Go To Statement Considered Harmful

Key Words and Phrases: go to statement, jump instruction, branch instruction, conditional clause, alternative clause, repetitive clause, program intelligibility, program sequencing

CR Categories: 4.22, 5.23, 5.24

EDITOR:

For a number of years I have been familiar with the observation that the quality of programmers is a decreasing function of the density of go to statements in the programs they produce. More recently I discovered why the use of the go to statement has such disastrous effects, and I became convinced that the go to statement should be abolished from all "higher level" programming languages (i.e. everything except, perhaps, plain machine code). At that time I did not attach too much importance to this discovery; I now submit my considerations for publication because in very recent discussions in which the subject turned up, I have been urged to do so.

My first remark is that, although the programmer's activity ends when he has constructed a correct program, the process taking place under control of his program is the true subject matter of his activity, for it is this process that has to accomplish the desired effect; it is this process that in its dynamic behavior has to satisfy the desired specifications. Yet, once the program has been made, the "making" of the corresponding process is delegated to the machine.

My second remark is that our intellectual powers are rather geared to master static relations and that our powers to visualize processes evolving in time are relatively poorly developed. For that reason we should do (as wise programmers aware of our limitations) our utmost to shorten the conceptual gap between the static program and the dynamic processes, to make the correspondence between the program (spread out in text space) and the process (spread out in time) as trivial as possible.

Let us now consider how we can characterize the progress of a process. (You may think about this question in a very concrete manner: suppose that a process, considered as a time succession of actions, is stopped after an arbitrary action, what data do we need to describe the process in a helpful and manageable way. The unbridled use of the go to statement has an immediate consequence that it becomes terribly hard to find a meaningful set of coordinates in which to describe the progress of a process.

Why do we need such independent coordinates? The reason is—and this seems to be inherent to sequential processes—that we can interpret the value of a variable only with respect to the progress of the process. If we wish to count the number, $n$, say, of people in an initially empty room, we can achieve this by increasing $n$ by one whenever we see someone entering the room. In the in-between moment that we have observed someone entering the room but have not yet performed the subsequent increase of $n$, its value equals the number of people in the room minus one!

The progress of the process can always be uniquely characterized by a (mixed) sequence of textual and/or dynamic indices. The main point is that the values of these indices are outside programmer's control; they are generated (either by the write-up of his program or by the dynamic evolution of the process) whether he wishes or not. They provide independent coordinates in which to describe the progress of the process.

The go to statement as it stands is just too primitive; it is too superfluous, because we can express repetition with the aid of recursive procedures. For reasons of realism I don't wish to exclude them: on the other hand, repetition clauses can be implemented quite comfortably with present day finite equipment; on the other hand, the reasoning pattern known as "induction" makes us well equipped to retain our intellectual grasp on the processes generated by repetition clauses. With the inclusion of the repetition clauses textual indices are no longer sufficient to describe the dynamic progress of the process. With each entry into a repetition clause, however, we can associate a so-called "dynamic index," inexorably counting the ordinal number of the corresponding current repetition. As repetition clauses (just as procedure calls) may be applied nestedly, we find that now the progress of the process can always be uniquely characterized by a sequence of textual and/or dynamic indices.

The difficulty is that such a coordinate, although unique, is utterly unhelpful. In such a coordinate system it becomes an extremely complicated affair to define all those points of progress where, say, $n$ equals the number of persons in the room minus one! The unbridled use of the go to statement has an immediate consequence that it becomes terribly hard to find a meaningful set of coordinates in which to describe the process progress. Usually, people take into account as well the values of some well chosen variables, but this is out of the question because it is relative to the progress that the meaning of these values is to be understood! With the go to statement one can, of course, still describe the progress uniquely by a counter counting the number of actions performed since program start (viz. a kind of normalized clock).

The difficulty is that such a coordinate, although unique, is utterly unhelpful. In such a coordinate system it becomes an extremely complicated affair to define all those points of progress where, say, $n$ equals the number of persons in the room minus one!

The go to statement as it stands is just too primitive; it is too much an invitation to make a mess of one's program. One can regard and appreciate the changes considered as bridling its use. I do not claim that the clauses mentioned are exhaustive in the sense that they will satisfy all needs, but whatever clauses are suggested (e.g. abortion clauses) they should satisfy the requirement that a programmer independent coordinate system can be maintained to describe the process in a helpful and manageable way.

It is hard to end this with a fair acknowledgment. Am I to...
Language Protection by Trademark Ill-advised

Key Words and Phrases: TRAC languages, procedure-oriented language, proprietary software, protection of software, trademark, copyright protection, patent protection, standardization, licensing, Mooers doctrine

CR Categories: 2.12, 2.2, 4.0, 4.2

EDITOR:
I would like to comment on a policy published 25 August 1967 by the Rockford Research Institute Inc., for trademark control of the TRAC language "originated by Calvin N. Mooers of that corporation": "It is the belief at Rockford Research that an aggressive course of action can and should be taken to protect the integrity of its carefully designed languages." Mr. Mooers believes that "well-drawn standards are not enough to prevent irresponsible deviations in computer languages," and that therefore "Rockford Research shall insist that all software and supporting services for its TRAC languages and related services be furnished for a price by Rockford, or by sources licensed and authorized by Rockford in a contract arrangement." Mooers' policy, which applies to academic institutions as well as commercial users, includes "authorized use of the algorithm and primitives of a specific TRAC language; authorization for experimentation with the language . . . ."

I think that this attempt to protect a language and its software by controlling the name is very ill-advised. One is reminded of the Comtir language, whose developers (under V. Yngve) restricted its source-level distribution. As a result, that effort was bypassed by the people at Bell Laboratories who developed SNOMOL. The latter language and its software were inevitably superior, and were immediately available to everyone, including the right to make extensions. Later versions benefited from "meritorious extensions" by "irresponsible young people" at universities, with the result that SNOMOL today is an important and prominent language, while Comtir enjoys relative obscurity.

Mr. Mooers will find that new TRAC-like languages will appear whose documentation, because of the trademark restriction, cannot mention TRAC. Textbook references will be similarly inhibited. It is unfortunate.

BERNARD A. GALLER
University of Michigan
Ann Arbor, Mich. 48104

Mr. Mooer's Reply

EDITOR:
Professor Galler's letter, commenting on our Rockford Research policy statement on software protection of 25 August 1967, opens the discussion of what may be a very significant development to our computing profession. This policy statement applies to our TRAC (TM) computer-controlling languages. The statement includes a new doctrine of software protection which may be generally applicable to a variety of different kinds of complex computer systems, computer services, languages, and software. Already it is evident that this doctrine has a number of interesting legal and commercial implications. It is accordingly appropriate that it be subject to critical discussion.

The doctrine is very simple. For specificity, I shall describe it in regard to the TRAC languages which we have developed: (1) Rockford Research has designated itself as the sole authority for the development and publication of authentic standards and specifications for our TRAC languages; and (2) we have adopted TRAC as our commercial trademark (and service mark) for use in connection with our computer-controlling languages, our publications providing standards for the languages and any other related goods or services.

The power of this doctrine derives from the unique manner in which it serves the interests of the consuming public—the people who will be using computer services. The visible and recognized TRAC trademark informs this public—the engineers, the sociologists, the business systems people, and the nonprogrammers everywhere—that the language or computer capability identified by this trademark adheres authentically and exactly to a carefully drawn Rockford Research standard for one of our TRAC languages or some related service. This is in accord with a long commercial and legal tradition.

The evils of the present situation and the need to find a suitable remedy are well known. An adequate basis for proprietary software development and marketing is urgently needed, particularly in view of the doubtful capabilities of copyright, patent, or "trade secret" methods when applied to software. Developers of valuable systems—including languages—deserve to have some vehicle to give them a return. On the user side the nonexistence of standards in the computer systems area is a continuing nuisance. The proliferation of dialects on valuable languages (e.g., SNOMOL or FORTRAN) is sheer madness. The layman user (read "nonprogrammer") who now has access to any of several dozen computer facilities (each with incompatible systems and dialects) needs relief. It is my opinion that this new doctrine of autonomous standardization coupled with resort to commercial trademark can provide a substantial contribution to remedying a variety of our problems in this area.

Several points of Professor Galler's letter deserve specific comment. The full impact of our Rockford Research policy (and
No Trouble with Atlas I Page-Turning Mechanism

Key Words and Phrases: Atlas I, page-turning procedures
CR Categories: 4.2, 4.22

EDITOR:

The editorial on "The European Computer Gap," Comm. ACM 10 (April 1967), 203, tells of "paper designs that could never be converted into operational systems," and among these includes "the page-turning procedures proposed with the original design of the Atlas."

Not merely does this do injustice to Atlas, but it is in fact quite wrong. The Atlas I machine we have here has a one-level storage made up of 48K words of 2μ cores and 96K words on drums. The paging and page-turning mechanism has worked without any trouble at all almost from the beginning—so well that it is something we hardly ever think about. To give an idea of how intensively the system is used let me say that since 1964 we have been running a service for research workers in all British universities with a very mixed load of programs in all the major languages. We put about 2500 jobs through the machine each week, and the system efficiency is around 70 percent. By this last figure I mean that, of all the instructions obeyed by the machine over a long period, 70 percent go into either the compiling or execution of users' programs. The figure can rise to over 90 percent with a favorable job-mix.

J. HOWLETT
Atlas Computer Laboratory
Chilton, Didcot
Berkshire, England

Dealing with Neely's Algorithms

Key Words and Phrases: algorithm, computation of statistics, truncation error, Neely's comparisons
CR Categories: 4.0, 5.5, 5.11

EDITOR:

When we decided to use the method of Welford [1] in our FORTRAN programs we made some comparisons, but arrived at a conclusion which contradicts Peter Neely's [2]. This was an invitation to us to scrutinize Neely's work. His remark, "The inaccuracy noted for \( M_z \) may be due to IBM-FORTRAN, which does not compile a floating round," is one pointer to the source of inaccuracy. Indeed, with a compiler which does compile a floating round, Welford's method gives results equivalent to those obtained with the two-pass method recommended by Neely. If a floating round is not compiled, the use of Kahan's trick [3] will give excellent results even on those machines, such as an IBM 1620 which truncates before normalizing a floating point sum.

Another source of inaccuracy, however, is due to the way Welford's formulas are programmed. In particular we found that the formulas as given by Welford and programmed by Neely are not the best available.
The best versions for programming purposes seem to be the following:

\[ m_0 = 0; \quad m_i = m_{i-1} + (x_i - m_{i-1})/i, \quad i = 1, n; \quad M_2 = m_2 \]  

\[ s_0 = 0; \quad s_i = s_{i-1} + (x_i - m_{i-1})^2 - (x_i - m_{i-1})/i, \quad i = 1, n; \quad S_2 = s_n \]

and \( P_2 \) similar to (2). Of these equations (1) is most important and addition using Kahan’s trick will give an error-free answer.

Not using Kahan’s trick will give results for variable \( x_{i,10} \) not as good as those obtained with the two-pass method, but since we think this kind of variable is not likely to occur in practical work, we recommend (1) and (2) for calculation of the mean and corrected sum of squares. Since we found that from (1) and (2) \( \Sigma x \) and \( \Sigma x^2 \) are more accurately retrieved than when computed directly, we think that (1) can be used in numerical integration too, if the result afterwards is multiplied by the number of intervals.

References:

A. J. van Reeken
Rekencentrum
Katholieke Hogeschool
Tilburg, The Netherlands

Abbreviations for Computer and Memory Sizes

Key Words and Phrases: memory, thousand

CR Categories: 2.44, 6.34

Editor:
The fact that \( 2^{19} \) and \( 10^9 \) are almost but not quite equal creates a lot of trivial confusion in the computing world and around its periphery. One hears, for example, of doubling the size of a 32K memory and getting a 65K (not 64K) memory. Doubling again yields a 131K (not 130K) memory. People who use powers of two all the time know that these are approximations to a number they yield a 131K (not 130K) memory. People who use powers of two memory and getting a 65K (not 64K) memory, Doubling again, we wouldn’t have to approximate. I suggest that \( \kappa \) (kappa) be used for this purpose. Thus a 32K memory means one with exactly 32,768 words. Doubling it produces a 64K memory which is to say one with exactly 65,536 words. As memories get larger and go into the millions of words, one can speak of a 32\( \kappa \) (33,554,432-word) memory and doubling it will yield a 64\( \kappa \) (67,108,864-word) memory. Users of the language will need to have at their fingertips only the first nine powers of 2 and will not need to explain the discrepancies between what they said and what they meant.

Donald R. Morrison

Computer Science, Division 5256
Sandia Corporation, Sandia Base
Albuquerque, N. Mex.

In Defense of Langdon’s Algorithm*

Key Words and Phrases: lexicographic permutation

CR Categories: 5.39

Editor:
Ord-Smith [Letter to the Editor, Comm. ACM 10, 11 (Nov. 1967), 684] makes some impertinent remarks on the subject of Langdon’s algorithm [1]. The main point of the letter “that these does not appear to be any combinatorial advantage of circular ordering over lexicographic ordering” is hardly relevant. The problem attacked by Langdon is not to find combinatorial advantage but rather computational advantage, which Langdon’s algorithm most certainly provides.

Most of the score or so of ALGOL algorithms published in CACM on the subject of lexicographic succession have been badly written; they contain only the sketchiest of theoretical discussion, and the obscurity of their construction masks their essentially simple methodology. In contrast, Langdon gives a clear and concise theoretical discussion and logic diagram. The relative brilliance of Langdon’s paper may be taken as an indication that formal papers and logic diagrams are a superior method for presentation of this subtle type of arithmetic. The essential point that Ord-Smith seems to have missed is that Langdon’s algorithm uses rotation rather than transposition as the basis of iteration, thus taking advantage of the hardware design of modern computers which perform rotation much more efficiently than transposition. The ALGOL language, however, does not give the user access to the rotation registers and hence will not implement this algorithm efficiently with respect to running time. The fact that the transposition methods give shorter running times indicates not superior algorithms but a fundamental weakness of the ALGOL language for this type of numeric manipulation. Given access to the rotation registers, Langdon’s algorithm would be efficient in both coding compactness and running time.

References:

B. E. Rodden
Defense Research Establishment
Toronto, Ontario, Canada

* DRET Technical Note No. 686

Endorsing the Illinois Post Mortem Dump

Key Words and Phrases: ALCOR post mortem dump

CR Categories: 4.12, 4.42

Editor:
The authors of “The ALCOR Illinois 7090/7094 Post Mortem Dump” [Comm. ACM 10, 12 (Dec. 1967), 804–808] have presented a technique for producing post mortem dumps which, in my opinion, should be incorporated in all high level programming languages. A similar technique has been in operation for several years at Manchester [1] and has proved to be extremely useful, especially for student programmers.

Reference:

S. R. Clark
Department of Computing Science
The University of Manitoba
Winnipeg, Canada

Donald R. Morrison

Computer Science, Division 5256
Sandia Corporation, Sandia Base
Albuquerque, N. Mex.

Communications of the ACM

Volume 11 / Number 3 / March, 1968