# Additional comments on a problem in concurrent programming control

# Letters to the Editor

## Format Effectors in ISO7 and ASCII

EDITOR:

Backspace is awkward to deal with. But compound characters do not necessarily require backspacing. One way is to define certain graphics as nonspacing [see Barron, D. W. *Comput. J. 7* (1965), 281]. Another way is to have a format effector which inhibits the spacing forward when the following graphic is printed. Thus to underline an $x$ one has nospace, $x$, underline. This is more general than having special nonspacing graphics, and requires no more characters than using backspace.

The advantages of using nospace instead of backspace are:

1. It gives prior warning of a compound symbol.
2. It avoids the necessity for line reconstruction.
3. It entails less movement on printers, and consequently is probably mechanically simpler.

I submit therefore that in ISO7 and in any revision of ASCII, a place be found for the format effector "nospace."

I would point out that character-by-character printers almost inevitably have a nospace mechanism, which is automatically engaged when the printing head reaches the right-hand margin. My proposal amounts to having a control character which activates this mechanism. As an alternative to having a single "nospace" effective for just the next graphic, there could be two format effectors: "stop-spacing" and "start-spacing." Use of these might be mechanically easier, and more compact for overprinted sequences of four or more graphics, but it is less suited for the most common case of two overprinted graphics.

My basic point is that if ISO7 contains "backspace" only, then the opportunity is lost for overprinting graphics on devices which do not have back-space mechanisms, but do have no-space mechanisms.

I. C. PYLE
*Atomic Energy Research Establishment*
*Harwell, Didcot, Berks., England*

## "Pāṇini-Backus Form" Suggested

EDITOR:

Knuth, in a recent Letter to the Editor of CACM [1], makes the point that the metasyntactic notation used in, e.g., the ALGOL 60 report [2] should be renamed. In particular, he observes the well-acceded fact that the so-called Backus Normal Form is, indeed, not a normal form in any sense. The purpose of this letter is to observe that Backus was not the first to use the form with which his name has become associated, although he did, indeed, discover it independently.

Dr. Alexander Wilhelmy has called to my attention [3] a work by Pāṇini [4]. Pāṇini was a scholar who flourished between 400 B.C. and 200 B.C.; perhaps his most significant work was the compilation of a grammar of Sanskrit. In order to describe the (rather complicated) rules of grammar, he invented a notation which is equivalent in its power to that of Backus, and has many similar properties: given the use to which the notation was put, it is possible to identify structures equivalent to the Backus "|" and to the use of the meta-brackets "<" and ">" enclosing suggestive names. Pāṇini avoided the necessity for the character "::=" by writing the meta-result on the right rather than the left (see, e.g., [5] and [6] for a similar notation).

Since it is traditional in professional circles to give credit where credit is due, and since there is clear evidence that Pāṇini was the earlier independent inventor of the notation, may I suggest the name "Pāṇini-Backus Form" as being a more desirable one? Not only does it give due credit, but it also avoids the misuse of the word "Normal."

REFERENCES:

1. KNUTH, DONALD E. Backus normal form vs. Backus Naur form. *Comm. ACM 7*, 12 (Dec. 1964), 735–736.
2. NAUR, P. [ED.]. Revised report on the algorithmic language ALGOL 60. *Comm. ACM 6*, 1 (Jan. 1963), 1–17.
3. WILHELMY, A. Private commuuication dated 5 November 1966.
4. KĀVYATĪRTHA, NĀRĀYAṆA RĀMA ĀCĀRYA (ED.) Pāṇinimuni-praṇītaḥ aṣṭadhyāyīsūtrapāṭhaḥ vārtikapāṭhasamalaṅkṛtaḥ. Bombay, India, 1954 (See also [7], supplied by Dr. Donald Knuth.) [Kāvya, N.R.A. (ED.) Pāṇini—Reading of Rules in Eight Chapters, Embellished by His Pupils].
5. IRONS, E. T. Maintenance manual for PSYCO—part one. Institute for Defense Analyses, Princeton, N.J.
6. INGERMAN, P. Z. *A Syntax-Oriented Translator*. Academic Press, New York, 1966.
7. PĀṆINI. *The Ashtādhyāyī*. Edited and translated into English by Srisa Chandra Vasu, Delhi, India, 1962.

PETER ZILAHY INGERMAN
*Manager, Language Systems*
*Standards & Research*
*Radio Corporation of America*
*Cherry Hill, N.J. 08034*

## Additional Comments on a Problem in Concurrent Programming Control

EDITOR:

In D. E. Knuth's solution [*Comm. ACM 8*, 5 (May, 1966), 321–322, Letter to the Editor] of Dijkstra's problem [*Comm. ACM 8*, 9 (Sept. 1965), 569] it is not quite easy to check that any computer waiting for its critical section has to wait at most $2^{N-1}$ turns (the word "turn" refers to a computer performing its critical section). It occurred to me that by a small change in his program that number can be reduced to $\frac{1}{2}N(N-1)$, with the extra advantage that for this new program it is easier to see why and how it works. The change consists of replacing

*L3*: $k := i$;
critical section;
$k :=$ **if** $i=1$ **then** $N$ **else** $i-1$;

by

*L3*: critical section;
**if** control $[k] = 0 \vee k=i$ **then** $k :=$ **if** $k = 1$ **then** $N$ **else** $k-1$;

and requiring that the initial value of $k$ is one of the numbers $1, \cdots, N$, instead of 0.

With these alterations we find:

(i) If at a certain moment $k$ has a value $i$, and if control $[i] \neq 0$, then $k$ does not change its value before computer $i$ performs its critical section.

(ii) In a time interval where $k$ is constant, no computer can pass its critical section twice. Assuming computer $j$ passes twice, we have $j \neq k$ and control $[k] \neq 0$ (otherwise $k$ would have changed the first time); computer $k$ does not pass its critical section before $j$ does (for otherwise the value of $k$ would change before $j$ gets its second turn); hence control $[k] \neq 0$ all the time between the

two turns of $j$, and this means that $j$ cannot get to $L2$ after its first turn.

From (i) and (ii), it follows that if computer $i$ has control $[i] \neq 0$, then it has to wait at most $N(N-1)$ turns. The actual maximum is $\frac{1}{2}N(N-1)$, however. This we prove for $i = 1$.

(iii) If $j$ has one of the values $2, \cdots, N$, then the following holds. In a time interval throughout which control $[1] \neq 0$ and $j \geq k \geq 1$, computer $j$ can pass its critical section at most once. For, after its first passage we cannot have $k = j$, (even if $j = N$ the value of $k$ cannot jump from 1 to $N$ under these circumstances) and from that moment on we have $j > k \geq 1$, which implies that $j$ cannot get to $L2$ before control $[1] = 0$.

From (i), (ii), and (iii), it follows that in a time interval where control $[1] \neq 0$, computer $j$ can have at most $N-j+1$ turns ($2 \leq j \leq N$). Hence computer 1 has to wait at most $\sum_{j=2}^{N} (N-j+1) = \frac{1}{2}N(N-1)$ turns. It is not difficult to show that this waiting period can indeed occur.

N. G. de Bruijn
*Technological University*
*Eindhoven, The Netherlands*

## Call for Information: Law and Data Processing

Editor:

I am writing this letter in the hope that your readers will be kind enough to help me help them and the data processing community.

I am now engaged in a survey study and collection of material in the broad field of law and data processing. The results of the study and the accompanying bibliography are expected to have a wide circulation. Unfortunately, published works in this field are scattered among many journals in different fields and even incomplete bibliographies are difficult to come by. Many works appear to exist in unpublished form and information about court decisions and actual experience in this field are not readily available.

I would greatly appreciate hearing from any reader who has information about unpublished works or publications not widely known, bibliographies in the field, court decisions, personal experiences with legal problems concerning data processing, or any information which might be useful to the study. Any assistance will be gratefully acknowledged in the study. I would also like to learn of any legal problems in this area which your readers feel have not been treated in the available literature and which are nevertheless important to them.

If their firm's lawyers are not regular readers of the *Communications of the ACM*, I hope your readers will bring my request to their attention also.

John F. Banzhaf III
*Computer Program Library*
*509 Fifth Ave.*
*New York, N.Y. 10017*

## Aesop and the Referee: A Fable

Editor:

Once upon a time, a referee received a paper for review. The paper was laden with Theorems which were proved by reference to unpublished technical reports and "to be published" documents. As the referee looked at the piles of unread journals on his desk, he decided that he did not have the time required to decipher the Theorems and he put the paper aside. Many moons passed and he finally received a prodding letter from the editor. The pile of unread journals was even larger, so he devoted even less time to the review. When he had found several instances of poor notation or missing references and had suggested minor changes, his job was complete. Many moons passed and that paper joined the piles of unread journals across the country.

*Question:* Were the Theorems correct???

*Morals:* (1) Referees should spend real time, not just turnaround time at their jobs. (2) Authors should include copies of every cited technical report or unpublished document with copies of their papers.

R. Y. Kain
*University of Minnesota*
*Minneapolis, Minnesota 55455*

## More on Processing 64-Character Cards

Editor:

This letter is in reply to Robert F. Rosin's letter, "Bridging the Equipment Gap for Processing 64-Character Cards" [*Comm. ACM 9*, 9 (Sept. 1966), 694].

The suggestion for overprinting two symbols to supply extra characters especially for PL/I and EBCDIC is impractical since only a minority of potential users have *space suppression* equipment on their printers. There is, however, a clear need for some scheme for a large number of users.

I have made a program available for 1401 (and 360 in 1401 mode) on the IBM Program Library (Program Number 01.4.203) called LATCH (List All The CHaracters) which uses a double line to print special characters for PL/I and EBCDIC character sets.

Details of the symbols used are listed below. They may be user-modified.

| *Character* | *EBCDIC Representation* | *2-Line Representation* | *Comments* |
|---|---|---|---|
| Or | \| | 1<br>1 | (One over one 11<br>Note: concatenation 11) |
| Left parenthesis* | ( | /<br>*L* | Gives visual impression of left parenthesis. |
| Plus | + | .<br>& | Period over ampersand ("Resembles" principle) |
| Greater than | > | *G*<br>*T* | Normal convention (also in Fortran and PL/I 48 character set) |
| Less than | < | *L*<br>*T* | Normal convention (also in Fortran and PL/I 48 character set) |
| Right parenthesis* | ) | 1<br>/ | One over slash resembles right parenthesis |
| Not | ¬ | .<br>7 | Seven resembles "not" symbol; Period over seven |
| Semicolon | ; | .<br>, | Period over comma |
| Quote | ' | ,<br>(blank) | Comma over blank |
| Equals* | = | —<br>— | Dash over dash |
| Colon | : | .<br>. | Period over period |
| Underscore (break) | — | .<br>— | "Resembles" principle; Period over dash |

*Some of these special symbols *are* available in correct form on some machines.

Tom Scharf
*A/S Datasentralen*
*Boks 3654*
*Oslo 1, Norway*

*Letters are continued on page 148*

LETTERS—cont'd. from p. 138

## Should There Be a CS Undergraduate Program?

EDITOR:

Having read the reports of the two symposia on the impact of computing on undergraduate mathematics instruction [*Comm. ACM 9*, 9 (Sept. 1966)], I must comment.

In the remarks of Professor Givens and Professor Murray there is little to which I take exception. Professor Atchison, however, says, "It should be recognized that there are still some people who are questioning whether there should be an undergraduate program in Computer Science."

I am one of them, and in fact I want to argue that there should not be such a program.

Probably every undergraduate nowadays should learn something about computers and their uses, as part of his knowledge of man's tools; certainly everyone who intends to go into science should learn the rudiments of numerical analysis and programming. Furthermore, it is right and inevitable that the existence of computers and of computer-oriented methods should influence the content of mathematics courses. Professor Murray has treated this matter quite well.

My opinion, however, is that the computer professional needs to know, or at least to encounter, almost everything that is in the modern undergraduate mathematics curriculum. If he adds to this curriculum (which should, obviously, include a good course in logic) a course or two in physics or electronics, and perhaps a look at some field such as psychology or economics, in which he may someday have to do computing, he is not going to have much time left over.

I have read the paper which outlines the C³S preliminary recommendations. This curriculum contains a great deal of worthwhile material, but most of it belongs at the graduate level, where it can be done better any way. I simply do not see that much of it can be included in an undergraduate program except by skimping on basic science and by excluding from the student's experience all but the most perfunctory contact with nontechnical subjects.

Professor Murray puts the matter succinctly, "College and university education should be aimed at the intellectual development of the student. Indeed, it is because the impact of computers is significant for the broad development of the student that changes are required."

We have heard a good deal of muttering about the social responsibilities of the computing profession. A part of this responsibility, I feel, is to abstain from pressuring the colleges and universities into answering the supposed needs of industry by grinding out narrow, semi-literate technicians.

The would-be computer scientist can learn about syntax-directed compilers on the job or in graduate school. In college, Birkhoff and MacLane, Aristotle, and T. S. Eliot will do him more good.

L. FULKERSON
*IBM Watson Research Center*
*Yorktown Heights, N. Y. 10598*

### *A Reply to Fulkerson's Comments*

EDITOR:

I would like to offer a few comments concerning Mr. Fulkerson's Letter on an undergraduate curriculum in computer science.

First, let me say that all of the points made by Mr. Fulkerson are very familiar to all of us on the Curriculum Committee, in fact some members may even agree with him. The facts are, however, that numerous undergraduate programs are being established and the Curriculum Committee feels an obligation to propagate the best ideas possible for such programs. We have stressed, and will continue to stress, that such programs may not be appropriate at all schools and that they should emphasize the educational rather than the training philosophy. Perhaps the best answer as to whether or not there should be an undergraduate program in computer science will be revealed after the passage of a few more years. Certainly this is the approach we have adopted here at the University of Maryland. We are first setting up our Masters Degree program, with the intent of developing a Doctoral program next. After these have been established, we can consider the question of an undergraduate program. This is the approach being taken by many schools.

Now I will try to comment explicitly on some of Mr. Fulkerson's points. I certainly agree with him that every undergraduate should learn something about computer science. Our Curriculum Committee in conjunction with the Education Committee is now considering a lower level computer course for all college students, but it is not in our current suggested curriculum, and probably will not get into our next publication.

I do not concur with Mr. Fulkerson's opinion that it is necessary for a computer professional to know or at least encounter almost everything in the modern undergraduate mathematics curriculum. Although there is much that is mathematical, included in or necessary for computer science, we have had considerable testimony, including that of many well qualified mathematicians, to the effect that much of computer science depends very little on the mathematics presently being taught in our universities. On the other hand, a great many mathematical concepts are needed which are seldom found in mathematics programs. It is my opinion that a potential computer science major, depending on what he is going to do, should get one or more courses in mathematics beyond the calculus, but I am sure you know that many people think that even this much is not needed. I regard this as something of an average answer to the amount of mathematics needed for an undergraduate major in computer science. Our Curriculum Committee has discussed this point at some length, and we feel that it is entirely possible for the undergraduate to take this much mathematics, a sufficient amount of computer science, and considerable additional material, all within the normal Bachelor of Science degree requirements.

Relative to Mr. Fulkerson's remark that most of the curriculum material belongs at the graduate level, I suspect that too much undergraduate level material in computer science is already being offered at the graduate level. Most of the material we propose is indeed already being taught successfully at the undergraduate level. A more adequate undergraduate education in computer science—even a good minor—would make better graduate programs possible. It is my opinion that the computer science material is in the process of drifting from the higher educational levels to the lower ones as is typical in all areas.

I have spent a considerable amount of time thinking about the intellectual development of a possible undergraduate student in computer science. I would agree that his intellectual development is extremely important, but it seems to me that it is entirely possible that an undergraduate student may even be better prepared intellectually by taking a good undergraduate major in computer science than by taking an undergraduate major in a more traditional science or engineering subject. It is conceivable to me that a *good* undergraduate program in computer science may do a better job than a traditional major in imparting to the student wisdom, knowledge, and the ability to organize his thoughts and facts for future application to this real and somewhat disorganized world. It is my conjecture that the newer developments in the computer and information sciences may well yield very significant new approaches to the intellectual realms of knowledge. Such a broad approach, if we can indeed achieve this in computer science, and I think we can, will not yield a semi-literate technician, but an educated intellectual.

WILLIAM F. ATCHISON
*Computer Science Department*
*University of Maryland*
*College Park, Md. 20740*