Letters to the Editor

On the Development of a New Common Computer Language

Key Words and Phrases: programming languages, language criteria, standardization, PL/I
CR Categories: 4.22

EDITOR:

Congressman Brooks, in his letter of December 5th [*Comm. ACM 11*, 1 (Jan. 1968) 55, 56], states that "Independent criteria identifying the characteristics of a new-generation common computer language must be developed." In commenting on this point in a letter to Congressman Brooks [*SIGPLAN Notices 2*, 12 (Dec. 1967), 55, 56], I found it necessary to strip away his supporting arguments and restate his proposal more briefly, but largely in his own words, as follows:

Moved by the likelihood that "a less than the best" language, PL/I, "could become by default, the *de facto* language of the next generation," you propose the setting up of a group of "the best minds in the entire data processing community" to develop "independent criteria identifying the characteristics of a new generation common computer language." Whereupon, "under improved USASI procedures, ... changes in PL/I" (or perhaps even a brand new language) would be proposed "to meet the general criteria," and "it would then be reasonable to expect IBM ... to adopt" the results.

Certainly PL/I has many faults. It is poorly defined, it lacks a certain parsimony of design, it does not always make full and obvious use of many of the powerful concepts it employs, and it is by no means as widely applicable as it could be. Yet it is a valuable language, and who can tell if these faults detract greatly from its practical utility as a programming tool?

Nevertheless, to develop a better language to eventually replace it is surely a worthwhile and, I believe, an important national goal. A Federal initiative toward this goal should be welcome—though I expect some will not welcome it. But the main problem is not in establishing better criteria and designing an improved language; in addition, a more *valuable* language must be created. And the economic value of a language is determined not so much by its technical excellence, or the lack of it, but by how much time, money and intellectual effort people can be persuaded to invest in developing and using it.

With the technical authority of a group of acknowledged experts behind it, and with the full economic authority of the major computer user, the Federal Government, supporting it, success in establishing an improved and more valuable language can no doubt be achieved—as the history behind the current investment in COBOL might seem to indicate, except that this very investment makes the present job harder. Yet it is an important job—too important, I believe, to be decided by the commercial interests of one manufacturer, and I wish Congressman Brooks success in mobilizing the resources needed to achieve this vital goal of a better, more valuable, computer programming language.

CHRISTOPHER J. SHAW
System Development Corporation
2500 Colorado Avenue
Santa Monica, California 90406

Standardization of Hand-Coding Needed for Man-to-Machine Communication

EDITOR:

At present, USASI Working Group X3.6.3 is developing a standard for hand-coding for man-to-man communications. This is a significant step forward.

But, the job ought not end here; the scope of the group must be broadened. USASI X3.6 should include all hand-coding—specifically man-to-machine readers of hand-coded characters—in its charge to X3.6.3. Anything less is shortsighted in this era of expanding technology.

I therefore urge individual ACM'ers to contact Mr. R. W. Bemer [see *Comm. ACM*, August 1967] on this matter. The time it takes will be far less than the time to find and correct one miscoded input.

E. J. ORTH, JR., *Chairman*
ACM Birmingham Area Chapter
600 N. 18 Street
Birmingham, Alabama 35202

Proposed Abbreviation for 1024: bK

Key Words and Phrases: memory, thousand
CR Categories: 2.44, 6.34

EDITOR:

Morrison's suggestion [Letter to the Editor, *Comm. ACM 11*, 3 (Mar. 1968), 150] that kappa (κ) be used as a symbol for 2^{10} is a good one. The argument for precision in terminology is compelling as one deepens the scientific content of any field. Fortunately, convenience and clarity coincide here with increased precision in expression.

A Greek letter symbol has a disadvantage when used in connection with a computer. For some years I have been using and urging others to use a different symbol for 2^{10}. The symbol is: bK. It may be read as "binary thousand" or just "bee kay." Using a lowercase b is almost as objectionable as a Greek letter since it too is not available on most computer printers. We are, however, accustomed to using uppercase equivalents in computer printing of more readable and more easily remembered typewritten symbols. The proposed symbol is distinctive, easily remembered as a combination of "b" for the "small" number 2 and the conventional "K" for "thousand" or "kilo." Writing bK or BK does not seem to me a significant choice, but I prefer the former and either to κ since it is less distinctive and already has many mathematical and scientific uses. It appears unobjectionable to use the convention that $(bK)^2$ may be written as bK^2, etc.

If *Communications of the ACM* and *Computing Reviews* were to adopt as part of their style manual some symbol for 2^{10}, it would go a long way toward assuring its acceptance.

WALLACE GIVENS
Applied Mathematics Division
Argonne National Laboratory
Argonne, Illinois 60439

On the Evaluation of Multiplicative Combinatorial Expressions

Key Words and Phrases: combinatorial expressions
CR Categories: 5.30

EDITOR:

Evaluating multiplicative arithmetic expressions that arise in combinatorial theory (multinomial coefficients, probabilities, and coupling coefficients) by straightforward computation can lead to difficulties with overflow even when the magnitude of the final result is representable. The method suggested here is fast and does not cause unnecessary overflow. It can be used in formulae involving integer factors not greater than some given N (a typical value of 52 occurs in problems concerning the distribution of playing cards).

Three arrays are declared—*ex*, *hfac*, *lfac*[2:*N*]. *ex*[*n*] contains the exponent of n in the result. For all n, *hfac*[*n*] contains the largest prime factor of n and *lfac*[*n*] contains $n \div hfac[n]$.

To begin, zero the array *ex* and set up the factors in *lfac* and *hfac*. Evaluate the expression by modifying the exponents in *ex*.

For example, to divide by $k!$:

```
    for i := 2 step 1 until k do ex[i] := ex[i] − 1;
```

When the result is complete, decompose the composite integer factors in decreasing order of magnitude into their prime factors. The final numerical result may then be obtained. The result is an integer if the exponents are all nonnegative (and no division will be required) otherwise the result is a rational fraction reduced to primitive form.

```
comment  if den is 1 the result is num, otherwise the result is
  a rational fraction = num/den;
  num := 1;
  den := 1;
  for k := N step −1 until 2 do
    begin
      if hfac[k] > 0 then
        begin
          ex[hfac[k]] := ex[hfac[k]] + ex[k];
          ex[lfac[k]] := ex[lfac[k]] + ex[k];
          ex[k] := 0
        end;
      if ex[k] > 0 then num := num × xk↑ ex[k];
      if ex[k] > 0 then den := den xk↑ (−ex[k])
    end
```

I wish to thank the referee for his helpful suggestions and the ALGOL example.

J. K. S. McKAY,
Atlas Computer Laboratory
Science Research Council, Chilton
Didcot, Berkshire, England

An Auxiliary Program to Analyze LISP 1.5 Programs

Key Words and Phrases: LISP, list processing language, debugging, program analysis, cross references, program
CR Categories: 4.22, 4.29, 4.42, 4.49

EDITOR:

We have had some difficulties in the past analyzing other people's LISP 1.5 programs and recognizing what their internal interrelationship was. We have now designed and tested a LISP program which performs just this analysis and enables us to gain more insight into other LISP programs.

Most LISP programs lack comments and, of course, our program cannot provide missing comments. (This might be a rather interesting artificial intelligence problem!) However, we wished to be able to split up a complex program into smaller subparts, that are easy to understand apart, and therefore we were looking for a detailed cross-reference table of the LISP functions used. This is of even more use than in, e.g. FORTRAN, because nearly all LISP programming is done by means of functions.

Our program will give two cross-reference tables, one that is characterized by the clause "x refers to y" and another characterized by "x is referenced by y" with x standing for a function name and y standing for a set of functions. We have included references to quoted S-expressions. The program will produce a listing of the top level functions and a listing of some APVAL definitions. Of course this program can check a given program merely statically but not dynamically. Thus functions whose definitions will be set up at evaluation time, probably will not be detected, but the functions for setting up probably will.

We have employed the program to find out: (1) where modifications had to be made; (2) where certain error comments originated from; (3) where references to input/output were located; and (4) why certain expressions were handled erroneously. The program appears to be a useful means of first aid and has been running successfully. The program is now undergoing a general revision and may be extended to cover some additional features. Possibly we will give a more detailed report some time later.

We would be interested to hear whether somebody else has written a program of the type discussed, and we would greatly appreciate any contact. Persons interested in obtaining the program may address inquiries to the author.

KNUT BAHR
Deutsches Rechenzentrum
Rheinstrasse 75
6100 Darmstadt
Germany

Generating Permutations by Nested Cycling

Key Words and Phrases: permutations
CR Categories: 5.39

EDITOR:

The purpose of this letter is two-fold: first to give due credit to the Tompkins-Paige algorithm, and second to clarify a comment by Hill, CR Review 13891 on "Programs for Permutations" [*Comput. Rev. 9*, 3 (Mar. 1968), 165]. Hill states, "No references are given in this paper, nor in a simultaneously published English paper by Langdon [1], which outlines the nested cycle permutation algorithm previously reported by Peck and Schrack [2], subsequently improved by Trotter, and formulated recursively by Boothroyd."

Hill is correct in the "no references" portion of his statement, but I should have referenced Tompkins [3], and not Peck and Schrack. Langdon [1] and Algorithm 86 are implementations of different versions of the Tompkins-Paige algorithm. However, Langdon [1] is not a direct implementation because the bookkeeping is considerably simplified by a "trick" which works *only* on its version of the Tompkins-Paige algorithm.

Many authors, myself included, have discussed nested cyclic permutations but have neglected to reference the Tompkins-Paige algorithm. I hope I have clarified the interrelationship of Langdon [1], Algorithm 86, and the Tompkins-Paige algorithm.

References:

1. LANGDON, G. G. An algorithm to generate permutations. *Comm. ACM 10*, 5 (May 1967), 298–299.
2. PECK, J. E. L., AND SCHRACK, G. F. Algorithm 88 PERMUTE. *Comm. ACM 5*, 4 (Apr. 1962), 208.
3. TOMPKINS, C. Machine attacks on problems whose variables are permutations. Sec. 3. Proc. Sixth Symp. Appl. Math., Numerical Anal. McGraw-Hill (for AMS), New York, 1956, pp. 198–211.

GLEN G. LANGDON, JR.
IBM SDD Laboratory, Dept. 156
P.O. Box 6
Endicott, N. Y. 13760