technical contributions

AFTER THE GOTO DEBATE

Hilmer W. Besel
Mathematics Department
Loma Linda University
Riverside, California 92505

The current discussions about structured programming and related topics are indeed interesting and constructive. Undoubtedly others, like myself, are awaiting some eventual consensus on these issues as, for example, whether strict single-entry single-exit blocks or some modification of these are most suitable (1)(2)(3). I think I would vote for the modification.

My purpose in this contribution is to raise a somewhat tangential question and hopefully to provide a suitable answer. The line of thought to be presented began some years ago with a comment by Dr. Loren P. Meissner, a colleague during a past employment, namely that a flowchart ought to be "typeable." The idea, of course, was that then the flow chart could also be computer produced. This was not taken very far by us at that time partly because the only mechanism to show program structure seemed to be indentation, and this alone does not easily nor neatly satisfy all requirements.

My question is:

> after the debates of program structure and variable binding(4), etc., have been at least somewhat settled, how do we capitalize on these new insights in the mundane, everyday, down-to-earth programming activities?

What I mean is:

> What form shall the actual document take that the programmer is working with day by day and that he leaves for posterity?

I am aware of various schemes and programming systems which routinely produce flow-charts of the old variety using large boxes and diamonds, etc., made from lines of asterisks and such. Presumably, these kinds of systems could equally well produce the new flow-chart symbols such as proposed in (3) and (5). I also realize that computer graphics systems can draw symbols like these. However, I do not consider either of these approaches entirely satisfactory. Both imply that there exists a program listing in addition to the flow chart. Also graphics systems are considerably fewer in number than typewriter terminals. I agree with the criticisms of present systems expressed by Ehrman (6).

A solution to the problem should maintain the goals espoused in the current discussions, including: perspicuity with related simplicity(2) and convenient revision facilities. Moreover, the language processor and/or system facilities should do the external as well as internal bookkeeping associated with program production. And, ideally, the detailed version of the flow chart should be part of, or closely associated with, the program text itself on the same external hard copy sheets.

The following proposal is, I think, in keeping with these goals.

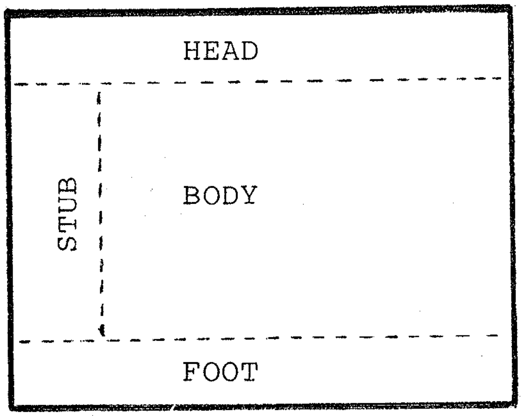
PROPOSAL.

The editing facilities associated with a language processor should:

1. accept as input
2. modify as directed
3. maintain intact internally
4. be able to print out

the program text in units of "pages" roughly compatable with the typed contents of, say, an 8 1/2 by 11 inch sheet of paper. Such an editor would ideally be a powerful on-line, interactive program. "Page" refers merely to a convenient sub-unit of a program, eg. a procedure, as also suggested in (2).

Further, such a "page" should have defined within it certain standard regions which correspond to portions of the sheet of paper and which regions are handled by the editor in ways peculiar to each such region. I will name these regions respectively the HEAD, STUB, BODY, and FOOT, placed as shown in the diagram.



The HEAD contains page identification information for the reader of the program, with the page number being in some form of Dewey decimal scheme so that page insertions and deletions are easy. The STUB contains the line numbers of the program text for such languages as ALGOL and PL/1 that do not use line numbers for internal referencing. In addition the STUB contains program flow information. The BODY contains the program text, and the FOOT contains indentifier cross-reference information in the form of foot-notes.

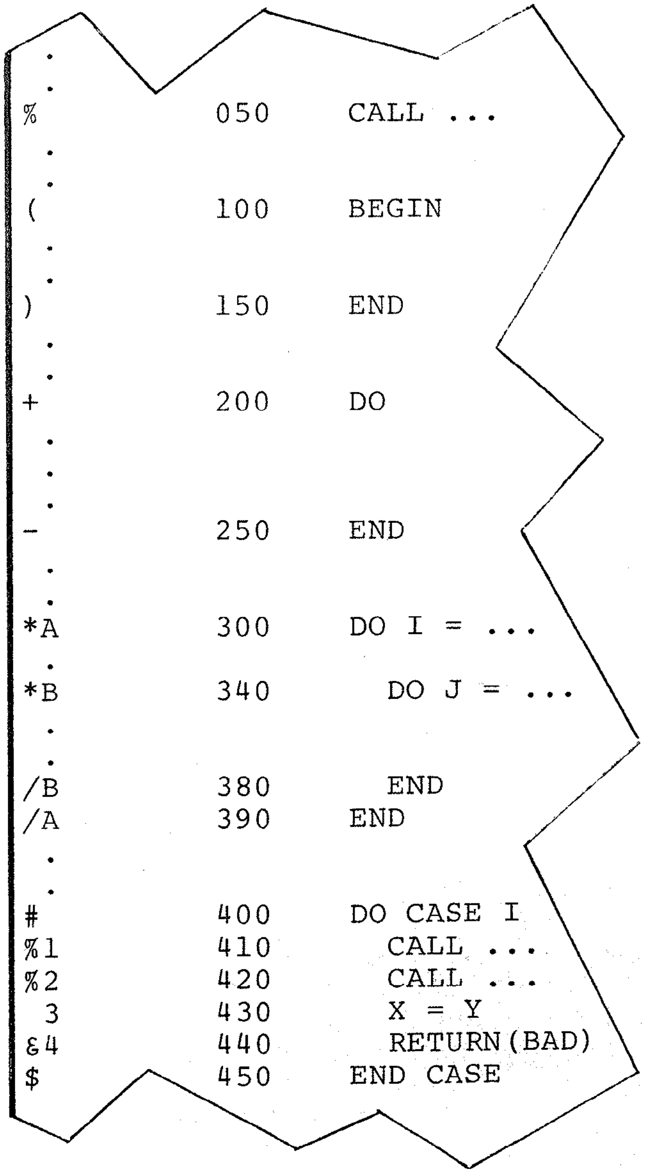
When called upon to supply program text for the language processor, the editor transmits only the BODY portion of each successive "page", but maintains intact the other portions. The editor might also generate and/or update the information in these other portions as discussed below. The complete text handled by the editor can be basically a string of characters, with a special character connoting the major separations: a double such character simultaneously marks the beginning of a new page and the end of the last, after which the first occurrence of this character marks the end of the HEAD and the second occurrence the end of the BODY. The STUB is separated from the BODY by an implied (and perhaps dynamically definable) tab which is easily implemented by a count from every carriage return within the BODY.

Possible variations of this scheme include letting the STUB extend upward and/or downward to the paper's edges. Also a right hand STUB could be defined in addition to or in place of the left. The BODY could be divided into sub-regions for declarations and imperative text as was done in NELIAC (7). Note that line numbering may repeat on each "page".

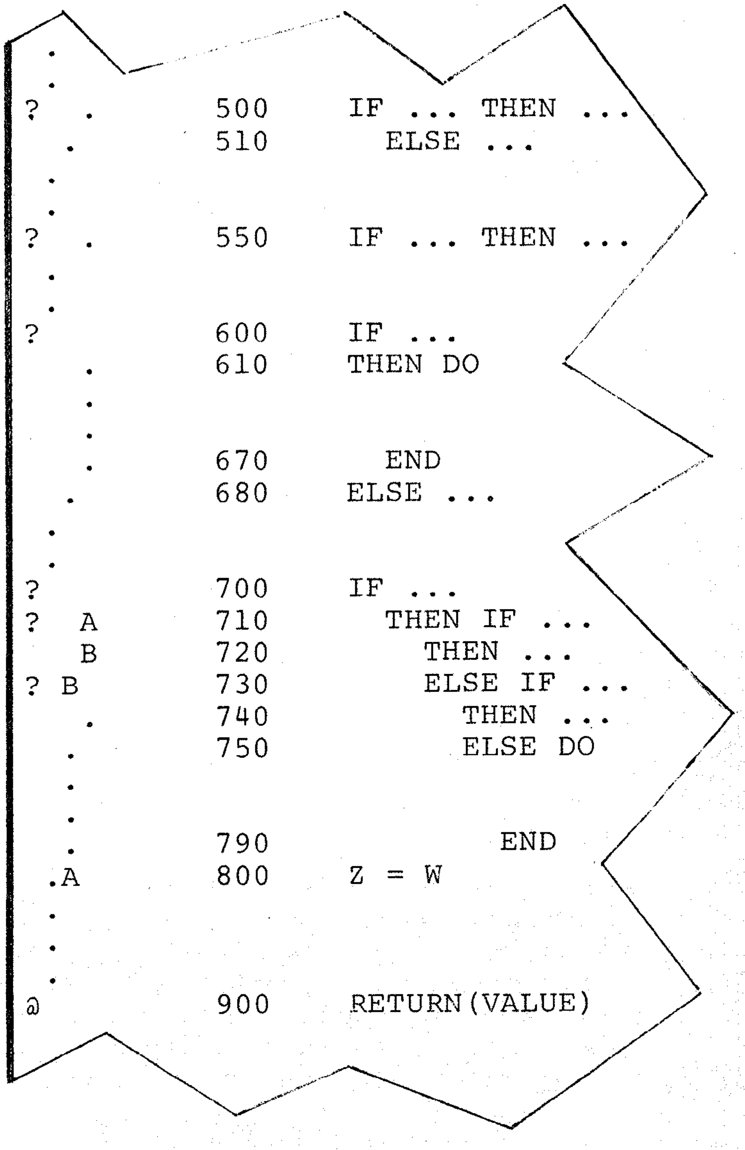
The foot-notes in the FOOT region can be numbered, and their antecedent markea in the BODY by enclosing the numbers in special characters. For example, in PL/1, the otherwise unused double quote character would do nicely. Thus in the BODY, an identifier, say, would be referenced as ... ❮indent❯ "5" ... , and in the FOOT would appear: 5 ❮cross-reference information❯ . In this way also, variable binding information might be documented. Of course, the editor should delete such foot-note references when transmitting BODY text to a language processor.

The implementation of an editor such as envisioned here could be on at least two levels. The least sopisticated approach would not provide for any feedback from the language processor to the regions of a "page." The programmer would simply provide the text for the various regions to the editor so as to come up with a well documented final copy, but would have to revise the various entries on line as changes are made to the program. At a higher level of sophistication the editor could receive information back from the language processor and automatically produce or modify the documentation in the HEAD, STUB and FOOT regions.

The various regions on a "page" have been discussed in a general way. By far the most important aspect of this scheme is the specific flow information which appears in the STUB. Since this is best shown by examples, my contribution concludes with some samples which are self-explanatory, and include various flow-units discussed in (3).



```
 .
 .
%        050    CALL ...
 .
 .
(        100    BEGIN
 .
 .
)        150    END
 .
 .
+        200    DO
 .
 .
 .
-        250    END
 .
 .
*A       300    DO I = ...
 .
*B       340      DO J = ...
 .
 .
/B       380      END
/A       390    END
 .
 .
#        400    DO CASE I
%1       410      CALL ...
%2       420      CALL ...
 3       430      X = Y
&4       440      RETURN(BAD)
$        450    END CASE
```



```
 .
 .
?  .         500    IF ... THEN ...
  .          510      ELSE ...
 .
 .
?  .         550    IF ... THEN ...
 .
 .
?            600    IF ...
   .         610    THEN DO
   .
   .
   .         670      END
  .          680    ELSE ...
 .
 .
?            700    IF ...
?  A         710      THEN IF ...
   B         720        THEN ...
? B          730        ELSE IF ...
   .         740          THEN ...
  .          750          ELSE DO
  .
  .
  .          790            END
 .A          800    Z = W
 .
 .
 .
@            900    RETURN(VALUE)
```

A system such as outlined above would make programming a delight, and go a long way toward reducing the cost of building programs.

REFERENCES

(1) McCracken, Daniel D., "Revolution in Programming: An Overview," Datamation, 19, 12, December, 1973, p. 50.

(2) Donaldson, James R., "Structured Programming," Datamation, 19, 12, December, 1973, p. 52.

(3) Nassi, J. and Shneiderman, B., "Flowchart Techniques for Structured Programming," SIGPLAN Notices, 8, 8, August, 1973, p. 12.

(4) George, James E. and Sager, Gary R., "Variables--Bindings and Protection," SIGPLAN Notices, 8, 12, December, 1973, p. 18.

(5) Celko, Joe, Letter to Editor, SIGPLAN Notices, 8, 10, October, 1973, p. 3.

(6) Ehrman, John R., "System Design, Machine Architecture, and Debugging", SIGPLAN Notices, 7, 8, August, 1972, p. 8.

(7) Halstead, Mauris Howard. Machine Independent Computer Programming. Washington: Spartan Books, 1962.