Workshop: Programming With Dyalog APL

ECUST 上海, July 5th 2018

Morten Kromberg
CXO, Dyalog Ltd.

(based on work by Jay Foad, Roger Hui & John Scholes)

Agenda

Morning

- Verify APL Installations – or see http://tryapl.org
- Download course materials from
  - https://github.com/mkromberg/ECUST2018
- A few slides: Introduction to APL
- Exercises until Lunch

After Lunch

- A quick look at tools for
  - Importing Data from Excel
  - Charts
- Review of MatLab code (SA_Cal) converted to APL

Gitte + Morten depart for airport around 2PM

History of APL

Kenneth E. Iverson
1920-2004

- Canadian of Norwegian Descent
- Born on a small farm in Alberta (Canada)
- Finished one-room school after 9th grade and worked on the farm
- Army 1942, Flight Engineer in Air Force from 1943
  - Almost finished High School in the service
  - Promised his officers and mates that he would pursue an academic career after the war
- B.A. from Queens University, Kingston Ontario
  - Ken didn't know there was such a thing as University before he joined the army!

Nothing on this slide in China

History of APL, continued

- Doctoral work at Harvard with Aiken and Leontief
- Taught at Harvard for 6 years,
  - frustrated with inadequacies of mathematical notation
- Developed ”Iverson Notation” in response
  - Published ”A Programming Language” in 1962

ACM Turing award in 1979:

“For his pioneering effort in programming languages and mathematical notation resulting in what the computing field now knows as APL, for his contributions to the implementation of interactive systems, to educational uses of APL, and to programming language theory and practice.”

Syntaxes of Mathematics

$a\,b$

$Mat1 \cdot Mat2$

Problems:

- *Wide variety of syntactical forms*
- *Strange and inconsistent precedence rules*
- *Things get worse when you deal with matrices*

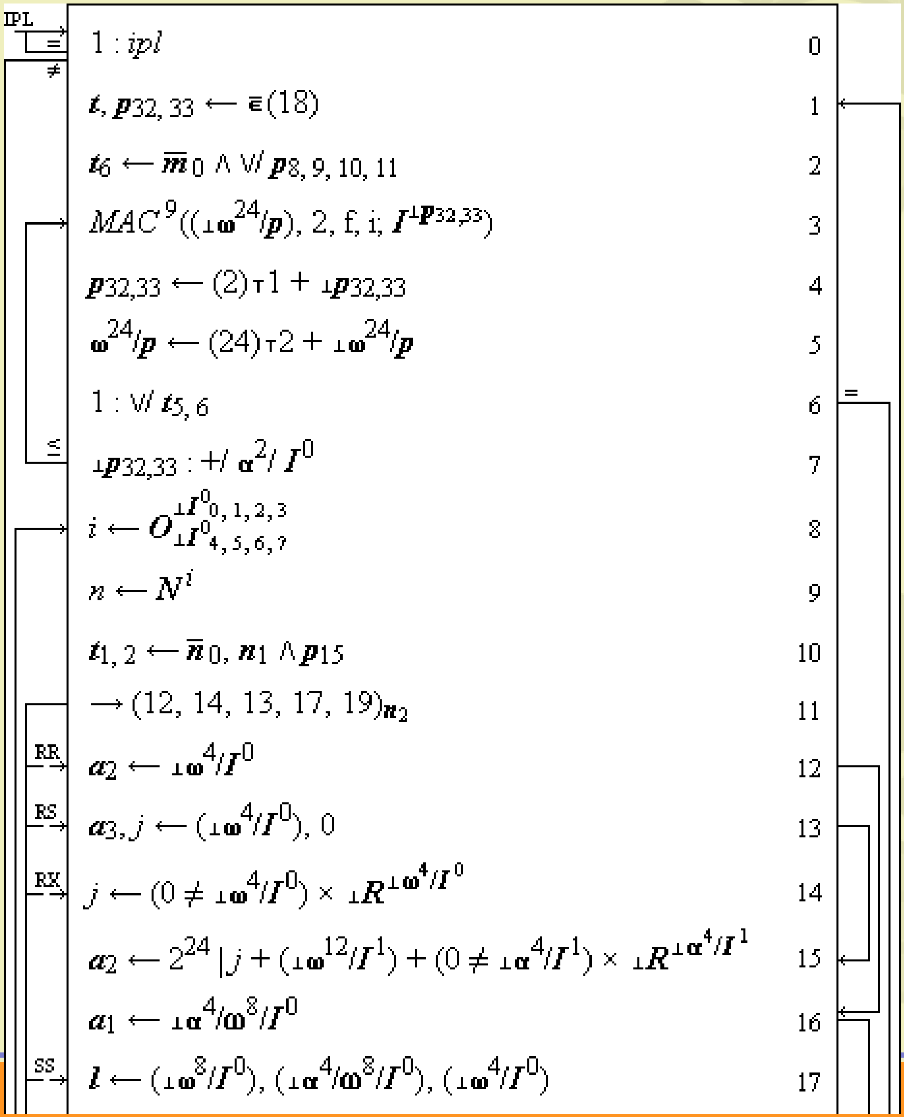
See http://www.jsoftware.com/papers/EvalOrder.htm

$\sum_{n=1} 4n$

$\prod_{i=1} 4i$

$\frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$

Iverson Notation: Description of IBM\360



IPL $1 : ipl$ — 0

$\boldsymbol{t}, \boldsymbol{p}_{32,33} \leftarrow \epsilon(18)$ — 1

$t_6 \leftarrow \overline{\boldsymbol{m}}_0 \wedge \vee/ \boldsymbol{p}_{8,9,10,11}$ — 2

$MAC^9((\perp\boldsymbol{\omega}^{24}/\boldsymbol{p}), 2, f, i; \boldsymbol{I}^{\perp\boldsymbol{p}_{32,33}})$ — 3

$\boldsymbol{p}_{32,33} \leftarrow (2)\top 1 + \perp\boldsymbol{p}_{32,33}$ — 4

$\boldsymbol{\omega}^{24}/\boldsymbol{p} \leftarrow (24)\top 2 + \perp\boldsymbol{\omega}^{24}/\boldsymbol{p}$ — 5

$1 : \vee/ \boldsymbol{t}_{5,6}$ — 6

$\perp\boldsymbol{p}_{32,33} : +/ \boldsymbol{\alpha}^2/ \boldsymbol{I}^0$ — 7

$i \leftarrow O^{\perp\boldsymbol{I}^0_{0,1,2,3}}_{\perp\boldsymbol{I}^0_{4,5,6,7}}$ — 8

$n \leftarrow N^i$ — 9

$t_{1,2} \leftarrow \overline{\boldsymbol{n}}_0, \boldsymbol{n}_1 \wedge \boldsymbol{p}_{15}$ — 10

$\rightarrow (12, 14, 13, 17, 19)_{\boldsymbol{n}_2}$ — 11

RR $a_2 \leftarrow \perp\boldsymbol{\omega}^4/\boldsymbol{I}^0$ — 12

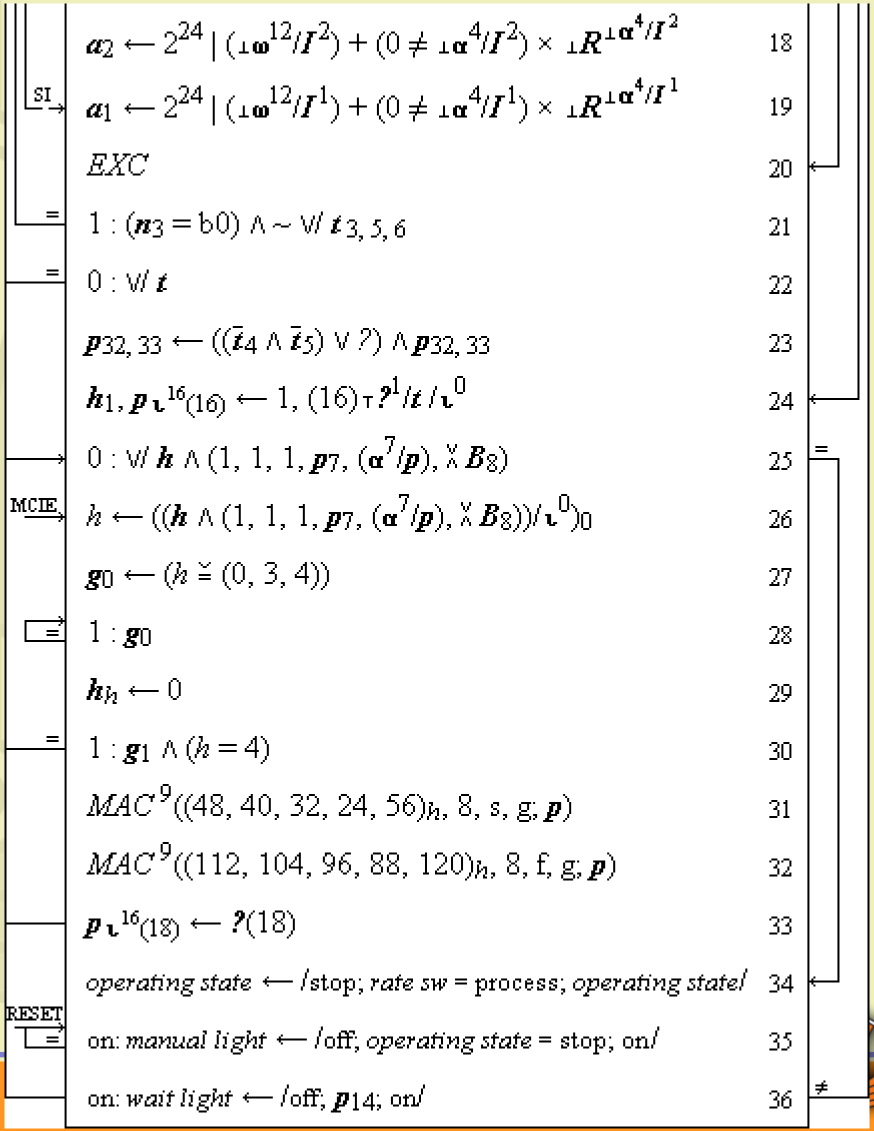
RS $a_{3,j} \leftarrow (\perp\boldsymbol{\omega}^4/\boldsymbol{I}^0), 0$ — 13

RX $j \leftarrow (0 \neq \perp\boldsymbol{\omega}^4/\boldsymbol{I}^0) \times \perp R^{\perp\boldsymbol{\omega}^4/\boldsymbol{I}^0}$ — 14

$a_2 \leftarrow 2^{24} | j + (\perp\boldsymbol{\omega}^{12}/\boldsymbol{I}^1) + (0 \neq \perp\boldsymbol{\alpha}^4/\boldsymbol{I}^1) \times \perp R^{\perp\boldsymbol{\alpha}^4/\boldsymbol{I}^1}$ — 15

$a_1 \leftarrow \perp\boldsymbol{\alpha}^4/\boldsymbol{\omega}^8/\boldsymbol{I}^0$ — 16

SS $\boldsymbol{l} \leftarrow (\perp\boldsymbol{\omega}^8/\boldsymbol{I}^0), (\perp\boldsymbol{\alpha}^4/\boldsymbol{\omega}^8/\boldsymbol{I}^0), (\perp\boldsymbol{\omega}^4/\boldsymbol{I}^0)$ — 17



$a_2 \leftarrow 2^{24} | (\perp\boldsymbol{\omega}^{12}/\boldsymbol{I}^2) + (0 \neq \perp\boldsymbol{\alpha}^4/\boldsymbol{I}^2) \times \perp R^{\perp\boldsymbol{\alpha}^4/\boldsymbol{I}^2}$ — 18

SI $a_1 \leftarrow 2^{24} | (\perp\boldsymbol{\omega}^{12}/\boldsymbol{I}^1) + (0 \neq \perp\boldsymbol{\alpha}^4/\boldsymbol{I}^1) \times \perp R^{\perp\boldsymbol{\alpha}^4/\boldsymbol{I}^1}$ — 19

EXC — 20

$1 : (\boldsymbol{n}_3 = b0) \wedge \sim \vee/ \boldsymbol{t}_{3,5,6}$ — 21

$0 : \vee/ \boldsymbol{t}$ — 22

$\boldsymbol{p}_{32,33} \leftarrow ((\overline{\boldsymbol{t}}_4 \wedge \overline{\boldsymbol{t}}_5) \vee ?) \wedge \boldsymbol{p}_{32,33}$ — 23

$\boldsymbol{h}_1, \boldsymbol{p}\,\boldsymbol{\iota}^{16}(16) \leftarrow 1, (16)\top ?^1/\boldsymbol{t}/\boldsymbol{\iota}^0$ — 24

$0 : \vee/ \boldsymbol{h} \wedge (1, 1, 1, \boldsymbol{p}_7, (\boldsymbol{\alpha}^7/\boldsymbol{p}), \overset{\vee}{\wedge} B_8)$ — 25

MCIE $h \leftarrow ((\boldsymbol{h} \wedge (1, 1, 1, \boldsymbol{p}_7, (\boldsymbol{\alpha}^7/\boldsymbol{p}), \overset{\vee}{\wedge} B_8))/\boldsymbol{\iota}^0)_0$ — 26

$\boldsymbol{g}_0 \leftarrow (h \,\underline{\in}\, (0, 3, 4))$ — 27

$1 : \boldsymbol{g}_0$ — 28

$\boldsymbol{h}_h \leftarrow 0$ — 29

$1 : \boldsymbol{g}_1 \wedge (h = 4)$ — 30

$MAC^9((48, 40, 32, 24, 56)_h, 8, s, g; \boldsymbol{p})$ — 31

$MAC^9((112, 104, 96, 88, 120)_h, 8, f, g; \boldsymbol{p})$ — 32

$\boldsymbol{p}\,\boldsymbol{\iota}^{16}(18) \leftarrow ?(18)$ — 33

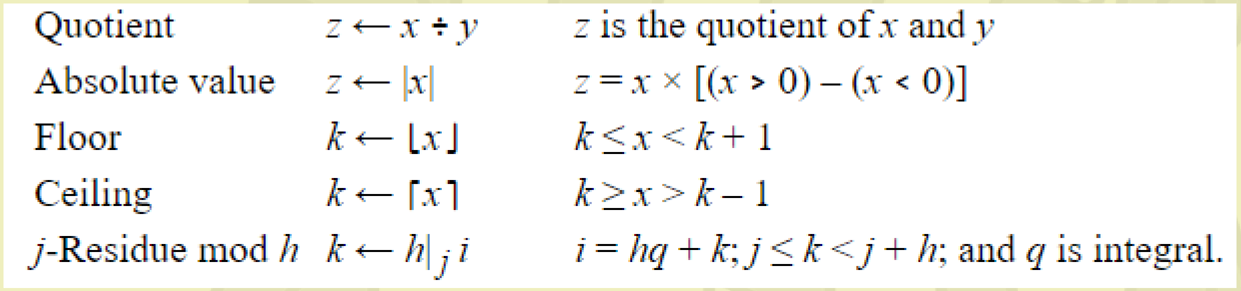
operating state ⟵ /stop; *rate sw* = process; *operating state*/ — 34

RESET on: *manual light* ⟵ /off; *operating state* = stop; on/ — 35

on: *wait light* ⟵ /off; $\boldsymbol{p}_{14}$; on/ — 36

Slide 8: Lucky Linearization

- The 5: Ken Iverson, Adin Falkoff, Larry Breed, Dick Lathwell, Roger Moore.
  Operated by “Quaker Consensus”.



Quotient	$z \leftarrow x \div y$	z is the quotient of x and y
Absolute value	$z \leftarrow \lvert x \rvert$	$z = x \times [(x > 0) - (x < 0)]$
Floor	$k \leftarrow \lfloor x \rfloor$	$k \leq x < k + 1$
Ceiling	$k \leftarrow \lceil x \rceil$	$k \geq x > k - 1$
j-Residue mod h	$k \leftarrow h \vert_j i$	$i = hq + k; j \leq k < j + h;$ and q is integral.

A Programming Language (for Mathematics)

```
a×b
```
ab

```
*x
```
e^{x}

```
x÷y
```
$\frac{x}{y}$

```
b⍟a
```
$\log_b a$

```
a*÷n
```
$\sqrt[n]{a}$

$Mat1 \cdot Mat2$
```
Mat1 +.× Mat2
```

```
f g x
```
$f\,g\,x$

```
(f+g) x
```
$(f+g)\,x$

```
(3○x)*2
```
$\tan^{2} x$

```
+/4×⍳6
×/4×⍳6
```

$$\sum_{n=1}^{6} 4n$$

$$\prod_{i=1}^{6} 4i$$

$$\frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

```
(2×a)÷⍨(-b)(+,-)0.5*⍨(b*2)-4×a×c
```

Exercises

- The folder "Exercises" contains numbered files, starting with "00-syntax.txt"
- We will do one exercise at a time, followed by a discussion. For each line in the file:
  - Enter the expression, but ...
  - BEFORE you hit <enter>, try to predict the result
  - When you are surprised, think about it
- Remember: You can get help for APL symbols by hovering over the "language bar" or putting the cursor on a symbol and hitting F1

Saving Your Work

- Historically, APL users have saved workspaces containing code and data as a single file
  - Similar to an Excel Spreadsheet
  - Takes a ”snapshot of the VM”
  - Beware: also saves the execution stack if there is one
- Save your work using

```
      )save /path/mywsname[.dws]
```

- Load it again with

```
      )load /path/mywsname
```

- You can extract named objects from a workspace:

```
      )copy /path/mywsname foo goo x y
```

- Saved workspaces can have a ”latent expression” ⎕LX, which is executed when the workspace is loaded, unless you

```
      )xload /path/mywsname
```

Saving Your Work, continued

- In the last few years, it has become more popular to use Unicode text files (and SVN/GIT), especially for source code
- You can save a fn/var, namespace or class using `]save` (`]save` is a "user command" - written in APL):
  ```
  ]save name /path/name[.dyalog]
  ```
  (it is customary to use the same name for the file)
- You can bring it back into the active workspace using
  ```
  ]load /path/name
  ]load /path/*
  ```
- If you edit objects that were `]load`ed, the system will offer to update the file each time you make a change
- From version 15.0, the interpreter (editor) knows how to open and view source files without user commands.

APL vs MatLab – Differences (1/2)

APL

```
      X ← 1 0 3
      Y ← 10 1 2
      X ÷ 10
0.1 0 0.3
      X ÷ Y
0.1 0 1.5
      X ⌹ Y
0.13333
```

MatLab

```
      X = [1,0,3]
      Y = [10,1,2]
      X / 10
0.1 0 0.3
      X / Y
0.13333
      X ./ Y
0.1 0 1.5
```

- In APL, ÷ always divides an element in the left argument with an element of the right argument
- In MatLab, / changes meaning depending on the shape of the arguments.

APL vs MatLab – Differences (2/2)

APL

```
      X ← 1 0 3
      Y ← 10 1 2

      X × 10
10 0 30
      X × Y
10 0 6



      X +.× Y
16
```

MatLab

```
      X = [1,0,3]
      Y = [10,1,2]

      X * 10
10 0 30
      X * Y
[error]
      X .* Y
10 0 6
      X * Y'
16
```

- In APL, × works like +−÷! and ALL the "scalar functions"
- In MatLab, * changes meaning depending on the shape of the arguments – and in a different way from how / changes.

APL vs. MatLab

- APL was invented between 1960-1966
  - MatLab 1965-1970
  - (both have developed a lot since then)
- APL is simple and regular
  - MatLab has retained some of the irregularities of Math.
- *In my opinion*, this makes MatLab
  - Less surprising to the beginner
  - Messy when things get complicated
- MatLab is a good choice if there is already a module which solves your problem
  - If you have to write your own algorithms, APL may be cleaner and faster – and help you think about the problem
  - It is *much* easier to deploy solutions in APL than MatLab

Reading APL

```
CB←{⍵[1+(⍴⍵)|X∘.+X←(⍳⍺)-1]}
```

Imagine arguments:

```
8 CB ' ⎕'
```

Reading APL

What does this function do?

```
{(~R∊R∘.×R)/R←1↓⍳⍵}
```

Multi-Line dfns

```
      )ed sign ⍝ to start the editor

      sign←{
        ⍵<0:'negative'
        ⍵>0:'positive'
        'zero'
      }

      sign 1
positive
```

Procedures / Tradfns

Monadic:

```
     ∇ R←Sum X
[1]    R←+/X
     ∇
```

Dyadic:

```
     ∇ R←A MatMult B
[1]    R←A+.×B
     ∇
```

Niladic:

```
     ∇ Run
[1]    ⎕←'Boo Hiss!'
     ∇
```

Procedures / Tradfns

”Ambi-valent” (+ use a control structure)

```
     ∇ R←{Window} Sum X
[1]    :If 0=⎕NC 'Window' ⋄ R←+/X
[2]    :Else ⋄ R←Window +/ X
[3]    :EndIf
     ∇
```

Procedures / Tradfns

Name Elements of Right Argument

+ Local Variable

```
     ∇ r←Round (n decimals);base
[1]    base←10*decimals
[2]    r←(⌊0.5+n×base)÷base
     ∇
```