Comment on Analytic Differentiation by Computer

Dear Editor:

In the article by James W. Hanson, et al., "Analytic Differentiation by Computer" [*Comm. ACM 6* (June 1962), 349–355], the authors make the following statement with respect to the final or output phase of the program: "The precedence list for the operators is employed in this algorithm in order to avoid the insertion of redundant parentheses into the output string." If one uses, in output, the precedence list given in the article, parentheses will not always be inserted correctly. The expression

$$(X * A - (A * X + B))/XP2$$

when broken down into the parentheses-free form by the Ershov algorithm and then reconstructed by the output algorithm (without differentiating) becomes

$$(X * A - A * X + B)/XP2 .$$

This, and similar errors, may be corrected by using the following precedence list,

$$p(.) > p(P) > p(\boxminus) > p(/) > p(*) > p(-) > p(+) > p(\,(\,) = p(\,)\,) > p(\vdash) = p(\dashv).$$

In passing, it should be noted that in the flow chart given in Fig. 3, there should be a $(j \leftarrow j - 1)$ immediately after the test $(j > \delta + 1)$.

GARY BOSWELL
Bell Helicopter Co.
Fort Worth 1, Tex.

An Algebraic Compiler for the FORTRAN Assembly Program

Dear Editor:

An algebraic compiler has been written which may be added to the FORTRAN Assembly Program. This compiler will expand all algebraic statements with the following operations: addition, subtraction, multiplication and division. It will compile multilevel expressions in floating-point arithmetic (this can easily be revised to fixed-point). It is called for by a pseudo-operation (EXPR) requesting a compilation of the expression found in the variable field. The method of compiling has been fashioned after that described by D. J. Dijkstra [1]. In brief, the expression is translated into a Polish string and the string is then compiled. The compilation is in floating point and the number of arguments is a function of table lengths.

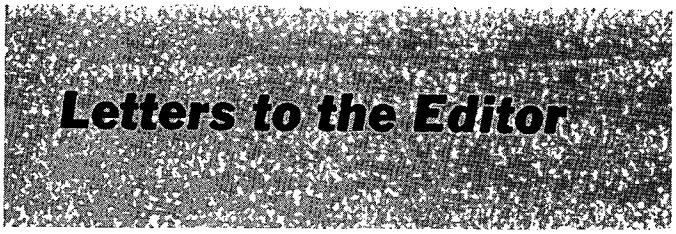
One additional feature of the compiler is the ability to compile indirect addressing into an expression. At the suggestion of W. H. Wattenburg, the character string left parenthesis, name, right parenthesis, i.e., (name), can uniquely define indirect addressing on the name within the parentheses. This feature has been incorporated into the compiler. It has been thought that it will be useful since the compiler is contained in an assembly program.

Another feature is the ability to handle multiple assignments in a number of ways. The following expressions are equivalent:

```
EXPR  A=B=C+D/E
EXPR  A=(B=C+D/E)
EXPR  A=C+D/E=B
```

Also, there is no necessity for an assignment character. If there is no assignment, the result will be left in the accumulator register. A by-product of multiple assignments is that a *portion* of an expression may be stored. Thus:

```
EXPR  A=(B=C+D)/E
```



It is felt that this will be a useful tool in scientific computation.

Example:

```
SQRT   SXA    XR4,4
       STO    N
       AXT    4,4
       EXPR   XZERO=5.  1st approximation to root
ITER   EXPR   XZERO=XZERO*XZERO
       EXPR   XZERO=XZERO((3*N+XZERO2)/
                (3*XZERO2+N))
       TIX    ITER,4,1
       CLA    XZERO
XR4    AXT    **,4
       TRA    1,4
N      PZE
XZERO  PZE
       END
```

REFERENCE:
1. DIJKSTRA, D. J. Making a Translator for ALGOL 60. Automatic Programming Information #7 (May, 1961), Mathematisch Centrum, Amsterdam.

A. D. STIEGLER
Lockheed Missiles and Space Co.
Palo Alto, Calif.

More on Testing BCD Words with FORTRAN

Dear Editor:

The article "Low Level Language Subroutines for Use Within FORTRAN", by M. P. Barnett appeared in the November 1961 issue of the *Communications* (pp. 492–495) and was subsequently followed by letters to the editor from Otto Mond and R. E. Dickie in the February 1962 (p. 78) and June 1962 (p. 364) issues respectively. In each instance a method for comparing two bcd words was discussed. This note points out a rather convenient method of performing this test.

Since some installations may not have the available software, it should be noted that this method requires the 32K 709/7090 FORTRAN system having the Boolean arithmetic statement feature.

Using the FORTRAN Boolean operations, the test may be derived using the following notion of set theory: Let A and B be two arbitrary BCD words; then $A*(-B) \equiv 0$ in case $A < B$ (i.e., an appropriate masking of B will yield A); similarly $-A*B \equiv 0$ in case $B < A$. Hence, the expression $A*(-B) + (-A)*B$ will be zero in case A is identical to B, leading to the **if** statement

$$B \quad \textbf{if } (A*(-B) + (-A)*B)\ N_1\ ,\ N_2\ ,\ N_3$$

where N_2 is the statement transferred to in case A is identical to B, and N_1 is the statement transferred to otherwise.

C. A. OSTER
EDP Operation
General Electric Co.
Richland, Wash.