APL
PRESS
Box 27, Swarthmore, PA 19081

2

1 2

1976

APL NEWS

APL PRESS

BOX 27, SWARTHMORE, PA 19081 (215) 328-9771

This second issue of APL News presents a language extension proposal by Doug Forkes, and a new column by Jeff Shallit, to be devoted to recreations. Correspondence on any of these features will be welcomed.

Beginners in APL criticized the first issue because it contained nothing of interest to them. The excerpt on Reading APL is offered for such readers; comment on it would be appreciated.

Many readers assumed that receipt of the first issue implied that they would continue to receive the newsletter. This is not so, since many of the names were on rented mailing lists and cannot be reused. So if you wish to receive the newsletter and have not sent in your name, do so now. In particular, if you did not receive a copy of our book list you are not now on our mailing list.

EDUCATION CONFERENCE

A conference on APL in Engineering Education was held at the Sagamore Conference Center, August 18-20. Organized by Professors Wilbur LePage and Garth Foster, and sponsored by the Department of Electrical and Computer Engineering of Syracuse University, the conference drew about fifty participants, mostly from mathematics and engineering faculty in Canada and the northeastern United States. Further information is available from the organizers; in particular, they are preparing a report for submission to APL Quote-Quad.

© APL PRESS, 1976

SCALAR CONFORMABILITY

Doug Forkes
I. P. Sharp Associates

Definition. Two arrays A and B satisfy scalar conformability if their shapes agree or if at least one is a scalar or one-element vector. I propose extending the condition of scalar conformability to a third case where the ranks agree and ∧/((ρA)=ρB)∨(1=ρA)∨1=ρB. The shape of the result in this case is (×(ρA)⌊ρB)×(ρA)⌈ρB.

It is difficult to express the result in conventional APL (thus illustrating at least one use for the extension, albeit circular). Stated informally, the dimensions which are equal to 1 have scalar extension applied to them. For example, if A←7 3 1ρι21 and B←7 1 5ρι35, then A+B ←→ A[;;5ρ1]+B[;3ρ1;].

The above expression is in 1-origin, as are all expressions in this article. It is worth noting that many of the uses of this extended conformability replace origin-dependent expressions with origin-independent ones.

A new function. We now define a function ω such that AωB ←→ ((~A)⌈A\ρB)ρB. That is, ω expands the shape vector of B, injecting 1 where there is a 0 in A. For example, if B←1 2ρ3, then 1 1 0 1 0ωB ←→ 0 1 1 2 1ρB. Notice that no change is made to the number or order of the elements of B; the shape is merely expanded.

This new function is useful in producing arrays satisfying the extended conformability rule. The usefulness of ω in this article suggests that it would be worth implementing as a primitive if this extended scalar conformability were adopted.

Examples. If MX←3 3ρι9 and VC←ι3, then:

MX+(ρMX)ρVC ←→ MX+0 1ωVC (adding matrix to row vector)

MX+⍉(⌽ρMX)ρVC ←→ MX+1 0ωVC (adding matrix to column vector)

MX∘.+VC ←→ (1 1 0ωMX)+0 0 1ωVC (outer product)

MX+.×VC ←→ +/MX×0 1ωVC (inner product)

Of course these last two examples do not represent extended capabilities, but rather show the close relationship of this definition to other APL concepts. However, they point the way to solutions to problems related to inner and outer product which are currently awkward. For example, compute *A*+.α*B*, where α is some scalar function but not a primitive function. For example, if *A*α*B* ↔ 1⌈*A*×*B*, then:

MX+.α*VC* ↔ +/1⌈1 2 2⍉*MX*∘.+*VC*

↔ +/1⌈*MX*×0 1ω*VC*

Consistency with other APL concepts. This definition of conformability is used in some implementations of inner product [Pakin, 1972]. Two arrays are conformable if their inner dimensions are equal, or if one or the other equals 1. In the latter case the array is extended in exactly the manner proposed here. Notice that this operation also benefits from the new function ω. For example, suppose that *P* is a vector of primes and that *M* is a vector of integers. Then (×*P*⌊.|(1,ρ*M*)ρ*M*)/*M* gives the elements of *M* that are relatively prime to every element of *P*. The phrase (1,ρ*M*)ρ*M* can be replaced by the slightly neater phrase 0 1ω*M*, so that we have (×*P*⌊.|0 1ω*M*)/*M*.

In those cases where the extended conformability coincides with present conformability, the results are identical with the conventional definition. This would seem to be desirable for any such extension, although a similar proposal [Breed, 1971] did not have this property.

Implementation considerations. The function ω is easily expressed in present APL and is easily implemented. Implementations which allow multiple references to the same data structure and separate references to data vector and shape vector would be especially suited to the implementation of ω. The extended scalar conformability would pose few difficulties, being essentially a matter of nested loops rather like those used in indexing; indeed it was no coincidence that I used indexing in my example definition.

REFERENCES

Pakin, Sandra, APL 360 Reference Manual, Science Research Associates, 1972, page 42.

L.M. Breed, "Generalized Scalar Extension", APL Quote-Quad, Vol 2, Number 6, March, 1971, page 5.

Jeff Shallit

This is the first APL-PLAY column, and I would like to describe some of the kinds of things we will be doing:

A. Problems -- consisting of both writing (defining functions to perform specified tasks) and reading (to explicate a given function or expression.) For example, what is the significance of the expression ∧/*M*=⌊*M* for a matrix *M*? For any array *M*?

B. Games -- listings of games you can play on an APL machine, with suggested modifications left as problems.

C. Proofs -- of new or old identities using APL notation. For example, I could use a neat proof that *A*⍉*B*⍉*C*↔*A*[*B*]⍉*C*.

D. APL Trivia -- including notes about APL and its current implementations, known bugs (such as a domain error for =/'*ABC*'), and novel uses of scan, inner and outer products, and other functions.

Except for this first column, which is a potpourri, each column will have some unifying theme such as number theory, graphics, or geometry. I invite your participation.

1. Give a clear explanation of lines 2, 11, and 12 of the following embodiment of the popular game of Mastermind:

```
    ∇ MASTERMIND;⎕IO;TABLE;GUESS;INPUT;N;S
[1]    ⎕IO←1
[2]   START:TABLE←⍉1+(SLOTSρ1↑ρCOLORS)⊤¯1+ι(1↑ρCOLORS)*SLOTS
[3]    N←0
[4]   GUESS:N←N+1
[5]    GUESS←,TABLE[S←?1↑ρTABLE;]
[6]    TABLE←(S≠ι1↑ρTABLE)⌿TABLE
[7]    ⎕←,(COLORS,' ')[GUESS;]
[8]    'ENTER NUMBER OF BLACK FOLLOWED BY NUMBER OF WHITE.'
[9]    INPUT←⎕
[10]   →(SLOTS=INPUT[1])/END
[11]   TABLE←(INPUT[1]=TABLE+.=GUESS)⌿TABLE
[12]   TABLE←((+/INPUT)=(+/[2]TABLE∘.=ι1↑ρCOLORS)+.⌊+⌿GUESS
          ∘.=ι1↑ρCOLORS)⌿TABLE
[13]   →(0≠1↑ρTABLE)/GUESS
[14]   'YOU HAVE MADE A MISTAKE.  WE ARE STARTING OVER.'
[15]   →START
[16] END:'I GUESSED YOUR ARRANGEMENT IN ',(⍕N),' TRIES.' ∇
```

```
      COLORS              ρCOLORS            SLOTS
RED                 6 6                4
BLUE
YELLOW
GREEN
BLACK
WHITE
```

2. The golden ratio, *G*, is the positive root of the quadratic equation `0=X⊥1 ¯1 ¯1`.

   a) Write an APL expression for *G* using any APL primitives, decimal points, the digit 5 (any number of times), and no other digits.

   b) Repeat part a for the negative root, $\underline{G}$.

3. There are two commonly used ways to represent lists of names of arbitrary lengths: as the rows of a matrix, or as segments of a vector separated by one or more spaces.

The canonical form for a matrix of names has each row left justified and a minimum width. The canonical form for a vector has a minimum number of spaces.

   a) Write a function *CANON*, to return its argument in canonical form, whether it be a matrix or a vector.

   b) Write a function *CONVERT*, to convert a matrix in canonical form to its vector equivalent or vice versa, depending on the rank of the argument.

4. Write the briefest possible "self-reproducing function" *SRF*, whose execution produces a print-out identical to that produced by `∇SRF[⎕]∇`, where the brevity of a function *F* is measured by the expression `+/,∨\' '≠⌽⎕CR 'F'`. Except for its length, the following example serves:

```
    ∇ Z←SRF
[1]   Z←(5 3 ρ'   [1][2][3]'),(5 3ρ' '),(⎕CR 'SRF'),[1] ' '
[2]   Z[1;5]←'∇'
[3]   Z[5;5]←'∇'
    ∇
```

READING APL

from Elementary Analysis

Learning to read expressions. The only effective way to learn a language or notation is to use it. Use of the notation in the exercises thus far should provide familiarity with each of the functions and the symbols used to denote them. The reader may, however, still have difficulty in interpreting compound expressions involving a number of functions.

The main point to be appreciated is that the few rules and definitions given in the preceding five pages are complete and mechanical in application; no subtle interpretation is needed. In this respect, readers familiar with conventional notation may find their greatest bar to learning psychological; one is so accustomed to the need for subtle interpretation that it is difficult to apply precise rules mechanically. For example, one is used to the fact that `X(X-3)` means something quite different from `F(X-3)`, and that values of variables cannot always be substituted for the variables in an expression (thus if `X` is `3` and `Y` is `4`, the expression `XY` is equivalent to `3Y` but not to `X4` nor `34`, which normally denote something entirely different).

Mechanical interpretation of an expression can be carried out in the following steps:

1. Determine the first function to be executed. According to item 9 at the beginning of this section, this function is the rightmost which can (subject to the rules for parentheses) be executed.

2. Execute the first function and replace it and its argument, or arguments, in the expression by the value obtained.

3. Repeat steps 1 and 2 until no more functions remain in the expression.

Consider, for example, the evaluation of the expression `N=+/(1↓ιN)×0=(1↓ιN)|N` for the case where `N←6`. The steps in the interpretation are shown below, with a line under each portion being evaluated:

```
N=+/(1↓ιN)×0=(1↓ιN)|N

N=+/(1↓ιN)×0=(1↓ιN)|6

N=+/(1↓ιN)×0=(1↓0 1 2 3 4 5)|6

N=+/(1↓ιN)×0=1 2 3 4 5|6

N=+/(1↓ιN)×0=0 0 0 2 1

N=+/(1↓ιN)×1 1 1 0 0

N=+/(1↓0 1 2 3 4 5)×1 1 1 0 0

N=+/1 2 3 4 5×1 1 1 0 0

N=+/1 2 3 0 0

N=6

6=6

1
```

With experience one of course collapses the process by combining simple steps and by executing independent expressions in parentheses simultaneously. For example:

```
N=+/(1↓ιN)×0=(1↓ιN)|N

6=+/1 2 3 4 5×0=1 2 3 4 5|6
```

Finally, one learns to grasp the significance of entire phrases. For example, in the underlined phrase of the last expression above, it is clear that `1 2 3 4 5|6` yields remainders on dividing `6`, and that the entire underlined phrase therefore yields 1's and 0's, the 1's indicating the elements of the vector `1 2 3 4 5` that are divisors of `6`. Moreover, the vector `1 2 3 4 5` clearly includes all possible divisors of `6` (except `6` itself). The succeeding multiplication by `1 2 3 4 5` therefore yields a vector which includes only zeros and the divisors of `6`. Hence the summation (`+/`) yields the sum of all divisors of `6`. This sum is finally compared with `6` yielding a `1` if `6` equals the sum of its (other) divisors.

More generally, for any positive integer value of *N* it appears that the expression yields `1` if *N* is a <u>perfect number</u>, that is, a number which equals the sum of its other divisors. In other words, the expression is a test for perfect numbers. Moreover, the actual evaluation can now be performed by an efficient shortcut; since the expression to the right of the equal sign yields the sum of the divisors, they may simply be listed and summed. Thus, for `N←27`, the expression becomes `27=+/1 3 9`, and clearly yields `0`.