(Wilkes, 1973) has an elegant solution in the case of a single process, running in a fixed amount of program store, and programmed in a language whose only control structures are simple sequence and the **while** loop. In this situation it is almost certain which pieces of program will be executed next, and the obvious assumptions about what will not be used for longest lead to the following strategy.

Initially program text is loaded from the beginning until all the available store is used. The process can then be run until it needs text which has not been loaded. The structure of loops enclosing the current point of execution is then examined, and store reallocated so that as many levels of inner loop as possible are completely resident in store, at the expense of outer levels. There will be at most one level which is partly in store and within this level the store is allocated to text next to be executed at the expense of text most recently used.

This strategy minimizes the number of occasions when store is reallocated, under the assumption that each loop is executed a large number of times. It is particularly relevant to overlay from moving head devices, where transfer of a large block of information takes little more time than that of a small block. The number of backing store accesses is L/T per loop cycle for each loop too large to fit entirely in store, where L is the length of the loop and T the total store available.

The other control structures necessary in a practical programming language turn out to cause relatively minor complications. A procedure is deemed to have its body at whichever of its calls is at the innermost level. This still works for recursive procedures, because only those loops enclosing the current point of execution are considered in the allocation strategy. We are therefore concerned only with the actual depth of recursion, not the potential depth.

The if then else and **case** structures pose more of a problem, but it can plausibly be argued that to minimize the number of backing store accesses one should apply the rule of thumb—'if the probability of a piece of program text being used exceeds its length divided by T it should be loaded; if not, not'. The determination of probabilities is, of course, the nub of the whole allocation problem, but the situation where the sum of only two probabilities must be 1.0 is somewhat simpler than that where the entire program is to be considered at once, and one would expect automatically collected statistics to be correspondingly more reliable.

The trade-off of space between program and data, and between independent processes competing for space is not solved by this method, but the ability to estimate the incremental effect of changes of T on each process (particularly the discontinuities at L/T = 1) may well contribute to more effective distribution of store between processes.

Yours faithfully,

M. A. SABIN

Computers and Automation Institute
Hungarian Academy of Sciences
H 1502 Budapest
XI. Kende u. 13-17
Hungary
4 July 1975

Reference

WILKES, M. V. (1973). The dynamics of paging. *The Computer Journal*, Vol. 16, No. 1, pp. 4-9.

To the Editor
The Computer Journal

Sir

Structured programming in APL

A recent paper by Lim and Lewis (1974) claims to present a review of the capabilities of APL with respect to the environment it provides to aid the implementation of structured programs. In this paper, the authors appear to correctly recognise that there are few criteria available to enable a subjective evaluation or comparison of programming languages to be made. However, in their discussion of this point, they have totally ignored the concept of structural entropy as a criterion for estimating the relative degree of disorder in a computer program—and the consequences of this disorder. Entropy concepts provide a starting point for attempting a subjective evaluation of computer programs and hence, indirectly, a comparison of programming languages. The structural entropy concept may be deduced using the work of More (1973) and Rissanen (1973) as a basis for the development of the conceptual schema.

The comprehensiveness of their review is doubtful since:

(*a*) no mention is made of the facilities that exist to enable the implementation of the widely accepted principle of process decomposition, which, in programming circles, turns up in the synonymous form of 'top down' design/implementation methodology, and,

(*b*) the extensive work of Kelly (1972, 1973a, 1973b) or Harris (1973), the earliest of which dates back to 1972, receives no mention.

While overlooking (b) is forgivable, a total dismissal of (*a*) is not as it is now a somewhat aged principle.

Yours faithfully,

P. G. BARKER

Department of Computing
University of Durham
Science Laboratories
South Road
Durham DH1 3LE
28 July 1975

References

LIM, A. L., and LEWIS, G. R. (1974). Toward Structured Programs in APL, *The Computer Journal*, Vol. 18, No. 2, p. 140.

MORE, T. (1973). Axioms and Theorems for a Theory of Arrays, *IBM J. Res. Devel.*, Vol. 17, p. 135.

RISSANEN, J. (1973). Bounds for Weight Balanced Trees, *IBM J. Res. Devel.*, Vol. 17, p. 101.

KELLY, R. A. (1972). APLGOL, A Structured Programming Language for APL, Report No. 320-3299, Palo Alto Scientific Centre.

KELLY, R. A. (1973). APLGOL, An Experimental Structured Programming Language, *IBM J. Res. Devel.*, Vol. 17, p. 69.

KELLY, R. A., and WALTERS, J. R. (1973). A Structured Programming System for APL, Report No. G320-3318, Palo Alto Scientific Centre.

HARRIS, L. R. (1973). A Logical Control Structure for APL, Proceedings of the 1973 APL Congress, p. 203, North-Holland/American Elsevier.

To the Editor
The Computer Journal

Sir

The paper 'Insuring Computers', by K. J. Allen, Vol. 18, No. 3 of August 1975, was a clear and comprehensive exposition on this subject; but I felt that when summarising the less familiar types of insurance under 'Miscellaneous Risks' he should have mentioned the potential importance of Fidelity Guarantee insurance when computers are being used in financially sensitive areas.

Briefly, the purpose of Fidelity Guarantee insurance is to indemnify the employer for money stolen, obtained by fraud, or embezzled, by employees. This risk is not normally covered by other types of policy. In view of several spectacular computer frauds, some involving very large sums, this type of insurance should be considered.

The employer has a choice between 'blanket' policies and individual policies. A blanket policy covers all employees, or all employees of a class, while individual policies cover named employees. When large numbers of employees are concerned, blanket policies are usually cheaper because the insurance company does not usually make enquiries into the backgrounds of the individual employees concerned; but many employers take the view that they want such enquiries made and are prepared to pay for the service. This can be achieved through individual policies (or by paying extra premium for such enquiries to be made under a blanket policy).

Fidelity Guarantee policies may cover only frauds and embezzlements which are committed during the currency of the policy. When a policy is introduced covering persons who are already employees of the company, consideration should be given to covering what insurance people call the 'retro' risk, i.e. the retrospective possibility that a fraud or embezzlement had already been com-

mitted by the persons concerned, but had not yet been detected.

Yours faithfully,

A. PARKIN

School of Mathematics Computing & Statistics
City of Leicester Polytechnic
P.O. Box 143
Leicester LE1 9BH
7 October 1975

Mr. Allen replies:

The point made by Mr. Parkin is a valid one, but in practice fidelity guarantee policies for all but the smallest risks are issued to cover all employees, and would include computer personnel. It is, therefore, likely to be more a problem for the insurer than the insured, although the computer user would be well advised to check that the insurance does cover computer staff.

To the Editor
The Computer Journal

Sir

I am looking for a computer program written in FORTRAN to solve large scale capacitated network problems. I have noted the ALGOL program published in *The Computer Journal*, Vol. 16, August 1973, but would be most grateful if you would send me, or refer me to a FORTRAN equivalent thereof.

Yours faithfully,

B. JOFFE

B. L. Joffe Associates (PTY) Ltd.
49a Garden Road
Orchards
Johannesburg 2001
South Africa
25 November 1975

To the Editor
The Computer Journal

Sir

I read with interest R. A. Earnshaw's article entitled 'Is APL a viable programming language?' in Vol. 18, No. 4 of the *The Computer Journal*. However, the conclusion attributed to Streeter (1972) that execution of APL programs usually costs a factor of ten to a hundred times more than equivalent FORTRAN or PL/1 compiled program code hardly does justice to the interpretive efficiency of current APL processors.

My own substantial experience in writing APL systems and providing support to client users of the Atkins Computing time-sharing service has led me to a rather different conclusion. APL execution costs compare very favourably with FORTRAN compiled code. In fact, in a series of bench-mark tests involving large matrix operations, the APL code was actually faster than equivalent FORTRAN code—Mr. Earnshaw indicated why this might be possible in Section 6.1(*g*) Flexibility in execution.

I fully endorse Streeter's finding that program development time is considerably reduced using APL—a factor of three vis-a-vis FORTRAN or PL/1 is indicated. Even more impressive is the reduction in development computer costs that might be expected; our own experience, a tenfold reduction is not unusual. The same order of savings are to be expected for program maintenance or enhancement.

Taking both of these factors into account, the development of APL systems for long-term usage becomes a very attractive proposition, and one to which we at Atkins Computing are increasingly drawn. Moreoever, the base of APL applications implemented on our installation includes not only the 'accepted' scientific and technical areas, but also those within the commercial sphere.

In fairness to Mr. Earnshaw, I note that the article was submitted over a year ago, and that he acknowledges the implementation of many enhancements (including 'a faster processor'), which presumably were not available when the article was researched. I hope that my letter rectifies somewhat the disservice to APL.

Yours faithfully,

A. D. CROSSLEY

Atkins Computing Services Limited
Woodcote Grove
Ashley Road
Epsom
Surrey KT18 5BW
27 November 1975

Algorithms supplement

We are sorry to have to announce that, owing to pressure of work, Mr Shepherd has found it necessary to relinquish the editorship of the supplement. We are pleased, however, that the task is to be taken on by Professor Paul Samet.

To smooth the transition from one editor to another Mr Shepherd has agreed to complete the processing of all material in the pipeline; contributors should continue to send communications regarding current material to him.

New contributions should be sent to:

Professor P A Samet, Director, Computer Centre, University College London, 19 Gordon Street, London WC1H 0AH.

Mr Shepherd has edited the supplement for over four years. During this time it has continued to develop and we are most grateful for the time which he has devoted.