N73-19223

TREE-STRUCTURED INFORMATION FILE AND ITS SUBPROGRAM SUBTREE

Charles K. Mesztenyi
University of Maryland

The goal of automatic documentation of computer programs is to establish procedures, called documentation programs, that can be implemented by computer programs. These documentation programs may be divided into two categories: postmortem and developmental documentation programs. In the former case, a computer program is presented as input for documentation without any preparation; in the latter case, the program to be documented must be developed so that it contains information necessary for the documentation.

This paper is concerned only with the development documentation programs. A document tree is defined as the syntactic representation of a document when it is divided into subdivisions such as chapters and sections. A developmental tree is defined as a tree of information obtained during the course of the development of a computer program. The task of documenting a computer program is then made equivalent to a transformation of its developmental tree into a document tree. When this transformation is performed by a computer program, the documentation can be achieved automatically.

There is no attempt made in this paper to define the document tree more precisely. Only its tree structure is assumed. Efforts are concentrated on the developmental tree, specifically a subtree of it; the subprogram tree is illustrated in more detail.

GENERAL APPROACH

In the development of documentation programs, two objectives are paramount. Pieces of information about the program to be documented should be kept in a computer file during the development of the program, and this information should not be duplicated in the file. The importance of the first objective is obvious; the information should be in a computer-readable form for documentation. The importance of the second objective can be seen whenever a change is made during or after the development of the program to be documented. One can easily make the mistake of changing information in one place and forgetting about it in the other place. On the other hand, a change of information at a certain place may require changes in other information.

The goal of this project is to structure the developmental file of information in a tree structure (fig. 1) so that the nodes represent pieces of information. Any change in the

PRECEDING PAGE BLANK NOT FILMED

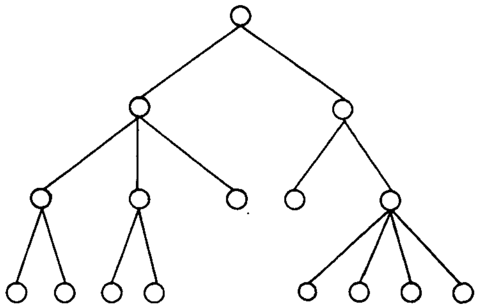

Figure 1.–Tree structure.

contents of a node may require changes in the subtree rooted in that node. In certain cases when the semantic structure is more complex, i.e., it may represent a directed graph, pointers may be used semantically.

The final documentation of a program is produced from its developmental tree of information. A special tree-traversing program, possibly interactive, selects out the contents of nodes or subtrees, invokes certain documentation programs to transform these data into special format, and stacks this information sequentially. The sequentially stacked information is processed by a listing program to produce the final printed document.

Obviously the main problem is the establishment of the developmental tree structure. At this time, a complete tree structure cannot be proposed. The definition of certain types of subtrees, however, has been accomplished. One of these, a source program subtree, is described in detail.

FLOWCHARTING AND PROGRAM LISTINGS

Any large computer program should be segmented into subprograms, subroutines, and procedures. The size of a subprogram may depend on its complexity and on its source language. Documentation of a subprogram is usually done in three different forms: textual description, flowchart, and source language listing.

The information should be structured as a tree. A source program is compiled (assembled), which generates a relocatable program. Figure 2 then defines the tree.

Certain information such as size, entry points, and external references can be obtained from the compiler-generated relocatable program. The rest of the information should be put into the source program. Textual information can easily be placed into the source program by grouped comment lines. Thus the source program may be defined as a tree, as seen in figure 3.

To combine the flowchart with the source program creates some problems. A special

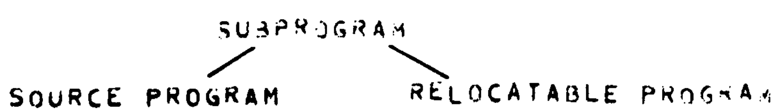

Figure 2.–Tree structure for subprogram.

Figure 3.–Tree structure for source program.

form called a sequence chart is used. This is not a complete flowchart in the standard sense, but it forces a tree on the otherwise graph-structured flowchart. Then there is no problem in listing a tree structure sequentially. The missing links of the graph structure, which appear as transfer statements in the source program, can be implemented by semantic comments. A special computer program for a source language can automatically flag these places.

Appendixes A, B, and C show the final printed forms of three different subprograms. The right side of the lists contains the actual program statements; the left side is stored internally as coded comments. The listing program takes care of this separation, but the actual sequential form is kept in the vertical direction. Those flow lines that represent the spanning tree of the program are shown with special characters, colons, periods, and asterisks. The groups of textual descriptions are separated by horizontal lines of asterisks. Both the names of the groups and the characters used for line drawing are made flexible by changing an internal table in the printing program. Special print programs are available: A "level" print gives only those lines that are not indented more than a certain input parameter. A "selective" print gives only a subtree; i.e., a defined group or a subtree of the body. The output of these print routines, formatted for a document processor, can be kept in the computer.

This form of documentation has been very helpful in the project from which these three examples were taken. During the debugging stage, it was easy to follow the sequence chart to locate a specific segment of a subprogram without turning pages back and forth.

Obviously, to get these forms, a good editing program capable of performing insertions and changes is needed. Appendixes D and E show appendix A in a developmental stage. In appendix D the initial sequence chart is defined. In appendix E an update procedure is shown. First the sequence chart is shown in a coding sheet geometrically; then its code is placed in front of it. The code for a line is composed by two fields. The first field defines either the depth of the text, 0 to 9, and blanks for program statements or contains special instructions, like group heading, change, and insert commands. The second field contains subcodes, such as line drawing codes for sequence charts and line numbers for updating commands. The text appears in the third field. In the actual input, the text field gets left adjusted. The lines will not be represented because they are already defined by codes.

This procedure for writing a program has the following advantages:

(1) It provides an up-to-date documentation of the program in the developmental stage.
(2) It forces a programmer to lay out his program so that it provides an automatic documentation at any level.
(3) It provides a form for a project leader to define subprograms without details that can be inserted by other programmers.
(4) It may be used for the present-day coded flowcharting programs.

Its main disadvantage is that it needs more work and discipline in the beginning.

SUMMARY

Printed documents have syntactic tree structures, such as titles, chapters, and sections. The semantic contents of the document may have more complex graph structures, but these structures are implemented by semantic references. A computer program has a graph structure also, but a spanning tree on this graph can be defined with semantic references to the missing links. This developmental tree of a program may have a different arrangement from a document tree. If the necessary information is contained in the developmental tree for the document tree, a transformation program can produce a document tree from the developmental tree. If the structures of the two trees are standardized, then this transformation can be achieved automatically. Otherwise, an interactive transformation routine can achieve a semiautomatic documentation.

APPENDIX A—PRINTED SUBPROGRAM: EXAMPLE 1

```
                                                              SUBROUTINE EXPRES (*, ISW)
*****************************************************
             TITLE
EXPRESSION TRANSLATOR, INFIX TO PREFIX
*****************************************************
             ABSTRACT
*** AUTHOR: C.K.MESZTENYI
*** DATE: JULY 21, 1970
*** LAGUAGE: FORTRAN 5
*** PROJECT: FORMAL - SUBROUTINE
*** SEARCH KEYS: NONE
*****************************************************
             DATA STRUCTURE
FORMAL.CMMN
                                                              INCLUDE CMMN
FORMAL.PWORD
                                                              INCLUDE PWORD
*** ARGUMENT:     *   ERROR RETURN
                ISW   INPUT ARGUMENT
*****************************************************
             SPECIFICATION
THIS IS A GENERALIZED EXPRESSION TRANSLATION ROUTINE FROM
INFIX TO PREFIX FORM. IT ASSUMES THAT THE CALLING ROUTINE
INITIALIZED THE SCANNER, THUS GSCANR GIVE THE CONSECUTIVE
LOGICAL SYMBOLS. THE ROUTINE MAY BE CALLED FROM 4 DIFFERENT
PLACES DEPENDING ON ISW:
ISW = 0 PROCESS AN ASSIGN STATEMENT: VARIABLE = EXPRESSION ;
    = 1 TRANSLATE THE EXPRESSION PART FROM A READ-IN DATA
        WHICH MAY BE IN THE FORM:  EXPRESSION ;
                            OR     VARIABLE = EXPRESSION ;
    = 2 PROCESS SUBSCRIPT EXPRESSION IN THE FORM:
                      EXPRESSION )
    = 3 PROCESS AN EXPRESSION IN THE FORM:
                 EXPRESSION ;
IN THE FIRST CASE, THE INFORMATIONS FOR THE VARIABLE ARE
STORED IN N1,N2,N3. IN THE SECOND CASE, ONLY THE EXPRESSION
PART IS RETAINED UPON RETURN. IN ALL CASES, THE TRANSLATED
AND SIMPLIFIED EXPRESSION IS PLACED ABOVE THE PUSH-DOWN
STACK WITH THE PUSH-DOWN STACK CONTAINING ONLY ONE ENTRY:
A COMMA WITH A COUNT CORRESPONDING THE NUMBER OF
EXPRESSIONS TO ACCOMODATE LISTS.
*****************************************************
             METHOD
AFTER INITIALIZATION, THE LOGICAL BCD SYMBOLS ARE OBTAINED
BY GSCANR AND PROCESSED ONE-BY-ONE IN A LOOP. PROCESSING A
SYMBOL IS DONE AS FOLLOWS:
   FIRST, IT IS CHECKED IF THE SYMBOL IS IN CORRECT TEXT;
   THEN
CONSTANTS- ARE LINKED IN ABOVE THE PUSH-DOWN STACK;
VARIABLES - THEIR VALUES ARE OBTAINED FROM THE SYMBOL
          TABLE AND LINKED ABOVE THE PUSH-DOWN STACK.
          IF THE VARIABLE IS SUBSCRIPTED, OR IT IS A
          FUNCTION IDENTIFIER, THEN THE NAME IS LINKED
          IN ABOVE THE PUSH-DOWN STACK, AND A LEFT
          PARENTH. IS PLACED IN THE PUSH-DOWN STACK WITH
          COUNT=1.
```

LEFT PARENTH. - IS PLACED IN THE PUSH-DOWN STACK WITH COUNT=0.

OPERATORS - THE PUSH-DOWN STACK IS EMPTIED OUT BY STKOUT UNTIL ITS TOP ELEMENT HAS PRECEDENCE NUMBER EQUAL TO OR LESS THAN THE PRECEDENCE NUMBER OF THE OPERATOR. THEN THE OPERATOR IS PLACED IN THE PUSH-DOWN STACK. SIMPLIFICATION IS PERFORMED BY STKOUT.

RIGHT PARENTH., RIGH BRACKET - THE PUSH-DOWN STACK IS EMPTIED OUT BY STKOUT UNTIL THE MATCHING LEFT PARENTH. IS FOUND. IF THAT HAS A COUNT=0, IT IS DISCARDED TOGETHER WITH THE RIGHT PARENTH. IF IT HAS A NON-ZERO COUNT, THEN IT INDICATES AN END OF SUBSCRIPTS (PAR.) OR END OF FUNCTION ARGUMENTS (BRACKET). IN CASE OF END OF SUBSCRIPTS, THE SUBSCRIPTS ARE COLLECTED AND THE VALUE OF THE SUBSCRIPTED VARIABLE IS OBTAINED FROM THE SYMBOL TABLE, WHICH IS LINKED IN. IN CASE OF END OF ARGUMENT LIST, THE FUNCTION IDENTIFIER IS OBTAINED AND LINKED IN

SEMICOLON - INDICATES THE END OF EXPRESSION. THE PUSH-DOWN STACK IS EMPTIED OUT BY STKOUT.

****

LOCAL VARIABLES

LOGICAL VARIABLE 'SB' IS TRUE WHENEVER THE SCANNED SYMBOL IS IN SUBSCRIPT LEVEL. 'SBC' VARIABLE CONTAINS THE DEPTH OF THIS LEVEL.
LOGICAL VARIABLE 'EQL' IS TRUE WHEN AN '=' HAD BEEN PROCESSED ALREADY, THUS IT MAY NOT APPEAR AGAIN. '=' MAY ALSO NOT APPEAR ON SUBSCRIPT LEVEL.
THE SYNTAX OF EXPRESSIONS IS CHECKED AT EVERY SCANNED SYMBOL BY MASKING 'TEST' WHICH WAS SET BY THE PREVIOUS SYMBOL. IF THE RESULT IS NOT ZERO THEN THE EXPRESSION HAS SYNTACTIC ERROR. IN THE FOLLOWING TABLE,'A' DENOTES AN ALPHANUMERIC NAME, 'N' DENOTES A NUMERIC CONSTANT,'I' DENOTES POSITIVE INTEGER:

SYMBOL	MASKING BITS (DEC.)	RESET TEST (DEC.)
INITIAL ASSIGN	---	1000000 (64)
INITIAL OTHERS	---	0100000 (32)
A	0001110 (14)	0001000 (8)
A(	0001110 (14)	0100000 (32)
A[	1001110 (78)	0100000 (32)
N	1001110 (78)	0000100 (4)
[I]	1001110 (78)	0000100 (4)
#I	1001110 (78)	0000100 (4)
(	1001110 (78)	0100000 (32)
=	1110101 (117)	0000001 (1)
UNARY +-	1011110 (94)	0010000 (16)
BINARY +-	1110001 (113)	0010000 (16)
* / **	1110001 (113)	0010000 (16)
,	1110001 (113)	0100000 (32)
) AS SEPARATOR	1110001 (113)	0000100 (4)
]	1110001 (113)	0000100 (4)
) AS END OF SUBS.	1110001 (113)	0000010 (2)

```
               1110001 (113)        ---

    'BRT' AND 'PAR' ARE USED TO COUNT THE BRACKETS AND
    PARENTHESIS, RESPECTIVELY.
    LOGICAL 'NEG' IS SET TO TRUE BY '-' FOR THE NEXT
    CHARACTER SCANNED ONLY.
                                                                  LOGICAL SB,EQL,NEG

*****************************************************
             SEQUENCE CHART

INITIALIZE
     PUSH-DOWN STACK WITH COMMA
          :                                                       NP=IGETF1($990)
          :                                                       NP0=NP
          :                                                       C(NP)=20K10
          :                                                       D(NP)=1
     SUBSCRIPT LEVEL
          :                                                       SB= .FALSE.
          :                                                       BRT = 0
          :                                                       PAR = 0
          :                                                       SBC=0
     LOGICAL VARIABLES EQL AND END, INITIAL TEST
          :                                                       EQL= ISW .GE. 2
          :                                                       TEST=32
          :                                                       IF (ISW .EQ. 0) TEST=64
          :                                                       NEG= .FALSE.
     GO TO SUBSCRIPT START IF ISW=2
          :                                                       IF (ISW .EQ. 2) GO TO 180
LOOP TO PROCESS CONSECUTIVE SYMBOL
*    GET SYMBOL
*         :                                                30     CONTINUE
*         :                                                       CALL GSCANR($990,IND,N1,ITC,ICC)
*    BRANCH BY TYPE OF SYMBOL
*    :    IND = 1,2,3,4 FOR
*    :    INTEGER, REAL, IDENTIFIER, SPECIAL CHARACTER
*    :       :                                                    GO TO (100,110,40,60),IND
*    :... INTEGER
*    :      :                                              100    I=0
*    :      :                                                     GO TO 120
*    :... REAL
*    :      :                                              110    I=3
*    :    LINK IN CONSTANT
*    :      :  :                                           120    IF (AND(TEST,78) .NE. 0) CALL FMLERR($990,N1,I,1)
*    :      :  :                                                  TEST=4
*    :      :  :                                                  J=ILINK1(NP,I,N1)
*    :    CHANGE SIGN IF NEG IS TRUE
*    :      :  :                                                  IF (NEG) D(J)=-D(J)
*    :      :  :                                                  NEG= .FALSE.
*    :      :  :                                                  GO TO 30
*    :... IDENTIFIER
*    :    CHECK IS -1 FACTOR SHOULD BE LINKED IN
*    :      :  :                                           40     INEG = 1
*    :      :  :                                                  GO TO 500
*    :    ERROR IF IT HAS MORE THAN 6 CHARACTERS
*    :      :  :                                           50     IF (ICC .NE. 0) CALL FMLERR($990,N1,1,2)
*    :    BRANCH BY TERMINATING CHARACTER
```

```
*    :    :                                                   GO TO (130, 180, 190), ITC + 1
*    :    :... IDENTIFIER NOT TERMINATED BY ( OR [
*    :    :         :                                  130    IF (AND(TEST,14) .NE. 0) CALL FMLERR($990,N1,1,1)
*    :    :         :                                         TEST=8
*    :    :         :                                         N2=0
*    :    :    CHECK IF ITS VALUE MUST BE LINKED IN
*    :    :    :                                       140    IF (EQL .OR. SB) GO TO 160
*    :    :    :... NO, GET ITS NAME AS VALUE
*    :    :    :         :                             150    IF (N2 .NE. 0) CALL ILINK1(NP,N2+7,N3)
*    :    :    :         :                                    J=6
*    :    :    :         :                                    IF (N2 .NE. 0) J=7
*    :    :    :         :                                    CALL ILINK1(NP,J,N1)
*    :    :    :         :                                    GO TO 30
*    :    :    :... YES, GET VALUE FROM SYMBOL TABLE
*    :    :         IF UNASSIGNED, THEN GET
*    :    :         ITS NAME AS ITS VALUE
*    :    :              :                             160    CALL SYMBOL($990,1)
*    :    :              :                                    IF (EPTR .EQ. 0) GO TO 150
*    :    :         COPY EXPRESSION AND
*    :    :         LINK IT WITHOUT LEADING COMMA
*    :    :              :                                    II=ICOPY0($990,EPTR)
*    :    :              :                                    I=NEXT(II)
*    :    :              :                                    J=LASTXX($990,II,1,0)
*    :    :         IS IT A LIST
*    :    :         :                                         IF (H2(II) .EQ. 1) GO TO 170
*    :    :         :... YES, EMPTY PUSH-DOWN STACK
*    :    :         :    COMBINE COUNT FOR COMMA
*    :    :         :         :                               CALL STKOUT($990,18)
*    :    :         :         :                               IF (ITYP(NP) .GT. 17) CALL FMLERR($990,N1,1,1)
*    :    :         :         :                               D(NP)=D(NP)+H2(II)-1
*    :    :         LINK IN EXPRESSION
*    :    :         :    :                             170    CALL RMOVF1(II)
*    :    :         :    :                                    CALL ILINKN(NP,I,J)
*    :    :         :    :                                    GO TO 30
*    :    :... IDENTIFIER TERMINATED BY LEFT
*    :    :    PARENTHESIS :   A(
*    :    :         :                                  180    IF (AND(TEST,14) .NE. 0) CALL FMLERR($990,N1,1,1)
*    :    :         :                                         PAR = PAR+1
*    :    :         :                                         TEST=32
*    :    :    SUBSCRIPTED VARIABLE, LINK IN NAME
*    :    :    AND PLACE '(' WITH COUNT 1 INTO THE
*    :    :    STACK, INCREASE SUBSCRIPT LEVEL
*    :    :         :                                         NP=ILINK1(NP,17,1)
*    :    :         :                                         CALL ILINK1(NP,7,N1)
*    :    :         :                                         SB= .TRUE.
*    :    :         :                                         SBC=SBC+1
*    :    :         :                                         GO TO 30
*    :    :--- IDENTIFIER TERMENATING WITH LEFT
*    :         BRACKET :   A[
*    :              :                                  190    IF (AND(TEST,78) .NE. 0) CALL FMLERR($990,N1,1,1)
*    :         GET FUNCTION IDENTIFIER,
*    :         BRANCH BY TYPE
*    :         :                                              I=IFUNCT(N1)
*    :         :                                              IF (I .EQ. 0) GO TO 210
*    :         :                                              BRT = BRT+1
*    :         :                                              IF (I .GT. 1) GO TO 200
*    :         :... DIFFERENTIAL FUNCTION
```

```
*   :      :      :                                                  NP=ILINK1(NP,23,0)
*   :      :      :                                                  GO TO 240
*   :      :... MATH. OR FORMAL FUNCTION
*   :      :      :                                          200     NP=ILINK1(NP,21,0)
*   :      :      :                                                  H1(NP)=I
*   :      :      :                                                  GO TO 240
*   :      :... NONE OF ABOVE, CHECK SYMBOL TABLE
*   :          :                                             210     N2=0
*   :          :                                             220     CALL SYMBOL($990,1)
*   :          :                                                     BRT = BRT+1
*   :          :                                                     IF (EPTR .EQ. 0) GO TO 230
*   :          :... DEFINED FUNCTION
*   :          :    LINK IN EXPRESSION
*   :          :          :                                          II=ICOPY0($990,EPTR)
*   :          :          :                                          I=NEXT(II)
*   :          :          :                                          J=LASTXX($990,II,1,0)
*   :          :          :                                          NP=ILINK1(NP,22,D(II))
*   :          :          :                                          CALL ILINKN(NP,I,J)
*   :          :          :                                          CALL IFREE1(II)
*   :          :          :                                          GO TO 240
*   :          :... UNDEFINED FUNCTION
*   :                     :                                  230     NP=ILINK1(NP,24,N1)
*   :               LINK IN COMMA FOR
*   :               THE ARGUMENTS FOLLOWING
*   :                     :                                  240     NP=ILINK1(NP,16,1)
*   :                     :                                          TEST=32
*   :                     :                                          GO TO 30
*   :... SPECIAL CHARACTERS
*       BRANCH BY THE SPECIAL CHARACTERS
*        :                                                   60      GO TO (270,280,290,270,300,340,350,270,440,270,270,270,390,400,
*        :                                                          1   420,270,270,270,270,430,270,460,460,410,270,270,270),N1
*        :... ILLEGAL SPECIAL CHARACTERS
*        :        :                                          270     CALL FMLERR($990,N1,1,1)
*        :... LEFT BRACKET OR ID. ENCLOSED IN BRACKETS
*        :      [   [I]
*        :        :                                          280     IF (AND(TEST,78) .NE. 0) CALL FMLERR($990,N1,1,1)
*        :    UNDEFINED FUNCTION ARGUMENT OR SUBSCRIPTED
*        :    FUNCTION CHECK IF IT IS SUBSCRIPTED
*        :    FUNCTION
*        :    :... YES, GO TO FUNCTION PART
*        :    :    OF DEFINITION
*        :    :         :                                            IF (TEST .EQ. 2) GO TO 220
*        :    :... NO, IT IS A DUMMY ARGUMENT,
*        :         THEN IT MUST BE FOLLOWED BY
*        :         AN INTEGER AND RIGHT BRACKET
*        :              :                                            INEG = 2
*        :         CHECK IF -1 FACTOR IS NEEDED
*        :              :                                            GO TO 500
*        :              :                                    285     CALL GSCANR($990,IND,IDT,ITC,ICC)
*        :              :                                            IF (IND .NE. 1 .OR. IDT .LE. 0) CALL FMLERR($990,ITC,1,1)
*        :              :                                            I=ILINK1(NP,5,IDT)
*        :              :                                            CALL GSCANR($990,IND,IDT,ITC,ICC)
*        :              :                                            IF (IND .NE. 4 .OR. IDT .NE. 3) CALL FMLERR($990,ITC,1,1)
*        :              :                                            TEST=4
*        :              :                                            GO TO 30
*        :... RIGHT BRACKET   ]
*        :    END OF FUNCTION ARGUMENTS
```

```
*        :         :                        290   IF (AND(TEST,113) .NE. 0) CALL FMLERR($990,ITC,1,1)
*        :         :                              BRT = BRT-1
*        :         :                              IF (BRT .LT. 0) CALL FMLERR($990,ITC,1,4)
*        :         :                              TEST=4
*        :         :                              CALL STKOUT($990,17)
*        :         :                              IF (ITYP(NP) .NE. 16) CALL FMLERR($990,ITC,1,1)
*        :         :                              I=D(NP)
*        :         :                              J=NP
*        :         :                              NP=LAST(NP)
*        :         :                              CALL RMOVF1(J)
*        :         :                              J=ITYP(NP)
*        :         :                              IF (J .LT. 21) CALL FMLERR($990,ITC,1,4)
*        :         :                              IF (J .EQ. 24) ITYP(NP)=I+24
*        :         :                              IF ((J .EQ. 24) .AND. ((I+24) .GT. 31))
*        :         :                             1    CALL FMLERR($990,D(NP),1,3)
*        :         :                              IF (J .LT. 24) H2(NP)=I
*        :         :                              CALL STKOUT($990,21)
*        :         :                              GO TO 30
*        :... RIGHT PARENTHESIS    )
*        :         :                        300   IF (AND(TEST,113) .NE. 0) CALL FMLERR($990,ITC,1,1)
*        :    CHECK IF THIS IS AN END OF SUBSCRIPT
*        :    OR THE END OF A SUBEXPRESSION
*        :    :                                   PAR = PAR-1
*        :    :                                   IF (PAR .LT. 0) CALL FMLERR($990,ITC,1,4)
*        :    :                                   CALL STKOUT($990,18)
*        :    :                                   IF (ITYP(NP) .NE. 17) CALL FMLERR($990,ITC,1,4)
*        :    :                                   IF (D(NP) .NE. 0) GO TO 310
*        :    :... END OF SUBEXPRESSION,
*        :    :    REMOVE MATCHING '('
*        :    :         :                         I=NP
*        :    :         :                         NP=LAST(NP)
*        :    :         :                         CALL RMOVF1(I)
*        :    :         :                         TEST=4
*        :    :         :                         GO TO 30
*        :    :... END OF A SUBSCRIPT LIST
*        :         CHECK AND PACK SUBSCRIPTS
*        :              :                   310   TEST=2
*        :              :                         N2=D(NP)
*        :              :                         IF (N2 .GT. 4) CALL FMLERR($990,N2,0,5)
*        :              :                         N3=0
*        :              :                         SBC=SBC-1
*        :              :                         IF (SBC .EQ. 0) SB= .FALSE.
*        :              :                         DO 320 KK=N2-1,0,-1
*        :              :                         K=NEXT(NP)
*        :              :                         IF (ITYP(K) .NE. 0) CALL FMLERR($990,D(K),2,13)
*        :              :                         IF (D(K) .LT. 0 .OR. D(K) .GT. 511) CALL FMLERR($990,D(K),0,15)
*        :              :                         FLD(9*KK,9,N3) = D(K)
*        :              :                   320   CALL RMOVF1(K)
*        :         CHECK IF THIS IS THE END OF
*        :         TRANSLATION (ISW=2)
*        :         :                              IF ((ISW .EQ. 2) .AND. (.NOT. SB)) GO TO 330
*        :         :... NO, GO BACK TO VARIABLE PART
*        :         :    TO GET THE VALUE OF THE
*        :         :    SUBSCRIPTED VARIABLE
*        :         :         :                    J=NEXT(NP)
*        :         :         :                    N1=D(J)
*        :         :         :                    K=NP
```

```
*        :         :         :                            NP=LAST(NP)
*        :         :         :                            CALL RMOVFN(K,J)
*        :         :         :                            GO TO 140
*        :         :... YES, RETURN FOR ISW=2
*        :                   :                    330     CALL IFREEQ(NPO)
*        :                   :                            RETURN
*        :... MINUS   -
*        :    SET 'NEG= AND LINK IN +
*        :         :                              340     NEG= .TRUE.
*        :... PLUS    +
*        :    IS IT UNARY OR BINARY
*        :    :                                   350     IF (AND(TEST,94) .NE. 0) GO TO 360
*        :    :... UNARY PLUS OR MINUS
*        :    :         :                                 CALL STKOUT($990,18)
*        :    :         :                                 TEST=16
*        :    :         :                                 GO TO 30
*        :    :... BINARY + -
*        :              :                         360     J=18
*        :    COMMON PART FOR BINARY OPERATORS
*        :         :                              370     I=2
*        :         :                              380     IF (AND(TEST,113) .NE. 0) CALL FMLERR($990,ITC,1,1)
*        :         :                                      TEST=16
*        :         :                                      CALL STKOUT($990,J)
*        :         :                                      NP=ILINK1(NP,J,I)
*        :         :                                      GO TO 30
*        :... MULTIPLICATION   *
*        :         :                              390     J=19
*        :    GO TO BINARY OPERATOR
*        :         :                                      GO TO 370
*        :... EXPONENTIAL   **
*        :         :                              400     J=20
*        :    GO TO BINARY OPERATOR
*        :         :                                      GO TO 370
*        :... DIVISION   /
*        :         :                              410     I=-2
*        :         :                                      J=19
*        :    GO TO BINARY OPERATOR
*        :    SECOND ENTRY
*        :         :                                      GO TO 380
*        :... LEFT PARENTHESIS   (
*        :         :                              420     IF (AND(TEST,78) .NE. 0) CALL FMLERR($990,ITC,1,1)
*        :         :                                      PAR = PAR+1
*        :         :                                      TEST=32
*        :         :                                      INEG = 3
*        :    GO TO CHECK FOR -1 FACTOR
*        :         :                                      GO TO 500
*        :         :                              425     NP=ILINK1(NP,17,0)
*        :         :                                      GO TO 30
*        :... COMMA
*        :         :                              430     IF (AND(TEST,113) .NE. 0) CALL FMLERR($990,ITC,1,1)
*        :         :                                      TEST=32
*        :         :                                      CALL STKOUT($990,18)
*        :         :                                      D(NP)=D(NP)+1
*        :         :                                      GO TO 30
*        :... EQUAL SIGN   =
*        :         :                              440     IF (AND(TEST,117) .NE. 0) CALL FMLERR($990,ITC,1,1)
*        :         :                                      TEST=1
```

```
*          :          :
*          :          :
*          :       GET AND CHECK VARIABLE FOR ASSIGN
*          :       STATEMENT SAVE INFO. IN NN1,NN2,NN3
*          :          :
*          :          :
*          :          :
*          :       GO BACK TO TRANSLATE EXPRESSION
*          :          :
*          :...  SEMICOLON   ;
*          :...  APOSTROPHE  '
*                GO TO END OF TRANSLATION
*                     :
END OF LOOP

END OF TRANSLATION
       :
       :
     RETURN FOR 'ISW' = 1 AND 3
       :
     TRANSFER ASSIGNED VARIABLE INFORMATION
       :
     RETURN FOR 'ISW' =0
       :
CHECK IF -1 FACTOR MUST BE INSERTED
     INDICATED BY 'NEG'
     :... NO, GO TO RETURN TO CALLING PLACE
     :          :
     :... YES, LINK IT IN
                :
          RETURN TO CALLING PLACE
                :
ERROR RETURN
     :
     :
```

```
      IF (EQL .OR. SB) CALL FMLERR($990,ITC,1,1)
      EQL= .TRUE.
      CALL STKOUT($990,18)

      IF ((ITYP(NP) .NE. 16) .OR. (D(NP) .NE. 1) .OR. (LAST(NP) .NE.  ))
     1    CALL FMLERR($990,ITC,1,1)
      KK=NEXT(NP)
      IF (ISW .EQ. 1) GO TO 450
      IF ((KK .EQ. 0) .OR. (ITYP(KK) .LT. 6) .OR. (ITYP(KK) .GT. 7))
     1    CALL FMLERR($990,ITC,1,1)
      NN1=D(KK)
      NN2=0
      IF (ITYP(KK) .EQ. 6) GO TO 450
      NN3=NEXT(KK)
      NN2=ITYP(NN3)-7
      NN3=D(NN3)
450   CALL IFREEO(KK)
      NEXT(NP)=0

      GO TO 30


460   IF (AND(TEST,113) .NE. 0) CALL FMLERR($990,ITC,1,1)


      IF (PAR .NE.0 .OR. BRT .NE.0) CALL FMLERR($990,'()  []',1,4)
      CALL STKOUT ($990, 18)
      IF ((ITYP(NP) .NE. 16) .OR. (LAST(NP) .NE. 0))
     1         CALL FMLERR($990,ITC,1,1)

      IF (ISW .NE. 0) RETURN
      IF (.NOT. EQL) CALL FMLERR($990,ITC,1,1)

      N1=NN1
      N2=NN2
      N3=NN3

      RETURN


500   IF (.NOT. NEG) GO TO 510

      NP = ILINK1(NP,19,2)
      CALL ILINK1(NP,0,-1)
      NEG = .FALSE.

510   GO TO (50,285,425), INEG

990   CALL IFREEO(NP0)
      RETURN 1

      END
```

APPENDIX B—PRINTED SUBPROGRAM: EXAMPLE 2

```
****************************************************
             TITLE
MAIN PROGRAM FOR INTERACTIVE FORMAL SYSTEM

****************************************************
             SEQUENCE CHART
INITIALIZE BY CALLING FMLOPT
     :
LOOP TO GET NEXT INPUT LINE
*    READ LINE
*         :
*         :
*    IF IT STARTS WITH 'C ' (COMMENT), GO TO GET NEXT
*    LINE
*         :
*    IF IT STARTS WITH 'P ' (PRINT), GO TO 'P ' ENTRY
*         :
*         :
*    LOOP TO GET STATEMENT TYPE IN J
*    *
*    *
*    END OF LOOP
*    J=0, IT IS AN ASSIGN STATEMENT
*         :
*         :
*    REPRINT ERASE,OPTION,ROLOUT,SAVE AND RESET
*    STATEMENTS
*         :
*         :
*         :
*    BRANCH BY TYPE
*    :
*    :... READ STATEMENT
*    :         :
*    :         :
*    :... PRINT STATEMENT
*    :         :
*    :         :
*    :    'P ' = PRINT
*    :         :
*    :         :
*    :         :
*    :... DUMP STATEMENT
*    :         :
*    :         :
*    :         :
*    :         :
*    :         :
*    :... ERASE STATEMENT
*    :         :
```

```
         PARAMETER IDIM = 10
         DIMENSION IN(14), INN(14), ITAB(IDIM)
         EQUIVALENCE (IN(2), INN(1))
         DATA INN(14) / ';' /
         DATA ITAB /'READ  PRINT DUMP  ERASE OPTIONCOMMEN'
        + ,'ROLOUTNCOUNTSAVE  RESET '/

 99      CALL FMLOPT ('INT;',0)

 110     READ 100, END=200, IN
 100     FORMAT (13A6,A2)

         IF (FLD(0,12,IN(1)) .EQ. 1005K) GO TO 110

         IF (FLD(0,12,IN(1)) .EQ. 2505K) GO TO 22
         J = 0

         DO 111 I = 1,IDIM
 111     IF (IN(1) .EQ. ITAB(I)) J = I

         IF (J) ,60,
         GO TO (121, 121, 121, 120, 120, 110, 120, 121, 120, 120), J

 120     CONTINUE
         PRINT 101, IN
 101     FORMAT (XA6,':',13A6)

 121     GO TO (1, 2, 3, 4, 5, 110, 7, 8, 9, 10), J

 1       CALL FMLIO1 (INN,0)
         GO TO 110

 2       CALL FMLIO2 (INN,0)
         GO TO 110

 22      FLD(0,6,IN(1)) = 0505K
         CALL FMLIO2 (IN, 0)
         GO TO 110

 3       CALL ONDMP
         K = 'P'
         IF (INN(1) .NE. ' ') K = 0
         CALL DUMP(K)
         CALL OFFDMP
         GO TO 110

 4       CALL FMLERS (INN,0)
```

```
*    :         :
*    :... OPTION STATEMENT
*    :         :
*    :         :
*    :... ROLOUT STATEMENT
*    :         :
*    :         :
*    :... NCOUNT STATEMENT
*    :         :
*    :         :
*    :... SAVE STATEMENT
*    :         :
*    :         :
*    :... RESET STATEMENT
*              :
*              :
*    ASSIGN STATEMENT
*         :
*         :
*         :
*         :
END OF FILE READ - STOP
*    :
*****************************************************
```

```
         GO TO 110
5        CALL FMLOPT (INN, 0)
         GO TO 110
7        CALL FMLOUT (INN,0)
         GO TO 110
8        CALL COUNT
         GO TO 110
9        CALL FMLSAV (INN)
         GO TO 110
10       CALL FMLRES (INN)
         GO TO 99
60       PRINT 102, IN
102      FORMAT (X14A6)
         CALL FMLASG (IN,0)
         GO TO 110
200      STOP
         END
```

APPENDIX C—PRINTED SUBPROGRAM: EXAMPLE 3

```
**********************************************              CMMN PROC
              TITLE
COMMON DATA STRUCTURE FOR FORMAL SYSTEM
**********************************************
              DATA STURCRUE
ARRANGED IN 3 LABELED COMMONS
USED AS PROCEDURE, INCLUDED IN OTHER SUBPROGRAMS
                                                                    IMPLICIT INTEGER(A-Z)
                                                                    PARAMETER ERROR = ERRERR
1. LINKED STORAGE AREA
*     THE CORRESPONDING C(I)-D(I) WORDS ARE ALWAYS
*     USED IN PAIRS FOR STORING AN ITEM.
*     THE DIMENSION OF C-D, CDIM, MAY BE CHANGED
*     DURING INSTALLATION.
*     FIELDS IN THE C-D WORDS DEPEND ON THE USAGE,
*     THEY ARE DEFINED BY PROCEDURE 'PWORD', GENERALLY,
*     THE LAST 15 BITS IN C IS USED FOR LINKAGE OF
*     LINEAR ARRAYS.
*
*                                                                   PARAMETER CDIM = 2048
*                                                                   COMMON /FMLCM2/ C(CDIM)
*                                                                   COMMON /FMLCM3/ D(CDIM)
2. COMMON BLOCK FOR INDIVIDUAL POINTERS AND SWITCHES
*
*... FREE (AVAILABLE) STORAGE IN C-D                                 COMMON /FMLCM1/
*         C(NXNXNX) = FIRST
*         C(ILILIL) = LAST LOCATION
*         THE LINEAR ARRAY IS LINKED IN THE
*         LAST 15 BITS OF THE C-WORDS.
*             :                                                   +     NXNXNX, ILILIL
*... SYMBOL TABLE WITH TREE STRUCTURE IN 4 LEVELS
*         STORED IN C-D AREA. FIELDS IN THE C-WORD:
*             ITYPB - LAST - NEXT
*         NS = FIRST ENTRY IN C(NS)-D(NS)
*         :                                                       +     NS,
*         NSB = SUBROUTINE LEVEL POINTER
*         :     SUBPROGRAMS ARE IN ALPHABETIC ORDER
*         :                                                       +     NSB,
*         :... ITYPB(NSB) = 0
*         :... D(NSB) = ALPHANUMERIC NAME OF THE
*         :    SUBROGRAM
*         :... NSY = LAST(NSB) POINTER TO SYMBOL ENTRY
*         :    :    SYMBOLS ARE IN ALPHABETIC ORDER
*         :    :    NSY1 = POINTER TO PRECEEDING SYMBOL
*         :    :    ENTRY
*         :    :                                                  +     NSY,NSY1,
*         :    :... D(NSY) = ALPHANUMERIC NAME OF THE
*         :    :    SYMBOL
*         :    :... ITYPB(NSY)= TYPE OF SYMBOL,
*         :    :    :    SEE TABLE I.
*         :    :    :... >31, INDIRECT REFERENCE
*         :    :    :    LAST(NSY) POINTS TO AN OTHER
*         :    :    :    SYMBOL
*         :    :    :
*         :    :    :... SECOND AND THIRD BIT = 11
*         :    :    :    SUBSCRIPTED VARIABLE
```

```
*        :    :    :    NSU = LAST(NSY), POINTER TO
*        :    :    :    :    SUBSCRIPT ENTRY
*        :    :    :    :    SUBSCRIPTS ARE ORDERED BY
*        :    :    :    :    NUMBER OF SUBSCRIPTS, AND
*        :    :    :    :    BY ACTUAL SUBSCRIPTS, NSU1
*        :    :    :    :    = POINTER TO PRECEEDING
*        :    :    :    :    SUBSCRIPT ENTRY
*        :    :    :    :                                        +    NSU,NSU1,
*        :    :    :    :... D(NSU) = NUMERIC SUBSCRIPT,
*        :    :    :    :    SEE TABLE II.
*        :    :    :    :... FIRST 3 BITS OF C(NSU)=
*        :    :    :    :    :... =110, UNASSIGNED,
*        :    :    :    :    :    INDIRECTLY REFERENCED
*        :    :    :    :    :         LAST(NSU) POINTS
*        :    :    :    :    :         TO OTHER SYMBOL
*        :    :    :    :    :         ENTRY
*        :    :    :    :    :... =111, LAST SUBSCRIPT
*        :    :    :    :    :    ENTRY,
*        :    :    :    :    :         LAST(NSU) POINTS
*        :    :    :    :    :         BACK TO ITS
*        :    :    :    :    :         SYMBOL = NSY
*        :    :    :    :    :... =010, NORMAL ENTRY
*        :    :    :    :    :         LAST(NSU) POINTS
*        :    :    :    :    :         TO EXPRESSION
*        :    :    :    :    :         VALUE = EPTR
*        :    :    :    :... NEXT(NSU) = FORWARD LINK TO
*        :    :    :         NEXT SUBSCRIPT,ZERO FOR
*        :    :    :         THE LAST ONE
*        :    :    :... OTHERS, LAST(NSY) = EPTR
*        :    :         EPTR = EXPRESSION POINTER
*        :    :         :                                        +    EPTR,
*        :    :         :... ITYPB(EPTR) = 16
*        :    :         :... LAST(EPTR) = 0
*        :    :         :... H2(EPTR) = NUMBER OF
*        :    :         :    EXPRESSIONS (FOR LISTS)
*        :    :         :... H1(EPTR) =
*        :    :         :    :... =0, EXPRESSION IS IN
*        :    :         :    :    CORE
*        :    :         :    :... NOT ZERO, EXPRESSION
*        :    :         :              IS ON DRUM
*        :    :         :              INDEX = H1(EPTR)
*        :    :         :... NEXT(EPTR) FORWARD LINK
*        :    :              TO THE LINEARLY STORED
*        :    :              EXPRESSION WHEN IT IS
*        :    :              ON CORE.THE CONSECUTIVE
*        :    :              ENTRIES ARE ACCORDING
*        :    :              TO TABLE III.
*        :    :... NEXT(NSY) = FORWARD LINK TO NEXT
*        :         SYMBOL, ZERO FOR LAST ONE
*        :... NEXT(NSB) = FORWARD LINK TO NEXT
*             SUBPROGRAM, ZERO FOR LAST ONE
*
*... TEMPORARY VARIABLES FOR
*        N1 = NAME OF A VARIABLE
*        N2 = NUMBER OF SUBSCRIPTS
*        N3 = SUBSCRIPT WORD
*        IY = 3 OR 1 FOR SUBSCR. OR NOT
```

```
*
*... OPTION SWITCHES
*         XQTOPT = OPTION WORD FROM WAIT STATEMENT
*         PRODEX = EXPAND POWERS OVER PRODUCT
*         INTGSW = EVALUATE INTEGER VALUED FUNCTIONS
*         MATHSW = EVALUATE MATHEMATICAL FUNCTIONS
*         EXPDSW = USE DISTRIBUTIVE LAW
*         POWER  = EXPAND SUMS RAISED TO POS. INTEGERS
*         BASE   = 0,1,2,3 FOR BASE(0),(2),(10),(E)
*
*... MISCELLANEOUS
          SIMPSW = USED BY STOUT ROUTINE FOR RECURSIVE
                   SIMPLIFICATION
          BITSW  = USED BY STOUT ROUTINES
          IOUNIT = I/O UNIT NUMBER IF I/O STATEMENTS
          FTRARG = NUMBER OF FORTRAN TYPE ARGUMENTS
          DEFARG = NUMBER OF ARGUMENTS IN A DEFINED
                   FUNCTION
          DEFFUN = 1 IF DEFINED FUNCTION, 0 FOR VARIABLE
          NK = START OF ARGUMENT CHAIN IN C-D FOR LIST
               OF VARIABLES
          CBUF = I/O BUFFER
          NP = PUSH-DOWN STACK POINTER IN C-D AREA

*************************************************

     +    N1,N2,N3,IY,

     +    XQTOPT,PRODEX,INTGSW,MATHSW,EXPDSW,POWER,BASE,

     +    SIMPSW,BITSW,IOUNIT,FTRARG,DEFARG,DEFFUN,NK,CBUF,NP

      LOGICAL INTGSW,MATHSW,POWER,SIMPSW,BITSW,PRODEX
      REFERENCES ON
      END
```

APPENDIX D—DEFINITION OF INITIAL SEQUENCE CHART

Coding Form

The coding form is divided into three fields: Field 1 consists of one character, the general directive for input; field 2 contains special directives for flowchart elements and a label for program statements; field 3 contains the text.

An initial program is illustrated below:

```
T     EXPRESSION TRANSLATION
S     INITIALIZE
0D    LOOP TO PROCESS CONSECUTIVE SYMBOLS
1D    |    BRANCH BY TYPE OF SYMBOL
2B    |    ├INTEGER
2B    |    ├REAL
2B    |    ├IDENTIFIER
2BE   |    └SPECIAL CHARACTER
0     END OF LOOP
0     END OF TRANSLATION
           END
```

Input Form

The actual input does not contain the lines; the text is left adjusted in field 3:

```
T     EXPRESSION TRANSLATION
S     INITIALIZE
0D    LOOP TO PROCESS CONSECUTIVE SYMBOLS
1D    BRANCH BY TYPE OF SYMBOL
2B    INTEGER
2B    REAL
2B    IDENTIFIER
2BE   SPECIAL CHARACTERS
0     END OF LOOP
0     END OF TRANSLATION
      END
```

Output Form

The initial program can be listed with line numbers as follows:

```
      *****************************************
1 =            EXPRESSION TRANSLATION
      *****************************************
               SEQUENCE CHART
2 = INITIALIZE
```

```
 3 = LOOP TO PROCESS CONSECUTIVE SYMBOLS
 4 = :    BRANCH BY TYPE OF SYMBOL
 5 = :    : . . . INTEGER
 6 = :    : . . . REAL
 7 = :    : . . . IDENTIFIER
 8 = :    : . . . SPECIAL CHARACTERS
 9 = END OF LOOP
10 = END OF TRANSLATION
11 =                                        END
```

APPENDIX E—EXAMPLE OF AN UPDATING PROCEDURE

```
+1
            SUBROUTINE EXPRES (*, ISW)
+R7D        |
1B          |-IDENTIFIER NOT TERMINATED BY ( OR [
1B          |-IDENTIFIER TERMINATED BY (
1BE         └-IDENTIFIER TERMINATED BY [

+R8D        |
1B          |- -
            |  NEG = .TRUE.
1B          |- +
            |  J = 18
1B          |- *
            |  J = 19
1B          |- **
            |  J = 20
1BE         └- /
               J = 19
               I = -2
```

Note that the '+' is an insertion directive. The number following + indicates the line where the insertion is to be done. 'R' indicates that the levels of lines following to be inserted are defined relative to the line where the insertion occurs.

DISCUSSION

MEMBER OF THE AUDIENCE: I notice that you have many comments noted through there. It seems to be about a two-to-one comment per statement. Is that about correct?

MESZTENYI: It depends on the program. It depends on the language, too. The comments should be semantic, not repeated as an equation.

MEMBER OF THE AUDIENCE: Do you think that some of the discussions about what we can get out of the compiler would fall into this?

MESZTENYI: I would like to have the compiler in the subroutine. I would like to do that, but I would start here from the development point first, because this is where one defines the program first.

MEMBER OF THE AUDIENCE: It seems that the compiler could give you certain information, and you could add some personal comments and have better descriptive material. Is that true?

MESZTENYI: It depends on what standpoint you look at. As I look at it, I want an overall view from the beginning. Before I finish the program, I might want to give the specification a bigger flowchart type of definition that could be used right away.

MEMBER OF THE AUDIENCE: You are trying to get the flavor of the program that you are working on for a certain purpose. The compiler will only come out with standard words for any program. The compiler does not know what your program is, but you do. With personal comments added to the program, what you have would provide additional information.

MESZTENYI: I find it is hard for programmers to add something after they have written the program. When they write, they do not mind writing down their comments.

MEMBER OF THE AUDIENCE: I am working from the viewpoint that we now have difficulty at times getting any comments in, and if we provided a lead into the comments and they went down the list and it did not make too much sense to them from a general viewpoint, that they could add these rather well.

MESZTENYI: I agree that they could, and this is actually what is now done. I added this myself.

The other part I would like to focus on a little bit is the programming part. If you start from the sketch with those lines coming down and write, you make the programmer apply a little discipline to the subject of program placement. For example, I try to avoid any GO TO unless it is some kind of loop structure. I try to avoid going back. I find a loop for each logic curve that I process, but it is not a DO statement, and I jump directly back to the beginning. It probably would have been much nicer documenting it to go to the end of this loop and comment it, which goes back and gets the next one. In this way it forces the programmer to do a documented description because it is very hard to document a graph that points out the actual information. The text or the description of the program is sequential, but semantically it is a graph. A tree, which is sort of in-between, is much easier to represent. You have cross-references, but the form is still a tree, and this is what I tried to simulate.

MEMBER OF THE AUDIENCE: I think the speaker is trying to get the programmer to write down what is being accomplished and when. Once in the right-hand side, the language does not really matter. He is trying to read narrative text so that you get some concept of when things happen and what really is happening because the specification of the problem is written in a narrative form. He does that rather than deduce what was done from how something is being done. I do not think a programmer is going to do that very well because he is so involved in the mechanics that he cannot get out of them.

MEMBER OF THE AUDIENCE: It seems to me that here is a case where we can go from the rationale of a subroutine and in an automated way feed in the programming language statements. Is this what you had in mind? I could see how you actually tried to develop your subroutine. I can see how you can start with the rationale of the subroutine first and then by using the type of coding that you did, you could automatically call for the appropriate programming language statements.

MESZTENYI: Not automatically. I certainly think of more than just the semantic type of description that I want to accomplish. What I want to accomplish eventually is the statements.