TABLE 1

Size of Filial Set	Average Search Time Old	Average Search Time New	Relative Search Time Old	Relative Search Time New	Relative Cost Old	Relative Cost New
2	1.5	1.500000	1.2052	1.2052	1.8590	2.7886
3	2.0	1.888888	1.0138	.9575	1.1729	1.6616
4	2.5	2.208333	1.0043	.8871	1.0328	1.3684
5	3.0	2.479999	1.0381	.8581	1.0008	1.2410
6	3.5	2.716666	1.0878	.8444	1.0068	1.1722
7	4.0	2.926530	1.1448	.8375	1.0301	1.1304
8	4.5	3.115177	1.2052	.8343	1.0623	1.1031
9	5.0	3.286595	1.2673	.8330	1.0996	1.0842
10	5.5	3.443729	1.3302	.8329	1.1400	1.0706
12	6.5	3.723622	1.4568	.8345	1.2257	1.0532
14	7.5	3.967632	1.5827	.8372	1.3146	1.0431
16	8.5	4.184048	1.7073	.8404	1.4046	1.0371
18	9.5	4.378560	1.8304	.8436	1.4948	1.0334
20	10.5	4.555252	1.9520	.8468	1.5847	1.0312
25	13.0	4.937190	2.2492	.8542	1.8070	1.0294
30	15.5	5.256304	2.5380	.8606	2.0250	1.0300
35	18.0	5.530518	2.8196	.8663	2.2386	1.0317
40	20.5	5.771009	3.0949	.8712	2.4482	1.0338
45	23.0	5.985222	3.3649	.8756	2.6542	1.0360
50	25.5	6.178372	3.6302	.8795	2.8570	1.0383
55	28.0	6.354258	3.8913	.8830	3.0568	1.0405
60	30.5	6.515730	4.1487	.8862	3.2540	1.0427

The memory map corresponding to that given by Sussenguth is shown in Figure 3; the tree itself is shown in Figure 4. On a variable wordlength computer, such as the RCA 501, this revision would demand a 50 percent increase in storage requirements; less flexible machines might demand a 100 percent increase. Even in this latter case, the proposed method is superior as will be shown.

Consider a filial set of size s. If the ith member of the set is selected as the starting point in the search, the original set is partitioned into three subsets with 1, $s-n$ and $n-1$ members. The average search times for these sets are 1, $1+T_{s-n}$ and $1+T_{n-1}$ respectively. The average search time is therefore

$$T_s = \frac{1}{s}[1 + (1 + T_{s-n})(s - n) + (1 + T_{n-1})(n - 1)]$$

$$= 1 + \frac{1}{s}[(T_{s-n})(s - n) + (T_{n-1})(n - 1)],$$

a function of n. However, if the file is random, n could have assumed the values 1 to s with equal probability; therefore

$$T_s = 1 + \frac{1}{s^2}\sum_{n=1}^{s} [(T_{s-n})(s - n) + (T_{n-1})(n - 1)].$$

Because of symmetry, this is equivalent to

$$T_s = 1 + \frac{2}{s^2}\sum_{n=1}^{s} (T_{n-1})(n - 1).$$

If we define $i = n-1$, we have

$$T_s = 1 + \frac{2}{s^2}\sum_{i=0}^{s-1} i\,T_i\,,$$

which is equivalent to

$$T_s = 1 + \frac{2}{s^2}\sum_{i=1}^{s-1} i\,T_i\,.$$

Obviously, $T_i = 1$.

Now that the expected search time is known, relative cost can be computed on the same basis as Sussenguth uses, for direct comparison. These are shown in the accompanying table. With the 50 percent increase in storage requirements assumed in computing relative cost, the proposed method is superior when the average filial set size exceeds 9. If 100 percent storage increase is required, the break-even point is 16.

S & B Applications are continued on page 166

LETTERS TO THE EDITOR

The Dangling "else"

Dear Editor:

I cannot help feeling that Kaupe [A Note on the Dangling **else** in ALGOL 60, *Comm. ACM. 6*, 8 (Aug. 1963)] is tackling the problem in the wrong way. Admittedly it is possible to specify rules which allow us to interpret all his examples unambiguously, but is this really what is needed?

Computers and programmers exercise different processes in analyzing a program, and the most desirable feature of a programming language is that it should be impossible for a piece of program to mean one thing to the programmer and another quite different thing to the computer. One good example of this is a string such as $a/2 \times c$. Most, or at any rate many, mathematicians would interpret this naturally as meaning $a/(2 \times c)$, whereas an ALGOL compiler must treat it as $(a/2) \times c$.

Equally in the case of the dangling **else,** the construction

if . . . **then if** . . . **then** . . . **else** . . . ;

is inherently ambiguous in the same sense, and the vagaries of layout on the page can do a great deal to add to the confusion.

It seems clear that in this kind of situation the needs of the programmer are much better served by a restriction on what he may write, rather than by a definition of the presumed meaning of what he does write. The obvious solution is to require the insertion of brackets to define the required meaning. It is possible to do this in such a way as to allow Kaupe's constructions (2) and (4) while forbidding the ambiguous ones (1) and (3). However, this is probably not worthwhile, and the simplest way to rectify the situation seems to be to remove the construction ⟨if clause⟩⟨for statement⟩ from the definition of ⟨conditional statement⟩ in Section 4.5.1. of the revised Report.

On the question of the spirit of ALGOL 60, it seems to me that the omission of constructions (3) and (4) from ALGOL shows an awareness of the potential ambiguity on the part of the authors, and that the present ambiguity was the result of an oversight. So far from "resorting" to the introduction of **begin** and **end** symbols, I suggest that these are not only desirable but essential to the well-being of users of the language. To indulge in a misquotation, "Programs must not only be correct, they must be seen to be correct."

JOHN H. MATTHEWMAN
The University Mathematical Laboratory
Corn Exchange St.
Cambridge, England

Letters are continued on page 190

what overlays what. Implementation is correspondingly simplified. Storage can be allocated during one pass over the program. Compute time during compilation is reduced.

4. The size of a COMMON block is explicit in COMMON, DIMENSION, and TYPE statements; it may not be extended by EQUIVALENCE statements. This is particularly helpful in using FORTRAN IV named COMMON blocks, whose size may not vary from one subprogram to the next.

HAYDEN T. RICHARDS
Programmatics, Inc.
33 Malaga Cove Plaza
Palos Verdes Estates, Calif.

Letters to the Editor—Cont'd from page 165

Some Comments on the Aims of MIRFAC

Dear Editor:

Recently H. J. Gawlik [1] published an article on MIRFAC: A Compiler Based on Standard Mathematical Notation and Plain English. Its author is aware of earlier projects along analogous lines (MADCAP and COLASL [2]). When I heard of these earlier projects I was filled with amazement, for what they aimed at hardly seemed to be sensible. I did not raise my voice then, convinced and trusting that people would discover this for themselves in a very short time. Now, two and a half years later I am faced with the fact that the movement has not died its natural death as I had supposed it would. This discovery has given me some disappointment and I can only regret my earlier silence on the subject.

The justification for the project MIRFAC seems to be based on the opinion that what is right for communication from man to man should also be right for communication from man to machine. (This is the only interpretation which allows me to attach a meaning to Gawlik's statement "that a compiler should aim not merely to simplify programming, but to abolish it.") But this opinion should not pass unchallenged!

If we instruct an "intelligent" person to do something for us, we can permit ourselves all kinds of sloppiness, inaccuracy, incompleteness, contradiction, etc., appealing to his understanding and common sense. He is not expected to perform literally the nonsense he is ordered to do; he is expected to do what we intended to order him to do. A human servant is therefore useful by virtue of his "disobedience." This may be of some convenience for the master who dislikes to express himself clearly; the price paid is the non-negligible risk that the servant performs, on his own account, something completely unintended.

If, however, we instruct a machine to do something we should be aware of the fact that for the first time in the history of mankind, we have a servant at our disposal who really does what he has been told to do. In man-computer communication there is not only a need to be unusually precise and unambiguous, there is—at last—also a point in being so, at least if we wish to obtain the full benefits of the powerful obedient mechanical servant. Efforts aimed to conceal this new need for preciseness—for the supposed benefit of the user—will in fact be harmful; at the same time they will conceal the equally new possibilities in automatic computing, of having intricate processes under complete control.

I go on quoting Mr. Gawlik: ". . . MIRFAC has been developed to satisfy the basic criterion that its problem statements should be intelligible to nonprogrammers, with the double aim that the user should not be required to learn any language that he does not already know and that the problem statement can be checked for correctness by somebody who understands the problem but who may know nothing of programming."

I do not see the point of Mr. Gawlik's "basic criterion." Elsewhere [3] I have warned against the ". . . tendency to design programming languages so that they are easily readable for a semiprofessional, semi-interested reader." (Symptoms of this tendency are languages whose vocabulary includes a wild variety of English words to be used in a nearly normal sense, and some translators that even allow a steadily expanding list of synonyms and misspellings for these words. Particularly, languages designed under commercial pressure have suffered seriously from this tendency.) It looks so attractive—"Everybody can understand it immediately." However, giving a plausible semantic interpretation to a text which one assumes to be correct and meaningful is one thing; writing down such a text and expressing exactly what one wishes to say may be quite a different matter! On comparable grounds, John McCarthy calls "COBOL . . . a step up a blind alley on account of its orientation towards English which is not well suited to the formal description of procedures." [4]

Furthermore, to accept Mr. Gawlik's double aim is a mistake. Standard mathematical notation has been designed to describe relations; we now have to define processes. Plain English has grown out of a need of interhuman communication to be vague and ambiguous, to tell jokes and to sing nursery rhymes, but is obviously unfit to express what has to be expressed now. One can borrow mathematical notations, one can borrow English words, but completely new semantics must be attached to them and despite superficial similarities one creates a new language. I think the similarities are more misleading than clarifying.

The dangers are revealed by Mr. Gawlik's second aim of having the problem statement checked for correctness by somebody who understands the problem but who may know nothing of programming. Of course such a person can check it, but the crucial point is whether he will find the errors! Of course he will not find them because in human communication one is constantly trained to try to understand another's intentions and not to notice the nonsense. The corrector who understands the problem but knows nothing of programming will be misled by the familiarity of the characters and the words and he will, in all probability, be satisfied if he recognizes the problem.

I am all in favor of clear and convenient algorithmic languages but, please, let them honestly be so—to disguise them in clothes which have been tailored to other purposes can only increase the confusion.

REFERENCES:

1. GAWLIK, H. J. MIRFAC: A compiler based on standard mathematical notation and plain English. *Comm. ACM 6*, 9 (Sep. 1963).
2. WELLS, M. B. MADCAP: A scientific compiler for a displayed formula textbook language. *Comm. ACM 4*, (Jan. 1961).
3. DIJKSTRA E. W. On the design of machine independent programming languages. In *Annual Review in Automatic Programming, Vol. III*, Goodman, R. (Ed.), Pergamon Press, 1961.
4. McCARTHY, J. A basis for a mathematical theory of computation, preliminary report. Western Joint Comput. Conf., 1961.

E. W. DIJKSTRA
Department of Mathematics
Technological University
Eindhoven, Netherlands