Additional comments on a problem in concurrent programming control

Citation for published version (APA):

DOI:
10.1145/363162.363167

Document status and date:
Published: 01/01/1967

Document Version:
Publisher's PDF, also known as Version of Record (includes final page, issue and volume numbers)

Please check the document version of this publication:

• A submitted manuscript is the version of the article upon submission and before peer-review. There can be important differences between the submitted version and the official published version of record. People interested in the research are advised to contact the author for the final version of the publication, or visit the DOI to the publisher's website.
• The final author version and the galley proof are versions of the publication after peer review.
• The final published version features the final layout of the paper including the volume, issue and page numbers.

Link to publication

General rights
Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

• Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
• You may not further distribute the material or use it for any profit-making activity or commercial gain
• You may freely distribute the URL identifying the publication in the public portal.

If the publication is distributed under the terms of Article 25fa of the Dutch Copyright Act, indicated by the “Taverne” license above, please follow below link for the End User Agreement:
www.tue.nl/taverne

Take down policy
If you believe that this document breaches copyright please contact us at:
openaccess@tue.nl
providing details and we will investigate your claim.

Download date: 07. Jan. 2020
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 probably be reduced to

3. It entails less movement on printers, and consequently is probably mechanically simpler.

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 nospace mechanisms.

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. 1966), 569] it is not quite easy to check that any computer waiting for its critical section has to wait at most \( \frac{1}{2}N(N-1) \) 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

\[
L3: \quad k := i;
\]

critical section;

\[
k := \text{if } i = 1 \text{ then } N \text{ else } i - 1;
\]

by

\[
L3: \quad \text{critical section;}
\]

if control \([k] = 0 \lor k = i \) then \( k := \text{if } k = 1 \text{ then } N \text{ else } k - 1 \);

and requiring that the initial value of \( k \) is one of the numbers 1, \( \ldots, N \), 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

References:
4. Kāvyāśīrtha, Nārāyanā Rāma Ācārya (Ep. Pāṇinimuni
prājñātah antadhyāyaśūtrapāthah vārtikāpāṭhasamālaṅkṛtaḥ. Bombay, India, 1964 (See also [7], supplied by Dr. Donald Knuth.) [Kāvyā, N.R.A. (Ep.) Pāṇinī—Reading of Rules in Eight Chapters, Embellished by His Pupils].

Peter Zilahy Ingerman
Manager, Language Systems
Standards & Research
Radio Corporation of America
Cherry Hill, N.J. 08034

Srisa Chandra Vasu
Manager, Language Systems
Standards & Research
Radio Corporation of America
Cherry Hill, N.J. 08034
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 \( 2, \ldots, N \), 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. BACHHAF 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 turn-around 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 to print special characters for PL/I and EBCDIC character sets.

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

<table>
<thead>
<tr>
<th>Character</th>
<th>EBCDIC Representation</th>
<th>ASCII Representation</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>Or</td>
<td>(</td>
<td>)</td>
<td>( / )</td>
</tr>
<tr>
<td>Left parenthesis*</td>
<td>( ( )</td>
<td>( / )</td>
<td>Gives visual impression of left parenthesis.</td>
</tr>
<tr>
<td>Plus</td>
<td>( + )</td>
<td>&amp;</td>
<td>Period over ampersand (&quot;Resembles&quot; principle)</td>
</tr>
<tr>
<td>Greater than</td>
<td>( &gt; )</td>
<td>( T )</td>
<td>Normal convention (also in FORTRAN and PL/I 48 character set)</td>
</tr>
<tr>
<td>Less than</td>
<td>( &lt; )</td>
<td>( L )</td>
<td>Normal convention (also in FORTRAN and PL/I 48 character set)</td>
</tr>
<tr>
<td>Right parenthesis*</td>
<td>( )</td>
<td>( / )</td>
<td>One over slash resembles right parenthesis</td>
</tr>
<tr>
<td>Not</td>
<td>( \neg )</td>
<td>( 7 )</td>
<td>Seven resembles &quot;not&quot; symbol; Period over seven</td>
</tr>
<tr>
<td>Semicolon</td>
<td>;</td>
<td>.</td>
<td>Period over comma</td>
</tr>
<tr>
<td>Quote</td>
<td>( ' )</td>
<td>(blank)</td>
<td>Comma over blank</td>
</tr>
<tr>
<td>Equals*</td>
<td>( = )</td>
<td>( - )</td>
<td>Dash over dash</td>
</tr>
<tr>
<td>Colon</td>
<td>:</td>
<td>( . )</td>
<td>Period over period</td>
</tr>
<tr>
<td>Underscore(break)</td>
<td>( -- )</td>
<td>( &quot;Resembles&quot; principle; Period over dash</td>
<td></td>
</tr>
</tbody>
</table>

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

TOM SCHARF
A/S Datacentralen
Boks 364
Oslo 1, Norway

Letters are continued on page 148
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 CS 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.

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 an-