Correspondence

To the Editor
The Computer Journal

Sir

The postage stamp problem

With reference to W. F. Lunnon's article (this *Journal*, Vol. 12, p. 377) and J. L. Seldon's letter (this *Journal*, Vol. 15, p. 361), I should like to point out that the solution to V (8, 4) is 221 not 213 as reported by M. L. V. Pitteway.

The unique set of stamp values obtained were, 1, 3, 8, 12, 32, 54, 73, 99.

The program was written in CORAL 66 and run on a Modular One computer. The serial method of computation was employed and total processing time was approximately 100 hours.

Yours faithfully,

B. P. PHILLIPS

Telecomms Technical Officer
ATCEU
Hurn Airport
Christchurch
Dorset
13 January 1975

To the Editor
The Computer Journal

Sir

A note on APL

In the face of the current low-profile sales campaign for APL, I would like to argue that the trend to greater use of APL is undesirable. This is not a statement made without due consideration: I speak as an ex-addict whose access to APL was forcibly cut off when the money to pay for the terminal ran out, and having got over the withdrawal symptons I can now take an objective view.

APL arouses strong passions, and people either love it or hate it. However, it tends to be assessed as an indivisible package and accepted or rejected on the basis of one feature (e.g. array operations, or right-to-left evaluation). In fact the APL 'package' contains a number of separate components that can be judged independently. These include:

(*a*) the terminal interface to the user
(*b*) the array operations
(*c*) the concise notation.

The terminal interface is excellent. The system responds in an intelligent way, provides very good editing facilities, uses line-image techniques to ensure that what the user sees is what the computer sees, and it is almost impossible to fool the system with illegal commands or keystrokes. The only criticism is that the error messages take brevity to the extreme. There are only four different messages, and each consists of two words, one of which is ERROR. Indeed, the response of the APL system to a program error is reminiscent of the mourner on an Irish funeral procession, who on being asked 'who's dead?' replied 'The man in the box up front'. With this exception the human engineering of the terminal interface is superb, and much of the acceptance of APL is probably due to the fact that its interface is so much better than most other terminal systems, and incomparably better than other IBM systems. However, this is really nothing to do with APL as a language, and there is no reason why other language systems should not be similarly engineered. (The trouble is that, as in many other aspects of computing, the mass of users just don't know what is possible.)

The array operations are a great attraction of APL, and although it is undoubtedly convenient and powerful to be able to manipulate arrays as a whole, this is not unique to APL—similar facilities can be provided, for example, in ALGOL 68 by a library prelude. Moreover APL (like FORTRAN and ALGOL 60) forces everything into arrays, even when a record structure would be more appropriate.

The array capability of APL is a major contribution to the conciseness of an APL program. Another contribution to conciseness is the extended character set, and the large number of built in operations that are identified by a single character. Whilst it can be argued that APL takes conciseness to undue extremes (in the same way that COBOL extols the virtues of verbosity) there is a lesson that could be learned with advantage by other language designers and implementers. For how much longer are languages to be constrained by using a character set that may have been good enough for Herman Hollerith in 1890, but is pathetically inadequate for the expression of scientific algorithms?

The third contribution to conciseness is the syntax (or lack of it) and the notorious 'right-to-left' evaluation rule. Many critics of APL base their criticism solely on the right-to-left rule, seeming not to realise that you could have a left-to-right rule without changing any of the other features of the language. The real criticism of the language, in my view, does not lie in the 'syntactic salt' of order of evaluation, but is to be found in the impoverished syntax and the consequent possibility of ingenious coding tricks (in both of which APL resembles assembler language). The syntax restricts functions to have zero, one or two arguments, and if they have two arguments the function must be written in infix form. Moreover, function names have to be alphanumeric: here as in the right-to-left rule APL violates mathematical tradition, since in mathematics infix operators are almost invariably non-alphabetic symbols. (It is interesting to note that the designers of POP-2 were aware of this notational convention: in POP-2 the class of names includes sequences of signs e.g. ++, ***, <=> etc., so that true infix operators can be defined.) The APL syntax provides no control structures other than **go to,** the destination of which can be a computed line number. Although the facility of operating on whole arrays largely removes the need for a **for** construction, the **repeat while** is still needed, and has to be achieved by a conditional jump which in my experience is invariably opaque.

Thus the only structuring of APL programs is into functions: within a function there can be no structure. The power of the language is such that it encourages all sorts of trickery (variously described as 'the one-line syndrone' and 'pornographic programming') and therein lies its fascination and its greatest danger. By encouraging the use of APL we are encouraging the production of unstructured programs that are incomprehensible to other programmers (and possibly to their originators). We are discouraging collective 'ego-less' programming: we are going back to the worst days of the assembly-language whizz kids. There is much in APL that we could profitably absorb into other languages (notably the array facilities) but, like marijuanha and alcohol, the unrestrained use of APL is socially unacceptable, and requires restraint.

Yours faithfully,

D. W. BARRON

Department of Mathematics
The University
Southampton SO9 5NH
10 June 1975

To the Editor
The Computer Journal

Sir

Efficient Automatic Overlay—an added advantage of goto-less programming languages

The problem of allocating store dynamically to program segments