what I have classed as 'successful' has been in organisations that have a great deal of technological 'know-how' and have realised that a similar approach might be useful here also. The engineer's attitude to quality control and product testing also has much from which the computing profession could learn with profit.

We must learn from both the successes and the failures. It is for these reasons that I think scale and approach are so very important, and that the requisite knowledge and experience must be available within the university.

Before I close, I want to mount one last hobby-horse. So much of one's efforts in preparing a problem for a machine go into the organisation of the necessary arithmetic that many people appear to lose sight of their final objective. The intermediate entities are all numbers and it is all too easy to produce vast quantities of printed sheets, filled with numbers but very difficult to interpret. Humans have developed some skill in calculating but not many are good at interpreting complex relationships when these are expressed only in numerical form. If the related parts are several pages apart the task is practically impossible. Yet the presentation of output in an intelligible format is rare.

Graphical output is most valuable for presenting results in an informative way and microfilm plotters, producing films, are excellent for showing the behaviour of time dependent processes. There is no other way of doing this, pages and pages of printed figures are useless in this respect. It is generally believed that such plotters are very expensive, yet their cost is comparable with that of the very high speed printers that are now available and which are regarded as indispensable by most installations. I find there is very little realisation of the potentialities of such film-making devices.

Although my own experience has been confined almost entirely to scientific use of computers I believe that much of what I have said about the approach to and organisation of computing projects applies equally to commerce.

To sum up my remarks I can do no better than quote Richard Hamming's famous and elegant epigram, that 'The purpose of computing is insight, not numbers'.

Correspondence

To the Editor
The Computer Journal

Sir,

In spite of the *ad hoc* nature of the FORTRAN language, it is widely used: many proposals for 'improving' the language have unfortunately been adopted unilaterally by compiler writers. Chambers (1971) is to be congratulated for exposing his proposals to public scrutiny, and I should like to add some comments to the inevitable clamour which will follow. Paragraphs of Chambers' paper are referenced in square brackets [ ].

[2.1] *Character data:*
It is proposed that quote signs be accommodated in a character constant by the use of a processor-dependent mechanism. This defies the requirement of upward compatability: the escape mechanism should be fixed by the standard. Actually, it would appear that the old Hollerith constant remains unambiguous in the new language and could be kept as an alternative representation of a character constant.

[2.1(d)] Chambers proposes operators for comparing character expressions of arbitrary length: his footnote concedes that the results will be implementation-dependent. We ask (A) can the dependence be avoided and (B) can a similar result be achieved in a different way. Certainly (A) can be achieved, either by demanding that ASCII be the internal representation or by having a complicated comparison algorithm which effectively translates into ASCII before making the comparison. Let us, however, ask what is the purpose of X ω Y, where ω is a relational operator and X and Y are character expressions of possibly-different length. One major application would be sorting character strings into 'alphameric' order. But a comparison technique which encourages the user to compare the whole string at once is likely to be less efficient than one where characters are compared one at a time from the left. The only essential facility is the comparison of single characters in ASCII order. I propose that this be done, not by hiding a complicated algorithm in a simple-looking facility, but by providing a built-in function IFIXCH, of type INTEGER, which takes a single argument of type CHARACTER and returns the ASCII code value. Note that the internal representation remains arbitrary but the function always returns the ASCII code. Then operations of the type .GT. on the resulting integers will be performed efficiently and will be implementation-independent.

On this arguement alone, the proposal for IFIXCH does not look very exciting. But it permits characters to be used for indexing—a very powerful technique which would be impossible under Chambers' proposals. Hash table management would also be possible. (An inverse of IFIXCH would be provided.)

On the other hand, relational expressions can be formed between character expressions using .EQ. or .NE. without regard to the internal representation, and this construction could therefore be implemented in the new language.

[2.2] *Internal formatted conversion*, and [2.6] *Data transmission:*
To eliminate unnecessary restrictions, I propose that, instead of the format statement number f, the specification should permit a label expression $\mathscr{L}$ (specifying a labelled format statement in the current program body) or a character expression $\mathscr{C}$ (specifying the characters of a format specification explicitly). A FORMAT statement could then be selected on the basis of an indexed label array, or a FORMAT specification generated dynamically in a character array by use of ENCODE.

[2.7] *Array assignment:*
The footnote to section [2.7] mentions an objection made by the referee. Unfortunately the cure is worse than the disease! Chambers' proposal would require the results of the right hand side to be built up in temporary storage and only afterwards transferred to the array specified on the left hand side. Surely the only practical cures are:

(*a*) to complete the evaluation of one element of the array at a time, and to determine the result of pathological cases by fixing the order of evaluation in the specification, or
(*b*) to state in the specification that the effects of pathological cases such as A = A / A(1, 1) are undefined.

Yours faithfully,

A. J. Flavell

Max-Planck Institut für Physik und Astrophysik
8 München 23
Föhringer Ring 6
31 August 1971

Reference

Chambers, J. M. (1971). Another round of FORTRAN, *The Computer Journal*, Vol. 14, pp. 312-314.