Correspondence

To the Editor
The Computer Journal

Sir

I was delighted to see Chung-Phillips' and Rosen's note on dynamic storage allocation in FORTRAN appear in the literature (*The Computer Journal*, Vol. 18, pp. 342-343). As the authors correctly point out, the utility and flexibility of the majority of applications programs written in FORTRAN are marred by that language's woefully static memory management philosophy.

In the interests of readability, modularity, and portability, however, I wish to call attention to three details of implementation which tend to make the technique more serviceable in practice.

1. Responsibility for storage allocation should be delegated to a separate routine, say IALOCF (ARRAY, LENGTH) which simply returns the index of the first word of an allocation relative to the specified base ARRAY. This not only isolates the user program from the details of memory management, but also permits easy generalisation of the technique to several dynamic storage areas simultaneously.
2. Most modern operating systems provide mechanisms to allow user programs to request and/or release actual core store during the course of a job (viz. RFL command in SCOPE/KRONOS) and many loaders will arrange to link blank COMMON last, in order that it may be subsequently extended. Making use of these features in conjunction with (1) above, truly dynamic systems are easily achieved.
3. Finally, I must protest that, according to the 1966 ANSI FORTRAN standard, X3.9-1966, variable dimensioning information cannot be communicated through COMMON (blank or labelled) but must be passed via the formal parameter list (section 7.2.1.1.2). On some machine architectures, particularly those without indirect addressing, use of such multi-dimensional array arguments can be up to twice as expensive computationally as to those in common storage; but in today's world of exponentially diverging hardware and software costs, even this loss is probably tolerable in the interests of flexible program design.

In closing, if I may be so bold, I should like to propose that a proper survey article of data structuring/management techniques in FORTRAN, particularly as they affect production scientific codes, is long overdue.

Yours faithfully,

J. T. HASTINGS

Atmospheric Technology Division
National Centre for Atmospheric Research
PO Box 3000
Boulder
Colorado 80303
USA
4 February 1976

To the Editor
The Computer Journal

Sir

With reference to the article 'A note on dynamic data storage in FORTRAN IV' by Alice Chung-Phillips and R. W. Rosen in *The Computer Journal* (Vol. 18, pp. 342-343; November, 1975), I was most surprised to realise that the technique described had not been published before although I for one have been using it for almost as long as I can remember (since about 1963!) when variable dimension statements were first allowed in FORTRAN.

I am writing to explain how with a slight extension it is possible greatly to improve the efficiency of programs in which array sizes are dependent on the input parameters. C-P and R state that if the size of preassigned storage space is not suitable then only *two statements* need to be adjusted. This is neither necessary nor desirable—I sometimes make a mistake in changing just one statement!—and the program has to be recompiled because of these changes. The following method requires a once and for all compilation.

The Control Data Cyber 76 computer, using the standard SCOPE operating system, works in a multiprogrammed mode and it is advantageous, for getting fast throughput for a given program, for the program to be as small as possible. SCOPE is able to schedule small programs for execution more easily than large ones. In the Cyber 76 small core memory (SCM) is used for execution and storage of some data but large core memory (LCM) is used for extensive data and for working space by SCOPE. When elements in data arrays are to be accessed sequentially by a program the machine works faster if the data are in LCM—for hardware reasons. The Cyber 76 FORTRAN compiler is such that blank COMMON blocks are placed at the end of all other information—program and labelled COMMON—this is the key to success for the technique.

First let us assume we have decided to put the data in LCM, this is achieved by using the FORTRAN LEVEL statement and—using the notation of the referenced paper—we write LEVEL 2, A. Also we put the array A into *blank* COMMON thus COMMON/ /A(1) and note particularly that the dimension of A is one—the smallest possible value. Now we follow C-P and R and compute the sub-array sizes and in particular ITOTAL, the total space required. We can check this against NTOTAL but in our case NTOTAL is pre-set once and for all as the *maximum* number of words of LCM that any program can use. If in the statement IF(ITOTAL.GT.NTOTAL) CALL EXIT the CALL EXIT is executed then we must rethink our problem but if not we encounter the statement CALL MEMORYL (ITOTAL). This causes SCOPE to change the LCM space allocation to be ITOTAL which is the size we now know to be required and we have never asked for more space than is absolutely necessary. In fact, even NTOTAL and the IF statement are redundant because the system knows how much LCM is available and if the CALL MEMORYL statement cannot satisfy the requirement then the job aborts with an appropriate message.

We can go further, suppose in the example we do not require KEY and IANS after a certain stage in the program (such arrays are deliberately placed last in the order of the sub-arrays) then at this state we execute CALL MEMORYL(I1) and scope will reduce the LCM allocation accordingly.

It remains only to explain that if we do not wish to use LCM because either our data is relatively small or it is not accessed sequentially then the only changes required to the above in order to use SCM instead are:

(i) omit the LEVEL statement and

(ii) insert IPROG = MEMORYS(0)
and ITOTAL = 12 + NS*NP − 1 + IPROG
CALL MEMORYS(ITOTAL)
in place of ITOTAL = 12 + NS*NP − 1
and CALL MEMORYL(ITOTAL) used previously.

The function MEMORYS(0) returns the current size of SCM i.e. after compilation and the allocation of space for arrays with fixed dimensions, hence the ITOTAL computed is now the total SCM required for everything. CALL MEMORYS(ITOTAL) reserves the total SCM space required.

In conclusion, I should point out that this technique is also applicable to Control Data lower Cyber computers and probably to many others but in some cases there is insufficient flexibility in the monitor to allow this dynamic adjustment of storage space. Furthermore, most charging algorithms take account of *how much* SCM and LCM is used and for *how long*, so it is to the advantage of the