Letters to the Editor

ALGOL: Pleasure through Pain

Dear Editor:

After working through the description of ALGOL 60 and also through your special issue on compiler techniques, I feel impelled to bring to the attention of your readers a word which they may find quite useful in the next few years. As defined by *Webster's New International Dictionary*, it is

Algolagnia: The finding of pleasure in inflicting or suffering pain.

G. M. WEINBERG
IBM Systems Research Institute
787 United Nations Plaza
New York 17, N. Y.

Negative Binomial Probability Distribution Tables

Dear Editor:

A FORTRAN program to compute Negative Binomial Probability Distribution Tables is now available.

The program operates in two modes: (1) input of a single mean and variance to output a single table, and (2) input of a single mean and initial, incremental, and final variances to output a series of tables. Unlimited input in a chosen mode is restricted to one card (record) per input.

Output tables include a heading, page number, associated mean and variance, the numerical value of the variables Q, K, $1 - \sum_i P_i$, P_0, and four successive probabilities across each line. Each supplementary page of a table has its own page number and a modified heading. Up to 1,000 probabilities can be computed and printed in floating-point mode, from P_0 to P_{999}, with a numerical threshold chosen by the user.

Copies of the program, with operating instructions, will be provided upon request. All inquiries should be addressed to:

Logistics Research Project
The George Washington University
707 22nd Street, N. W.
Washington 7, D. C.
Attn: M. Hershkowitz

M. HERSHKOWITZ

Concerning Ferguson's Paper on Fibonaccian Searching

Dear Editor:

Concerning the paper "Fibonaccian Searching" by David E. Ferguson which appeared in the December issue of the *Communications*, some readers may be interested to know that the algorithm given is optimal not only as $n \to \infty$ but also for any finite n. See J. Kiefer, "Sequential Minimax Search for a Maximum," *Proc. Am. Math. Soc. 4* (1953), 502–6, and Selmer Johnson, RAND Corp. Paper P-856 (1956), where a simpler proof of the same result is given. Further discussion appears in the text *Dynamic Programming* by Richard Bellman, p. 34 (Princeton (1957)).

M. L. JUNCOSA
RAND Corporation

SHARE Committee

Dear Editor:

The March issue of the *Communications* summarized the work of the SHARE UNCOL Committee. The report failed to show that this work was the joint contribution of the entire SHARE Committee.

As most of the readers of the *Communications* are aware, the current members of the SHARE Committee are:

PHILIP BAGLEY,	The Mitre Corporation
MORT BERNSTEIN,	The RAND Corporation
ELAINE BOEHM,	International Business Machines Corporation
WILLIAM DOBRUSKY,	System Development Corporation
THOMAS B. STEEL, JR.,	System Development Corporation
FRANCIS V. WAGNER,	North American Aviation, Inc.

All of them have made important contributions to UNCOL. I apologize for the oversight which resulted in the omission of their names.

T. B. STEEL, Jr., *Chairman*
SHARE UNCOL Committee

On Grau's Recursive Processes and On Compiling Expressions in ALGOL

Dear Editor:

I have found the January, 1961 issue of the *Communications* very interesting and helpful. In particular, I have carefully studied the article by A. A. Grau, "Recursive Processes an ALGOL Translation". There seem to be a few errors in this article, but I have figured out changes that will make Grau's method work properly (see below).

However, in checking over this method, I discovered something about ALGOL which seems to increase the difficulty of compiling expressions. Consider, for example, the following expression:

$$\textbf{if } B \textbf{ then } x \textbf{ else } y < 0 \qquad (1)$$

This is obviously a boolean expression because the relational operator $<$ appears. Also, y is evidently an "arithmetic" variable (*real* or *integer*), and B is evidently a *boolean* variable. But, it is not possible to tell from the context whether x is *boolean* or arithmetic. Furthermore, this expression cannot be compiled properly without knowing what type of variable x is. If x is *boolean*, then the expression is equivalent to:

$$\textbf{if } B \textbf{ then } x \textbf{ else } (y < 0) \qquad (2)$$

On the other hand, if x is *real* or *integer*, then the expression is equivalent to:

$$(\textbf{if } B \textbf{ then } x \textbf{ else } y) < 0 \qquad (3)$$

In the latter case, the expression in parentheses is a conditional arithmetic expression.

Grau's method handles expression (1) as though x were a boolean variable. It is possible that other ALGOL compilers would have trouble with this type of expression.

One can, of course, design a compiler to handle such expressions. Grau's method can do it if one increases the number of control states, and separates the arithmetic operators and the boolean and relational operators into different classes, instead of lumping them all together.

Another solution to the difficulty would be to make a change in ALGOL. All that is needed is to change the definition of "relation" (Section 3.4.1) to:

⟨relation⟩ :: = ⟨simple arithmetic expression⟩ ⟨relational operator⟩ ⟨simple arithmetic expression⟩

With this change, expression (1) would be treated as in (2), whereas to get the meaning of (3) it would be necessary to use parentheses, or a form such as:

if B **then** x < 0 **else** y < 0

I do not think this restriction of ALGOL would impose undue hardships, and it would certainly simplify the translation problem.

Corrections to Grau's process:

Procedures COMPEX, IF, and THEN should be changed to the following forms:

COMPEX **if** prec(γ[g]) ≦ prec (ω) **then begin** EXB; rep **end**
else begin Ent(E2, γ[g]); Ent(0) **end**; **go to** next.

IF {**if** ¬ C(α[a]) **then go to** ——} → π;
if α[a] = 'η[h]' **then** h := h − 1;
α[a] := address of this instruction.

THEN Set address in C(α[a]) to next larger instruction address;
a := a − 1.

In the translation matrix, the entry in column E1 opposite ω should be Ent(E2,ω) | Ent(0).

GILBERT A. BACHELOR
Oregon State University
Corvallis, Oregon

A Program Rack

Dear Editor:

The unexpected discovery that additional instructions must be sandwiched in between lines of a sequence, presumed complete, is not an infrequent experience. The contrivance pictured here was constructed in an experimental attempt to expand the pages of a program, that is, to impart flexibility to them. It provides for manual rearrangement of the sequence of instructions and for the insertion or removal of part of them. It may be used either during coding or debugging.

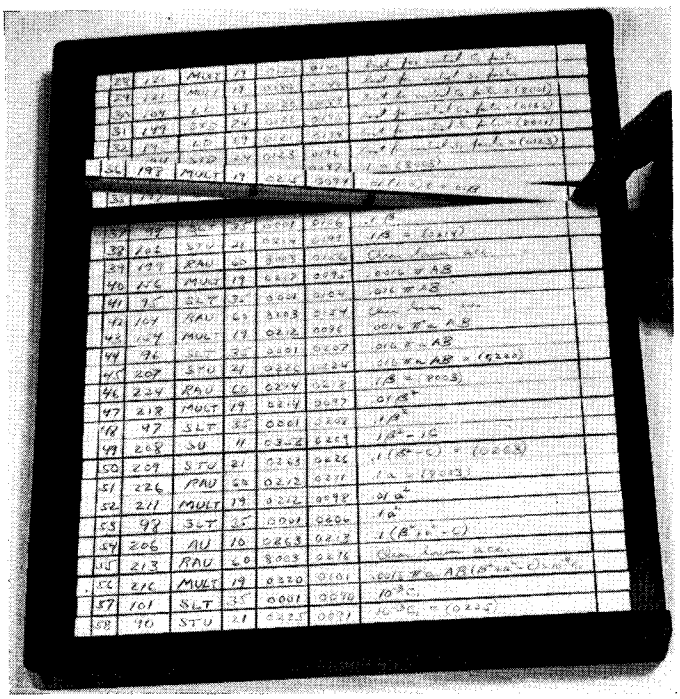
Each element within the rack can contain one line of instruction, that is, one code word. (In this case the elements are scribed for basic IBM 650 coding.) The white plastic surface accepts handwriting with an ordinary pencil. Marks can be erased as quickly and easily as those on paper. The under side of each element is tapered near its outer edge so that depression at this point will cause the opposite end to kick up and separate itself from the others.

Three small cylindrical magnets are built into the body of each element. The ends of two of these can be seen in the photograph.

Grouped upon a desk the elements—as many as needed—can be positioned to show the path of the computation. Thus, they serve in the dual role of program and flow chart and, in effect, constitute a combination of the two. This combination is particularly helpful for those of us who like to write the program first and construct the flow chart in finished form afterward. If photographic equipment, practicable for the purpose, is available, the completed program can be recorded and then wiped away to ready the devices for re-use.

Actually this system aids in programming problems from the inside out, that is, in coding the elementary routines before attending to their linkage with the main program. It may serve, also, as a training device for students of programming. Replacement of instructions by their modified form can be performed in analogy to the varying paths taken within a computer during the course of a computation.

For practical use the pilot model illustrated here proved to be a little heavy and somewhat cumbersome in handling. The thickness, 0.3 inches, of the elements is excessive. Writing upon them when the hand is resting upon their plane of support is comparable to writing upon the last line of a full pad. All these difficulties can be eliminated, of course, with further words on design.



Among other possible improvements is the inclusion of columnar headings in the rack. The magnets could have been positioned symmetrically about the center of the long dimension in order to utilize their repelling forces when an element is unintentionally inverted.

The principle of the system can be economically applied by perforating along the horizontal lines of program sheets. Use of heavy paper stock for the purpose would eliminate much of vulnerability to window blasts or to persons breezing by the desk. The elements of the completed program could be placed upon a sheet with a surface treated for adherence.

GEORGE E. REYNOLDS
Air Force Cambridge Research Laboratories
Cambridge, Mass.

(Letters Continued on Page 284)

ON THE APPROXIMATION OF CURVES BY LINE SEGMENTS USING DYNAMIC PROGRAMMING

The RAND Corp., Santa Monica, California
RICHARD BELLMAN

Introduction

In a recent paper [1], Stone considered the problem of determining the $2N + 2$ constants, $a_i, b_i, \quad i = 1, 2, \cdots, N + 1$ and the N points of subdivision $u_1, u_2, \cdots, u_N$ so as to minimize the function

$$F(a_1, a_2, \cdots, a_{N+1}; b_1, b_2, \cdots, b_{N+1}; u_1, u_2, \cdots, u_N) = \sum_{j=1}^{N+1} \int_{u_{j-1}}^{u_j} (g(x) - a_j - b_j x)^2 \, dx, \tag{1}$$

where $u_0 = a, \quad u_{N+1} = b$ and $a \leqq u_1 \leqq u_2 \leqq \cdots \leqq u_N \leqq b$. For the case where $g(x)$ is quadratic, one can look forward only to a simple computational algorithm for the solution. In this note we wish to show how dynamic programming provides a solution in these terms.

Basic Recurrence Relation

Let us write, for fixed a and $b \geqq a$,

$$f_N(b) = \min_{[a_i, b_i, u_i]} F(a_1, \cdots, a_N; b_1, \cdots, b_N; u_1, \cdots, u_N). \tag{1}$$

Then

$$f_1(b) = \min \left[\int_a^{u_1} (g(x) - a_1 - b_1 x)^2 \, dx + \int_{u_1}^{b} (g(x) - a_2 - b_2 x)^2 \, dx \right], \tag{2}$$

where the minimum over $-\infty < a_1, a_2, b_1, b_2 < \infty$, $a \leqq u_1 \leqq b$. This function is readily determined since we can compute the minima over the a_i and b_i and then minimize over u_1 by means of a discrete search.

It is easy to see that for $N \geqq 2$ we have

$$f_N(b) = \min_{0 \leqq u_N \leqq b} \left[\min_{[a_N, b_N]} \int_{u_N}^{b} (g(x) - a_N - b_N x)^2 \, dx + f_{N-1}(u_N) \right]. \tag{3}$$

This is a particular application of the principle of optimality [2, 3]. Since the minimum over a_N, b_N can be calculated explicitly, we can write (3) in the form:

$$f_N(b) = \min_{a \leqq u_N \leqq b} [h(u_N, b) + f_{N-1}(u_N)] \tag{4}$$

A computational solution along these lines requires a few seconds per stage, where N is the number of stages; see [3].

Extensions

It requires very little additional effort to approximate to $g(x)$ by quadratic polynomials, or by polynomials of any fixed degree. Similarly, we can compute the minimum of

$$\sum_{j=1}^{N+1} \int_{u_{j-1}}^{u_j} (g(x) - h(x, a_j, b_j))^2 \, dx, \tag{1}$$

provided that we know how to minimize the function

$$\int_{u_N}^{b} (g(x) - h(x, a_N, b_N))^2 \, dx \tag{2}$$

over a_N and b_N, or the minimum of

$$\sum_{j=1}^{N+1} \max_{u_{j-1} \leqq x \leqq u_j} |g(x) - h(x, a_j, b_j)|. \tag{3}$$

REFERENCES

1. STONE, H. Approximation of curves by line segments. *Math. Comput. 15* (1961), 40–47.
2. BELLMAN, R. *Dynamic Programming.* Princeton University Press, Princeton, N. J., 1957.
3. BELLMAN, R., AND DREYFUS, S. *Computational Aspects of Dynamic Programming.* Princeton University Press, Princeton, N. J.; to appear.

Letters (Continued from page 253)

Irons' Procedure DIAGRAM

Dear Editor:

I would like to make some comments on the excellent and stimulating article, "A Syntax Directed Compiler," by Edgar T. Irons in the *Comm. ACM* issue of Jan. 1961, a preprint of which has been used here in the design of an ALGOL 60 translator for GIER, a new Danish computer.

I think that the ALGOL formulation of the procedure DIAGRAM given at the end of the article should have the following corrections:

(1) line 17 *should begin* **if** i ≠ 0.....
(2) line 19 *should begin* **if** STAB [i + 1] =.....
(3) line 27 *should begin* **if** i ≠ 0.....

(4) OTCEL should be assigned some "unconfusing" value before the statement labelled START is obeyed, as otherwise it will be undefined in the statement labelled FOUND in the case when STAB [i] = INPUT [j] and STAB [i + 1] ≠ LEFTBRACE.

It must be common to many languages that could be translated with the aid of this procedure that SUCCR [INPUT [j], INPUT [j]] is always *false* and so lines 10–15 of DIAGRAM can be greatly simplified.

B. H. MAYOH
Regnecentralen
Gl. Carlsbergvej 2
Copenhagen Valby, Denmark