APL
one of the coolest programming languages ever
. . . inspired by the "Lithp" talk. . .

Bernd Ulmann
`ulmann@vaxman.de`

YAPC::Europe 2006
30th August – 1st September 2006
Birmingham

Ken Iverson, 17.12.1920 – 19.10.2004

In 1957 IBM employee *Ken Iverson* started the development of a "language" to simplify mathematical notation which might be even used to describe computer architectures.
This language was internally known as *Iverson's Better Math* – IBM obviously did not like this name and forced Iverson to come up with something better:

A Programming Language
APL for short.

- Since Iverson's initial goal was to implement a more consistent mathematical notation he decided to use *normal* mathematical symbols, i.e. *rather weird* symbols which makes typing in an APL program bit of a challenge.

  To be honest, APL code looks like line noise...

- APL does not care about data types too much (this sounds very familiar).
- APL's basic datastructure is the vector.
- APL does not need loops, conditionals and the like (normally) – everything is mapped to vector/matrix operations.

How does a really, really simple APL program look like? Let us compute the following sum (which was solved for the general case by little Gauss when he was barely six years old – how depressing. . .)

$$\sum_{i=1}^{100} i$$

using APL:

$$+/\iota 100$$

The goal: Generate a list of all prime numbers between 2 and R...

...without any loops or conditionals – and with as few characters as possible!

$$(\sim \mathtt{R} \in \mathtt{R} \circ . \times \mathtt{R})/\mathtt{R} \leftarrow 1 \downarrow \iota \mathtt{R}$$

How does this work?

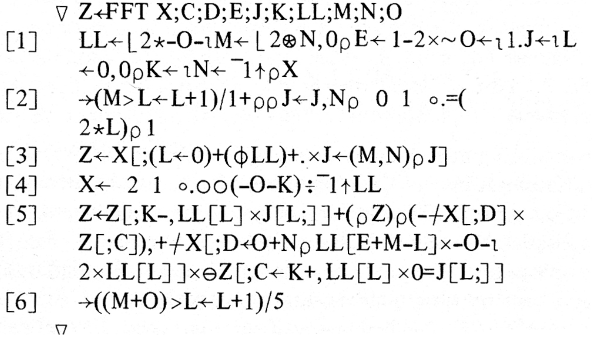
- Create a vector of the form $(2, \ldots, R)$ and save it in R (again) by $\mathtt{R} \leftarrow \mathtt{1} \downarrow \iota\mathtt{R}$.
- Create a matrix of the form

$$\begin{pmatrix} 4 & 6 & 8 & 10 & \ldots \\ 6 & 9 & 12 & 15 & \ldots \\ 8 & 12 & 16 & 20 & \ldots \\ 10 & 15 & 20 & 25 & \ldots \\ \vdots & \vdots & \vdots & \vdots & \ddots \end{pmatrix}$$

  by creating the outer product of the vector R like this: $\mathtt{R} \circ . \times \mathtt{R}$. This matrix contains everything but prime numbers!

- Now create a bit vector with elements corresponding to the elements of R, containing a 1 for every value being prime (thus not being an element of the matrix above): $(\sim \mathtt{R} \in \mathtt{R} \circ . \times \mathtt{R})$.
- Finally, use this bit vector to select all prime elements from R using the selection operator $/$.

Performing a fast Fourier transformation (FFT):



```
      ∇ Z←FFT X;C;D;E;J;K;LL;M;N;O
[1]     LL←⌊2*-O-⍳M←⌊2⍟N,0⍴E←1-2×~O←⍳1.J←⍳L
        ←0,0⍴K←⍳N←¯1↑⍴X
[2]     →(M>L←L+1)/1+⍴⍴J←J,N⍴ 0 1 ∘.=(
        2*L)⍴1
[3]     Z←X[;(L←0)+(⌽LL)+.×J←(M,N)⍴J]
[4]     X← 2 1 ∘.○○(-O-K)÷¯1↑LL
[5]     Z←Z[;K-,LL[L]×J[L;]]+(⍴Z)⍴(-⌿X[;D]×
        Z[;C]),+⌿X[;D←O+N⍴LL[E+M-L]×-O-⍳
        2×LL[L]]×⊖Z[;C←K+,LL[L]×0=J[L;]]
[6]     →((M+O)>L←L+1)/5
      ∇
```

How does this work? Do not even ask! (Have a look into "Programmieren in APL" by Wolfang K. Giloi, deGruyter, Berlin, 1977, p. 212.)

- Being able to write APL programs does not normally imply that you are able to read APL programs – not even your own!
- This has APL earned the notation of being a *write only language*.
- People thinking of Perl as an unreadable language have obviously never seen any APL code (yet).

As with LISP there were some attempts to build APL machines, i.e. implement processors with special features to speed up the execution of APL programs.
On of the earliest attempts was an APL implementation in microcode (real programmers write microcode – Assembler is a high level language) for the IBM S/360 model 25 (described by Hassit).
A real hardware implementation was the (commercially unsuccessful - why?) Control Data STAR-100 vector processor shown in the next slide (a *real* computer – LISP machines are quite boring compared to that :-)).

APL is far from being dead!

- There even is a recent APL implementation available for free and for most common platforms (unfortunately not for OpenVMS) from Morgan Stanley: `http://www.aplusdev.org`
- Ken Iverson's last brainchild, called **J**, is available for a lot of platforms, too (even OpenVMS – ported by me :-)) – it is even more weird than APL since it does not need any special symbols but (ab)uses everything available in ASCII in very inventive ways (cf. `http://www.jsoftware.com`).