Letters to the Editor—Cont'd from page 297

A Comment

Dear Editor:

I wish to comment on the significant number of contributions to Pracniques, Letters to the Editor and other departments of *Communications* that are concerned solely with methods of circumventing unpleasant characteristics of IBM 7090 series machines or of some specific operating system for those machines. There can be no question that this information is of great importance to a large segment of the profession. Since such material seldom provides the slightest contribution to the state of the art, however, and since a mechanism, the *SHARE Secretary Distribution*, exists for exactly this purpose, I question its suitability in a professional publication.

Although the SSD does not offer its contributors the prestige of formal publication, it does offer two overwhelming advantages: speed of distribution and distribution to exactly the intended audience. Thus, those who distribute such material through the SSD clearly consider service to the profession to be the more important consideration.

Rejection of such material by the *Communications* would, furthermore, avoid cluttering up a professional publication with unprofessional filler, and would repudiate what could well become a precedent for absurdity. (Has anyone yet proposed an ACM SIG for IBSYS users?)

CONRAD H. WEISERT
Applied Physics Laboratory
The Johns Hopkins University
8621 Georgia Avenue
Silver Spring, Maryland

ED. COMMENT. Contributions of temporary interest to particular users groups should certainly not be published in the *Communications*. Yet technical points about widely used languages and machines are appropriate in a Techniques Department. When does a technical point become too specialized?—C.C.G.

On the Recursive Programming Techniques

Dear Editor:

J. A. Ayers' "Recursive Programming in FORTRAN II" [*Comm. ACM 6* (Nov. 1963), 667] is a clever and potentially useful device. It is, however, subject to pitfalls if the object code produced by the compiler is not carefully considered.

Consider the example below of a recursive routine to calculate binominal coefficients from the formula $C(m, n) = C(m, n-1)(m-n+1)/n$:

```
 1  SUBROUTINE BCOEFN (M, N, KOEFN, DUMMY)
 2  IF (N) 3, 4, 6
 3  CALL EXIT
 4  KOEFN = 1
 5  GO TO 10
 6  CALL STORE (N)
 7  CALL DUMMY (M, N-1, KOEFN, DUMMY)
 8  CALL RSTOR (N)
 9  KOEFN = KOEFN * (M-N+1)/N
10  RETURN
```

When statement 7 is executed and the routine is reentered, the original initialization of N is overwritten by an assignment to a temporary cell within the routine BCOEFN. When statement 9 is executed, the expressions $(M-N+1)$ and N are assigned to the same location, giving incorrect value to KOEFN.

It is, of course, easy to code around this particular difficulty by replacing statement 7 by two statements:

```
N = N-1
CALL DUMMY (M, N, KOEFN, DUMMY)
```

Mr. Ayers used this device even though his example did not require it. The present example illustrates the requirement that the argument list in the CALL DUMMY statements should be identical with the one in the SUBROUTINE statement.

If the recursive routine contains variable subscripts, DO loops, or computed GO TO statements, index registers 1 and 2 will not be properly restored. This will ordinarily give trouble in the program calling on the recursive procedure. Variable subscripts may be avoided by referring to arrays through separate subprograms such as FETCH and STORER, given below:

```
SUBROUTINE FETCH (A, I, B)
DIMENSION A(1)
I = I
B = A(I)
RETURN
SUBROUTINE STORER (A, I, B)
DIMENSION A(1)
I = I
A(I) = B
RETURN
```

O. C. JUELICH
Solid Mechanics Research
North American Aviation, Inc.
4300 E. Fifth Ave.
Columbus 16, Ohio

Dear Editor:

Mr. Juelich has discovered two important limitations of the recursive programming technique [*Comm. ACM 6* (Nov. 1963), 667]: the necessary identicality of the argument lists of the SUBROUTINE statement and the CALL DUMMY statement, and the destruction of the index register 1 and 2 information.

I see no remedy for the first difficulty, but the second may be avoided by dimensioning the LD variable and saving LD(1), LD(2), LD(3) in the STORE subroutine and restoring LD(3), LD(2), and LD(1) in the RSTOR subroutine.

JAMES A. AYERS
Mathematics Dept.
Research Laboratories
General Motors Corp.
12 Mile and Mound Rds.
Warren, Michigan

More on "Simple I/O" Statements

Dear Editor:

As the author of the program in question, I should like to reply briefly to Prof. Galler's remarks concerning the "complicated" state of affairs with the CNV format-free input subroutine, in his letter in the January 1964 issue of the *Communications*.

As he states in his letter, the key to the implementation of simple free format I/O is in the use of a symbol table obtained at compile-time indicating mode, dimension and storage allocation.

Unfortunately, we at M.I.T. were not in the happy position of writing our own compiler. Our aim was to alleviate the input format situation in the already existing IBM FORTRAN II compiler without becoming involved in tinkering with it (and possibly also the loader). Therefore, we had no access to any such symbol table.

Herein lies the reason for the majority of the complications apparent in the October 1963 article by Barnett, Futrelle and myself, since we are thus forced to ask the user to transmit the necessary symbol table type information to the CNV subroutine through CALL's to its LIST, LIST1D, LIST2D and LIST3D entry-points, when he proposes to have order-independent or array input.

If he is prepared to pay such a price, then, the user of the regular IBM FORTRAN may now also have format-free input as well as his MAD counterpart. (He may have order-*dependent* input *without* setting up his symbol table.) Moreover, I think that on input he will have as much power and flexibility.

To make a direct comparison, take for example the data-card cited in Prof. Galler's letter:

```
A = 3.2,    C = $AB$,    M(3) = 8.12, 9.34, 1.2*
```

To achieve the same effect the CNV subroutine user would punch his data-card:

```
A = 3.2,    C = 2HAB,    M(I) (I = 3, 5) ARE 8.12, 9.34,
                         1.2
```

Instead of "READ DATA" he would write "CALL CINCV (NTAPE, ISEOF)" where NTAPE and ISEOF are tape and end-of-file condition specifications, respectively. In addition, the following two statements would be required at the head of the program for the symbol table, but are only necessary once, however many subsequent inputs to A, C and M occur.

```
CALL LIST (1HA, A, 1HC, C)
CALL LIST1D (1HM, M, N)
```

where N is the dimension of M.

Finally, I must say that it is heartening to discover that at least one *writer* of compilers appreciates the need for format-free input (and output!) in many situations. Also, when incorporated *into a language* such a feature should, I agree, be kept simple, even though this may not be possible otherwise.

MICHAEL J. BAILEY
IBM Data Systems Division
545 Technology Square
Cambridge 39, Mass.

On Polyphase Sort

Dear Editor:

We have recently (November, 1963) implemented a polyphase sort on KDF9, using backwards-reading of the magnetic tapes (any number from 3 to 9). This may be of interest to your readers since as far as we know no backwards-read polyphase sort has previously been implemented, although the possibility was discussed by Gilstad [2].

The logic was worked out independently during 1962: the string distribution is done as in Malcolm's paper [1], except that we have used "horizontal" dummy distribution, so that whenever possible dummies are merged with dummies; but the method of adjusting string directions is not that of Gilstad [2].

We started with a slightly more general requirement than is referred to in [2], namely, the output could be written to any chosen tape, possibly even the input (allowing optional change of reels). The first string on each tape is written in the opposite order to that of the final output file. The state of the tapes at the end of the internal sort phase, using Malcolm's notation, is that either all the t's are odd (output for any tape, or chosen to be on the input tape) or that one t is odd and the remainder are even (which can occur whatever destination is wanted). In the former case merging can start at once, but in the latter, adjustments must be made to the tape with odd t.

Consider first the case where the output is not chosen to be on the input tape. Normally we assume that the dummies on each tape are at the back end of the real strings so that the dummies are read back first. The adjustment that is now done is to transfer one of these to the front end of the data, reversing its (notional) direction, so that the real strings get "shifted back" one string, making the direction of the first real or dummy string to be read back the same as on the other tapes. (So, it is essential that the dummies are notional and not represented physically on tape.) Of course this process works only when there is at least one dummy on the deck with odd t. This we guarantied by never allowing the sum of d's to reduce to zero during the presort when the output is specified for a tape other than the input, and by ensuring that the last tape to have its d not zero is the deck with odd t. (For writing the next string we made the rule: choose the deck with greatest d, giving priority to one with even t.)

In the case where the output is to be for the input tape and only one t is odd at the end of the presort, then the strings on the tape with odd t must be passed to the (former) input tape. We considered that this pass of a fraction of the data is more efficient than building up to the next "all t's odd" pattern during the presort.

REFERENCES:

1. MALCOLM, W. D., JR. String distribution for the polyphase sort. *Comm. ACM 6*, 5 (May 1963), 217–220.
2. GILSTAD, R. L. Read-backward polyphase sorting. *Comm. ACM 6*, 5 (May 1963), 220–223.

D. T. GOODWIN AND J. L. VENN
English Electro-Leo Computers, Ltd.
Kidsgrove
Stoke-on-Trent, Staffordshire
England

More on SLIP

Dear Editor:

Mr. L. D. Yarbrough raised some questions about SLIP in his letter in the January 1964 issue of *Communications*. We have also obtained a deck of SLIP and compiled it on both our 1604 and 3600 computers. The only change we had to make to the source deck was to declare START in line 634 to be TYPE INTEGER. The other errors which Mr. Yarbrough detected are allowable statements to our compiler. The respective compile times were 3 minutes 20 seconds on the 1604, and 59 seconds on the 3600. In addition, we have correctly executed several test problems for SLIP on the 1604.

SANFORD ELKIN
Control Data Corporation
3330 Hillview Ave.
Palo Alto, California