ele é multiplicado por 2 para equilibrar. Havendo igualdade a trinca é pitagórica. Senão não. Como as 2 reduções não têm indicação de eixo, elas se aplicam ao último eixo. Isto permite chamar esta função com um array, como no segundo caso do exemplo.

Mais um exemplo, agora recursivo, do cálculo do fatorial

```
   fact← {0=⍵:1 ⋄ ⍵×∇ ⍵-1}
   fact 5
120
   fact¨ ⍳10    ⍝ fact é aplicado a cada elemento de 0 a 9
1 1 2 6 24 120 720 5040 40320 362880
```

Uma dfn é formada por um conjunto de expressões, expressões guardadas ou apenas guardas. Uma guarda é uma condição que retorna 0 ou 1. A guarda é avaliada e se o resultado for 1, a expressão associada é executada. A dfn termina após a primeira expressão não guardada que não seja um assinalamento (←) OU após a primeira expressão guardada que tenha respondido 1 OU quando não há mais expressões.

O resultado de uma dfn é o resultado da última expressão avaliada. Se a última expressão avaliada termina em um assinalamento, o resultado é tímido (*shy*) ou seja não é mostrado na sessão. (Mas vai para uma variável se isto estiver no código).

Todas as variáveis citadas na dfn são locais a ela, e não há risco de conflito de nomes com variáveis globais. Há uma atribuição especial ⍺←expressão que só é executada se a dfn for chamada monadicamente. Se ela for chamada com 2 operandos, este comando não é executado. Este caso sinaliza o valor *default* para a variável esquerda.

6.2 Operador direto

Usa mais ou menos as mesmas regras, mas agora o operando esquerdo é alfa-alfa (⍺⍺) enquanto o direito é ômega-ômega (⍵⍵). Já a auto-chamada na recursão é representada por del-del (∇∇). Isto tudo nos permite escrever o seguinte resumo

{⍺ função ⍵}	{⍺⍺ operador ⍵⍵}	:guarda
⍺ argumento esquerdo	⍺⍺ operando esquerdo	:: erro-guarda
⍵ argumento direito	⍵⍵ operando direito	⍺←arg esq default
∇ auto-referencia	∇∇ auto-referencia	s←resultado tímido

6.3 Guarda de erro

O exemplo seguinte mostra como manipular os códigos de erro associados. Veja o exemplo

```
mais←{
  tx←'catch all' ⋄  0::tx
  tx←'domain'    ⋄ 11::tx
  tx←'length'    ⋄  5::tx
  ⍺+⍵
}
   2 mais 3              ⍝ sem erro
5
```

```
   2 3 4 5 mais 'treis'  ⍝ tamanhos dos argumentos não combinam
length
   2 3 4 5 mais 'tres'   ⍝ não se somam letras a números
domain
   2 3 mais 3 4⍴5        ⍝ não dá para somar vetor e matriz
catch all
```

Em APL o erro número 5 é ”length error”. O erro número 11 é ”domain error”e o erro número 0 é qualquer erro entre 1 e 999.

O exemplo mostra como estabelecer o valor de um erro de guarda. O nome `tx` é local e apenas retêm o erro correto.

6.4 Funções trad

Este é o nome de funções definidas as usual, usando um editor. Eu chamei-as há pouco de funções MAXI. As diferenças que interessam no caso:

- Uma dfn pode ser anônima, uma tradfn sempre tem nome.
- Uma dfn é nomeada pelo assinalamento. Uma tradfn têm seu nome estabelecido na primeira linha de sua definição.
- Os nomes em uma dfn são locais. Em uma tradfn são globais, a menos que tenham sido citados no header (primeira linha) da definição.
- Uma dfn pode chamar uma tradfn e vice-versa. Uma dfn pode ser definida em uma tradfn e vice-versa.

O Dyalog APL chama estas funções de funções procedurais ou funções tradicionais, ou abreviadamente trad-funs.

6.5 Editor

O Dyalog tem um editor built-in usado para editar objetos. Sua chamada em

```
)ed coisa
```

edita a FUNÇÃO de nome `coisa`. Mas, o mesmo editor pode ser chamado para editar outras coisas, a depender de um caracter escrito entre `)ED` e o nome da coisa. Acompanhe na tabela

caracter	edita o que ?
nada	função
∇	função
-	matriz de texto
→	vetor de texto
∈	vetor de vetores de texto (1 sub-vetor por linha)

 ©75-20, Pedro Kantek

Capítulo 7

Funções do sistema

Este capítulo é extenso, vamos abreviar.

função	significado	detalhes e exemplo
⎕	entrada e saída de caracteres	⎕←'2+2' ◇ ⎕←'=' ◇ ⎕←4 2+2=4
⍞	entrada e saída do APL	ao contrário do APL2 serve para alfa.
⎕A	caracteres alfabéticos	⎕A ABCDEFGHIJKLMNOPQRSTUVWXYZ
⎕AI	informações de contabilização	[1]=identificação do usuário; [2]=tempo de computador em milésimos de segundo; [3]=tempo de conexão; [4]=tempo de digitação. Aqui em casa: ⎕AI 0 7203 9682597 9378999
⎕AN	nome do usuário	⎕AN Pedro
⎕ARBIN	Input arbitrário	pág. 298
⎕ARBOUT	Output arbitrário	pág. 300
⎕AT	atributos	Informa atributos do objeto (tipo de resultado e valência)
⎕AV	Vetor atômico	Depreciado e trocado por ⎕UCS
⎕AVU	Vetor atômico Unicode	Vetor de inteiros com 256 elementos com os codepoints dos caracteres em ⎕AV
⎕BASE.Y	Base Class	Informa a classe base do nome Y
{X}⎕C Y	Conversão de caso	Se $X = 1$ devolve Y em maiúsculo. Se $X = ^{-}1$ em minúsculo.
{X}⎕CLASS Y	Referências a classes e interfaces	Se monádico retorna a hierarquia de classes de Y. Se diádico, Y é uma classe e X é uma referência a uma interface
⎕CLEAR	Clear	Limpa o workspace e dá-lhe o nome de CLEAR WS
{R}←⎕CMD Y	Executa comando	Z←⎕CMD'dir' ↑Z' ...segue o dir...
{R}←X ⎕CMD Y	Inicia processador auxiliar	Sinônimo de ⎕SH. X é o nome do processador auxiliar. Y é ignorado em windows.
⎕CR Y	Representação canônica	da função Y
{R}←{X}⎕CS Y	Change Space	Y é a referencia a um nomespace. X é o conjunto de objetos a exportar para o namespace.
{R}←{X}⎕CSV Y	Valores separados por vírgula	Importa e exporta valores separados por vírgula. Se monádico Y especifica a origem dos valores. Se diádico X contém os valores a converter a CSV. Y neste caso é o destino.
⎕CT	Tolerância de comparação	Default é 10^{-14}
{R}←{X}⎕CY Y	Copia de workspace	Y é o nome do workspace salvo. X é opcional e pode identificar nomes APL do workspace.
⎕D	Dígitos	vetor de caracteres: '0123456789'
⎕DCT	Tolerância de comparação decimal	Usado quando ⎕FR é 1287. Nos demais casos usa-se ⎕CT
{R}←⎕DF Y	Forma de display	Estabelece como mostrar um namespace, objeto GUI, instância ou classe

`⎕DIV`	Tratamento da divisão por zero	Se 0, a divisão por zero produz erro de domínio, exceto em `0÷0` que vale 1. Se igual a 1, a divisão por 0 retorna 0. O default é 0.
`{R}←⎕DL Y`	Atraso	Pausa de Y segundos.
`⎕DM`	Mensagem de diagnóstico	vetor de 3 elementos: a mensagem, a linha errada e a posição do caret
`⎕DMX`	Mensagem de diagnóstico extendido	Inclui informações de threads
`{R}←⎕DQ Y`	aguarda e processa eventos.	Uso em objetos GUI
`{X}⎕DR Y`	Representação do dado	Se monádico, retorna o tipo do argumento Y. Se diádico converte Y no tipo especuificado por X
`X ⎕DT Y`	Data e tempo	Valida ou converte data e tempo entre formatos
`{R}←{X} ⎕ED Y`	Edita objeto	Invoca o editor. Y é o objeto a editar e o opcional X é o que o editado deve virar.

`⎕EM Y`	Mensagem de evento	Converte um código em mensagem de erro
		`⎕EM 11`
		`DOMAIN ERROR`

`⎕EN`	Evento número	Retorna o número do último erro ou interrupção
`⎕EXCEPTION`	Exception	Retorna a última exceção de um objeto NET
`{R}←⎕EX Y`	Expunge	Elimina o objeto Y
`{R}←{X}⎕EXPORT Y`	Exportação de objeto	Exporta funções ou operadores
`{R}←X ⎕APPEND Y`	Component file append	Acrescenta X no arquivo de número (tie) Y
`⎕FAVAIL`	File system available	Se retornar 1 é porque o componente de File System está disponível. Se voltar 0, não está
`{X} ⎕FCHK Y`	File check & repair	valida e repara arquivos de componentes de número Y
`X ⎕FCOPY Y`	Cópia de arquivo	Cria uma cópia do arquivo de número Y com o nome X
`{R}←X ⎕CREATE Y`	Cria o arquivo de nome X com o número Y	
`{R}←⎕FDROP Y`	Elimina arquivo	de número Y
`{R}←X ⎕FERASE Y`	Elimina arquivo	de número Y
`⎕FHIST Y`	História do arquivo	Indica data e hora dos principais eventos do arquivo
`{R}←⎕FHOLD Y`	Hold de arquivo ou variável externa	Para processamento sincronizado
`{R}←{X}⎕FIX Y`	Fixa script	Estabelece namespaces, interfaces e funções do script Y
`⎕FLIB Y`	Biblioteca de arquivos componentes	No diretório Y, mostra os arquivos
`{X}⎕FMT Y`	Formatação	Transforma numérico em caracter. págs 388-394
`⎕FNAMES`	File names	Retorna os nomes dos arquivos abertos (tied)
`⎕FNUMS`	File numbers	Retorna os números dos arquivos abertos (tied)
`X ⎕FPROPS Y`	File properties	Retorna as propriedades X do arquivo número Y
`⎕FR`	Representação em ponto flutuante	O valor de `⎕FR` determina como são executadas as oprações de ponto flutuante
`⎕FRDAC Y`	Acesso de leitura a arquivo	Devolve a matriz de acesso do arquivo de número Y
`⎕FRDCI Y`	Informação de arquivo de leitura	Informa tamanho, último leitor e última leitura do arquivo
`⎕FREAD Y`	Leitura	Lê o registro $Y[2]$ do arquivo de número $Y[1]$
`X ⎕FRENAME Y`	Renomear	Renomeia o arquivo de número Y (tied) para o nome X
`{R}←X ⎕FREPLACE Y`	Substituição de registro	Troca o registro $Y[2]$ do arquivo de número $Y[1]$ para o conteúdo X
`{R}←{X} ⎕FRSIZE Y`	Troca de tamanho	Troca o tamanho do arquivo de número Y. X determina o tamanho máximo em bytes. O valor 0 indica o máximo possível. Esta operação também compacta o arquivo
`⎕FSIZE Y`	Dados do arquivo	Informa: o número do primeiro componente; o número (+1) do último, o tamanho atual e o limite do arquivo
`X ⎕FSTAC Y`	Estabelece Acesso	Usando uma matriz válida em X estabelece o acesso a Y

`{R}←X ⎕FSTIE Y`	Compartilhamento e abertura	Associa o arquivo de nome X ao número Y. Exemplo `'SALES' ⎕FSTIE 1`
`{R}←X ⎕FTIE Y`	Abertura exclusiva	Abre e associa o arquivo X ao número Y. Exemplo `'SALES' ⎕FTIE 1`
`{R}←⎕FUNTIE Y`	Fechamento	Fecha o arquivo. Se $Y = 0$, fecha tudo
`{R}←⎕FX Y`	Estabelece (fixa)	o objeto Y
`⎕INSTANCES Y`	Instâncias	Retorna uma lista das instâncias da classe Y
`⎕IO`	Origem dos índices	Pode ser 0 ou 1
`{X}⎕JSON Y`	Dados JSON	Importa e exporta dados neste formato
`⎕LC`	Line count	Vetor de números informando o indicador de estado
`⎕LOAD Y`	Load de ws	Carrega o workspace salvo como Y
`{R}←{X}⎕LOCK Y`	Lock	Tranca a função ou operador. X indica em que condições...
`⎕LX`	Expressão latente	O que é executado automaticamente quando o workspace é carregado
`{X}⎕MAP Y`	Map File	Associa um arquivo mapeado com um array APL
`{R}←{X} ⎕MKDIR Y`	Make directory	Cria um diretório de nome Y de acordo com indicações de X
`⎕ML`	Migartion level	Determina o grau de migração entre este APL e o APL2 da IBM
`{R}←X ⎕MONITOR Y`	Monitoramento	X identifica as linhas da função/operador Y que devem ser monitoradas. A resposta é a acumulação de estatísticas de tempo para estas linhas. Se $X = 0$, cancela tudo. Se monádico, a resposta é para toda a função. Pág. 435
`{R}←{X}⎕NA Y`	Associação de nome	Permite o APL acessar funções compiladas dentro de uma dll. Págs 436-462
`{R}←X ⎕NAPPEND Y`	Append em arquivo nativo	$Y[1]$ é um inteiro negativo do arquivo. X é o conteúdo e $Y[2]$ indica qual o tipo de conversão de X antes de gravar
`⎕NC Y`	Classificação de nome	Y é classificado: -1:inválido; 0:não usado; 1:label; 2:variável; 3:função; 4:operador; 8:evento; 9:objeto
`{R}←X ⎕NCOPY Y`	Copia de arquivo nativo	Copia o arquivo de nome Y para o local X
`{R}←X ⎕NCREATE Y`	Cria arquivo nativo	Y é o número negativo pelo qual será conhecido. Se for 0, o sistema escolhe e devolve em R. X é o nome
`{R}←{X} ⎕NDELETE Y`	Elimina arquivo nativo	Elimina o arquivo Y
`{R}←X ⎕NERASE Y`	Apaga arquivo nativo	Y é o número negativo e X o nome igual a quando ele foi criado/aberto
`⎕NEW Y`	Cria instância	de classe, diálogo com objeto GUI ou tipo NET especificado por Y
`⎕NEXISTS Y`	Existe ?	Responde se o arquivo de nome Y existe ou não
`{X} ⎕NGET Y`	Lê arquivo de texto	X pode ser 'ASCII', 'UTF-8', entre outros e Y tem o nome e o flag. Se 0, R é um vetor de caracteres. Se é 1, R é um nested array. Veja o exemplo `aux ← ⎕NGET (arq) 1` `r←↑⊃aux[1]`
`{X} ⎕NINFO Y`	Informação de arquivo nativo	Y pode ser número ou nome
`{X} ⎕NL Y`	Name List	Informa quais os nomes do tipo Y possivelmente na ordem X
`{R}←X ⎕NLOCK Y`	Lock de arquivo	controla a atualização de um intervalo de bytes no arquivo. X diz o que fazer no arquivo $Y[1]$ a partir do byte $Y[2]$ pelo tamanho $Y[3]$
`X ⎕NMOVE Y`	Move arquivo	Similar a `⎕NCOPY` mas apaga na origem
`⎕NNAMES`	Nomes de arquivos nativos	Retorna nomes de arquivos nativos
`⎕NNUMS`	Números de arquivos nativos	Retorna números de arquivos nativos
`{X} ⎕NPARTS Y`	Nomes parciais	Y é um ou mais nomes de acordo com o sistema operacional. X indica se os nomes devem ser normalizados. $R[1]$ é o path; $R[2]$ é o nome base e $R[3]$ é a extensão
`{R}←X ⎕NPUT Y`	Escreva em arquivo de texto	$Y[1]$ é o nome do arquivo. $Y[2]$ é o flag: 0 se o arquivo já existe dá erro. 1 se o arquivo já existe é sobreescrito e 2 se o arquivo já existe ele é apendado. Exemplo `LF←⎕UCS 10` `(,a,LF) ⎕NPUT (arq) 1`
`{R}←{X} ⎕NQ Y`	Desenfileira evento	
`⎕NR Y`	Representação aninhada	Mostra a representação nested da função ou operador

©75-20, Pedro Kantek

`⎕NREAD Y`	Leitura arquivo nativo	$Y[1]$=número negativo do aqruivo; $Y[2]$=código de conversão; $Y[3]$=contador e $Y[4]$=início. Se omitido ou -1 o dado é lido da posição corrente
`X ⎕NRENAME Y`	Renomeia	Troca o nome do arquivo número negativo Y pelo nome X
`{R}←X ⎕NREPLACE Y`	Troca	O dado X é posto em Y, onde [1]=número negativo do arquivo; $Y[2]$=start byte. -1 aqui indica posição atual e $Y[3]$=conversão
`{R}←X ⎕NRSIZE Y`	Resize	Troca o tamanho do arquivo número Y para o tamanho X
`{R}←{X}⎕NS Y`	Namespace	
`⎕NSI`	Namespace indicator	
`⎕NSIZE Y`	Tamanho	Informa o tamanho do arquivo de número negativo Y
`X ⎕NTIE Y`	Abertura de arquivo nativo	X é o nome do arquivo. $Y[1]$ é o número negativo que ele deverá ter. $Y[2]$ é opcional e indica 0=leitura, 1=gravação e 2=ambos
`⎕NULL`	Nulo	Valor nulo
`{R}←⎕NUNTIE Y`	Fechamento	Fecha o arquivo negativo Y
`{R}←{X} ⎕NXLATE Y`	Translate de arquivo nativo	
`⎕OFF`Desliga	a sessão APL	
`⎕OR Y`	representação do objeto	Y
`⎕PATH`	Caminho	Mais ou menos análogo à variável PATH do windows
`{X}⎕PFKEY Y`	Tecla de função	Exemplo:`(')RESET',⊂'ER') ⎕PFKEY 2`
`⎕PP`	Precisão de impressão	Informa o número de decimais na saída, com 1..34
`{R}←{X} ⎕PROFILE Y`	Profile aplication	Págs 537-543
`⎕PW`	Print width	Tamanho da linha de saída
`{X}(A ⎕R B) Y`	Replace	Y é a entrada. Troca A por B. Págs 548-568
`{X}(A ⎕S B) Y`	Search	Págs. 549-568
`⎕REFS Y`	Referência cruzada	Y é uma função ou operador e a resposta é uma lista de nomes
`⎕RL Y`	random link	Valor inicial da randomização
`⎕RSI`	Indicador de espaço	
`⎕RTL`	Limite de timeout	Segundos de `⍞`, `⎕ARBIN` e `⎕SR`
`{R}←{X}⎕SAVE Y`	save workspace	Y é o nome do ws a salvar. Se X é 0, o)SI não é salvo, e se é 1 sim
`⎕SD`	Screen dimensions	2 inteiros com as linhas e colunas da tela
`⎕SE`	Namespace da sessão	
`{R}←⎕SHADOW Y`	Nome sombra	
`⎕SI`	State indicator	indicador de estado
`{R}←{X} ⎕SIGNAL Y`	Sinal de evento	
`⎕SIZE Y`	Tamanho de objeto	
`⎕SM`	Screen map	
`{X}⎕SR Y`	Screen read	
`⎕SRC Y`	Source	
`⎕STACK`	Pilha de SI	
`⎕STATE Y`	Estado de um objeto	
`{X} ⎕STOP Y`	Parada	X são as linhas de Y onde a execução deve parar. Se monádico estou perguntando em quais linhas ele para
`X ⎕SVC Y`	Set Access control	
`⎕SVC Y`	Query access control	
`{X} ⎕SVO Y`	Shared variable offer	Se monádico, estou perguntando o grau de acoplamento
`⎕SVQ Y`	Shared variable query	
`⎕SVR Y`	Shared variable retract offer	
`⎕SVS Y`	shared variable state	
`⎕TC`	Terminal control	depreciado. Substituir por $TC[1] = UCS[8]$, $TC[2] = UCS[10]$ e $TC[3] = UCS[13]$
`⎕TCNUMS Y`	thread child numbers	
`{R}←{X}⎕TGET Y`	Get tokens	
`⎕THIS`	This space	
`⎕TID`	Current thread identity	

{R}←{X} □TKILL Y	kill thread	
□TNAME	Current thread name	
□TNUMS	Thread numbers	
□TPOOL	Token pool	
{R}←{X}□TPUT Y	Put tokens	
{R}←{X} □TRACE Y	Estabelecer trace	Mostre as linhas X da função Y. Se monádico estou perguntando
□TRAP	Trap evento	
□TREQ Y	Token request	
□TS	Time stamp	ano, mês, dia, hora, minuto, segundo e milissegundo
□TSINC Y	Espera para a thread terminar	
{X}□UCS Y	Conversão unicode	Converte caracteres Unicode em inteiros e vioce-versa. Usada monadicamente converte caracteres Unicode em codepoints e vice versa. Se X é UTF-8 a tradução é entre caractres unicode e os bytes do caso
□VR Y	Representação em vetor	da função ou operador Y
{X}□VFI Y	verificação e fix da entrada	Converte e fixa números
□WA	Tamanho do workspace	Padrão aqui é 200MB
{R}←{X}□WC Y	Janela de criação de objeto	Para uso em GUI
{X}□WG Y	Obtenção de propriedades da janela	
{X}□WN Y	Nomes das janelas filho	
{R}←{X}□WS Y	estabelecimento de propriedades da janela	
□WSID	Nome do workspace	
□WX	Exposição de janela	
{X} □XML Y	Converte string XML em array APL	
□XSI	Indicador de estado extendido	
{X}□XT Y	Seta variável externa	Se monádico estou perguntando

 ©75-20, Pedro Kantek

Capítulo 8

SALT

Introduzido a partir da versão 11 do Dyalog, o SALT (Simple APL Libray Toolkit) permite que coisas (funções, operadores, variáveis... APL) sejam guardadas na forma de arquivos planos Unicode e possam ser carregados e usados em uma sessão APL. O que tecnicamente é chamado na indústria de *script*. As vantagens deste procedimento são evidentes

- Sistemas APL podem ser desenvolvidos usando qualquer editor.
- Podem ser armazenados em bibliotecas como arquivos de texto e podem ser lidos e mantidos de maneira independente.
- Podem ser enviados e recebidos por e-mail.
- Podem ser publicados e descarregados em páginas web.
- São carregados no APL quando necessários.
- Podem ser centralizados e geridos por um bibliotecário.
- Pode-se assinalar uma versão a cada módulo, seguindo-se uma gerência de versão adequada.

Vamos por partes (Como disse o Jack, aquele estripador)

8.1 Criando o arquivo de script

Usando um editor, que obviamente deve ter suporte a Unicode, deve-se criar um arquivo cuja extensão deve ser `.dyalog`. É possivel usar outra extensão, mas daí ela precisará ser explicitamente declarada no `load`. Mas, não é uma boa idéia.

Este arquivo precisa iniciar por uma de 3 cláusulas, a saber `:Namespace`, `:Class` ou `:Interface` e deve encerrar por `:EndNamespace`, `:EndClass` ou `:EndInterface` respectivamente.

Logo após esta cláusula vem o nome interno do arquivo que será utilizado mais tarde como qualificador das coisas aqui definidas. Há também o nome externo, que será aquele do sistema operacional (antes da extensão `.dyalog` acima citada), Nada impede que ambos nomes sejam o mesmo. Vamos fazer isto aqui, chamando o arquivo de `trevas.dyalog`.

Assim, este arquivo é

```
:Namespace trevas
∇ r←media v;s
s←+/,v
r←s÷⍴,v
∇
primos←{1↓(~v∈1 1↓v∘.×v)/v←⍳⍵}
hoje ← 22 7 2020
:EndNamespace
```

e ele será salvo como `trevas.dyalog`. Note que neste arquivo definiram-se 1 função direta de nome `primos`, uma função tradicional de nome `media` e uma variável de nome `hoje`. Se quiser inserir os números de linha na função tradicional pode fazê-lo, mas eles serão descartados mais a frente.

8.2 Carga Posterior

Tudo começa fazendo um

```
      )clear
clear ws
```

Para termos certeza de que nenhum fiapo solto vai ficar... Agora, cria-se uma variável qualquer contendo o caminho até o arquivo SALT, como em

```
Path←'c:\p\dapl2\'
```

e finalmente, usando a função de sistema ⎕SE que atribui um nome ao namespace da sessão atual.

```
      ⎕←⎕SE.SALT.Load Path,'trevas'
#.trevas
```

As referências futuras ao conteúdo do arquivo SALT são feitas apendando no início o nome `trevas` como em

```
       trevas.primos 100
2 3 5 7 11 13 17 19 23 29 31 37 41 43 47 53 59 61 67 71 73 79 83 89 97
```

ou calculando alguma média

```
      trevas.media 1 2 3 4 5 6
3.5
```

Ou simplesmente perguntando que dia é hoje

```
      trevas.hoje
22 7 2020
```

É meio óbvio, que havendo um problema (erro) qualquer no código, volta-se ao editor unicode, altera-se o arquivo, salva-se o mesmo, recarrega-se o arquivo SALT e segue o baile. Se você clicar 2 vezes no nome da função que deu erro, o arquivo original será reaberto e se você tiver autoridade (questão do sistema operacional) poderá atualizar e salvar o arquivo.

A qualificação dos nomes no namespace é adequada para impedir conflitos neste workspace. Assim, nada impede (embora não seja uma boa idéia) criar uma função local com o mesmo nome como em

```
      ∇primos
[1]  33,55,66∇
```

E, a seguir

```
      primos
33 55 66
      trevas.primos 10
2 3 5 7
```

Capítulo 9

Comandos de usuário

É uma facilidade (junto com a idéia de SALT = arquivos plenos Unicode) de desenvolver comandos que funcionam como os comandos `)xxxx` mas que são programados diretamente pelo usuário. A diferença entre uns e outros é que o caracter inicial é um "fecha colchetes" e portanto tais comandos têm a cara de `]xxxx`. Não podem ser emitidos dentro de funções do usuário e o Dyalog apresenta a seguinte lista de comandos prontos

```
ARRAY               User commands that relate to arrays or variables:
  Compare            Compare two variables/arrays
  Edit               Edit a simple or mixed array of enclosures using
                     the Array Editor from davidliebtag.com

 CALC                User commands that relate to manipulation of data:
  Factors            Determine the prime factors of the argument
  FromHex            Convert a list of hexadecimal representations of
                     integers to a numeric vector
  PivotTable         Create a pivot table from an appropriate matrix
  ToHex              Convert integer(s) to a vector of text vectors
                     containing the hexadecimal representation of each
                     number

 DEVOPS              User commands that relate to development and operations:
  DBuild             Run one or more DyalogBuild script files (.dyalogbuild)
  DTest              Run (a selection of) functions named test_* from a
                     namespace, file or directory

 EXPERIMENTAL:
  Config             Display value of configuration parameters (optionally
                     with origin)

 FILE                User commands that relate to files:
  CD                 Report (and, optionally, change) the current directory
  Collect            Merge all the files with the specified path/name{-nnn}
                     into a single file
  Compare            Compare two component files
  Edit               Open the specified native file as an editable text file
  Find               List line numbers and lines that contain the specified
                     string for each file that has a match
  Open               Open a file or directory with the specified application
  Replace            Replace strings in files and return the number of changes
                     made
  Split              Split a single file into (up to 999) smaller files
  ToLarge            Transform component files in the specified directory
                     from small-span to large-span
  ToQuadTS           Convert a component file timestamp (single float number)
                     to ⎕TS format (vector of 7 numbers)
  Touch              Check whether the specified file exists in the current/specified
                     location and create it if it cannot be found
```

```
FN                 User commands that relate to functions and operators:
 Align             Align end-of-line comments
 Calls             Produce the calling tree of a function in a
                   class/namespace/scriptfile
 Compare           Compare two functions
 Defs              List single-line dfns, dops, derived functions and trains
 DInput            Define a multi-line dfn, dop or derived function/operator,
                   or execute a multi-line expression
 Latest            List functions modified since a specified date (default
                   is today), most-recently-changed first
 ReorderLocals     Sort local names in the header of tradfns and tradops

LINK               User commands for namespace-directory synchronisation
                      (see https://github.com/dyalog/link/wiki):
 Add               Associate item in linked namespace with new file/directory
                   in corresponding directory
 Break             Break link between namespace and corresponding directory
 Create            Link a namespace with a directory (create one or both if absent)
 Export            Export a namespace to a directory (create the directory if
                   absent); does not create a link
 Expunge           Erase item and associated file
 GetFileName       Return name of file associated with item
 GetItemName       Return name of item associated with file
 Import            Import a namespace from a directory (create the namespace
                   if absent); does not create a link
 List              List active namespace-directory links
 Refresh           Fully synchronise namespace-directory content

MSWIN              User commands that relate to the Microsoft Windows operating system:
 Assemblies        List all .NET assemblies currently loaded into memory
 Caption           Query or change window captions
 CopyReg           Copy registry entries to file and, optionally, between
                   Dyalog versions
 FileAssociations  Change the file associations of Dyalog workspaces, applications
                   and sources (.dws, .dyapp, .apl? .dyalog files)
 GUIProps          List property names and values for given or current object
 KeyPress          Return message arguments of KeyPress events

NS                 User commands that relate to namespaces:
 ScriptUpdate      Synchronise namespace/class script to match current content
 Summary           Summarise (scope, size, syntax) the functions in a
                   namespace/class/scriptfile
 Xref              Describe the inter-object cross-references in a
                   class/namespace/scriptfile

OUTPUT             User commands that affect the way items are displayed:
 Box               Display output with borders indicating shape, type and structure
 Boxing            Display output with borders indicating shape, type and structure
 Disp              Display specified array with borders indicating sub-array
                   shape and type
 Display           Display specified array with borders indicating array and
                   sub-array shape and type
 Find              Precede output with a reference to the line of code that
                   generated it
 Format            Format text into vector of text vectors (or matrix if called
                   from Session) to fit ⎕PW using specified  margins
 HTML              Render HTML or SVG
 Layout            Format text into vector of text vectors (or matrix if
                   called from Session) to fit ⎕PW using margins inferred from the text
 Plot              Plot data
 Rows              Cut, wrap, fold or extend the display of output lines to
                   fit the Session window
```

```
PERFORMANCE        User commands that relate to consumption of computing resources:
 Profile           Report performance details (optionally through GUI)
 RunTime           Report execution time of one or more expressions
 SpaceNeeded       Compute memory needed to run expression(s)

SALT               User command covers for SALT functionality:
 Boot              Boot from a file containing instructions or a function
 Clean             Remove SALT tags from the ws or specific items so that
                   SALT no longer saves changes to associate files
 Compare           Compare two versions of a SALTed item
 List              List files (default: .dyalog only) and directories in
                   the specified directory
 Load              Load item from native text file
 Refresh           Reload all SALTed items from their associated files
 RemoveVersions    Remove one or more versions of a file managed by SALT
 Save              Save item in a native text file (default: same place
                   if already SALTed)
 Set               Return one or all parameters or set one parameter
 Settings          Return one or all parameters or set one parameter
 Snap              Save all new or modified items to native text files
                   (unscripted namespaces become directories)

TOOLS              Development and presentation user commands:
 ADoc              Automated documentation generation
 Calendar          Display calendar
 Chart             Run the chart wizard on argument APL expression
 Demo              Provide playback mechanism for live demonstrations
 Help              Browse or get link to information about any APL concept
 Version           Report version numbers of APL, OS, SALT, UCMD, .NET
                   and/or workspace

TRANSFER           User commands that convert workspaces between APL
                   versions and dialects:
 In                Import a workspace from a workspace transfer file
 Out               Export the current workspace to a workspace transfer file

UCMD               User commands that manage the user command framework:
 UDebug            Facilitate debugging of user commands
 ULoad             Load a user command's script into the current namespace
 UMonitor          Gather user command execution data
 UNew              Create one or more new user commands (optionally using a GUI)
 UReset            Refresh cache of all user command definitions
 USetup            Run Setup from setup.dyalog
 UVersion          Report version information of a user command

WS                 User commands that relate to workspaces:
 Check             Perform workspace integrity check
 Compare           Compare two workspaces
 Document          List (part of) the workspace content, displaying
                   each item separately
 FindRefs          Follow references in the workspace until all references
                   have been found
 FnsLike           List functions and operators matching a pattern
 Locate            Locate (and, optionally, replace) strings in the workspace
 Map               Display namespace treeview
 NamesLike         List names followed by their class matching the pattern
 Nms               List names followed by their class matching the pattern
 ObsLike           List objects matching the pattern
 Peek              Execute expression in temporary copy of workspace
 SizeOf            Report size of variables in descending order
 VarsLike          List variables matching the pattern

]        ⍝ for general user command help
```

 ©75-20, Pedro Kantek

```
]cmd -?  A for info on the "Cmd" command or group
```

Capítulo 10

Estruturas de Controle

Uma função do usuário (programa) APL executa do primeiro ao último comando. Eventuais saltos podem ser implementados usando o formato `→n` onde n é o número da linha para onde se vai.

Eventuais melhoramentos deste desvio são:

- O desvio relativo (ao contrário do anterior – que é absoluto). Agora o desvio surge como `→label` e em algum ponto do código deve-se iniciar uma linha com a constante `label:`. Agora o desvio vem para cá. A vantagem é que ao renumerar o código – coisa que o APL faz de maneira automática cada vez que o código é alterado, o desvio absoluto corre o enorme risco de se perder, enquanto o desvio relativo segue íntegro.

- O desvio condicional. Agora o desvio só acontecerá se alguma condição for verdadeira.
  O formato é `→(condição)/label:`. Se a condição retornar 1, haverá desvio. Senão, o processamento segue a gravidade.

O APL original tinha apenas isto em termos de desvios. Era pouco, mas o APL se justificava dizendo que características da linguagem minimizavam esta necessidade. Um exemplo: o que em C++ se escreve

```
if (0==x%2) {
   z='par'; }
else {
   z='impar'; }
```

Em APL se escreve

```
z ← ((2ρ2|x),1 1 1)/'impar'
```

Acho até que era um cavalo de batalha dos APListas, dizendo mais ou menos assim, *olhe como somos diferentes.*

Mas, com exceção da IBM, toda a comunidade APL se rendeu ao fato de que não dá para ter uma linguagem de programação decente sem as ferramentas da programação estruturada. Por essa razão, quase todos implementaram tais ferramentas. Aqui vai-se ver como o Dyalog faz isso.

10.1 If

No formato

```
:If <condição>
   comando1
   comando2
   ...
[:Else
   comando11
   ... ]
:EndIf
```

Observações: As palavras `:If`, `:Else` e `:EndIf` são escritas com essa configuração de caixa alta/baixa. Mas, se você escrever diferente, o editor corrige para você. :Else é opcional e se ausente implicará que nada é executado se a condição original for falsa.

10.1.1 Variações

Há algumas, a saber:

:ElseIf Para simplificar longos textos de condicionais compostas. Acompanhe o exemplo

```
:If a>5 ◊ ...a maior do que cinco
:ElseIf a>3 ◊ ... a maior do que 3 e menor ou igual a 5
:ElseIf a>1 ◊ ... a maior do que 1 e menor ou igual a 3
:Else ◊ ... a menor ou igual a 1
:EndIf
```

:AndIf Permite incluir cláusulas AND à condicional original, veja

```
:If a>5 ◊ ... a maior que 5
:AndIf a<10 ◊ ... a maior que 5 E menor que 10
...
```

:OrIf Permite incluir cláusulas OR à condicional original

Eu particularmente prefiro deixar de usar :AndIf e :OrIf e usar condições compostas com as cláusulas and ($\wedge$) e or ($\vee$). Ao agir assim mantem-se a compatibilidade com outros ambientes e segue o baile (Teoria do homem comum).

10.2 While

No formato

```
:While <condição1>
   comando1
   comando2
   ...
:EndWhile [ou :Until <condição2>]
```

A regra é a de sempre. A condição1 é avaliada e se verdadeira, o bloco subordinado é executado. No caso do bloco se encerrar por EndWhile, ocorre um desvio incondicional para p início do bloco, quando então a condição1 volta a ser reavaliada.

No caso do bloco terminar por um Until, ocorre uma segunda condição. Se esta segunda condição for verdadeira, há uma saída do bloco e se ela for falsa, retorna-se ao início do bloco. Veja um exemplo

```
∇  teste2 x
 :While x<100
     x←x+2
     x
 :Until x>50
 'acabou' ∇
```

Neste caso, se x começar com 1 são impressos 3, 5, 7, ... 49, 51 e fim de processamento.

10.3 Repeat

No formato

```
:Repeat
   comando1
   comando2
   ...
   :Until condição
```

Haverá a repetição do bloco, pelo menos 1 vez e até que a condição seja verdadeira. Acompanhe

```
 teste3 x
 :Repeat
     x←x+2
     x
 :Until x>50
 'acabou'
```

Neste caso, se chamado com $x = 30$ haverá a impressão de 32, 34, 36, ..., 50, 52 e acabou o processamento. Note que :EndRepeat pode ser abreviado apenas com :End.

10.4 For

O formato agora é

```
:For <variável> :In <valores>
   comando1
   comando2
   ...
:EndFor
```

Quando a cláusula :For é encontrada, a expressão ao lado de :In é avaliada e o resultado armazenado. A variável recebe o primeiro valor do resultado armazenado e o bloco entre :For e :EndFor é executado. Quando :EndFor for achado, o segundo valor armazenado é atribuído à variável e segue o baile. O bloco termina quando não houver mais valores a assinalar. Veja

```
 teste4
 :For a :In ⍳10
     a
 :EndFor
 'acabou'
```

Neste caso, haverá a impressão de 1,2,3,..., 10.

10.5 Select

Para múltiplos valores, veja

```
:Select <variável>
:Case <valor1>
   comando1
:Case <valor2>
   comando2
:Else
   comando11
:EndSelect
```

10.6 With

Usado para simplificar referências a um objeto ou a um namespace. Termina por :EndWith.

10.7 Hold

Usado para sincronizar o acesso a blocos de dados ou recursos compartilhados entre várias threads.

10.8 Trap

Mecanismo para tratamento de erros a ser usado em conjunto com a função `⎕TRAP`.

10.9 GoTo

Implementa uma versão alternativa a `→n` ou `→label`.

10.10 Return

Encerra o processamento de uma função de maneira incondicional. Equivale a `→0`.

10.11 Leave

Usado para encerrar a execução de um bloco de comandos :For, :Repeat ou :While de maneira incondicional.

 ©75-20, Pedro Kantek

10.12 Continue

Usado para desprezar o ciclo atual e iniciar uma nova interação do bloco onde ele é emitido. (:For, :Repeat e :While).

10.13 Section

Usado para subdividir funções e scripts (classes, namespaces). Termina com :EndSecton. Este cara não altera a execução de nada.

10.14 Disposable

Usado para eliminar objetos da interface com .NET.

Capítulo 11

Graphical User Interface - GUI

Aqui vamos estudar como construir aplicações com a cara do Windows.

11.1 Propriedades

Objetos neste contexto possuem propriedades. Cada propriedade é constituída por um nome seguido de seu valor. Exemplos

```
'Posn' (10 10) ⍝ posição dentro do pai
'Size' (20 50) ⍝ tamanho
'Fcol' (0 0 192) ⍝ cor de fundo
'Font' ('Arial' 16) ⍝ fonte e seu tamanho
```

11.2 4 Funções básicas

São 4 as funções necessárias para manipular objetos

nome	significado	função
⎕WC	window create	criar objetos e especificar suas propriedades
⎕WS	window set	modificar propriedades ou especificar novas
⎕WG	window get	ler valores das propriedades
⎕WN	window names	obter nomes dos filhos

O nome do objeto é especificado à esquerda das 3 primeiras e como direito da última.

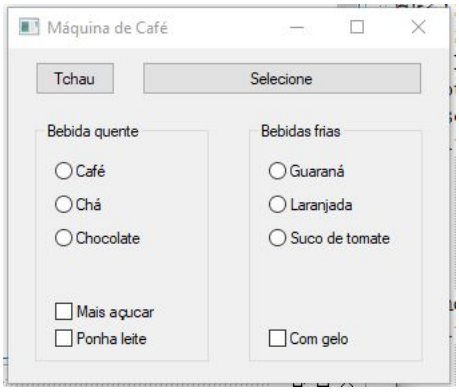
11.3 Um exemplo

Vamos criar um painel de aplicativo

```
r←CMDesign;Check;Radio
Check←'Style' 'Check'
Radio←'Style' 'Radio'
:With 'Bebidas'⎕WC'Form' 'Máquina de Café'('Size' 250 330)
    'bout'⎕WC'Button' 'Tchau'(10 20)(25 60)
    'bsel'⎕WC'Button' 'Selecione'(10 100)(25 210)
    :With 'Quente'⎕WC'Group' 'Bebida quente'(55 20)(180 130)
        'B1'⎕WC'Button' 'Café'(25 15)Radio
        'B2'⎕WC'Button' 'Chá'(50 15)Radio
        'B3'⎕WC'Button' 'Chocolate'(75 15)Radio
        'B4'⎕WC'Button' 'Mais açucar'(130 15)Check
        'B5'⎕WC'Button' 'Ponha leite'(150 15)Check
    :EndWith
    :With 'Frio'⎕WC'Group' 'Bebidas frias'(55 180)(180 130)
        'B1'⎕WC'Button' 'Guaraná'(25 15)Radio
        'B2'⎕WC'Button' 'Laranjada'(50 15)Radio
        'B3'⎕WC'Button' 'Suco de tomate'(75 15)Radio
        'B4'⎕WC'Button' 'Com gelo'(150 15)Check
    :EndWith
```

```
:EndWith
```

Para isto funcionar, antes execute `'#' ⎕WS 'Coord' 'Pixel'`. Se não fizer isso, o tamanho ficará errado e o painel não será visível. Daí chame a função CMDesign e você terá



Agora, vai-se detectar o que foi apertado. Duas estratégias aqui:

- Usar `⎕WG` para obter o valor da propriedade `'State'` de cada um dos botões.
- Usar a notação de Namespace para perguntar o valor da propriedade.

Por exemplo, para saber se o botão 'Café' foi ou não apertado, usando os 2 métodos:

```
      'Bebidas.Quente.B1' ⎕WG 'State'
0   ⍝ não foi
      Bebidas.Quente.B1.State
0   ⍝ mesma coisa
```

Para determinar o estado dos 5 botões localizados no grupo Quente:

```
      ⍴children ← ⎕WN 'Bebidas.Quente'
5     ⍝ volta como um vetor aninhado
      children ⎕WG ¨ ⊂ 'State'   ⍝ aplica ⎕WG'State' a cada filho
0 1 0 1 0   ⍝ escolha: cha com mais açucar
```

Usando a notação de Namespace a mesma coisa ficaria

```
      Bebidas.Quente.(B1 B2 B3 B4 B5).State
0 1 0 1 0
```

Para mudar propriedades, faz-se algo muito parecido. Por exemplo para trocar Chá por Chocolate, usando os 2 métodos

```
      'Bebidas.Quente.B2' ⎕WS 'State' 0
      Bebidas.Quente.B3.State ← 1
```

Ao executar a função acima, criou-se um GUI Namespaced de nome 'Bebidas' como se pode ver em

```
      ⎕NC ⊂'Bebidas'
9.2
```

Agora precisa-se resolver 2 problemas pendentes:

- É sempre visível. Para removê-lo há que fechá-lo.
- Está inativo. Não acontece nada quando se pressiona Tchau ou Selecione.

Para separar a definição do uso de um formulário, criemos a função

```
     ∇r←CMUse;Bebidas
[1]  CMDesign
[2]  ∇
     ⎕EX 'Bebidas'  ⍝ apagando o formulário global
     CMUse  ⍝ não acontece nada
```

Porque a variável Bebidas é local, ela é criada, mas ao final da função apagada (não é mais global). Precisamos fazer algo com a interação antes de apagar a variável (≡ encerrar a função).

11.3.1 Fila de eventos

Quando um ou mais box de diálogo estão ativos o usuário pode fazer muitas coisas: mover o mouse, apertar um botão, digitar coisas em um campo de entrada, mover a barra de rolamento, fechar um box de diálogo e por aí vai. Essas ações causam EVENTOS e eles são colocados em uma fila junto com informações apropriadas para determinar o que aconteceu.

Há aqui 2 filas envolvidas: uma gerenciada pelo Windows e outra pelo Dyalog.

Os eventos do APL são processados usando a função `⎕DQ`. `⎕DQ` deve ser seguido pelo (s) nome(s) do objeto. Se ele for seguido por um `'#'` (root) eventos de todos os formulários e caixas de diálogo criados nesta sessão serão manuseados.

Enquanto `⎕DQ` é executado, a sessão APL é temporariamente desabilitada e o foco transferido ao objeto listado no argumento. Veja

```
     ∇r←CMUse;Bebidas
[1]  CMDesign
[2]  ⎕DQ 'Bebidas'
[3]  ∇
```

A caixa de diálogo é mostrada, mas nada acontece. Por enquanto o máximo que você pode fazer é fechar a caixa de diálogo.

11.4 Funções de call-back

Para fazer a sua aplicação responsiva, é necessário especificar certos eventos que devem disparar certas ações o que em geral significa chamar certas funções APL. Certas funções são chamadas *call-back* porque elas são chamadas pelo `⎕DQ` para processar os eventos.

Um evento é identificado por uma palavra chave ou por um código numérico. Recomenda-se o uso da palavra chave exceto para eventos definidos pelo usuário que são especificados apenas por números. Veja

Ação	Palavra chave	Código
Teclar 1 tecla	KeyPress	22
Mover o mouse	MouseMove	3
Selecionar um botão numa caixa de diálogo	Select	30

A lista completa está disponível no help on-line.

A propriedade 'Event' pode ser pensada como um vetor contendo tantos elementos quantos são os tipos de eventos que podem ser gerados pelo objeto em questão. Um função de call-back é associada com um tipo particular de evento estabelecendo o item correspondente com a palavra chave ao nome da função.

```
('Event' 'Select' 'controle') ⍝ associa a função controle
 ⍝    quando o objeto for selecionado
('Event' 'KeyPresses' 'maiusca') ⍝ chama a função maiusca
 ⍝    cada vez que uma tecla é pressionada
```

A associação pode ser feita na criação (via `⎕WC`) ou adicionada mais tarde (via+ WS+).

Para um dado objeto, diversos eventos podem chamara a mesma função de call-back ou eventos diferentes podem chamar diferentes funções. Veja

```
('Event' (Evento1 Event2 Event3) 'callb')
('Event' (Event1 'callb1') (Event2 'callb2') (Event3 'callb3'))
```

As funções chamadas são funções APL comuns e podem fazer qualquer coisa (midar propriedades, deletar objetos, criar novos formulários, imprimir resultados e assim por diante.

Usualmente eventos disparam funções de call-back mas eles também podem ser associados com uma das 2 ações específicas representadas pelos valores `1` e `¯1`:

- `¯1` significa: ignore este evento como se ele não tivesse acontecido. Isto é chamado para proteger a aplicação a ações indesejáveis.
- `1` abandone o controle de `⎕DQ` e retorne ao fluxo de execução normal do APL. Este é o modo normal de sair do `⎕DQ`.

Vamos ver a aplicação fazendo isto: quando o botão selecione é ativado, a aplicação deve mostrar uma mensagem confirmando o tipo de bebida que a máquina deve preparar. Para obter isso, nossa função de call-back precisa saber o estado de todos os botões:

```
hotbin ← Bebidas.Quente.(B1 B2 B3 B4 B5).State
coldbin ← Bebidas.Frio.(B1 B2 B3 B4).State
```

e então usar estes 2 vetores booleanos para selecionar a descrição apropriada, como em

 ©75-20, Pedro Kantek

```
quentebeb←'Café' 'Cha' 'Chocolate' 'Mais açucar' 'adicione leite'
friobeb ← 'Guaraná' 'laranjada' 'suco de tomate' 'com gelo'
quentesel ← hotbin/quentebeb
friosel ← coldbin/friobeb
escolha←(⊂'Bebidas quentes: ',quentesel,(' ' 'Bebidas frias:'),friosel
```

Vamos mostrar o resultado como um objeto chamado MessageBox ou MsgBox ou

```
'Relatorio' ⎕WC 'MsgBox' 'Sua escolha atual é' escolha
```

A caixa de mensagem não é filha do formulário principal, e sim um objeto autônomo, então se queremos mostrá-lo usa-se `⎕DQ` de novo como em

```
      ⎕DQ 'Relatorio'
```

Integrando estas idéias a nossa aplicação, e mudando a versão (para '1' primeiro e depois para '2') de modo a manter as versões anteriores íntegras:

```
r←CMDesign2;Check;Radio
Check←'Style' 'Check'
Radio←'Style' 'Radio'
:With 'Bebidas'⎕WC'Form' 'Máquina de Café'('Size' 250 330)
    'bout'⎕WC'Button' 'Tchau'(10 20)(25 60)('Event' 'Select' 1)
    'bsel'⎕WC'Button' 'Selecione'(10 100)(25 210)('Event' 'Select' '#.CMReport2')
    :With 'Quente'⎕WC'Group' 'Bebida quente'(55 20)(180 130)
        'B1'⎕WC'Button' 'Café'(25 15)Radio
        'B2'⎕WC'Button' 'Chá'(50 15)Radio
        'B3'⎕WC'Button' 'Chocolate'(75 15)Radio
        'B4'⎕WC'Button' 'Mais açucar'(130 15)Check
        'B5'⎕WC'Button' 'Ponha leite'(150 15)Check
    :EndWith
    :With 'Frio'⎕WC'Group' 'Bebidas frias'(55 180)(180 130)
        'B1'⎕WC'Button' 'Guaraná'(25 15)Radio
        'B2'⎕WC'Button' 'Laranjada'(50 15)Radio
        'B3'⎕WC'Button' 'Suco de tomate'(75 15)Radio
        'B4'⎕WC'Button' 'Com gelo'(150 15)Check
    :EndWith
:EndWith

CMReport2;quentebeb;friobeb;quentesel;friosel;escolha;Report;hotbin;coldbin;buttons
buttons←Bebidas.Quente.(B1 B2 B3 B4 B5)
hotbin←buttons.State
quentebeb←buttons.Caption
buttons←Bebidas.Frio.(B1 B2 B3 B4)
coldbin←buttons.State
friobeb←buttons.Caption
quentesel←hotbin/quentebeb
friosel←coldbin/friobeb
escolha←↑(⊂'Bebidas quentes: '),quentesel,(' ' 'Bebidas frias:'),friosel
'Report'⎕WC'MsgBox' 'Sua escolha agora é 'escolha
⎕DQ'Report'

r←CMUse2;Bebidas
CMDesign2
DISPLAY ⎕DQ'Bebidas'
'Isto é tudo, pessoal'
```