СООБЩЕНИЯ
ОБЪЕДИНЕННОГО
ИНСТИТУТА
ЯДЕРНЫХ
ИССЛЕДОВАНИЙ

ДУБНА

E10 - 9938

D.C.Marinescu

STRUCTURED PROGRAMMING

1976

E10 - 9938

D.C.Marinescu

STRUCTURED PROGRAMMING

PURPOSE

Reliability,maintainability and extensibility are major requirements of today in the area of the software developement.

Structured programming is a technique for organizing and coding programs to make them easy to be understood and modified.

The cost of the maintenance, debugging and extension of structured programs is much lower than the one for conventional programs /7/ .

It is the intention of this paper to present the basic concepts of structured programming,which is considered as a remarkable invention and to describe an extension of FORTRAN which can be used as a structured language.

The source listing of a precompiler for the extended FORTRAN is presented in Appendix 1; since it is written in its own language the precompiler is also a good example of how to use the extended language.

CONTENTS

1. Historical considerations

The theoretical foundations of structured programming can be traced back to a paper written by Böhm and Jacopini /1/ who showed that only three structures are enough to write any program whatever sophisticated it might be :

1. - simple sequences with statements executed one after the other,
2. - selection clauses of the type IF...THEN...ELSE,
3. - repetition blocks with a DO . WHILE or DO.. UNTIL loop control mechanism.

Each structural block of the program must have only one entry point and one exit point. It follows that such blocks can be combined in such a way that the flow of control goes from the top to the bottom (or from the beginning to the end) without any backtracking. For this reason, structured programming is also called top-down programming.

Apparently, there are only two situations when the use of the pure three structures might lead to inefficiency:

- when only one of a set of functions is to be performed depending upon the value of a variable. This situation is handled by computed GO TO type statements in conventional programming. This situation can be avoided by generalising the IF..THEN..ELSE selection function from two to multiple valued;
- when abnormal termination of a repetition block is foreseen.

The absence of the GO TO like statements is not felt if the three structures are skilfully used; however the resulting programs are much easier to read, to understand and to debug.

The recognition of the idea of structured programming seems to be associated to the Dijkstra's letter /2/;he warned that

! GO TO statements were potentially hazardous to the state of mind of programmers in charge of debugging complex codes ' since it forces them to examine these codes in an unnatural way.

Perhaps, this is the reason why many people tend to consider the elimination of the GO TO statement as the whole point of structured programming; but, certainly, this is not true.

We must recall that H.Mills /3/ proved that if only the three basic structures are used and a program module has only one entry point and one exit point it becomes possible to prove whether a program is correct. Sometimes it is felt that the structured programming will help to switch the study of !the program proof of correctness! from the theoretical to the practical side. Up to now all attempts to prove analytically the correctness of a program failed to lead to a practical procedure and any skilfull programmer knows that there is only one way out: debugging on the machine.

If the question of the impact of structured programming upon real life arises , we must mention the 'chief programmer team' concept of Baker /7/ which is a new approach to the problem of the managerial framework of program production ,using structured programming.

A team of six IBM people applying Baker techniques developed over 83,000 lines of high level code in 22 months . But not only this unbelivable high rate of coding is impressive;the coding error rate was: one detected error per 10,000 lines of coding, or one per man-year.

Other big projects have since used this concept;the mission simulation system used in preparation for the Skylab operation (400,000 lines of source coding) reported the same level of productivity and of error coding rates.

2. Structured programming in conventional programming languages

As we all know the three control structures are either directly available or can be easily constructed in any programming language so that top-down programming can be done by anyone aware of its existence.

Certainly, the difficulty to do it varies from one language to another.

In any assembly language , especially when a macro facility is available structured programming can be practiced but with the known low productivity of assembly program writing.

Among the high level languages ALGOL and PL/1 have features that correspond directly to the three desired structures and thus these languages are most suitable for structured programming. COBOL has equivalent capabilities too.

As far as FORTRAN is concerned, in spite of the existence of IF and DO constructs we can state that it is not an ideal language for top-down writing but, nevertheless, by applying the basic ideas of structured programming better FORTRAN programs can be obtained.

We should now quote Daniel Mc Cracken /4/: 'I predict that within three to five year future there will be at long last , a swing to PL/1 , precisely because it is well suited for structured programming'.

But for the benefit of most FORTRAN addicts it is possible to extend standard FORTRAN to a structured language and to provide a precompiler for it. In this way one can have the advantages of a structured language and still be able to use FORTRAN.

3. SFOR - an extended FORTRAN for structured programming

3.1. SFOR function. SFOR is a precompiler developed at Bell Laboratories by D.M.O'Neill in 1974 /8/ .

The idea was to provide a mean to write structured FORTRAN programs without the need to change the FORTRAN compiler or to alter in any other way an existing programming system.

The input to SFOR is a file containing a program written in the extended language; SFOR scans this file for special operators and when found expands them to standard FORTRAN and produces an output file that can be compiled by any FORTRAN compiler.

3.2. Language specifications. The following notations will be used to describe the features of SFOR :

<lex> means any legal logical expression in FORTRAN

<aex> means any legal arithmetic expression in FORTRAN

S_1 any legal FORTRAN statement

3.2.1. Logical selection statement. The format of this statement is:

```
$IF(<lex>)
   $THEN S1
         :
         S2
   $ELSE S3
         :
   $ENDI S4
         S5
```

Note, that there is also one way selection statement:

```
$IF(<lex>)
   $THEN S1
         :
   $ENDI S2
         S3
```

$FOR evaluates the logical expression ‹lex› and if it has the value TRUE, the statements S_1 to S_2 and from S_5 on, are executed; if it has the value FALSE, the statements S_3 to S_4 and from S_5 on are executed. As far as the one way selection statement is concerned either the statements S_1 to S_2 and from S_3 on are executed (when TRUE) or only statements from S_3 on (when FALSE).

3.2.2. Arithmetic selection statement. The general format of this statement is:

```
$BRANCH(‹aex›)
        $CASE S1
               :
              S2
        $CASE S3
               :
              S4
        $CASE S5
               :
              S6
               :
        $CASE Sx
               :
        $ENDB Sy
              Sz
```

Here the arithmetic expression ‹aex› is evaluated and the value of it is then converted to an integer k. Suppose, that there are n blocks (a block is delimited by two consecutive $CASE operators or by a $CASE and a $ENDB; e.g., a block is the sequence of statements from S_1 to S_2, another block is the one starting with S_x and ending with S_y). If k is in the range 1 to n, the corresponding block is selected (e.g., if k=2, statements from S_3 to S_4 are selected). If k is either zero or greater than n, the control goes to the statement following the $ENDB (in our case S_z statement). Note, that each $CASE keyword may have an optional

two character identifier appended to it. For example $CASE1, $CASE10.

3.2.3. Repetition statements. The first type of repetition statement is a WHILE ... DO statement with the format:

```
$WHILE (‹lex›)
    $DO  S1
          :
    $ENDW S2
          S3
```

Here the evaluation of the logical expression is performed; if TRUE the block starting with statements S_1 and ending with statement S_2 is executed and the logical expression is again evaluated. If the value of it is FALSE control passes to statement S_3.

A REPEAT..... UNTIL structure is also available:

```
$REPEAT S1
         :
        S2
$UNTIL (‹lex›)
        S3
```

Here the block (S_1 to S_2) is repeated until the logical expression ‹lex› has the value TRUE. Then execution resumes with S_3.

3.2.4. Internal subroutines. Internal subroutines are to be defined whenever an identical block of code must be executed at several different places in a program. The use of internal subroutines shortens considerably the main program improving its readability.

In this implementation up to 20 internal subroutines may be defined and each can be called up to 9 times.

All internal subroutine definitions must appear at the end of the program using them. Nested definitions of subroutines are not allowed. Recursive calls of subroutines are prohibited; otherwise a subroutine can call another one.

The call of a subroutine is:

$CALL name

with name a 1 to 6 character name of an internal subroutine. Note, that a $CALL statement may not be the object of a logical IF statement.

The definition of an internal subroutine is:

```
$SUB name
      S1
      :
$ENDS S2
```

3.2.5. Restrictions. As far as the writing of SFOR statements is concerned the following rules must be observed:

R1. The statements must begin at column 7 or beyond.

R2. The statements must not extend past column 72.

R3. Continuation lines must have a non zero character in column 6.

R4. If an extended statement which contains a condition is continued over several lines ,only the last continuation line may end with a right parenthesis.

As far as the syntax of extended statements is concerned there are several restrictions:

SR1. Extended statements may not have a user defined statement label.

SR2. Only one extended statement may appear on a line.

SR3. If the $BRANCH construct is used,the user is not allowed to define variable or routine names of the form NBSWi ,$1 \le i \le 99$.

SR4. If internal subroutines are coded,names of the form NSUBi with $1 \le i \le 99$ are not allowed.

SR5. No label greater than 90000 is allowed in a program using the extended language.

3.3. The availability of the extended language.

A version of SFOR has been recently installed on the CDC 6400 machine existing at Dubna. It is available for general use .

A typical deck set up for a job requesting a precompilation, compilation and execution of a SFOR written program follows:

```
JOB CARD
ATTACH(SUSSY,MARINESCU63208,ID=LCTA,MR=1)
SUSSY.
REWIND(TAPE2)
FTN(I=TAPE2)
LGO.
7/8/9
source deck in SFOR
7/8/9
data
6/7/8/9
```

With only minor changes the extended FORTRAN can be installed on practically every machine provided with a FORTRAN compiler.

BIBLIOGRAPHY

/1/ C.Böhm and G.Jacopini - Flow Diagrams, Turing Machines and Languages with only two formation rules. CACM May 1966 pp 366-371

/2/ H.Mills - Top Down Programming in Large Systems. in 'Debugging Techniques in Large Systems' R.Rustin editor Prentice Hall 71

/3/ E.W.Dijkstra - GO TO Statements Considered Harmful. CACM March 68 pp 147-148

/4/ Mc.Cracken - Revolution in Programming. Datamation Dec 73 pp 50-52

/5/ J.R.Donaldson - Structured Programming. Datamation Dec 73 pp 52-53

/6/ E.Miller and G.E.Lindamond - Structured Programming : Top-Down Approach. Datamation Dec 73

/7/ F.Baker and H.Mills - Chief Programmer Team. Datamation Dec 73 pp 58-61

/8/ D.M.O'Neill - SFOR a Precompiler for the Implementation of a FORTRAN Based Structured Language. Bell Laboratories Internal Report 1974 (unpublished)

Received by Publishing Department on July 5, 1976.

APPENDIX 1 SFOR PRECOMPILER SOURCE LISTING

```
      PROGRAM SFOR(INPUT,OUTPUT,TAPE1=INPUT,TAPE5=OUTPUT,TAPE2)
C
C     THIS VERSION OF THE SFOR PRECOMPILER  HAS BEEN DESIGNED TO WORK
C     EFFICIENTLY ON THE CDC 6400 MACHINE. CHARACTERS ARE PACKED
C     TEN TO A COMPUTER WORD.
C     SEVERAL CHARACTER MANIPULATING FUNCTIONS FROM THE MACHINE LIBRARY
C     ARE USED
C
      INTEGER BLANK,DIGITS(1)
       INTEGER NAMES(15),NAMLEN(15),NAMTYP(15),NAMERR(4)
      INTEGER ASSIG(2),NSUBS(1),NIF(6),NBSW(1),CGOTO(2)
      INTEGER CONT(2),GOTO1(2),GOTO2(2),CASE(1),COMLPR(1),RPRCOM(1)
      INTEGER LINE(8),BUFF(8),BLANKS(8),SPTR,CURSUB,CALSUB
      INTEGER BPLACE,SPLACE
      INTEGER DOLLAR,ONE,CEE
      LOGICAL EQUAL,DIGIT,DONE,NEWSUB,CASE1,SUBSH,NOMORE
      INTEGER SUBNAM(9),SUBTAB(23,9),STACK(20),STATE(20)
C
      DATA SUBTAB/207*0/
      DATA SUBNAM/9*10H          /
      DATA MAXSUB,MXCALL,MAXSTK/9,20,20/
      DATA BLANKS/8*10H          /
      DATA BLANK/1L /
      DATA DIGITS/10H0123456789/
      DATA NAMES/2LIF,4LTHEN,4LELSE,4LENDI,5LWHILE,2LDO,4LENDW,6LREPEAT
     X ,5LUNTIL,6LBRANCH,4LCASE,4LENDB,3LSUB,4LENDS,4LCALL/
      DATA NAMLEN/2,3*4,5,2,4,6,5,6,2*4,3,2*4/
      DATA NAMTYP/0,3*1,0,2*2,0,3,0,2*4,3*0/
      DATA NAMERR/2LIF,5LWHILE,6LREPEAT,6LBRANCH/
      DATA ASSIG/10HASSIGN    ,10H    TO NSUB/
      DATA NSUBS/10HGO TO NSUB/
      DATA NIF/10H      IF((,10H      .LT.,10H1).OR.(    ,
     1 10H    .GT.   ),10H) GO TO   ,10H          /
      DATA NBSW/10HNBSW  =   /
      DATA CGOTO /10H      GO T,10HO((         /
      DATA CONT/10H      CONT,10HINUE      /
      DATA GOTO1/10H     1) GO,10H TO       /
      DATA GOTO2/10H      GO T,10HO         /
      DATA CASE/10H$CASE     /
      DATA COMLPR/10H,(        /
      DATA RPRCOM/10H),        /
      DATA CEE/1LC/
      DATA ONE/1L1/
      DATA DOLLAR/1L$/
C
C
      DONE=.FALSE.
      SPTR=0
      NSUB=0
      SUBSH=.FALSE.
```

```
      LABEL=89999
      NBRNCH=0
      BPLACE=1
C     THIS IS THE TOP OF THE MAIN PROCESSING LOOP
      IPAGE=1
      LINECT=0
      WRITE(6,9877) IPAGE
      $REPEAT CONTINUE
        $CALL READ
      LINECT=LINECT+1
      IF(LINECT.GT.56)  GO TO 1183
1182  CONTINUE
        $IF(.NOT.DONE)
C     IF THIS IS NOT AN EXTENDED STATEMENT,WRITE IT OUT IMMEDIATELY
          $IF(.NOT.EQUAL(LINE,IP1,DOLLAR,1,1))
          $THEN WRITE(2,9000) (LINE(I),I=1,8)
          $ELSE CONTINUE
C     DETERMINE THE TYPE OF STATEMENT AND BRANCH TO THE APPROPRIATE BLOCK
          $CALL $TYPE
          $BRANCH15 ITYPE
C     CASE 1 - $IF
                $CASE1 IP2=IP1+2
                LABEL=LABEL+2
                SPTR=SPTR+1
                IF(SPTR.GT.MAXSTK) GO TO 9010
                STACK(SPTR)=LABEL
                STATE(SPTR)=-1
                CALL IFPRNT(LINE,LEN,LABEL,0,IP1,IP2)
                WRITE(2,9000)  LINE
C     CASE 2 - $THEN
                $CASE2 CALL CHMOVE(LINE,IP1,BLANKS,1,5)
                WRITE(2,9000)  LINE
C     CASE 3 - $ELSE
                $CASE3 CALL CHMOVE(LINE,IP1,BLANKS,1,5)
                NXLAB=STACK(SPTR) + 1
                KOUNT=12
                $CALL BLBUFF
                CALL INTOUT(NXLAB,GOTO2,KOUNT,5)
                CALL CHMOVE(BUFF,IP1,GOTO2,7,11)
                KK=IP1+10
                WRITE(2,9000) BUFF
                KOUNT=0
                CALL INTOUT(STACK(SPTR),LINE,KOUNT,5)
                STACK(SPTR)=NXLAB
                WRITE(2,9000)  LINE
C     CASE 4 - $ENDI
                $CASE4 CALL CHMOVE(LINE,IP1,BLANKS,1,5)
                WRITE(2,9000)  LINE
                $CALL LABWRT
C     CASE 5 - $WHILE
```



```
                $CASE5 IP2=IP1+5
                LABEL=LABEL+2
                SPTR=SPTR+1
                IF(SPTR.GT.MAXSTK) GO TO 9010
                STACK(SPTR)=LABEL
                STATE(SPTR)=0
                KOUNT=0
                CALL IFPRNT(LINE,LEN,LABEL+1,LABEL,IP1,IP2)
                WRITE(2,9000)  LINE
C     CASE 6 - $DO
                $CASE6 CALL CHMOVE(LINE,IP1,BLANKS,1,3)
                WRITE(2,9000)  LINE
C     CASE 7 - $ENDW
                $CASE7 CALL CHMOVE(LINE,IP1,BLANKS,1,5)
                WRITE(2,9000)  LINE
                KOUNT=12
                $CALL BLBUFF
                CALL INTOUT(STACK(SPTR),GOTO2,KOUNT,5)
                CALL CHMOVE(BUFF,IP1,GOTO2,7,11)
                KK=IP1+10
                WRITE(2,9000) BUFF
                STACK(SPTR)=STACK(SPTR) + 1
                $CALL LABWRT
C     CASE 8 - $REPEAT
                $CASE8 CALL CHMOVE(LINE,IP1,BLANKS,1,7)
                LABEL=LABEL+2
                SPTR=SPTR+1
                IF(SPTR.GT.MAXSTK) GO TO 9010
                STACK(SPTR)=LABEL
                STATE(SPTR)=1
                KOUNT=0
                CALL INTOUT(STACK(SPTR),LINE,KOUNT,5)
                WRITE(2,9000)  LINE
C     CASE 9 - $UNTIL
                $CASE9 IP2=IP1+5
                CALL IFPRNT(LINE,LEN,STACK(SPTR),0,IP1,IP2)
                SPTR=SPTR-1
                WRITE(2,9000)  LINE
C     CASE 10 - $BRANCH
                $CASE10 KF=IP1+7
                NCASES=0
C     DETERMINE THE NUMBER OF CASES
                $REPEAT DIGIT=.FALSE.
                I=0
                $REPEAT I=I+1
                $IF(EQUAL(LINE,KF,DIGITS,I,1))
                $THEN DIGIT=.TRUE.
                $ENDI NCASES=NCASES*10+I-1
                $UNTIL(DIGIT.OR.(I.EQ.10))
                KF=KF+1
```

```
                IF(KF.GT.LEN) GO TO 9020
                $UNTIL(.NOT.DIGIT)
                IF(NCASES.EQ.0) GO TO 9020
                CALL CHMOVE(LINE,IP1,BLANKS,1,KF-IP1)
                IF((NCASES.LT.2).OR.(NCASES.GT.99)) GO TO 9030
C     BUILD THE ASSIGNMENT STATEMENT FOR THE $BRANCH SWITCH
                LABEL=LABEL+NCASES+3
                SPTR=SPTR+1
                IF(SPTR.GT.MAXSTK) GO TO 9010
                STACK(SPTR)=LABEL-1
                NBRNCH=NBRNCH+1
                IF(NBRNCH.EQ.10) BPLACE=2
                STATE(SPTR)=NCASES+2
                KOUNT=4
                CALL INTOUT(NBRNCH,NBSW,KOUNT,BPLACE)
                CALL CHMOVE(LINE,IP1,NBSW,1,7)
                WRITE(2,9000)  LINE
                CALL CHMOVE(NIF,11,NBSW,1,6)
                CALL CHMOVE(NIF,28,NBSW,1,6)
                KOUNT=37
                CALL INTOUT(NCASES,NIF,KOUNT,2)
                KOUNT=46
                CALL INTOUT(STACK(SPTR),NIF,KOUNT,5)
      WRITE(2,9000) NIF
C     BUILD THE #COMPUTED GO TO #
                $CALL BLBUFF
                CALL CHMOVE(BUFF,1,CGOTO,1,12)
                KOUNT=12
                NXLAB=STACK(SPTR)-STATE(SPTR)+2
                CALL INTOUT(NXLAB,BUFF,KOUNT,5)
                I=1
                $REPEAT I=I+1
                CALL CHMOVE(BUFF,KOUNT+1,COMLPR,1,1)
                KOUNT=KOUNT+1
                NXLAB=NXLAB+1
                CALL INTOUT(NXLAB,BUFF,KOUNT,5)
                $IF(I.NE.NCASES)
                $IF(MOD(I,8).EQ.0)
                $THEN KOUNT=KOUNT+1
                CALL CHMOVE(BUFF,KOUNT,COMLPR,1,1)
                WRITE(2,9000) BUFF
                       $CALL BLBUFF
                       CALL CHMOVE(BUFF,6,ONE,1,1)
                       KOUNT=12
                       NXLAB=NXLAB+1
                       CALL INTOUT(NXLAB,BUFF,KOUNT,5)
                       $ENDI I=I+1
                       $ENDI CONTINUE
               $UNTIL(I.EQ.NCASES)
               CALL CHMOVE(BUFF,KOUNT+1,RPRCOM,1,2)
```



```
                CALL CHMOVE(BUFF,KOUNT+3,NBSW,1,6)
                KK=KOUNT+8
                WRITE(2,9000) BUFF
                CASE1=.TRUE.
                $CALL READ
                IF(.NOT.EQUAL(LINE,IP1,CASE,1,5)) GO TO 9040
C     CASE 11 - $CASE
                $CASE11 CONTINUE
11              IF(STATE(SPTR).EQ.2) GO TO 9050
                KF=IP1
C     GET THE $CASE IDENTIFIER (IF ANY)
                $WHILE(.NOT.EQUAL(LINE,KF,BLANK,1,1))
                $DO KF=KF+1
                $ENDW IF(KF.GT.LEN) GO TO 9070
                IF(KF-IP1.GT.7) GO TO 9060
                CALL CHMOVE(LINE,IP1,BLANKS,1,KF-IP1)
C     FOR SECOND AND SUCCEEDING $CASES, END THE PREVIOUS $CASE BLOCK
                NXLAB=STACK(SPTR)-STATE(SPTR)+2
                STATE(SPTR)=STATE(SPTR)-1
                $IF(.NOT.CASE1)
                $CALL BLBUFF
                KOUNT=12
                CALL INTOUT(STACK(SPTR),GOTO2,KOUNT,5)
                CALL CHMOVE(BUFF,IP1,GOTO2,7,11)
                KK=IP1+10
                $ENDI  WRITE(2,9000) (BUFF(I),I=1,8)
                CASE1=.FALSE.
                KOUNT=0
                CALL INTOUT(NXLAB,LINE,KOUNT,5)
                WRITE(2,9000)  LINE
C     CASE 12 - $ENDB
                $CASE12 IF(STATE(SPTR).NE.2) GO TO 9080
                CALL CHMOVE(LINE,IP1,BLANKS,1,5)
                WRITE(2,9000)  LINE
                $CALL LABWRT
C     CASE 13 - $SUB
                $CASE13 IF(SUBSW) GO TO 9090
                IP1=IP1+1
                SUBSW=.TRUE.
                CALL CHMOVE(LINE,1,CEE,1,1)
                WRITE(2,9000)  LINE
C     DETERMINE THE $SUB NAME, AND SEE IF IT HAS BEEN REFERENCED
                IP1=IP1+4
                CALL BLTRIM(LINE,IP1,LEN,IP2)
                IF(IP2.EQ.0) GO TO 9100
                NLEN=LEN-IP2+1
                IF(NLEN.GT.6) GO TO 9100
                NEWSUB=.TRUE.
                I=1
                $WHILE(I.LE.NSUB)
```

```
                $IF((NLEN.EQ.SUBTAB(1,I)).AND.EQUAL(LINE,IP2,
     1          SUBNAM(1,I),1,NLEN))
                $THEN KOUNT=3
                CALL INTOUT(99995+I,CONT,KOUNT,5)
                WRITE(2,9000)  CONT
                CURSUB=I
                SUBTAB(2,I)=1
                NEWSUB=.FALSE.
                $ENDI I=NSUB
                $ENDW I=I+1
                $IF(NEWSUB)
                $THEN IP2=IP2-1
                WRITE(6,9110)  LINE
                CURSUB=0
                $ENDI CONTINUE
C     CASE 14 - $ENDS
                $CASE14 IF(.NOT.SUBSW) GO TO 9120
                CALL CHMOVE(LINE,IP1,BLANKS,1,5)
C     IF IT HAS BEEN REFERENCED, BUILD THE #ASSIGNED GO TO#
C     TO HANDLE THE RETURN LINKAGE
                $IF(CURSUB.GT.0)
                $THEN  WRITE(2,9000) LINE
                SUBSW=.FALSE.
                NCALL=SUBTAB(2,CURSUB)
                $IF(NCALL.NE.0)
                $CALL BLBUFF
                CALL CHMOVE(BUFF,7,NSUBS,1,10)
                KOUNT=16
                SPLACE=1
                IF(CURSUB.GT.9) SPLACE=2
                CALL INTOUT(CURSUB,BUFF,KOUNT,SPLACE)
                CALL CHMOVE(BUFF,KOUNT+1,COMLPR,1,2)
                KOUNT=KOUNT+2
                J=0
                $REPEAT J=J+1
                CALL INTOUT(SUBTAB(3+J,CURSUB),BUFF,KOUNT,5)
                KOUNT=KOUNT+1
                        CALL CHMOVE(BUFF,KOUNT,COMLPR,1,1)
                        $IF((MOD(J,8).EQ.0).AND.(J.NE.NCALL))
                        $THEN WRITE(2,9000) BUFF
                        $CALL BLBUFF
                        CALL CHMOVE(BUFF,6,ONE,1,1)
                        $ENDI KOUNT=SPLACE+18
                        $UNTIL(J.EQ.NCALL)
                        CALL CHMOVE(BUFF,KOUNT,RPKCOM,1,1)
                        $ENDI WRITE(2,9000) BUFF
                $ENDI CONTINUE
C     CASE 15 - $CALL
                $CASE15 IP1=IP1+5
                CALL BLTRIM(LINE,IP1,LEN,IP2)
```



```
                IF(IP2.EQ.0) GO TO 9140
                NLEN=LEN-IP2+1
                IF(NLEN.GT.6) GO TO 9140
                NEWSUB=.TRUE.
                I=1
C     DETERMINE IF THIS $CALL REFERENCES A NEW $SUB, AND IF SO
C     MAKE AN ENTRY IN THE $SUB TABLE, OTHERWISE GET ITS INDEX
                $WHILE(I.LE.NSUB)
                $IF((NLEN.EQ.SUBTAB(1,I)).AND.EQUAL(LINE,IP2,
     1          SUBNAM(1,I),1,NLEN))
                $THEN IF(SUBTAB(2,I) .GE.MXCALL) GO TO 9130
                        CALSUB=I
                        NEWSUB=.FALSE.
                        $ENDI I=NSUB
                        $ENDW I=I+1
                        $IF(NEWSUB)
                        $THEN NSUB=NSUB+1
                        IF(NSUB.GT.MAXSUB) GO TO 9140
                        SUBTAB(1,NSUB)=NLEN
                         CALL CHMOVE(SUBNAM(NSUB),1,LINE,IP2,NLEN)
                $ENDI CALSUB=NSUB
                CALL CHMOVE(LINE,1,CEE,1,1)
                WRITE(2,9000)  LINE
C     BUILD THE #ASSIGN LABEL TO# AND SAVE THE RETURN LABEL
                LABEL=LABEL+2
                $CALL BLBUFF
                KOUNT=7
                CALL INTOUT(LABEL,ASSIG,KOUNT,5)
                CALL CHMOVE(BUFF,7,ASSIG,1,20)
                KOUNT=26
                SPLACE=1
                IF(CALSUB.GT.9) SPLACE=2
                CALL INTOUT(CALSUB,BUFF,KOUNT,SPLACE)
                WRITE(2,9000) BUFF
                KOUNT=12
                CALL INTOUT(99990+CALSUB,GOTO2,KOUNT,5)
                $CALL BLBUFF
                CALL CHMOVE(BUFF,7,GOTO2,7,11)
                WRITE(2,9000) BUFF
                KOUNT=0
                CALL INTOUT(LABEL,CONT,KOUNT,5)
                WRITE(2,9000) CONT
                NCALL=SUBTAB(2,CALSUB)+1
                SUBTAB(2,CALSUB)=NCALL
                SUBTAB(3+NCALL,CALSUB)=LABEL
                $ENDB CONTINUE
                $ENDI CONTINUE
                $ENDI CONTINUE
      $UNTIL(DONE)
C     PERFORM WRAP-UP
```

```
      $IF(SPTR.EQ.0)
      $IF(SUBSW)
C     A $SUB BLOCK WAS NOT PROPERLY TERMINATED
      $IF(CURSUB.GT.0)
                    $THEN  WRITE(6,9150)  SUBNAM(CURSUB)
                    $ELSE  WRITE(6,9153)
                    $ENDI CONTINUE
C     CHECK TO SEE WHETHER ALL REFERENCED $SUBS WERE DEFINED
      $ELSE WRITE(6,9160)
                    I=1
                    $WHILE(I.LE.NSUB)
                    $IF(SUBTAB(3,I).NE.1)
                    $THEN WRITE(6,9170)  SUBNAM(I)
                    $ENDI CONTINUE
                    $ENDW I=I+1
                    $ENDI CONTINUE
      $ELSE CONTINUE
C     ONE OR MORE BLOCKS WERE NOT PROPERLY TERMINATED
                    $REPEAT ITYP=STATE(SPTR)+2
                    IF(ITYP.GT.3) ITYP=4
                    WRITE(6,9180)  NAMERR(ITYP)
                    SPTR=SPTR-1
                    $UNTIL(SPTR.EQ.0)
                    $ENDI CONTINUE
      STOP
C     THIS $SUB READS INPUT LINES UNTIL A NON-COMMENT, NON-NULL
      $SUB READ
C     LINE IS FOUND
      $REPEAT READ(1,9000) LINE
                    IF(EOF(1))  20,654
654                 WRITE(6,9876)  (LINE(I),I=1,8)
9876                FORMAT(1H ,8A10)
                    NOMORE=.FALSE.
                    $IF(.NOT.EQUAL(LINE,1,CEE,1,1))
                    $THEN LEN=72
                    $WHILE(EQUAL(LINE,LEN,BLANK,1,1))
                    $DO LEN=LEN-1
                    $ENDW IF(LEN.EQ.1) LINE(1)=DOLLAR
                    CALL BLTRIM(LINE,7,LEN,IP1)
                    IF(IP1.GT.0) NOMORE=.TRUE.
                    $ELSE WRITE(2,9000) LINE
                    $ENDI CONTINUE
                    $UNTIL(NOMORE)
   15 CONTINUE
      $ENDS CONTINUE
20                  WRITE(6,9879)
9879                FORMAT(1H-,5X,*  END OF SOURCE PROGRAM LISTING  *,5X)
                    DONE=.TRUE.
      GO TO 15
C     THIS $SUB DETERMINES THE TYPE OF THE EXTENDED STATEMENT
```



```
C     THE TYPE INDEX CORRESPONDS TO THE VARIOUS $CASES
      $SUB $TYPE
                    ITYPE=-1
                    I=0
                    $REPEAT I=I+1
                    $IF(EQUAL(LINE,IP1+1,NAMES(I),1,NAMLEN(I)))
C     IF THE EXTENDED $-OPERATOR IS NOT A BLOCK HEADER, MAKE SURE
C     THAT IT MAY LEGALLY FOLLOW THE PRECEEDING $-STATEMENT
                    $IF((SPTR.EQ.0).AND.(NAMTYP(I).GT.0))
                    $THEN ITYP=NAMTYP(I)
      WRITE(6,9190) NAMERR(ITYP)
                    SPTR=0
                    DONE=.TRUE.
                    $ELSE JTYP=STATE(SPTR)+2
                    IF(JTYP.GT.3) JTYP=4
                    ITYP=NAMTYP(I)
                    $IF((ITYP.GT.0) .AND. (JTYP.NE.ITYP))
                    $THEN WRITE(6,9200) NAMERR(ITYP)
                    SPTR=0
                    DONE=.TRUE.
                    $ELSE ITYPE=I
                    $ENDI CONTINUE
                    $ENDI CONTINUE
                    I=15
                    $ENDI CONTINUE
                    $UNTIL(I.GE.15)
C     IF THE $-WORD IS UNRECOGNIZABLE, PRINT A DIAGNOSTIC
                    $IF(.NOT.DONE.AND.(ITYPE.EQ.-1))
                    $THEN WRITE(6,9210) LINE
                    $ENDI CONTINUE
      $ENDS CONTINUE
C     THIS $SUB PLACES A LABEL IN THE LINE BUFFER
C     BEFORE WRITING IT OUT
      $SUB LABWRT
                    KOUNT=0
                    $CALL BLBUFF
                    CALL INTOUT(STACK(SPTR),BUFF,KOUNT,5)
                    CALL CHMOVE(BUFF,IP1,CONT,7,8)
                    KK=IP1+7
                    WRITE(2,9000) BUFF
      $ENDS SPTR=SPTR-1
      $SUB BLBUFF
      IALPHA=BLANKS(1)
      DO  4400 K=1,8
4400  BUFF(K)=IALPHA
      $ENDS CONTINUE
C     DIAGNOSTIC STATEMENTS
9000  FORMAT(8A10)
 9010 WRITE(6,9015) MAXSTK
 9015 FORMAT(//# ERROR - MORE THAN #,I3,# NESTED STATEMENTS IS ILLEGAL#)
```

```
      STOP
 9020 WRITE(6,9025)
 9025 FORMAT(//# ERROR - BAD $BRANCH STATEMENT#)
      STOP
 9030 WRITE(6,9035)
 9035 FORMAT(//# ERROR - ILLEGAL NUMBER OF $CASES#)
      STOP
 9040 WRITE(6,9045)
 9045 FORMAT(//# ERROR - A $CASE STATEMENT MUST IMMEDIATELY FOLLOW #,
     1#THE $BRANCH STATEMENT#)
      STOP
 9050 WRITE(6,9055)
 9055 FORMAT(//# ERROR - MORE $CASES THAN WAS SPECIFIED#)
      STOP
 9060 WRITE(6,9065)
 9065 FORMAT(//# ERROR - BAD $CASE STATEMENT#)
      STOP
 9070 WRITE(6,9075)
 9075 FORMAT(//# ERROR - $CASE IDENTIFIER TOO LONG#)
      STOP
 9080 WRITE(6,9085)
 9085 FORMAT(//# ERROR - LESS $CASES THAN WAS SPECIFIED#)
      STOP
 9090 WRITE(6,9095)
 9095 FORMAT(//# ERROR - NESTED INTERNAL SUBROUTINE DEFINITION#)
      STOP
 9100 WRITE(6,9105)
 9105 FORMAT(//# ERROR - ILLEGAL INTERNAL SUBROUTINE NAME#)
      STOP
 9110 FORMAT(//#WARNING - INTERNAL SUBROUTINE #,6A1,# WAS NOT#,
     1,#REFERENCED IN THIS PROGRAM#/)
 9120 WRITE(6,9125)
 9125 FORMAT(//# ERROR - MISSING $SUB STATEMENT#)
      STOP
 9130 WRITE(6,9135) MXCALL,SUBNAM(CURSUB)
 9135 FORMAT(//#ERROR - MORE THAN #,I3,# $CALLS ON ROUTINE #,A10)
      STOP
 9140 WRITE(6,9145) MAXSUB
 9145 FORMAT(//# ERROR -#,I2,# IS THE MAX. NUMBER OF DIFFERENT $SUBS#)
      STOP
 9150 FORMAT(//# ERROR - MISSING $ENDS FOR SUBROUTINE #,6A6)
 9160 FORMAT(//# NORMAL END OF PREPROCESSING#)
 9170 FORMAT(//# ERROR - INTERNAL SUBROUTINE #,A10,# WAS NOT DEFINED#)
 9180 FORMAT(//# ERROR - MISSING $#,A10,# BLOCK TERMINATOR#)
 9190 FORMAT(//# ERROR - MISSING $#,A10,# STATEMENT#)
 9200 FORMAT(//# ERROR - MISSING $#,A10,# STATEMENT OR PREVIOUS BLOCK#,
     1# NOT PROPERLY TERMINATED#)
 9210 FORMAT(//# ERROR - UNKNOWN STATEMENT TYPE>#/1X,8A10)
 1183 LINECT=0
      IPAGE=IPAGE+1
```



```
      WRITE(6,9877)  IPAGE
 9877 FORMAT(1H1,5X,* SFOR VERSION 2  -  SOURCE PROGRAM LISTING*,10X,I5)
      GO TO 1182
      END
C
C
      SUBROUTINE BLTRIM(LINE,KF,KL,IP1)
      LOGICAL EQUAL
      INTEGER LINE(1),BLANK
      DATA BLANK/1L /
      IP1=KF
  777 IF(.NOT.(EQUAL(LINE,IP1,BLANK,1,1))) GO TO 888
      IP1=IP1+1
      IF(.NOT.(IP1.GT.KL)) GO TO 999
      IP1=0
      RETURN
  999 CONTINUE
      GO TO 777
  888 CONTINUE
      RETURN
      END
C
C
      SUBROUTINE INTOUT(N,LINE,KOUNT,NPLACE)
C
      INTEGER LINE(1),BLANK,DIGITS(1)
      DATA BLANK/1L /
      DATA DIGITS/10H0123456789/
      NUMBER=N
      IPLACE=NPLACE-1
  777 NXDIGT=NUMBER/10**IPLACE
      NUMBER=NUMBER-NXDIGT*10**IPLACE
      IPLACE=IPLACE-1
      KOUNT=KOUNT+1
      NXDIG=NXDIGT+1
      CALL CHMOVE(LINE,KOUNT,DIGITS,NXDIG,1)
      IF(.NOT.(IPLACE.EQ.-1))  GO TO 777
      RETURN
      END
C
C
      SUBROUTINE CHMOVE(BUFF1,IP1,BUFF2,IP2,NCHARS)
C
      INTEGER BUFF1(1),BUFF2(1)
      INTEGER WORD1,WORD2,CHAR1,CHAR2,XY
      DATA XY/2LXX/
C     DECODE THE POINTERS IP1 AND IP2 INTO WORD AND CHAR
      IF(IP1.GT.10)  GO TO 77
      WORD1=1
      CHAR1=IP1
```

```
      GO TO 88
77    WORD1=IP1/10+1
      CHAR1=IP1-(WORD1-1)*10
      IF(CHAR1.EQ.0) GO TO 111
      GO TO 88
111   CHAR1=10
      WORD1=WORD1-1
88    IF(IP2.GT.10)  GO TO 101
      WORD2=1
      CHAR2=IP2
      GO TO 102
101   WORD2=IP2/10+1
      CHAR2=IP2-(WORD2-1)*10
      IF(CHAR2.EQ.0) GO TO 112
      GO TO 102
112   CHAR2=10
      WORD2=WORD2-1
102   CALL XMOVE(XY,BUFF1(WORD1),BUFF2(WORD2),NCHARS,CHAR1,CHAR2)
      RETURN
      END
C
C
      LOGICAL FUNCTION EQUAL(S1,IP1,S2,IP2,LEN)
C
      INTEGER S1(1),S2(1)
      INTEGER WORD1,WORD2,CHAR1,CHAR2,XY
      DATA XY/2LXX/
C     DECODE THE POINTERS IP1 AND IP2 INTO WORD AND CHAR
      IF(IP1.GT.10)  GO TO 77
      WORD1=1
      CHAR1=IP1
      GO TO 88
77    WORD1=IP1/10+1
      CHAR1=IP1-(WORD1-1)*10
      IF(CHAR1.EQ.0) GO TO 111
      GO TO 88
111   CHAR1=10
      WORD1=WORD1-1
88    IF(IP2.GT.10)  GO TO 101
      WORD2=1
      CHAR2=IP2
      GO TO 102
101   WORD2=IP2/10+1
      CHAR2=IP2-(WORD2-1)*10
      IF(CHAR2.EQ.0) GO TO 112
      GO TO 102
112   CHAR2=10
      WORD2=WORD2-1
102   EQUAL=.TRUE.
      CALL XCOMP(XY,S1(WORD1),S2(WORD2),STATUS,LEN,CHAR1,CHAR2)
```



```
      IF(STATUS.NE.0.0)  EQUAL=.FALSE.
      RETURN
      END
```