PROBLEM SOLVING WITH *APL*

In Parabola, Vol. 7, No. 3, an example was given of the use of the computer to solve the problem of finding all integers less than some upper limit (10000 was used) which can be represented as KKK (written to the base J > K) in two different ways. The computer language used was FORTRAN.

During the last ten or twelve years, a new language called *APL* (a programming language) has been developed. *APL* is much more powerful than any other language, because the number and type of operators available is much greater. At the same time, the notation is more closely related to that of mathematics, but is more consistent.

Since many readers of Parabola are now learning some computer programming, a brief outline of some of the things *APL* can do may be of interest.

APL uses the conventional notations + - × and ÷. In addition, a negative symbol (¯) is used. Thus 2 +¯1 equals 1. All expressions are evaluated from right to left, eliminating rules of precedence such as the usually accepted one about multiplications and divisions being performed before additions and subtractions. Hence ¯1 + 2 is 1, -1 + 2 is ¯3 and 3×4 + 2 is 18.

APL can handle arrays of numbers (vectors or matrices) as easily as it can handle single numbers (scalars). Thus

1 2 4 + 3 6 9 ↔ 4 8 13.

In addition to the usual mathematical functions, some others have been introduced, and the four common ones have been extended. New symbols are used, some of which take two arguments (these are called dyadic functions) and some of which take just one (monadic). Some symbols can be used to denote both monadic and dyadic functions.

The following table gives some examples of the functions, and the corresponding results. Can you work out the general rule for each?

<u>Symbol</u>	<u>Name</u>	<u>Example</u>	<u>Result</u>
⍳	Iota	⍳4	1 2 3 4
		⍳6	1 2 3 4 5 6
⌊	Floor	⌊4.3	4
		⌊9.9	9
⌈	Ceiling	⌈4.3	5
		⌈9.9	10
⌊	Minimum	3⌊6.1	3
⌈	Maximum	3⌈6.1	6.1

There are many more symbols than these, and there isn't the space to discuss them all here. But it is worth mentioning that *APL* is designed to be used at an electric typewriter which is directly connected to a computer. The symbol ⎕ (pronounced "quad") is used in a programme for both input and output. (This symbol suggests a window through which you can see into the programme.) So *A*←⎕ means "type one (or more) numbers on the typewriter and store them in the computer with the name *A*". Similarly, ⎕←*B* instructs the computer to type the value of the variable *B*.

Because, in many calculations, we may want to add a list of numbers, or multiply them together, or find the largest (or smallest) of them, *APL* has a simple way of doing this:

```
+/1 3 4 2 9     ↔     19
×/1 3 4 2 9     ↔    216
⌈/1 3 4 2 9     ↔      9
⌊/1 3 4 2 9     ↔      1
```

In effect, the function symbol (+ , × etc.) is placed between each number in the list, and the resulting expression is then evaluated.

Another useful function is ⍴ (the Greek letter rho) which means "find the 'shape', or dimensions, of whatever expression follows". Thus

```
⍴ 1 3 4 2 9   ↔   5
```

because 1 3 4 2 9 is a vector of length ("shape") five elements.

Look at this *APL* expression:

```
⎕ ← (+/A) ÷ ⍴A ← ⎕
```

Remembering that *APL* evaluates expressions and programme statements from right to left, we see that this means:

(a) read in a list of numbers and call it *A*;
(b) find the shape of *A* (i.e., how many numbers there are in the list);
(c) divide this into (+/*A*) which is the sum of all the numbers in the list;
(d) type out the result.

This simple expression, therefore, finds the average of the numbers which are entered. The "conversation" with the computer would look like this:

```
      ⎕←(+/A)÷⍴A←⎕
⎕:
      2.34  5.67  8.9  6.54  3.21
5.332
```

Notice that *APL* uses just the number of significant figures required for the answer. The maximum precision to which it works is sixteen figures.

The *APL* programme to solve the problem mentioned at the beginning of this article uses some function symbols which have not been defined here. Nevertheless, you might like to see what it looks like. It will be recalled that the FORTRAN programme was sixteen lines long. We use an *APL* instruction to display the programme, which I have called *TRIPLETS*:

```
      ∇TRIPLETS[⎕]∇
    ∇ TRIPLETS M
[1]   (A=1⌽A)/A←A[⍋A←(A≤M)/A←((,(J∘.>L))/,N∘.×L←⍳⌈/(J-1)⌊⌊M÷N←1+J+(J←1+⍳98)*2)]
    ∇
```

The programme takes just one line (even if it is a long one!). To execute it, we type its name - *TRIPLETS* - followed by the value of *M*. The solution immediately follows:

```
      TRIPLETS 10000
273  546  931  3549  3783  4557  7566  9114
```

APL was first developed as a system of notation, and was subsequently implemented by IBM for use as a computer language in the manner described. Some other computer companies now have their own versions of *APL* as well. As a matter of interest, this article was prepared and edited using an *APL* text editing programme.

Graham de Vahl Davis

* * *

The author is Associate Professor in the School of Mechanical Engineering, University of New South Wales.