TRANSLATION OF THE SPARKS PREPROCESSOR
FROM FORTRAN TO SPARKS.

by

RICHARD MANSCH STEDIEL

E.S., UNIVERSITY OF TEXAS, ARLINGTON, 1972

-----------------------------

A MASTER'S REPORT

submitted in partial fulfillment of the
requirements for the degree

MASTER OF SCIENCE

Department of Computer Science

KANSAS STATE UNIVERSITY

Manhattan, Kansas

1978

Approved by:

Myron A. Hallen

MAJOR PROFESSOR
# TABLE OF CONTENTS

## CHAPTER 1 INTRODUCTION
1.0 PURPOSE ................................. 1
1.1 EXPLANATION OF THE SPARKS PREPROCESSOR ................................. 1
1.2 HISTORY OF SPARKS ................................. 2
1.3 TRANSLATION OF THE PREPROCESSOR ................................. 2
1.4 CONCLUSION ................................. 3

## CHAPTER 2 TRANSLATION INTO SPARKS
2.0 INTRODUCTION ................................. 4
2.1 TRANSLATION OF FORTRAN STATEMENTS ................................. 4
2.2 IMPLEMENTATION ................................. 15
2.3 PROGRAM IMPROVEMENTS ................................. 16
2.4 CONCLUSION ................................. 20

## CHAPTER 3 PROGRAM DOCUMENTATION
3.0 INTRODUCTION ................................. 21
3.1 MAIN AND SUBROUTINE SPECIFICATIONS ................................. 24
3.2 PROGRAM VARIABLES ................................. 38
3.3 COMMON BLOCKS ................................. 41

## CHAPTER 4 SPARKS USER'S MANUAL
4.0 INTRODUCTION ................................. 43
4.1 SAMPLE PROGRAM ................................. 44
4.2 LANGUAGE DEFINITIONS ................................. 46
4.3 SPARKS IF STATEMENT ................................. 47
4.4 SPARKS CASE STATEMENT ................................. 49
4.5 SPARKS LGCP STATEMENTS ................................. 50
4.6 SPARKS EXIT AND CYCLE STATEMENTS ................................. 53
4.7 LABELLING IN SPARKS ................................. 56
4.8 SPARKS ECJ STATEMENT ................................. 56
4.9 SPARKS COMMENTS ................................. 56
4.10 GENERAL FEATURES ................................. 57
4.11 CONCLUSIONS ................................. 59

## APPENDIX A: SPARKS PREPROCESSOR IN SPARKS ................................. A1

## APPENDIX B: SPARKS PREPROCESSOR IN FORTRAN ................................. B1
ILLEGIBLE

THE FOLLOWING DOCUMENT (S) IS ILLEGIBLE DUE TO THE PRINTING ON THE ORIGINAL BEING CUT OFF

ILLEGIBLE
CHAPTER 1

INTRODUCTION

1.0 Purpose.

The purpose of this report is to describe the translation of the SPARKS preprocessor from FORTRAN to the SPARKS language. The report is organized such that the chapters dealing with the documentation of the preprocessor and the updated user's guide can be detached. The preprocessor written in SPARKS is included at appendix A. The FORTRAN translation of that code is at appendix B.

1.1 Explanation of SPARKS Preprocessor.

The sole purpose of the SPARKS preprocessor is to translate SPARKS statements into FORTRAN. Why SPARKS? And for that matter, why FORTRAN? Answering the second question first, FORTRAN has become established as the primary language for scientific and engineering computation. It also has the distinction of being the earliest 'high' level programming language, developed about 1957 at IBM. Because of its age, FORTRAN has many positive attributes such as:

a. It is almost always available and its compilers are often good.

b. There is a language standard which allows a degree of portability not obtainable with other languages.

c. There are extensive subroutine libraries available.

d. There is a large labor force already familiar with
CHAPTER 1

FORTRAN.

However, FORTRAN has certain negative aspects such as the fact that it was developed before the concepts of structured programming became established. It is essentially an unstructured and nonmodular language. Now, why SPARKS? The SPARKS language adds the improved syntactical constructs to FORTRAN which allow structured and modular code. Thus, the SPARKS preprocessor provides a simple means to obtain a nicely structured language while preserving the virtues of FORTRAN.

1.2 History of SPARKS.

SPARKS originated as a language for describing algorithms in the textbook Fundamentals of Data Structures by Ellis Horowitz and Sartaj Sahni, published by Computer Science Press, Woodland Hills, California. The SPARKS preprocessor which grew out of this prototype language was originally written in SPARKS. The program was then hand translated into FORTRAN to produce the first working model. During the hand translation and implementation, the language was revised and no longer exactly matches the original textbook version.

1.3 Translation of the Preprocessor.

Rewriting the SPARKS preprocessor in SPARKS brings to the translation program itself the structure for easy
CHAPTER 1

readability and understanding and the necessary modularity
to make modifying the program easier. The modularity of
SPARKS can be illustrated by the ease in which the 20
separate cases of the main program case statement can be
rearranged. The cases can be moved about within the case
statement by simply taking the cards belonging to that case
and replacing them where desired. This same action in the
FORTRAN copy would be a major undertaking as many labelled
statements and GO TO's would have to be changed.

1.4 Conclusion.

The measure of a computer language, good or bad, is the
ease with which it allows a programmer to describe a real
world problem in a natural way. The SPARKS language
improves FORTRAN toward this end. The preprocessor itself,
written in SPARKS, is a program much easier to read,
understand, and modify.
CHAPTER 2

TRANSLATION INTO SPARKS

2.0 Introduction.

The writing of the translator in SPARKS involved two distinct steps. The overall structure of a block of the FORTRAN was recognized and then its sub-structures found. The preprocessor was hand translated to SPARKS using this top-down approach.

2.1 Translation of FORTRAN Statements.

The first step in the translation process was to recognize the overall structure of each routine. For example, the large main routine consists of a single loop forever with a case statement within the loop. The loop covers 12 pages of code and the case statement has 20 separate cases over those 12 pages of code. Once the overall structure of the routine was discovered, the individual constructions within the routine were recognized and converted to equivalent SPARKS code. This method of finding the structures (large to small) was followed to produce the entire translator in handwritten SPARKS code. Shown in the following paragraphs is how the actual FORTRAN code was recognized as convertible to a particular SPARKS construction. Also discussed, are those instances where the machine translation of the SPARKS code will not reproduce the exact FORTRAN of the original. The FORTRAN IF statement
was generally easy to convert to SPARKS as shown with the nested IF statement example.

**ORIGINAL FORTRAN**

IF (INDFLG .NE. BKWED) GC TO 12000
SSTRT=SSTRT-INCR
IF (CURFLG .EQ. NEW) CALL SFRNT
NESTNC=NESTNC-1
BLKNO=BLKNC+1

1200C CONTINUE

**TRANSLATION TO SPARKS**

IF INDFLG .EQ. BKWED THEN
SSTRT=SSTRT-INCR
IF CURFLG .EQ. NEW THEN
CALL SFRNT
ENDIF
NESTNC=NESTNC-1
BLKNO=BLKNC+1
ENDIF

The inner IF statement is an example where the machine translation will not reproduce the original FORTRAN code. SPARKS has no translation to the inner IF construction. The
following illustrates the preprocessor conversion.

SPARKS
IF CURFLG.EQ.NEW THEN
CALL SPRNT
ENDIF

MACHINE TRANSLATION TO FORTRAN
IF(.NOT.(CURFLG.EQ.NEW)) GO TO 11000
CALL SPENT
11000 CONTINUE

Although the meaning of the original FORTRAN and the SPARKS preprocessor generated FOMTRAN is the same, it is apparent that care must be taken when hand translating to SPARKS. The relational operator may or may not require negation depending on the type of FORTRAN IF to be translated. A particular sequence of code with the FORTRAN IF can signal the SPARKS IF/THEN/ELSE construction. The following FORTRAN sequence clearly indicates the IF/THEN/ELSE construct. ECOND is a logical.

FORTRAN
IF(.NOT.ECOND) GC TC 1920
CALL ERBCS(2,ECO,2,1,0,0)
TOEND=TCEND+1
GO TO 1910

-6-
CHAPTER 2

1920 TCEND=TCEND+2
1910 CONTINUE

TRANSLATION TO SPARKS

IF ECOND THEN
    CALL ERRCR (2,DOO,2,1,0,0)
    TCEND=TCEND+1
ELSE
    TCEND=TCEND+2
ENDIF

Again, the relational operator can be changed or negated to reverse the THEN and ELSE cases but care must be used so as not to change the meaning of the original code.

Many nested IF statements or IF statements which branch to another IF on the true branch are convertible to the SPARKS case statement. The improved readability and modularity of SPARKS is nicely shown by the CASE example.

FORTRAN

IF (STTOP.NE.FFOR).GO TO 2090
LAB1=STACK (TCP-5)
LAB2=STACK (TCP-2)
LAB3=STACK (TCP-1)
TCP=TOP-6

-7-
CHAPTER 2

LBPTR=LELF-20
GO TO 2000

2090 IF (STTOP.NE.LLOCP) GO TO 2080
LAB1=STACK (TCE-3)
LAB2=STACK (TCP-2)
LAB3=STACK (TCE-1)
TOP=TOP-5
LBPTR=LEFTR-15
GO TO 2000

2080 IF (STTOP.NE.WHILE) GC TO 2070
LAB1=STACK (TCP-3)
LAB2=STACK (TCE-2)
LAB3=STACK (TCP-1)
TOP=TCP-15
LBPTR=LEFTR-15
GO TO 2000

2070 CONTINUE

CALL ERECR (3,REPT,6,4,REFT,6)

2000 CONTINUE

TRANSLATION TC SFARKS

CASE

:STTOP.EG.FPCB:
LAB1=STACK (TCE-5)
LAB2=STACK (TOP-2)
LAB3=STACK (TCE-1)

-8-
CHAPTER 2

TOP=TCF-6
LBPTR=LEPTR-20

:STTOP.EQ.LLCOP:
LAB1=STACK(TCF-3)
LAB2=STACK(TCF-2)
LAB3=STACK(TCF-1)
TOP=TCF-5
LBPTR=LEPTR-15

:STTOP.EQ.WHILE:
LAB1=STACK(TCF-3)
LAB2=STACK(TCF-2)
LAB3=STACK(TCF-1)
TOP=TCF-5
LBPTR=LEPTR-15

:ELSE:
CALL ERROR(3,FEET,6,4,REPT,6)

ENDCASE

SPARKS allows four loop constructions: The FOR loop, the ICOP/REPEAT, the WHILE loop, and the LOOP/UNTIL. The task of recognizing a loop construction in FCRTRAN, other than the obvious DO loop, involves finding a test condition and a backward GC TC. The following examples show how the SPARKS loop constructions were used to convert the FCRTRAN to SPARKS. Treating the easier DO loop first, it is replaced by the SPARKS FOR loop.
CHAPTER 2

FORTRAN

DC 1980 I=1,5
L=LAB3+I-1
1980 FBUF (I) = LABEL (L)

TRANSLATION TO SPARKS

FOR I=1 TO 5 DC
L=LAB3+I-1
FBUF(I) = LABEL(L)
REPEAT

This is another case where the machine translation of the SPARKS will not reproduce the original FORTRAN. There is no SPARKS construction that converts to a FORTRAN DO loop. The SPARKS FOR loop is more powerful than shown above in that it will allow incrementation by any integer, either positive or negative. The above illustration shows the default incrementation of +1. The SPARKS preprocessor handles the translation of the FOR loop in an interesting way. The example below is of a negative incrementing loop and its FORTRAN translation.

SPARKS

FOR I=10 TO 2 BY -2 DO
J=LAB3+I-1
REPEAT

-10-
CHAPTER 2

MACHINE TRANSLATION TO FORTRAN

I=10
GO TO 100
102 I=I+(-2)
100 IF(.NOT.(I-(2))*(-2).LE.0) GO TO 101
   J=LAB3+I-1
   GC TO 102
101 CONTINUE

The example illustrates that although the FORTRAN DO loop can be easily replaced by a SPARKS FOR loop, the machine translation of the FOR loop into FORTRAN in no way resembles the original code. The other loop structures were also used in the translation to replace the FORTRAN conditional tests and backward GC TC's. A WHILE loop conveniently replaces the FORTRAN code shown below.

FORTRAN

3401 IF(SPIN(TOBEG).NE.BLANK) GO TO 3402
   TOBEG=TCBEG+1
   IF(TOBEG.GT.IMAX) GO TO 3403
   GO TO 3401
3403 TCEND=IMAX+1
3402 CONTINUE

TRANSLATION TO SPARKS

-11-
CHAPTER 2

WHILE SPIN(ICEEG).EQ.BLANK DO
  TOBEG=TCBEG+1
  IF TOBEG.GT.IMAX THEN
    TOEND=IMAX+1
    EXIT
  ENDDIF
REPEAT

The SPARKS construction EXIT, used above, allows premature leaving of a loop structure. Similarly, the SPARKS CYCLE statement is used to transfer to the bottom of a loop from anywhere within the loop structure. The SPARKS ICCE/REPEAT and the EXIT are used to translate the following FORTRAN code.

FORTRAN
1090 IF(K.GT.IMAX) GO TO 2140
  IF(SPIN(K).NE.BLANK) GO TO 1091
  K=K+1
  GO TO 1090
1091   I=ISNUM(SPIN(K))
  GO TO 2150
2140   I=0
2150   CONTINUE

TRANSLATION TO SPARKS
CHAPTER 2

LCOP

IF K.GT.IMAX THEN

I=0
EXIT
ENDIF

IF SPIN(K).NE.BLANK THEN

I=ISNUM(SEIN(K))
EXIT
ENDIF

K=K+1
REPEAT

The last SPARKS loop structure, the LOCF/UNTIL is used to convert the FORTRAN below.

FORTRAN

10651 TTOP=TTOP-STFRM(TTOF)

IF(TTOP.GT.0) GO TO 10652

CALL ERROR(3,EEEND,3,4,EEEND,3)
ECOND=.TRUE.
RETURN

10652 CCNTINUE

STTOP=STACK(TTOP)

IF(STTOP.NE.SUBFN) GC TO 10651

TRANSLATION TO SPARKS

-13-
CHAPTER 2

LCOP

IF TTOP.LE.0 THEN
    CALL ERROR (3, EEEND, 3, 4, EEEND, 3)
    ECOND=.TRUE.
ENDIF

STTOP=STACK(TTOP)
UNTIL STTOP.EQ.SUBFN REPEAT

Some of the FORTRAN code was modified during the original hand translation from SPARKS to FORTRAN to the extent that it did not match any SPARKS constructions. The subroutine DETNX is cited as an example. This subroutine's overall structure was not convertible to any single SPARKS construction. Essentially, the intent of the FORTRAN code had to be determined exactly and SPARKS code generated as required. The overall construction of the subroutine in SPARKS consists of two nested LCOP-REPEAT constructions which bear little resemblance to the original FORTRAN code. There are other instances where it was necessary to rearrange the SPARKS code completely in order to eliminate retaining the FORTRAN GO TO. The completed translation to SPARKS does not, in any case, retain the FORTRAN IF statement, the GC TC in any form, or the DO loop in the program. Of course, the statements common to both FORTRAN and SPARKS were retained to avoid punching duplicate cards. The first SPARKS deck varied less than four per cent in size.
CHAPTER 2

from the original FORTRAN deck.

2.2 Implementation.

Loading and debugging of the SPARKS language preprocessor was the next step in the translation process. It was decided that the SPARKS language preprocessor should be loaded and debugged one subroutine at a time. This procedure would side-step the problem of trying to find errors in the 1845 card SPARKS deck. The original FORTRAN language preprocessor was copied onto a temporary disk file. The SPARKS preprocessor originally brought to Kansas State University was not disturbed and provided the means to translate the new SPARKS language version to FORTRAN. The new subroutines were processed through and linked to the FORTRAN copy on the temporary disk. After each subroutine was converted, a test deck was run. Each subroutine was completely debugged before another was added. When all subroutines had been replaced, the main routine was replaced and the temporary disk was complete with the machine translation of the new SPARKS language preprocessor. At this point, two copies of the SPARKS deck were made. One copy was retained as a record copy and one copy was provided to another Graduate Student, John J. Martin, for use in his Master's project.

During the initial loading, two errors were discovered
in the original FORTRAN version of SPARKS: Large programs could not be loaded as there was an error in the method of reclaiming the label pointer positioning. Consequently, large programs would over-drive the label table and the program would terminate unsuccessfully. Also, the SCAN1 subroutine which normally stops on a semicolon incorrectly translated format and data statements containing semicolons and produced fatal FORTRAN errors. These errors made it impossible to load the entire preprocessor as a single program, and FORTRAN flags (F in column 1) had to be placed around all format and data statements. It was found that John Martin had already developed corrections for these two errors as a part of his Master's project and those corrections were incorporated in the working deck.

2.3 Program Improvements.

Once a copy of the preprocessor written in SPARKS was obtained, the code was examined in detail. Many occurrences of redundant code were found as well as several unused variables. The following variables were eliminated from the program:

a. HECH—was not used.
b. COOLN—duplicate variable name, not needed
d. NINE—retained as a subroutine local variable, only.
e. CODE0—was not used.
CHAPTER 2

f. CODE9 was not used.
g. IBLD-eliminated as unnecessary.
h. ITIN, ITOT, ITRD, INRD, ITERS, ITPRF, INFRS, INPFF, INFPR, INSPR, INSN1-statistics gathering variables, all eliminated.

The variable STBEG, which marks the beginning of all input statements, was set once in each separate case of the Main Program. It was noted that the beginning of the line is determined prior to entering the Main Routine CASE statement. Therefore, STBEG could be set prior to entry into the CASE statement and 19 redundant cards removed from the Main Routine. The following block of code was found to appear frequently in the Main Routine:

\[
\text{FSTAT} = \text{FSTAT} + \text{INCR} \\
\text{IF FSTAT} \gt \text{IMAX THEN} \\
\text{FSTAT} = \text{FSTAT} - \text{INCR} \\
\text{ENDIF}
\]

The block was rewritten as follows saving two unnecessary statements.

\[
\text{IF FSTAT} + \text{INCR} \lt \text{IMAX THEN FSTAT} = \text{FSTAT} + \text{INCR} ; \text{ENDIF}
\]

It was found that blanks were placed in the first six columns of the FORTRAN output buffer (FEUF) by almost every separate case of the Main Routine by repetition of the following code:

\[
\text{FOR I=1 TO 6 DO}
\]

-17-
CHAPTER 2

FBUF(I) = BLANK

REPEAT

A location in the first subroutine, DETAX, was located where the buffer could be filled for each output line. The above code was then eliminated from the Main Routine. Instances were also found where identical flags were set either in both the true and false branches of an IF statement or repeated in consecutive IF statements. In all of these instances, the flags to be set were moved so as to eliminate repetitious code.

Examination of the CYCLE and EXIT statement-handling routines in the Main Routine CASE statement revealed that the code was practically identical for both statements. A subroutine was created to process both the CYCLE and EXIT statements. The only code remaining in the Main Routine for the EXIT and CYCLE statements is a call to the CHECK Subroutine and a call to the new subroutine, TYHAND.

Several occurrences were found where the SPARKS code could be improved by rearranging and changing the construction used. Reference is made to the CASE conversion example of section 2.1, where it can be seen that the code for the LCCF case and the WHILE case is identical. Also, the ELSE case was redundant as the CHECK Subroutine had been called prior to entry into the case. With these
observations, the CASE can be replaced by the IF-THEN-ELSE construction using much less code.

```
IF STTOP.EQ.FFGR THEN
    LAB1=STACK(TOE-5)
    LAB2=STACK(TOF-2)
    LAB3=STACK(TOF-1)
    TOP=TOP-6
    LEPTR=LBPTR-20
ELSE
    LAB1=STACK(TOF-3)
    LAB2=STACK(TOE-2)
    LAB3=STACK(TOF-1)
    TOP=TOP-5
    LBPTR=LBPTR-15
ENDIF
```

The last items to be investigated were the incorrect error messages, the method of declaring integers and arrays, and the alignment of COMMON blocks. The error messages output by the preprocessor were either incorrect or had missing words or letters. The problem found was that the error variables had been equivalenced to positions in the program's reserved word table, but apparently the ordering within the table had been changed and subsequently the equivalences were now to the wrong positions. The problem
was solved by eliminating the equivalence statements and assigning values to the ERROR subroutine's variables by DATA statements in the BLOCK DATA routine. The integer declarations were made originally by long lists of integer statements listing every variable and array. Also, the arrays were dimensioned in the integer statements. Each variable and array was again listed in the COMMON blocks. This double listing of the variables in each routine could be eliminated by the use of the implicit integer feature. It should be noted that the program is no longer ANSI Standard FORTRAN with the inclusion of the implicit integer feature. The dimensioning of the arrays was included in the COMMON blocks. A closer look at the named COMMON blocks indicated that they were not optimized for the routines that used them; there were many occurrences where a named common of up to 21 variables was included in a routine which used only one of the variables. Thus, in addition to adding the dimensioning to the common blocks, they were re-aligned and/or split so that each routine included fewer common statements.

2.4 Conclusion.

The resulting SPARKS language preprocessor is a 1470 line program versus the 1875 lines of the original FORTRAN version. It has one less case in the main routine and one additional subroutine.
CHAPTER 3

PROGRAM DOCUMENTATION

3.0 Introduction.

The SPARKS preprocessor is a 1470 line program consisting of a Main Routine, a Block Data, 11 Subroutines, and three Integer Functions. The preprocessor written in the SPARKS language is at APPENDIX A. The machine translation of this program into FORTRAN is at APPENDIX B. The module access diagram for the program is shown in Figure 1. The data flow of the preprocessor is in Figure 2. The global variables used by the program and the routines in which they are used are included in Paragraph 3.2 of this chapter. The common blocks and the variables they include are shown in Paragraph 3.3. Figures 1 and 2 are given on the next two pages.
CHAPTER 3

FIGURE 1
Module Access Diagram for the SPARKS Preprocessor.
CHAPTER 3

FIGURE 2

Data Flow Diagram for the SPARKS Preprocessor.
3.1 Main Routine and Subroutine Specifications.

In the paragraphs to follow, each routine of the program is outlined as to its function, the common blocks it includes and pseudo code showing what the routine does.

**MAIN ROUTINE**

*FUNCTION:* The main routine causes a card to be read (subroutine DETNX reads the card) and the type of statement to be determined (again by subroutine DETNX which calls STYP). The main then processes the statements by type. The FORTRAN output is generated within the main routine CASE statement which is organized by statement type. The SPARKS output is accomplished at the end of the main for each input line after the FORTRAN for that line has been generated. The main routine includes the following common blocks: INPUT, SPRINF, FRTRNF, CENTRL, CRTMAT, INFRAT, STYF1, STYF2, BMNLC, KEYWD, TABLES, and TYP.

**ABSTRACT CODE**

BEGIN

Initialize program options

LCOF

Cause a SPARKS card to be read (Call to DETNX).

Determine statement type (Call to DETNX, STYP).

If a real, integer or logical, check for subroutine or function. If found, set type to

SUBFN.

CASE
: type is FORTRAN :
   Pass the FORTRAN to the FCRTRAN file.

: type is double slash :
   Process SPARKS comment.
   Translate comment to FORTRAN, pass to FORTRAN file.

: type is an IF :
   Place IF on stack.
   Translate to a FORTRAN IF and output to FORTRAN file.

: type is an ENDIF :
   Check stack top for an IF statement, if none found then error, CYCLE.
   Generate necessary labels and CONTINUES and output to the FORTRAN file.

: type is an ELSE :
   Check stack top for CASE or IF statement, if none found then error, CYCLE.
   Convert to FORTRAN and output to FORTRAN file.

: type is a FOR :
   Place FOR on stack.
   Search for key words TO, BY, and DO, if missing then error, CYCLE. Convert to FORTRAN and output to FORTRAN file.

: type is a REPEAT :
CHAPTER 3

Check stack top for an iterative statement;
   if none found then error, CYCLE.
Convert to FORTRAN, output to FORTRAN file.

: type is a CASE :
   Put CASE on stack.
Convert to FORTRAN, output to FORTRAN file.

: type is a COLON :
   Check stack for a CASE statement, if none
      found then error, CYCLE.
Convert to FORTRAN, output to FORTRAN file.

: type is an ENDCASE :
   Check stack top for a CASE statement, if
      none found then error, CYCLE.
Convert to FORTRAN, output to FORTRAN file.

: type is a LOOP :
   Put LOOP on stack.
Convert to FORTRAN, output to FORTRAN file.

: type is a WHILE :
   Put WHILE on stack.
Convert to FORTRAN, output to FORTRAN file.

: type is a CONTINUE :
   Convert to FORTRAN, output to FORTRAN file.

: type is an EXIT or CYCLE :
   Check stack top for iterative statement, if
      none found then error, CYCLE.
Convert to FORTRAN, output to FORTRAN file.
: type is an UNTIL :
   Check stack top for ICCP statement,
   if none found then error, CYCLE.
Convert to FORTRAN, output to FORTRAN file.
: type is a SUBROUTINE or FUNCTION :
Stack the SUBFN.
Re-initialize label numbers, skip to new
page, output to FORTRAN file.
: type is an END :
Check stack top for PROGRAM, SUBROUTINE, or
   FUNCTION; if not found then error, CYCLE.
Output to FORTRAN file.
: type is a FORTRAN comment :
Check for C in column 1 and output to FORTRAN
   file.
: type is an EOF :
Close the files.
STOP.
CHAPTER 3

the main with the statement type. The FORTRAN flag (F in column 1) is also handled by this routine. Code between the flags is passed directly to the FORTRAN file without processing. The subroutine includes the following common blocks: INPUT, PRINF, STYF2 and NMNIC.

ABSTRACT CODE

BEGIN

LCOF

LOOP

If the end of line pointer exceeds 72, EXIT.

If a semicolon is found then CYCLE.

If a C is found in column 1 then type is a COMMENT, RETURN.

Look for a blank or end condition (by a call to SCAN1).

CASE

: No blanks prior to end condition:
  Type is FORTRAN, RETURN.

: Column six is not blank:
  Type is a continuation card, RETURN.

: A blank is found prior to END condition:
  Call to Subroutine STYF to determine type.
  RETURN

: A label is found:

-28-
CHAPTER 3

Pass the label to the FORTRAN file.

ENDCASE

REPEAT

Read a card.

Output FORTRAN if FORTRAN flags are found.

REPEAT

END

SUBROUTINE STTYP

FUNCTION: The subroutine searches the SPARKS keyword table to find the type of statement. The subroutine is called only when a word is found surrounded by blanks which indicates a possible SPARKS statement. If the keyword is not found in the table, the type is assured to be FORTRAN. The keyword table is initialized in the BLOCK DATA. STTYP includes the following common blocks: INPUT, STTYP2, AND TABLES.

ABSTRACT CODE

BEGIN

Initialize local variables.

Use binary search of keyword table. If first letter match is found, check length character by character. If length matches then assign a value to TYPE and RETURN.

If a match is not found then TYPE is assigned as FORTRAN.
CHAPTER 3

RETURN

END

SUBROUTINE SCAN1

FUNCTION: The subroutine scans the input card for a requested character that is passed in. It either returns with the character found or with an END condition. SCAN1 includes the following common blocks: INFUT and STYP2.

ABSTRACT CODE

BEGIN

WHILE the line pointer is less than 72 DO

Search for the input character.

If an end condition is found (or a semicolon),
RETURN with a TRUE end condition.

If the input character is found, RETURN.

REPEAT

RETURN with a true end condition.

END

SUBROUTINE SCAN

FUNCTION: The subroutine searches the input string for a pattern that is passed to it. If the pattern is found, the subroutine returns with a false end condition. If the pattern is not found the end condition is returned true. SCAN includes the following common blocks: INFUT and STYP2.

ABSTRACT CODE
CHAPTER 3

BEGIN

LCOE

Match the input pattern, a character at a time.
If the pattern is found then end condition
is returned false.
Else the end condition is returned true.

REPEAT

END

SUBROUTINE SPRT

FUNCTION: The subroutine prints the processed
statements as SPARKS output. A 131 character line is used
to output the SPARKS code. SPRT includes the following
common blocks: INPUT, STTYP2, SPTINF, FRTINF, NMNJC, and
TYF.

ABSTRACT CODE

BEGIN

Fill the first six columns of output buffer.
Set beginning of line to be output.
Set end of line to be output.
Write line to printer.
Reset first six columns of output buffer
to blanks.

END

SUBROUTINE PHAND
CHAPTER 3

FUNCTION: The subroutine handles the FORTRAN statements. FHAND insures that column six is proper for the statement being processed, causes any incomplete FORTRAN statements to be completed, and causes the FORTRAN to be output. FHAND includes the following common blocks: INPUT, STYP2, FRTINF, and TYP.

ABSTRACT CODE

BEGIN

LOOP

Check for a semicolon or end condition.

If type is a continuation, copy the first six columns from input.

Complete and statements by calling FCUT.

RETURN when an end condition is found.

REPEAT

END

SUBROUTINE FCUT

FUNCTION: The subroutine completes FORTRAN statements by putting the label in columns one through five and introducing a C in column one if the line is a comment. If a blank line is found, the routine returns with blank line true. The routine causes FORTRAN to be output by FPRINT. FCUT includes the following common blocks: CENTR, FRTINF, STYP2, NMNLC, TYP, and FORPRT.

ABSTRACT CODE

-32-
CHAPTER 3

BEGIN

LCOF UNTIL the line has been completely scanned.
If the line pointer is at the end then
cause the line to be printed (call FPRNT).
Fill the output buffer with the label, if any.
If a continuation then fill column six.
If a comment put a C in column one.
Check for a blank line, if found then
RETURN blank line true.

REPEAT

END

SUBROUTINE FPRNT

FUNCTION: The subroutine writes the FORTRAN to the file that will be passed to the FCBCTRAN compiler for execution. FPRNT includes the following common blocks: INPUT, NMNIC, SPRINF, FRTINF, and FORERT.

ABSTRACT CODE

BEGIN

Check line pointers, if null output, RETURN.
Complete filling the output buffer, columns 1 to 80.
Check for blank line, if found RETURN.
Write line to FORTRAN output file.
Reset line pointers.
RETURN
CHAPTER 3

END

INTEGER FUNCTION ISNUM

FUNCTION: The function compares the input argument to the digits 0, 1, 2, ..., 9. If the input parameter is the alphanumerical representation of a number, the function returns a one. If the comparison fails, a zero is returned. ISNUM has no common blocks.

ABSTRACT CCDF

BEGIN

Set up comparison digets 0, 1, 2, ..., 9.

LCOF

Compare input parameter to each digit in turn.

If a match is found, then set function to one

and RETURN.

REPEAT

Set function to zero.

RETURN

END

INTEGER FUNCTION CCNV

FUNCTION: The Integer Function CCNV converts a number into its character representation so that it can be printed. CCNV has no common blocks.

ABSTRACT CCDF

BEGIN
CHAPTER 3

Set digits zero to nine in Holerith Fields.
Assign the function the digit in printable form.
RETURN
END

INTEGER FUNCTION GENLAE

FUNCTION: The function generates a label and stores that label in the label array. If there is a user defined label, that label is also stored in the label array. The function returns with a pointer to the label generated. GENLAE includes the following common blocks: INPUT and CENTRL.

ABSTRACT CODE

BEGIN
If there is not a user defined label then generate a label.
Put the label in the label array.
RETURN with the pointer to the label.
If the number of labels over-drives the label pointer then error.
If there is a user defined label then put that label in the label array.
RETURN with a pointer to the label.
END

SUBROUTINE CHECK
FUNCTION: The subroutine does the necessary checking of SPARKS statements. SPARKS keywords which begin SPARKS constructions are stacked for checking later when the end condition for that construction is encountered. The subroutine checks the stack top to insure the statement being checked is used in the correct context. CHECK includes the following common blocks: INPUT, PRFINP, SERINF, OOTPAT, STYFP1, CENTR, INFAT, TABLES, and KEYWD.

ABSTRACT CODE

BEGIN

Check stack, if empty then error.

CASE

: SPARKS type is a CCICN or ENDCASE :

If the stack top is a CASE then RETURN else error.

: SPARKS type is an ELSE or ENDIR :

If the stack top is an IF then RETURN else error.

: SPARKS type is an UNTIL :

If the stack top is a LOOP then RETURN, else error.

: SPARKS type is an END :

If the stack top is a program, subroutine, or function then RETURN, else error.

Unstack until a PROGRAM, SUBROUTINE, or FUNCTION is found, close the block.
CHAPTER 3

If the stack goes empty while unstacking and statement not found, then error. 
: SPARKS type is a REPEAT, EXIT, or CYCLE : 
If the stack top is an iterative statement then RETURN, else error. 

ENDCASE 
RETURN 
END

SUBROUTINE ERROR

FUNCTION: The Subroutine outputs SPARKS error messages, based on input arguments. Error contains the data statements necessary to build the appropriate error messages. ERROR includes the following common blocks: INPUT, STYF2, and NMNIC.

ABSTRACT CODE 

BEGIN 
Write the beginning of the error message. 
Match the input arguments to the error message data table. 
Write the remaining part of the error message. 
END

SUBROUTINE TYHANG

FUNCTION: The subroutine handles the processing of the EXIT and CYCLE statements. It causes the appropriate
CHAPTER 3

FORTRAN code to be output to implement the intent of the EXIT and CYCLE. TYHAND includes the following common blocks: INOUT, SFRINF, FRTINF, CTFAT, CENTR, and STYP2.

ABSTRACT CODE

BEGIN

Check for a label, if present then
   error, delete.

Convert the EXIT or CYCLE to FORTRAN by
generating a GO TO the appropriate
   LOOP label.

RETURN

END

3.2 Program Variables.

The global variables of the program are listed below along with the routines in which they appear. The arrays are listed after the variables. Local loop counters and temporary 'hold value' variables are not included.

Global Variables

<table>
<thead>
<tr>
<th>Variable</th>
<th>Using Routines</th>
</tr>
</thead>
<tbody>
<tr>
<td>BLANK</td>
<td>MAIN, BLCK DATA, DETNX, SCAN, SPRNT, FHND, FOUT.</td>
</tr>
<tr>
<td>BLNKLN</td>
<td>PRNT, FCUT.</td>
</tr>
<tr>
<td>BKWRD</td>
<td>MAIN, BLOCK DATA.</td>
</tr>
<tr>
<td>BLKNO</td>
<td>MAIN, SPRNT.</td>
</tr>
<tr>
<td>CCASE</td>
<td>MAIN, BLOCK DATA.</td>
</tr>
<tr>
<td>CEE</td>
<td>MAIN, BLCK DATA, SPRNT, FCUT.</td>
</tr>
<tr>
<td>CCLN</td>
<td>MAIN, BLOCK DATA.</td>
</tr>
<tr>
<td>COLON</td>
<td>MAIN, BLCK DATA, CHECK.</td>
</tr>
<tr>
<td>CCNT</td>
<td>MAIN, BLOCK DATA, DETNX, SPRNT, FOUT.</td>
</tr>
<tr>
<td>CCNT</td>
<td>MAIN, BLOCK DATA, DETNX, SPRNT,</td>
</tr>
<tr>
<td>Variable</td>
<td>Arguments</td>
</tr>
<tr>
<td>-----------------</td>
<td>------------------------------------------</td>
</tr>
<tr>
<td>CRDR</td>
<td>FHANE, BLOCK DATA, DETNX.</td>
</tr>
<tr>
<td>CURFLG</td>
<td>MAIN, BLOCK DATA, TYHAND.</td>
</tr>
<tr>
<td>DDBLSL</td>
<td>MAIN, BLOCK DATA, STTYP.</td>
</tr>
<tr>
<td>ECOND (LOGICAL)</td>
<td>MAIN, DETNX, SCAN1, SCAN, FHAND, CHECK.</td>
</tr>
<tr>
<td>ELSE</td>
<td>MAIN, BLOCK DATA, CHECK.</td>
</tr>
<tr>
<td>END</td>
<td>MAIN, BLOCK DATA, CHECK.</td>
</tr>
<tr>
<td>ENDDIF</td>
<td>MAIN, BLOCK DATA, CHECK.</td>
</tr>
<tr>
<td>EXIT</td>
<td>MAIN, BLOCK DATA, CHECK.</td>
</tr>
<tr>
<td>ENCASS</td>
<td>MAIN, BLOCK DATA, CHECK.</td>
</tr>
<tr>
<td>EOJ</td>
<td>MAIN, BLOCK DATA.</td>
</tr>
<tr>
<td>FILE</td>
<td>BLOCK DATA, DETNX, FPRNT.</td>
</tr>
<tr>
<td>FFLTR</td>
<td>MAIN, FPRNT, FOUT.</td>
</tr>
<tr>
<td>FFWD</td>
<td>MAIN, BLOCK DATA.</td>
</tr>
<tr>
<td>FSTRT</td>
<td>MAIN, FPRNT, CHECK.</td>
</tr>
<tr>
<td>IF</td>
<td>MAIN, BLOCK DATA, CHECK.</td>
</tr>
<tr>
<td>INDFLG</td>
<td>MAIN, BLOCK DATA, FOUT, FPRNT.</td>
</tr>
<tr>
<td>IMAX</td>
<td>MAIN, DETNX, SCAN1, SCAN1, SCAN.</td>
</tr>
<tr>
<td>LBFLG (LOGICAL)</td>
<td>MAIN, DETNX, SPRNT, FHAND, GBNLAE.</td>
</tr>
<tr>
<td>LBMX</td>
<td>MAIN, CHECK, GENLAB.</td>
</tr>
<tr>
<td>LEPTR</td>
<td>MAIN, CHECK, GENLAB.</td>
</tr>
<tr>
<td>NCYCLE</td>
<td>MAIN, BLOCK DATA, CHECK.</td>
</tr>
<tr>
<td>NESTNO</td>
<td>MAIN, SPRNT.</td>
</tr>
<tr>
<td>NEUTR</td>
<td>MAIN, BLOCK DATA.</td>
</tr>
<tr>
<td>NUMB</td>
<td>MAIN, BLOCK DATA, GENLAB.</td>
</tr>
<tr>
<td>OLD</td>
<td>MAIN, BLOCK DATA.</td>
</tr>
<tr>
<td>PRTR</td>
<td>MAIN, BLOCK DATA, DETNX.</td>
</tr>
<tr>
<td>PULOUT</td>
<td>MAIN, BLOCK DATA.</td>
</tr>
<tr>
<td>RINLO</td>
<td>MAIN, BLOCK DATA.</td>
</tr>
<tr>
<td>REPT</td>
<td>MAIN, BLOCK DATA, CHECK.</td>
</tr>
<tr>
<td>SEMI</td>
<td>BLOCK DATA, DETNX, SCAN1, FHAND.</td>
</tr>
<tr>
<td>SSTRT</td>
<td>SPRNT, CHECK.</td>
</tr>
<tr>
<td>STBEG</td>
<td>SPRNT.</td>
</tr>
<tr>
<td>STEND</td>
<td>SPRNT.</td>
</tr>
<tr>
<td>STMAX</td>
<td>MAIN, BLOCK DATA.</td>
</tr>
<tr>
<td>STMNO</td>
<td>MAIN, SPRNT, FPRNT.</td>
</tr>
<tr>
<td>SUBFN</td>
<td>MAIN, BLOCK DATA, CHECK.</td>
</tr>
<tr>
<td>TOBEG</td>
<td>MAIN, DETNX, STTYP, SCAN, FHAND.</td>
</tr>
<tr>
<td>TCEND</td>
<td>MAIN, DETNX, STTYP, SCAN, FHAND, CHECK.</td>
</tr>
<tr>
<td>TCP</td>
<td>MAIN, STTYP, CHECK.</td>
</tr>
<tr>
<td>TPITR</td>
<td>MAIN, CHECK.</td>
</tr>
<tr>
<td>TYPE</td>
<td>MAIN, DETNX, STTYP, SPRNT, FHAND, FCUT.</td>
</tr>
<tr>
<td>UTL</td>
<td>MAIN, BLOCK DATA, CHECK.</td>
</tr>
<tr>
<td>WHILE</td>
<td>MAIN, BLOCK DATA, CHECK.</td>
</tr>
</tbody>
</table>

-39-
### Local Variables

<table>
<thead>
<tr>
<th>SUBROUTINE</th>
<th>VARIABLES</th>
</tr>
</thead>
<tbody>
<tr>
<td>DETNX</td>
<td>EFF.</td>
</tr>
<tr>
<td>STYTP</td>
<td>BOTTCK, LENGTH, MIL, PNT.</td>
</tr>
<tr>
<td>SCAN1</td>
<td>CHAR.</td>
</tr>
<tr>
<td>SCAN</td>
<td>ISET, LEN, FAT.</td>
</tr>
<tr>
<td>SPRTNT</td>
<td>FIL, LEN1, LEN2, OVFL (LOGICAL), SEND.</td>
</tr>
<tr>
<td>FCUT</td>
<td>NINE, FAT, STRT.</td>
</tr>
<tr>
<td>CHECK</td>
<td>STYEL.</td>
</tr>
<tr>
<td>ERRORR</td>
<td>LEN, LEN1, LEN2.</td>
</tr>
</tbody>
</table>

### Global Arrays

<table>
<thead>
<tr>
<th>NAME</th>
<th>COUNT</th>
<th>VARIABLES</th>
</tr>
</thead>
<tbody>
<tr>
<td>DLNKS</td>
<td>131</td>
<td>MAIN, BLOCK DATA, SPRNT, FPRNT.</td>
</tr>
<tr>
<td>FEE</td>
<td>2</td>
<td>MAIN, ELCK DATA.</td>
</tr>
<tr>
<td>CASS</td>
<td>4</td>
<td>MAIN, BLOCK DATA, CHECK.</td>
</tr>
<tr>
<td>CCOLOM</td>
<td>5</td>
<td>MAIN, BLOCK DATA, CHECK.</td>
</tr>
<tr>
<td>CYCLE</td>
<td>5</td>
<td>MAIN, BLOCK DATA, CHECK.</td>
</tr>
<tr>
<td>CGOTO</td>
<td>9</td>
<td>MAIN, ELCK DATA.</td>
</tr>
<tr>
<td>CCNTI</td>
<td>8</td>
<td>MAIN, ELCK DATA.</td>
</tr>
<tr>
<td>DBSLLH</td>
<td>2</td>
<td>MAIN, ELCK DATA.</td>
</tr>
<tr>
<td>CCO</td>
<td>2</td>
<td>MAIN, BLOCK DATA.</td>
</tr>
<tr>
<td>EEEEND</td>
<td>3</td>
<td>BLOCK DATA, CHECK.</td>
</tr>
<tr>
<td>EEEXT</td>
<td>4</td>
<td>BLOCK DATA, CHECK.</td>
</tr>
<tr>
<td>ELZ</td>
<td>4</td>
<td>MAIN, ELCK DATA.</td>
</tr>
<tr>
<td>ENDCASS</td>
<td>7</td>
<td>MAIN, BLOCK DATA, CHECK.</td>
</tr>
<tr>
<td>ENIF</td>
<td>5</td>
<td>MAIN, ELCK DATA, CHECK.</td>
</tr>
<tr>
<td>ECL</td>
<td>1</td>
<td>MAIN, ELCK DATA.</td>
</tr>
<tr>
<td>FBUF</td>
<td>80</td>
<td>MAIN, DETNX, PHAND, FPRNT, FCUT.</td>
</tr>
<tr>
<td>GOTO</td>
<td>6</td>
<td>MAIN, BLOCK DATA.</td>
</tr>
<tr>
<td>IIIF</td>
<td>2</td>
<td>BLOCK DATA, CHECK.</td>
</tr>
<tr>
<td>IFNOT</td>
<td>9</td>
<td>MAIN, ELCK DATA.</td>
</tr>
<tr>
<td>ITERAT</td>
<td>9</td>
<td>BLOCK DATA, CHECK.</td>
</tr>
<tr>
<td>LABEL</td>
<td>200</td>
<td>MAIN, ELCK DATA, DETNX, FCUT, CHECK, GENLAB.</td>
</tr>
<tr>
<td>LEO</td>
<td>5</td>
<td>MAIN, ELCK DATA.</td>
</tr>
<tr>
<td>LLOOP</td>
<td>4</td>
<td>BLOCK DATA, CHECK.</td>
</tr>
<tr>
<td>MINUS</td>
<td>7</td>
<td>MAIN, ELCK DATA.</td>
</tr>
<tr>
<td>CNL</td>
<td>1</td>
<td>MAIN, ELCK DATA.</td>
</tr>
<tr>
<td>PLU</td>
<td>1</td>
<td>MAIN, ELCK DATA.</td>
</tr>
<tr>
<td>RREFT</td>
<td>6</td>
<td>BLOCK DATA, STYTP.</td>
</tr>
<tr>
<td>RESRD</td>
<td>200</td>
<td>BLOCK DATA, STYTP.</td>
</tr>
<tr>
<td>RSTAR</td>
<td>2</td>
<td>MAIN, BLOCK DATA.</td>
</tr>
<tr>
<td>SPIN</td>
<td>80</td>
<td>MAIN, ELCK DATA, DETNX,</td>
</tr>
</tbody>
</table>
CHAPTER 3

STYP, SCAN, SCAN, SPRNT, PHNL, FPRL, GENLAB, ERRCH.
STACK(200) MAIN, CHECK.
STFRM(6) BLOCK DATA, CHECK.
THENN(4) MAIN, ELOT DATA.
TCO (2) MAIN, ELOT DATA.
TRES(23) BKCK DATA, STYP.
UUNTL(5) MAIN, ELOT DATA, CHECK.

3.3 COMMON Blocks.

The COMMON blocks and the variables they include are listed below.

/LINEUT/ $EIN(80), \text{TCEG}, \text{TOEND}, \text{STEG}, \text{STEND}, \text{IMAX}, \text{LBFLG}$

/SPRINTF/ $SSTRT, \text{INC}, \text{STMNO}, \text{BIKNG}, \text{NESTNO}, \text{INDFLG}, \text{CURFLG}, \text{SNAX}$

/FRTINF/ $FBUF(80), \text{FPTR}, \text{FMAX}, \text{FSSTRT}, \text{BINKS}(131)$

/CENTNL/ $\text{STACK}(200), \text{LABEL}(200), \text{TCF}, \text{LBMAX}, \text{LBPTR}, \text{STPRE}(6), \text{TPTR}, \text{NEUTR}, \text{BKWRD}, \text{FRWRD}, \text{NEW}, \text{CLD}, \text{PULCUT}, \text{SNMAX}, \text{NUM}$

/OTPAT/ $\text{CGOIO}(9), \text{CONTI}(8), \text{ECL}(1), \text{GOTG}(6), \text{IFNOT}(9), \text{IG}(5), \text{MINUS}(1), \text{CNE}(1), \text{FLU}(1), \text{RSTAR}(2)$

/LINEAT/ $\text{CCLN}(1), \text{ELZ}(4), \text{DBSLH}(2), \text{DOO}(2), \text{BYE}(2), \text{THENN}(4), \text{ITO}(2)$

/STTYE1/ $\text{CCASE}, \text{CCLN}, \text{DBSL}, \text{EESE}, \text{EENL}, \text{ECJ}, \text{EEXIT}, \text{FFCR}, \text{IF}, \text{LCOF}, \text{RREPT}, \text{UNIL}, \text{WHILE}, \text{SUBFN}, \text{ENLNO}, \text{NCYCLE}, \text{ENCAS}, \text{EEND}$

/STTYE2/ $\text{CONT}, \text{CONT}, \text{FOTRN}, \text{BLANK}, \text{SEMI}$

/MMNIC/ $\text{CEE}, \text{PRTR}, \text{FPFL}, \text{CRDR}$

/KEYWD/ $\text{EEEND}(3), \text{EEEXT}(4), \text{CCYCLE}(5), \text{UUNTL}(5), \text{CASE}(4), \text{IF}(2), \text{ITERAR}(9), \text{ENCASS}(7), \text{ENIF}(5), \text{REPT}(6), \text{CCOLON}(5)$
CHAPTER 3

TABLE: TRES (23), RESRD (200)

TYPE: TYPE

FORTPRT: BLNKLN
CHAPTER 4

SPARKS USERS MANUAL

4.0 INTRODUCTION.

FORTRAN users, SPARKS is meant for you! As a FORTRAN user, you are taking advantage of a language that has been in use since 1957. You have, no doubt, come to rely on FORTRAN's established portability, highly-developed compilers, and wide-spread availability. You also know you are not alone, as there is an established large labor force familiar with FORTRAN. However, as you thumb through your FORTRAN program, haven't you wondered how nice it would be if you didn't have to keep jumping about following GO TO's? And, what of all the talk going around now about program structure and modularity? Don't you even feel a little bit guilty? You don't want to give up all of the advantages of FORTRAN just to eliminate GO TO's and get a little structure? Now you can guess why SPARKS is for you.

What is SPARKS? It is essentially a preprocessor for FORTRAN which inexpensively augments the language with improved syntactical constructs and other useful features. A FORTRAN preprocessor is a program which translates statements written in a language \( x \) into FORTRAN. In this case language \( x \) is SPARKS. SPARKS is classified as a preprocessor instead of a compiler because the source and the target language language have many statements in common. A translator such as SPARKS has many advantages. Most
importantly, it preserves a close connection with FORTRAN. Also many installations do not have a nicely structured language available and the translator provides a simple means for supplementing the existing FORTRAN capability.

4.1 Sample Program

Is a little convincing needed for you to give up the GO TO? Consider writing a program which searches for an integer $X$ in a sorted array of integers $A(N)$, $N$ less than or equal to 100. The output is the integer $J$ which is either zero if $X$ is not found or $A(J)=X$, $0<J<N+1$. The method used is the well known binary search algorithm. A reasonable attempt to write the program in FORTRAN follows:

```
SUBROUTINE FIND(A,N,X,J)
IMPLICIT INTEGER (A-Z)
DIMENSION A(100)
BOT=1
TOP=N
J=0
100 IF(BOT.GT.TOP) RETURN
MID=(BOT+TOP)/2
IF(X.GE.A(MID)) GO TO 101
TOP=MID-1
GO TO 100
101 IF(X.EQ.A(MID)) GC TC 102
ECT=MID+1
GO TO 100
```

-44-
102 J=MID
RETURN
END

Now, a look at this algorithm written in SPARKS.

SUBROUTINE PINS(A,N,X,J)
IMPLICIT INTEGER (A-Z)
DIMENSION A(100)
BOT = 1 ; TOP = N ; J = 0
WHILE BOT .LE. TOP DO
MID = (BOT + TOP) / 2
CASE
  : X .LT. A(MID) : TOP = MID - 1
  : X .GT. A(MID) : BOT = MID + 1
  ELSE : J = MID ; RETURN
ENDCASE
REPEAT
RETURN
END

The difference between the two algorithms is not dramatic, but it is significant. The WHILE and CASE statements allow the algorithm to be described in a more natural way. The program can be read from top to bottom without jumping up and down the page. When such improvements are consistently adopted in a large software project, the resulting code is bound to be easier to comprehend. Like to know more about SPARKS? Read on!
4.2 Language Definitions.

First the SPARKS language must be precisely defined. A distinction must be made between strictly FORTRAN statements and SPARKS statements. The latter are recognized by certain keywords and/or delimiters. Statements not containing SPARKS keywords are regarded as strictly FORTRAN and passed without alteration. SPARKS statements cause the translator to produce ANSI FORTRAN statements which accomplish the equivalent computation. Thus, SPARKS is compatible with FORTRAN and is, for all intents and purposes, a FORTRAN program. Hence, the local compiler ultimately defines the semantics of all SPARKS statements.

The method of inputting a SPARKS program is identical to FORTRAN; columns 7 to 72 are used for statements. Labels, except for format statements, are not normally used in SPARKS. However, they are input as are FORTRAN labels, in columns 1 to 5.

The SPARKS preprocessor outputs two temporary files. One file is the processed SPARKS code which is nicely indented. The second file is the FORTRAN translation of the SPARKS program. The FORTRAN is passed to the compiler for further translation. The SPARKS output is automatically provided to the user. The FORTRAN will normally be echoed by the local compiler depending on the installation.

SPARKS has a total of 20 keywords and special symbols. These words and symbols are listed below.
CHAPTER 4

**NOTE**

The above keywords/symbols MUST be surrounded by blanks to be recognized by the SPARKS preprocessor. Otherwise, they will be passed as FORTRAN variables. The keywords are not reserved in that they can be used as variables when not surrounded by blanks. However, to avoid confusion and possible errors, use of the keywords as variables is not recommended.

SPARKS statements can now be defined by giving their FORTRAN equivalents. The term statement, used below, and indicated by \( S, S_1, S_2, \) etc is meant to include any arbitrary number of both SPARKS and FORTRAN statements. There are six basic SPARKS constructions, two of which improve the testing of cases and four which improve the description of looping.

4.3 **SPARKS IF Statement**.

SPARKS

IF cond THEN
CHAPTER 4

S1
ELSE
S2
ENDIF

FORTRAN TRANSLATION
IF (.NOT. (cond)) GO TO 99999
S1
GO TO 99998

99999 CONTINUE
S2

99998 CONTINUE

The ELSE clause is optional, but the terminating ENDIF is mandatory. An IF statement without the ELSE clause is included below.

SPARKS
IF cond THEN
S1
GO TO 200
ENDIF
S3

200 CONTINUE

FORTRAN TRANSLATION
IF (.NOT. (cond)) GO TO 99999
S1
GO TO 200

99999 CONTINUE
CHAPTER 4

S3

200 CONTINUE
The GO TC in the SPARKS is included for those FORTRAN programmers who refuse to let go. It does serve to illustrate; however, that any legal FORTRAN statement can be mixed with SPARKS code. The 'cond' of the above IF statements must be a legal FORTRAN conditional.

4.4 SPARKS CASE Statement.
The second construction dealing with testing is the SPARKS CASE statement. This construct is most useful in consolidating nested test statements or strings of related conditional tests.

SPARKS
CASE
: cond1 : S1
: cond2 : S2
: condn : Sn
: ELSE : Sn+1
ENDCASE

FORTRAN TRANSLATION
IF(.NOT.(cond1)) GO TO 99999
   S1
   GO TO 99998
99999 IF(.NOT.(cond2)) GO TO 99997
   S2
CHAPTER 4

GO TO 99998
99997 IF (.NOT. (condn)) GO TO 99996
Sn
GO TO 99998
99996 CONTINUE
Sn+1
99998 CONTINUE

The cond1, cond2, etc., are any arbitrary number of legal FORTRAN conditionals. The ELSE case designates that the statements represented by Sn+1 will always be executed if all previous conditions are false. The ELSE clause is optional. The ENDCASE is mandatory.

4.5 SPARKS LOOP Statements.

The four LOOP structures of SPARKS are the WHILE, LOOP/UNTIL, LOOP/REPEAT, and the FOR loop.

* * * * * * * * N O T E * * * * * * * *

The programmer must provide for the incrementation or change in condition for proper termination of the WHILE loop and the LOOP/UNTIL. Also, the LOOP/REPEAT is a loop forever construction. The programmer must provide the means to get cut of the LOOP/REPEAT.

* * * * * *

The construction of the WHILE loop is shown below.
CHAPTER 4

**SPARKS**

**WHILE** cond DC

**S1**

**REPEAT**

**FORTRAN TRANSLATION**

99999 IF (.NOT. (cond)) GC TC 99998

**S1**

GO TO 99999

99998 CONTINUE

The **S1** indicates an arbitrary number of statements and must include a means for meeting the cond for termination. The cond must be a legal FORTRAN conditional. The **REPEAT** is mandatory.

The **LOCOP/UNTIL** is similar to the **WHILE** loop and is used when the test to exit is desired at the bottom of the loop.

**SPARKS**

**LOCOP**

**S1**

**UNTIL** cond **REPEAT**

**FORTRAN TRANSLATION**

99999 CONTINUE

**S1**

IF (.NOT. (cond)) GO TO 99999

The **S1** and cond imply the same meaning as in the **WHILE** loop above. Again, inclusion of the **REPEAT** is mandatory.

The **LOCOP/REPEAT** is convenient to provide a loop

-51-
structure where a exit condition can be encountered at several locations within the loop `body`. The programmer should be aware that this loop structure will not terminate unless some exit or stop condition is included in the loop `body`. The REPEAT is the loop delimiter and is mandatory.

**SPARKS**

```
LOOP
S1
REPEAT
```

**FORTRAN TRANSLATION**

```
99999 CONTINUE
S1
GO TO 99999
```

The SPARKS FCR loop is similar in its action to the FORTRAN DC loop but is much more powerful. It should be noted; however, that the translation of the FCR loop in no way resembles the DC loop.

**SPARKS**

```
FCR vble=exp1 TO exp2 BY exp3 DC
S1
REPEAT
```

**FORTRAN TRANSLATION**

```
vble=exp1
GC TO 99999
99998 vble=vble+exp3
```

-52-
CHAPTER 4

99999 IF(.NOT.(vble-(exp2))*(exp3).LE.0)
1GO TO 99997
S1
GO TO 99998
99997 CONTINUE

The '1GO TC 99997' is a continuation line. The three expressions (exp1, exp2, exp3) are allowed to be arbitrary FORTRAN arithmetic expressions. Similarly, vble may be any FORTRAN variable. Negative incrementation of the FOR loop is acceptable as shown in the example below.

FOR I=10 TC 2 BY -2 DO
   J=I+10
REPEAT

The phrase BY exp3 is optional and if not included the incrementation defaults to +1. Since exp2 and exp3 are re-evaluated each time through the loop, care must be taken not to modify the value of these expressions within the loop body. As with all SPARKS loops, the REPEAT is mandatory.

4.6 SPARKS EXIT and CYCLE Statements.

The EXIT statement is used in SPARKS to leave a loop structure prematurely. The EXIT causes control to be transferred to the first statement outside of the innermost loop statement that contains it. The EXIT statement can be used with any of the four SPARKS loop structures. An example of its use and the FORTRAN translation is shown
CHAPTER 4
below.

SPARKS

LOOP
S1
IF cond THEN EXIT
ENDIF
S2
REPEAT

FORTRAN TRANSLATION
99999 CONTINUE
S1
IF (.NOT. (cond)) GO TO 99998
    GO TO 99997
99998 CONTINUE
S2
    GO TO 99999
99997 CONTINUE

The CYCLE statement may also be used within any SPARKS loop construction. The CYCLE causes control to be transferred directly to the end of the innermost loop that contains it. Normal loop iteration is then continued. The following is example of both the EXIT and CYCLE statements. Note that the CYCLE does not transfer control outside of the loop that contains it.

SPARKS

-54-
CHAPTER 4

LCOP
S1
CASE
: cond1 : EXIT
: cond2 : CYCLE
ENDCASE
S2
REPEAT

FORTRAN TRANSLATION

99999 CONTINUE
S1

IF(.NOT.(ccnd1)) GO TO 99996
GO TO 99994
GO TO 99995

99996 IF(.NOT.(ccnd2)) GO TO 99995
GO TO 99998

99995 CONTINUE
S2

99998 GO TO 99999

99994 CONTINUE

The sharp FORTRAN programmer will immediately notice that this translation produces unreachable GC TO statements. The EXIT, CYCLE, and RETURN statements when used in a CASE or an ELSE of the IF/THEN/ELSE statement nested in a loop will generate unreachable GO TO statements. Many FORTRAN compilers will give non-fatal compiler warnings concerning
CHAPTER 4

the unreachable statements; however, the warnings should not effect the execution of the program.

4.7 Labelling in SPARKS.

The use of five digit labels in the examples has probably been noticed. SPARKS generates labels beginning with 99999 and decreases by one for each new label required. The numbering begins again for each new subroutine. Programmers who use labelled statements should avoid using large numbers to prevent label conflicts when translation occurs. User defined labels are passed without modification to the FORTRAN output.

4.8 The SPARKS EOJ Statement.

The SPARKS EOJ signals the translator that is the last card of the program. It must appear somewhere in columns 7 to 72 and be surrounded by blanks. The EOJ is used once per program.

4.9 SPARKS Comments.

SPARKS comments are delimited by the double slash. The following is an example of a proper SPARKS comment.

// THIS IS A SPARKS COMMENT //

All characters within the double slashes will be ignored. The comments are restricted to one line in columns 7 to 72. The double slashes must be surrounded by blanks. The double
CHAPTER 4

slash is taken as a single symbol and may not contain blanks between the two single slash marks. SPARKS comments appearing on the same line as other statements must be separated from those statements with a semicolon (see section 4.9). SPARKS comments will appear in the FORTRAN output as follows:

C // THIS IS A SPARKS COMMENT

The translator will automatically put the C in column one. Note that the trailing double slash is not passed to the FORTRAN output. However, failure to include the trailing double slash in the SPARKS will cause the translator to output an error message. FORTRAN comments are allowed and are output without modification.

4.10 General Information and Other Features

* * * * * * N O T E * * * * * *

The indention feature in SPARKS will sometimes cause a long line to exceed column 72. Normally this is not a problem as the translator automatically introduces a proper continuation card. However, if the long line contains a Hollerith field and the break occurs in that field, a fatal FORTRAN error may be introduced. The programmer should use caution when coding long statements such as FORMAT and DATA statements containing Hollerith fields.

* * * * * *
CHAPTER 4

If such statements cannot be avoided, the preprocessor operating mode can be altered to pass the statements without processing. A F in column one signals the preprocessor that subsequent lines contain only FORTRAN statements. Those statements are then passed without any checking or indentation. The next line with a F in column one returns the preprocessor to its normal mode. The cards containing the signal F must not contain other statements. The SPARKS output will have the lines surrounded by the F signal blocked out by the message:

+++++++BEGINNING OF FCRTRAN

Arbitrary number of FORTRAN Statements

+++++++END OF FCRTRAN

The FCRTRAN output will not contain any messages and it will not be indented.

The semicolon is used in SPARKS as a statement delimiter. Using the semicolon, more than one statement can be put on a single line. For example, the following line beginning in column 7 is a legal SPARKS input.

C=D+E ; A=B+C ; X=A ; // SPARKS COMMENT //

The preprocessor will translate the line into four lines of FCRTRAN as follows:

C=D+E
A=E+C
X=A
C // SPARKS COMMENT

-58-
CHAPTER 4

As a SPARKS key symbol, the semicolon must be surrounded by blanks.

4.11 Conclusions.

This user's guide is an adaptation of the user's guide developed by Dr. Ellis Horowitz and the SPARKS User Group, University of Southern California. Dr. Horowitz designed the SPARKS language originally for describing algorithms in the book *FUNDAMENTALS OF DATA STRUCTURES* by Ellis Horowitz and Sartaj Sahni, published by Computer Science Press, Woodland Hills, California. The original SPARKS preprocessor was developed from this prototype language.

This user's guide is an accurate description of the language features as currently supported by the revised version of the translator. The new version of the translator was rewritten in SPARKS by Richard M. Stroud and Dr. Myron A. Calhoun, Kansas State University, Manhattan, Kansas.
APPENDIX A

SPARKS PREPROCESSOR

WRITTEN IN

SPARKS
NEST  BLK  STNO

THIS IS A SPARKS PROGRAM

SPARKS TRANSLATION BY KSU SPARKS PREPROCESSOR AS MODIFIED BY R. STRCUD, M. CALHOUN, AND J. MARTIN.

// MAIN PROGRAM //
//
IMPLICIT INTEGER(A-Z)
LOGICAL LBFLG, ECCN, ECONE1, BAD
COMMON/INPUT/SPIN(80), TOBEG, TOEND, STBEG, STEND, IMAX, LRFLG
COMMON/SPRING/SSSTR, INCN, STMNC, BLKNC, NESTNO, IMINFLG, CURFLG, SMAX
COMMON/FRTINF/FRLF(80), FPTR, FMAX, FSRTL, BLNKS(131)
COMMON/CENTRL/STACK(20U), LABEL(20U), TCP, LBMAX, LPTR, STFM(6),
1  TPITR, NEUTR, BKWRC, FFWRD, NEW, CLD, PULCUT, STMAX, NUMB
COMMON/OLPAT/CGOTO(9), CONTI(8), ECL(1), GOTO(6), IFNOT(9), LEQ(5),
1  MINUS(1), CNE(1), FLU(1), RSTAR(2)
COMMON/INP/CLMN(1), ELZ(4), DBSFL(2), CCC(2), BYE(2), THENN(4), TOC(2)
COMMON/STTYPE/CCASE, COLCN, DBSFL, EELSE, EEND, EOJ, EEXIT, FFOR, IIF,
1  LLCCP, REPPT, LNL, WILE, SUBRN, RINLC, NCYCLE, ENDCAS, ENDIF
COMMON/STTP2/CNTL, CCNT, FCTR, BLNK, SEMI
COMMON/AMNC/CALL, PRTR, FFILE, CRDR
COMMON/KEYCL/EENCAS(3), EEXT(4), CCYCLE(5), UUNTL(5), CAS(4), IIF(7),
1  ITERAT(9), ENCALASS(7), EENIF(5), REPPT(6), CCCLCN(5)
COMMON/TABLES/TRES(23), RESRC(20U)
COMMON/TYP/TYPE
//
// WRITE TITLES ETC. //
WRITE(PRTR, 3)(SPIN(I), I=1, 80)
COMMON/FORMAT(4X, 4HNEST, 2X, 3HBLK, 2X, 4HSTNO, 4X, 30X, 80A1)
WRITE(PRTR, 4)
COMMON/FORMAT(32X, 'SPARKS TRANSLATION BY KSU SPARKS PREPROCESSOR',
1 ' AS'/'32X, ' MODIFIED BY R. STRCUD, M. CALHOUN, AND J. MARTIN./)
//
// SET UP OPTIONS //
IMAX=72
S MAX=100
FMAX=72
INCP=3
FSTRT=7+INCR
SSTRT=7+INCR
// EXTRACT OPTICALS GIVEN ON THE CONTROL CARD AND CHANGE ABOVE //
// DEFAULTS ACCORDINGLY--THIS COULD USE DEFTNX //
// INITIALISE THE PROGRAM RELATED VARIABLES. //
STACK(1)=SUBFN
TCP=1
TPTR=0
LPTR=6
TCEND=IMAX+1
BLKNC=0
STMNO=0
NESTNO=0
// START MAIN LOOP //
LCCP
1 1 49
FPTR=FSTRT
CALL DETNX(TYPE)
// PROCESS STATEMENTS ACCORDING TO TYPE //
// 1. TRANSLATE AND PRINT FORTRAN. //
// 2. SET INDENTATION FLAG---INDFLG. //
// 3. SET CURFLG NEW-OLD--CURRNT STATEMENT TO BE PRINTED IN //
// CLD CR NEW INDENTATION. //
//
// IF TYPE.EQ.RINLC THEN
2 2 58
A=TCBEG
B=TOENC
CALL DETNX(TYPE)
// IF THE SECOND TOKEN IS NOT FUNCTION THEN PASS AS FORTRAN. //
2 2 62
IF TYPE.NE.SUBFN THEN
3 3 63
TYPE=FCTRA
3 3 64
ENDIF
3 4 65
TOEND=B
3 4 66
TCBEG=A
3 4 67
ENDIF
1 5 68
STBEG=TOREG
1 5 69
// THE MAIN CASE STATEMENT //
CASE
  TYPE.EQ.FCTRAN :
    CALL FHAND
    STEND=TCEND
    CURFLG=CLD
    INDFLG=NEUTR
  TYPE.EQ.DBLSL :
    // LCK FOR A DOUBLE SLASH--SCAN IS NOT USED SINCE IT STOPS //
    // CN A SEMICOLON. //
    LCPP
    IF TOEND.GE.IMAX-1 THEN
      CALL ERRCR(2,DBSLH,2,1,0,0)
      // ERRCR--MISSING DOUBLE SLASH SUPPLIED AT END OF STATEMENT. //
      EXIT
    ENDF
    IF SPIN(TCEND+1).EQ.DBSLH(1).AND.SPIN(TOEND+2).EQ.DBSLH(1) THEN
      EXIT
    ENDF
    TCEND=TCEND+1
  REPEAT
  FBUF(1)=CEE
  // THE TYPE IS CHANGED SO THAT FOUT CAN CHECK IT AND PUT A C IN //
  // COLUMN ONE FOR LONG COMMENTS. //
  TYPE=CCMT
  CALL FCLT(SPIN,TOBEG,TCEND)
  CALL FPRNT
  INDFLG=NEUTR
  CURFLG=CLD
  TCEND=TCEND+2
  STEND=TCEND
  TYPE.EQ.IIF :
    LAB=GENLAB(1)
    CALL SCAN(THENN,4,ECOND)
    IF ECOND THEN
      // ABSENCE OF THEN ASUMED TO INDICATE FORTRAN IF. //
      TCBEG=TCBEG-2
    CALL FHAND
    ELSE
// Определяем постоянные:
CALL FCUT(IFNOT,1,9)
CALL FCUT(SPNT,TCBEG,TCENC)
CALL FCUT(CGTO,1,9)
CALL FCUT(LABEL,LAB3,LAB+4)
CALL FPRT
STACK(TCP+1)=LAB
STACK(TCP+2)=IIF
TCP=TCP+2
IF TCP+GT+STMAX THEN
    CALL ERRCR(6,0,0,5,0,0)
ENDIF
// STACK OVERFLOWED, PROGRAM TERMINATED. //</
TOEND=TCPEND+4
IF FSTRT+INCR.LT.IMAX THEN
    FSTRT=FSTRT+INCR
ENDIF
ENDIF
STENC=TCENC
CURFLG=CLUD
INOFILG=FRWRO
: TYPE.EQ.EENIDF :
CALL CHECK(EENIDF,BAD)
IF BAD THEN CYCLE ENDIF
FSTRT=FSTRT-INC
FPR=FSTR
IF LBFLG THEN
    // LABELLED STATEMENT--GENERATE 'LAB CONTINUE' //</
    CALL FCUT(CTNT,1,3)
    CALL FPRT
ENDIF
LAB=STACK(TOF-1)
TCF=TCP-2
LBPTR=LBPTR-5
FOR I=1 TO 6 DC
    L=LAB+I-1
3 28  146  
3 28  147  
3 29  148  \ FBUF(I)=LABEL(L)  
3 29  149  REPEAT  
3 29  150  \ FBUF(6)=BLANK  
3 29  151  CALL FCUT(CCINT,1,6)  
3 29  152  CALL FPRTN  
3 29  153  // GENERATE A LAB CONTINUE. //  
3 29  154  STEND=TCNAC  
3 29  155  INDFLG=RTKRD  
3 29  156  CURFLG=NEW  
3 29  157  TYPE.EQ.EELSE:\  
3 30  158  CALL CHECK(EELSE,BAC)  
3 31  159  IF BAD THEN \ 
3 32  160  CYCLE  
3 32  161  ENDIF  
3 32  162  FSTRT=FSTRT-INCR \  
3 33  163  IF LBFLG THEN \  
3 33  164  CALL ERROR(1,ELZ,4,2,0,0)  
3 33  165  // ERROR--LABELLING THE ELSE HAMPER PROGRAM CLEANLINESS-- //  
3 33  166  // DELETE FROM THE FORTRAN OUTPUT. //  
3 33  167  ENDIF  
3 34  168  LAB2=GENLAB(1)  
3 34  169  LAB1=STACK(TCP-1)  
3 35  170  FCR I=1 TO 6 DO  
3 35  171  \ FBUF(I)=BLANK  
3 36  172  REPEAT  
3 36  173  CALL FCUT(GOTO,1,6)  
3 36  174  CALL FCUT(LABEL,LAB2,LAB2+4)  
3 36  175  CALL FPRTN  
3 36  176  // GO TO LAB2 //  
3 36  177  FOR I=1 TO 5 DO \ 
3 37  178  \ L=LAB1+I-1 
3 37  179  \ FRUF(I)=LABEL(L)  
3 37  180  REPEAT  
3 38  181  FRUF(6)=BLANK  
3 38  182  CALL FCUT(CCINT,1,8)  
3 38  183  CALL FPRTN  
3 39  184  // LAB1 CONTINUE //  
3 39  185  STACK(TCP-1)=LAB2
STENC=TCENC
INDFLG=NEUTR
CURFLG=PULCUT
IF FSTRT+INCR.LT.IMAX THEN
  FSTRT=FSTRT+INCR
ENDIF

: TYPE.EQ.FFOR :
IF LDFLG THEN
  // LAB WILL HAVE IDENTIFYING LABEL IF ANY. //
  LAB=GENLAB(0)
  CALL FOLT(LCNTI,1,8)
  CALL FPRNT
ENDIF
TOBEG=TCENC+1
CALL SCANI(ECL,ECCND)
IF ECCND THEN
  CALL ERROR(2,EQL,1,3,0,0)
  // ERROR--MISSING EQUAL-STATEMENT SKIPPED. //
  CYCLE
ENDIF
VBLEA=TCBEG
VBLEB=TCENC
TCEND=TCENC+1
CALL SCANI(TCC,2,ECCND)
IF ECCND THEN
  CALL ERROR(2,TCC,2,3,0,0)
  // ERROR--MISSING 'TO' - STATEMENT SKIPPED. //
  TCEND=TCENC+1
  CYCLE
ENDIF
TCEND=TCENC+2
CALL FCUT(SP,VBLEA,TCEND-2)
CALL FPRNT
  // VARIABLE=EXPRESSION1. //
ITMP=TCEND
CALL SCANI(RYE,2,ECNCD)
EX2A=TOBEG
EX2B=TCEND
2   47   222
2   47   223
3   48   224
3   48   225
3   49   226
3   49   227
3   50   228
3   50   229
3   51   230
3   52   231
3   52   232
3   52   233
3   52   234
3   52   235
3   52   236
2   54   237
2   54   238
2   54   239
2   54   240
2   54   241
2   54   242
2   54   243
2   54   244
2   54   245
2   54   246
2   54   247
2   54   248
2   54   249
2   54   250
3   55   251
3   55   252
2   56   253
2   56   254
2   56   255
3   57   256
3   57   257
2   58   258
2   58   259

TCEND=TCENC+2
ENDIF
CALL SCAN(CCC,2,ECCND)
ENDIF
IF ECCND THEN
TCEND=ITMF
ENDIF
EX2R=TCEND
ENDIF
IF ECCND THEN
CALL ERROR(2,DOC,2,1,0,0)
// ERROR--MISSING 'DU'--SUPPLIED AT END OF STATEMENT. //
TCEND=TCEND+1
ELSE
TCEND=TCEND+2
ENDIF
LAB1=GENLAB(1)
LAB2=GENLAB(1)
LAB3=GENLAB(1)
LAB4=GENLAB(1)
// LAB IS THE IDENTIFYING LABEL, LAB4 IS THE TARGET OF EXIT, //
// AND LAB3 IS THE TARGET OF CYCLES. //
STACK(TCP+1)=LAB2
STACK(TCP+3)=LAB
STACK(TOP+4)=LAB3
STACK(TCP+5)=LAB4
STACK(TCP+6)=FFOR
STACK(TCP+2)=TPITR
TOP=TCP+6
IF TOP.GT.STMAX THEN
CALL ERROR(6,0,0,5,0,0)
ENDIF
// STACK OVERFLOWED---PROGRAM TERMINATED. //
TPITR=TCP
FOR I=1 TO 6 DC
FRUFI(1)=PLANK
REPEAT
CALL FCUT(GOTC,1,6)
CALL FCUT(LABEL,LAB1,LAB1+4)
CALL FPRNT
// GC TO LAB1. //
FOR I=1 TO 5 CC
  L=LAB2+I-1
  FB=FB(L)
REPEAT
  FB=FB(L)
  CALL FCUT(SPIN,VBLEA,VBLEB)
  CALL FOUT(EQL,1,1)
  CALL FCUT(SPIN,VBLEA,VBLEB)
  CALL FCUT(PLL,1,1)
IF ECOND THEN
  // THE 'BY' IS MISSING---DEFAULT INCREMENT OF EXPRESSION TO ONE. //
  CALL FOUT(CNE,1,1)
ELSE
  CALL FCUT(IFACT,9,9)
  CALL FOUT(SPIN,TCBEG,TCEND-2)
  CALL FCUT(RSTAR,1,1)
ENDIF
CALL FPRNT
// LAB2--INDICATES VARIABLE=VARIABLE+EXPRESSION3. //
FOR I=1 TO 5 CC
  L=LAB1+I-1
  FB=FB(L)
REPEAT
  FB=FB(L)
  CALL FCUT(IFACT,1,9)
  CALL FOUT(IFACT,9,9)
  CALL FCUT(SPIN,VBLEA,VBLEB)
  CALL FCUT(MINUS,1,1)
  CALL FCUT(IFACT,9,9)
  CALL FCUT(SPIN,EX2A,EX2B)
  CALL FCUT(RSTAR,1,1)
  CALL FCUT(RSTAR,1,1)
IF .NCT.ECOND THEN
  // IF 'BY' MISSING THEN GENERATE...(EXP2).LE.0... //
  // IN PLACE OF...(EXP2)*(EXP3).LE.0..... //
  CALL FCUT(RSTAR,2,2)
CALL FOLT(IFNOT,9,9)
CALL FOLT(SPIN,TCBEG,TCEND-2)
CALL FCLT(RSTAR,1,1)
ENDIF
CALL FCUT(LEC,1,5)
CALL FCUT(CGCTC,1,9)
CALL FCUT(LABEL,LAB3,LAB4+4)
CALL FPRNT

// LAB1...IF(.NOT.((VBLE-(EXP3)*.LE.0)) GO TO LAB3... //
STENC=TCEND
INDFLG=FR4RD
CURFLG=CLD
IF FSTRT+INCR.LT.IMAX THEN
  FSTRT=FSTRT+INCR
ENDIF

: TYPE.EQ.KREPT :
CALL CHECK(KREPT,BAG)
IF HAD THEN
  CYCLE
ENDIF
IF LBFLG THEN
  // LABELLEC STMT, GENERATED 'LAB CONTINUE' //
  CALL FOLT(CCNTI,1,8)
  CALL FPRNT
ENDIF
STTOP=STACK(TCP)
// LAB3=EXIT TARGET,LAB2=CYCLE TARGET,LAB1=LOOP BACK LABEL //
IF STTCP.EQ.FFCR THEN
  LAB1=STACK(TCP-5)
  LAB2=STACK(TCP-2)
  LAB3=STACK(TCP-1)
  TCP=TCP-6
  LBPTR=LBPTR-20
ELSE
  LAB1=STACK(TGP-3)
  LAB2=STACK(TGP-2)
  LAB3=STACK(TGP-1)
  TCF=TCP-5
LBPTP=LPTR-15
ENDIF
TPITR=STACK(TPITR-4)
FSRT=FSRT-INC
FPTR=FSRT
FOR I=1 TO 5 DC
    L=LAB2+I-1
    FBUF(I)=LABEL(L)
REPEAT
    FBUF(6)=BLANK
    CALL FCUT(GOTC,1,6)
    CALL FOUT(LABEL,LAB1,LAB1+4)
    CALL FPRT
    // GO TO LABEL 1 //
FOR I=1 TO 5 DC
    L=LAB3+I-1
    FBUF(I)=LABEL(L)
REPEAT
    FBUF(6)=BLANK
    CALL FCUT(CONTI,1,8)
    CALL FPRT
    // LAB3 CONTINUE //
STEND=TCEND
CURFLG=NEW
INDFLG=BKWPD
: TYPE.EQ.CCASE :
LAB=GENLAB(1)
// LABEL(1:5)=BLANKS. //
STACK(TCP+1)=1
STACK(TCP+2)=LAB
STACK(TCP+3)=CCASE
TOP=TCP+3
IF TOP.GT.STMAX THEN
    CALL ERROR(6,0,0,5,0,0)
ENDIF
// STACK OVERFLOWED---PROGRAM TERMINATED. //
STEND=TCEND
CURFLG=CLD
INDFLG=FRWD
IF LBFLG THEN
   // Labeled statement—generate 'LAB CONTINUE'. //
   CALL FGLT(CONTI,1,8)
   CALL FPRT
ENDIF
// Set the increment //
IF FSTRT+INCR LT IMAX THEN
   FSTRT=FSTRT+INCR
ENDIF
: TYPE.EQ.CCLCN :
CALL CHECK(CCLCN,BAD)
IF BAD THEN
   CYCLE
ENDIF
IF LBFLG THEN
   CALL ERROR(1,CASS,4,2,0,0)
   // Error—labelling the case hampers program cleanliness //
   // Delete from the FORTRAN output. //
   FOR I=1 TO 6 DO
      FBUF(I)=BLANK
   REPEAT
ENDIF
LAB1=STACK(TCP-1)
LAB2=STACK(TCP-2)
FSTRT=FSTRT-INCR
FPRT=FSTRT
// Case material printed one indent to the right. //
IF LAB2 NE 1 THEN
   CALL FGRT(GCTO,1,6)
   CALL FCUT(LABEL,LAB1,LAB1+4)
   CALL FPRT
   // GOTO LAB1. //
ENDIF
CALL SCAN(CCLN,ECCND)
IF ECCND THEN
   CALL ERROR(2,CCLN,1,1,0,0)
ENDIF
// ERRCF-MISSING CCLCA-SUPPLIEC AT END OF STATEMENT //
K=TOEND
TCEND=TCBEG
// SPIN(K)=IMAX OR SPIN(K+1)=SEMICOLON OR COLON. //
TMAX=IMAX
IMAX=K
CALL SCAL(ELZ,4,ECOND)
IMAX=TMAX
IF ECCND THEN
LAB3=GENLAB(1)
TCEND=K
// CLTPLT-LAB2-IF NOT CONDITION THE GO TO LAB3. //
FOR I=1 TC 5 DO
   L=LAB2+I=1
   FBUF(I)=LABEL(L)
REPEAT
FBUF(6)=BLANK
CALL FCUT(IFNOT,1,9)
CALL FOLT(SPIN,TCBEG,TCEND)
CALL FCUT(CGCTO,1,9)
CALL FOLT(LABEL,LAB3,LAB3+4)
CALL FPRTN
TCEND=TCEND+1
ELSE
// ELSE CLAUSE FOUND //
FOR I=1 TC 5 DO
   L=LAB2+I=1
   FBUF(I)=LABEL(L)
REPEAT
FBUF(6)=BLANK
CALL FCUT(CCINTI,1,8)
CALL FPRTN
// LAB2 CONTINUE //
TCEND=K+1
LAB3=1
// LABEL(1....5) ARE BLANKS. //
ENDIF
STACK(TCF-2)=LAB3
INDFLG=NELTR
CURLFLG=PULCUT
STEND=TCEND
// RESTORE THE INDENTATION. //
IF FSTPT+INCR.LT.IMAX THEN
  FSTRT=FSTRT+INCR
ENDIF

: TYPE=EG,ENDCAS:
CALL CHECK(ENCCAS,BAD)
IF BAD THEN
  CYCLE
ENDIF
FSTRT=FSTRT-INCR
FPTR=FSTRT
IF LBFLG THEN
  // LABELLED STATEMENT--GENERATE 'LAB CONTINUE' //
  CALL FCLT(CNTI,1,8)
  CALL FPRTN
ENDIF
LAB1=STACK(TCP-1)
LAB2=STACK(TCP-2)
TCP=TCP-3
LPTR=LAB1
FOR I=1 TO 5 DO
  L=LAB2+I-1
  FBUF(I)=LABEL(L)
REPEAT
  FBUF(6)=BLANK
END "LAB CONTINUE"
FOR I=1 TO 5 DO
  L=LAB1+I-1
  FBUF(I)=LABEL(L)
REPEAT
  FBUF(6)=BLANK
CALL FCUT(Conti,1,8)
CALL FPRTN
LAB1 = GENLAB(0)
LAB2 = GENLAB(1)
LAB3 = GENLAB(1)

STACK(TCP+2) = LAB1
STACK(TOP+3) = LAB2
STACK(TCP+4) = LAB3
STACK(TOP+5) = LLCOP
STACK(TOP+5) = TPITR
TCP = TCP+5

IF TOP.GT.STMAX THEN
CALL ERROR(6,0,0,5,0,0)
ENDIF

STACK OVERFLOWED, PROGRAM TERMINATED

FCR I = 1 TO 5 CO

L = LAB1 + I - 1
FBUF(I) = LABEL(L)
REPEAT
FBUF(6) = BLANK
CALL FOUT(CCNTI,1,8)
CALL FPNT
TPITR = TCP
CURFLG = CLD
INDFLG = FRWRD
STEND = TCEND
IF FSTRT + INCR.LT. IMAX THEN
FSTRT = FSTRT + INCR
ENDIF

LAB3 = GENLAB(1)
LAB2 = GENLAB(1)
LAB1 = GENLAB(0)
// LAB2 IS THE TARGET OF EXITS, LAB3 FOR CYCLES. //
CALL SCAN(CC,2,ECCND)
IF ECCND THEN
   CALL ERROR(2,CC,2,0,0)
   // CALL ERROR---MISSING 'DO'-SUPPLIED AT THE END OF STMT. //
   TOEND=TCEND+1
ELSE
   TCENC=TCEND+2
ENDIF
IF .NOT.LBFLG THEN
   FOR I=1 TO 5 DO
      L=LAB1+I-1
      FBUF(I)=LABEL(L)
      REPEAT
      FBUF(5)=BLANK
   ENDF
   CALL FCUT(IFNOT,1,9)
   CALL FCUT(SPIN,TOBEG,TCENC-2)
   CALL FCUT(CGCTC,1,9)
   CALL FCUT(LABEL,LAB2,LAB2+4)
   CALL FPRINT
   // GENERATE LAB1 AND IF TRUE CCND GO TO LAB2. //
   STACK(TCP+2)=LAB1
   STACK(TCP+3)=LAB3
   STACK(TOP+4)=LAB2
   STACK(TCP+5)=WHILE
   STACK(TCP+1)=TPITR
   TOP=TOP+5
   IF TCP.GT.STMAX THEN
      CALL ERRCR(6,0,0,5,0,0)
   ENDIF
   // STACK OVERFLOWED, PROGRAM TERMINATED. //
   TPITR=TOP
   CURFLG=CLD
   INDFLG=FR,WRD
   STEND=TOEND
   IF FSTRT+INCR.LT.IMAX THEN
      FSTRT=FSTRT+INCR
3 135 564
2 136 565
2 137 566
3 138 567
3 138 568
2 139 569
2 139 570
3 139 571
3 140 572
4 141 573
4 141 574
4 141 575
4 141 576
3 142 577
2 143 578
3 144 579
3 144 580
2 145 581
2 145 582
2 145 583
2 145 584
2 145 585
2 145 586
2 145 587
2 145 588
2 146 589
2 146 590
3 147 591
3 147 592
2 148 593
2 148 594
2 149 595
2 149 596
3 150 597
3 150 598
2 151 599
2 151 600
2 152 601

ENDIF
: TYPE.EQ.CONT:
FOR I=1 TO 6 DO
  FBUR(I)=SPIN(I)
REPEAT
TCBEG=7
WHILE SPIN(TEBEG).EQ.BLANK DO
  TCBEG=TCBEG+1
  IF TCBEG.GT.IMAX THEN
    TOEND=IMAX+1
    // BLANK CONTINUATION CARD, DELETED. //
    EXIT
  ENDIF
REPEAT
IF TOEND.GT.IMAX THEN
  CYCLE
ENDIF
// SPIN(TCBEG)=NOTBLANK; REST OF CARD CAN BE INDENTED. //
CALL FAND
FBUR(6)=BLANK
STREG=TCBEG
STEND=TOEND
CURFLG=CLD
INLFLG=NEUTR
: TYPE.EQ.EXIT:
CALL CHECK(EXIT,BAD)
IF BAD THEN
  CYCLE
ENDIF
CALL TYHANC(TYPE,EXIT,CCYCLE)
: TYPE.EQ.CYCLE:
CALL CHECK(EXIT,BAD)
IF BAD THEN
  CYCLE
ENDIF
CALL TYHANC(TYPE,EXIT,CCYCLE)
: TYPE.EQ.UTL:
CALL CHECK(UTL,BAD)
IF BAG THEN
  CYCLE
ENDIF
FSTR1=FSTR1-0RCH
FPTR=FSTR1
LAB1=STACK(TCP-3)
LAB2=STACK(TCP-2)
LAB3=STACK(TCP-1)
TOP=TOP-5
LBPTR=LBPTR-15
TPITR=STACK(IFITR-4)
IF LBFLG THEN
  CALL ERROR(1,UNITL,5,2,0,0)
  // ERRCR---LABELLING OF THE UNTIL HAMPER'S PROGRAM //
  // CLARITY, DELETED FROM FORTRAN OUTPUT. //
ENDIF
FOR I=1 TO 5 CC
  L=LAB2+I-1
  FBUF(I)=LABEL(L)
REPEAT
  FBUF(6)=BLANK
  CALL FCUT(IFACT,1,9)
  CALL SCAN(REPT,6,ECCEND)
  CALL FCUT(SPINT,OBEGL,TEND)
  IF ECCEND THEN
    CALL ERROR(2,REPT,6,1,0,0)
    // ERRCR---MISSING REPEAT-SUPPLIED AT THE END OF THE STATEMENT. /
    TEND=TEND+1
  ELSE
    TEND=TEND+6
  ENDIF
  CALL FCUT(CEOTO,1,9)
  CALL FCUT(LABEL,LAB1,LAB1+4)
  CALL FPRT
  // GENERATE IF(.NCT.((CCNC)) GO TO LAB1 //
  // GENERATE THE EXIT TARGET---LAB3 CONTINUE. //
FOR I=1 TO 5 CC
  L=LAB3+I-1
CALL FCUT(ASP, TOBEG, TCEND)
CALL FPRNT
// GIVE ANY END OF SUBROUTINE MESSAGES AND RESET ANY VARIABLES //
// IF NECESSARY. //
STEND=TCEND
INDFLG=0
CURFLG=NEW
: TYPE.EQ.CCMT :
FBUF(I)=CEE
TOBEG=2
TCEND=IMAX
LOOP
IF SPIN(TOBEG).NE.BLANK THEN
FCR I=2 TC 6 CO
FBUF(I)=BLANK
REPEAT
CALL FCUT(ASP, TOBEG, IMAX)
CALL FPRNT
STREG=TCBEGIN
EXIT
ENDIF
TOBEG=TCBEGIN+1
IF TOBEG.GT.IMAX THEN
// A BLANK COMMENT, OUTPUT A BLANK LINE. //
CALL FCUT(ASP, FPTR, FMAX)
CALL FPRNT
STREG=SSSTRT
EXIT
ENDIF
REPEAT
STEND=IMAX
CURFLG=0
INDFLG=NEUR
: TYPE.EQ.E0J :
// CLOSE THE FILE. //
WRITE(FPTR,1300)
FORMAT(/35H END OF SPARKS PROGRAM.....)
STOP
ENCCASE
IF CURFLG.EQ.PULGTL THEN
  SSTRT=SSTRT-INC
  CALL SPRNT
  SSTRT=SSTRT+INC
  IF SSTRT.GT.SMAX THEN
    SSTRT=SSTRT-INC
  ENDFEND
  BLKNO=BLKNC+1
ENDFEND
IF CURFLG.EQ.CLG THEN
  CALL SPRNT
ENDFEND
IF INDFLG.EQ.FRWR THEN
  SSTRT=SSTRT+INC
  IF SSTRT.GT.SMAX THEN
    SSTRT=SSTRT-INC
  ENDFEND
  IF CURFLG.EQ.NEW THEN
    CALL SPRNT
  ENDFEND
  BLKNO=BLKNC+1
  NESTNO=NESTNC+1
  ENDFEND
  IF INDFLG.EQ.BKWR THEN
    SSTRT=SSTRT-INC
    IF CURFLG.EQ.NEW THEN
      CALL SPRNT
    ENDFEND
    NESTNC=NESTNC-1
    BLKNC=BLKNC+1
    ENDFEND
  STMNC=STMNC+1
  REPEAT
  END
-1 202 751  BLOCK DATA
0 203 752     IMPLICIT INTEGER(A-Z)
0 203 753     LOGICAL LBFLG
0 203 754     CMMC/INPUT/SM/80), TOBEG, TOEND, STBE, STEN, IMAX, LBFLG
0 203 755     CMMC/SPRINT/SSRT, INCR, STMIO, BKNC, NESTNO, INDFLG, CURFLG, SMAX
0 203 756     CMMC/FRT INF/F3L(F30), FPTR, FMAX, FSSRT, BLNKS(131)
0 203 757     CMMC/CENTRL/STACK(20)), LABEL(200), TCP, LBMXX, LBPTR, STFRM(6),
1 203 758     TPTR, NEUTR, BKWR, FRWRD, NSW, CLD, Perala, SMAX, NUMP
0 203 759     CMMC/OPTUN/CGOTO(5), CONTI(8), EQL(1), GOTO(6), IFNOT(9), LEQ(5),
1 203 760     MINUS(1), ONE(1), PULC(1), RSTAR(2)
0 203 761     CMMC/INPUT/CCLN(1), ELZ(4), DLSL(2), DCC(2), BYE(2), THENN(4), TDO(2)
0 203 762     CMMC/STYP1/CCASE, COLON, DBLSL, EELSE, ENO, EOJ, EEJT, FFOR, IIF,
0 203 763     RREPT, UNTL, WHILE, SLFBN, RINLC, NCYCLE, ENDCA, EENDIF
0 203 764     CMMC/STYP2/CCMT, CCNT, FOTRN, BLANK, SEMI
0 203 765     CMMC/ТАблЕS/ТRS(23), RESE(22))
0 203 766     CMMC/NMNIC/CCE, PPTR, FFILE, CRDR
0 203 767     CMMC/NKEY/JEEEND(3), EEEJ(4), CYCLE(5), UNTL(5), CASS(4), IIIF(2),
1 203 768     ITERAT(5), ENCASS(7), ENIF(5), REPT(6), CCCLN(5)
0 203 769     // //
0 203 770     // CRITICAL CCODES--STFRM IS ACCESSED USING THE FOLLOWING: //
0 203 771     // CCASE=1, FFOR=2, IIF=3, LLOC=4, WHILE=5, SUBFN=6. //
0 203 772     // //
0 203 773     DATA SPIN/1HT, 1HH, 1HI, 1HS, 1H, 1HI, 1HS, 1H, 1HA, 1HN, 1HP, 1HA,
0 203 774     X 1HR, 1HK, 1HS, 1H, 1HP, 1HR, 1HG, 1HR, 1HA, 1HM,
0 203 775     1 1H, 1H, 1H, 1H, 1HH, 1HH, 1HI, 1HI, 1H, 1H, 1H, 1H, 1H, 1H,
0 203 776     2 1H, 1H, 1H, 1H, 1H, 1H, 1H, 1H, 1H, 1H, 1H, 1H, 1H, 1H,
0 203 777     3 1H, 1H, 1H, 1H, 1H, 1H, 1H, 1H, 1H, 1H, 1H, 1H, 1H, 1H,
0 203 778     4 1H, 1H, 1H, 1H, 1H, 1H, 1H, 1H, 1H, 1H, 1H, 1H, 1H, 1H,
0 203 779     DATA LNNKS/131*1H/
0 203 780     DATA STMAX, LBMXX, NMB/200, 200, 1GG000/
0 203 781     DATA NEUTR, BKWRD, FRWRD, NEW, CLD, PULOUT/1, 2, 3, 1, 2, 3/
0 203 782     // STACK FRAME SIZES:
0 203 783     // CASE FOR LOOP SUBFN WHILE //
0 203 784     DATA STFRM/3, 6, 2, 5, 1, 5/
0 203 785     DATA CGOTO/1H, 1H, 1H, 1H, 1H, 1H, 1H, 1H, 1H, 1H, 1H, 1H/
0 203 786     DATA CONTI/1HC, 1HC, 1HN, 1HT, 1FI, 1HN, 1HU, 1HE/
0 203 787     DATA EQL/1H=
DATA GOTC /1H,1HC,1H,1HT,1FC,1H /
DATA IFNOT/1HI,1HF,1H,1H,1HN,1HO,1HT,1H,1HF/ 
DATA LABEL/1H,1H,1H,1H,1H,1H,1F,1F,1F,195*1H /
DATA MINUS/1H-/
DATA CNE/1HI/
DATA PLU/1H+
DATA RSTAP/1HI,1H+/
DATA DBLSL,ELZ/1F,1H,1HE,1HL,1FS,1HE/
DATA COLN,COLON/1F:,1HC,1HO,1HL,1HC,1HN/
DATA CDE,DYE,THEAN,REPT,TCC/1HC,1HC,1HC,1F,1HY,1HT,1HN,1HE,1H,1HO,
DATA 1HE,1HF,1HI,1HC,1HE,1H,1H,1HO/
DATA CASE,CCLCN,CBSL,EELSE,SEND,EGJ,EEXIT,FF3R,COMT,
DATA CCNT,FOTR,F1F,LLCCP,REPT,UNTIL,WHILE,SUBFN,RINLC
DATA NCYCLE,ENCAS,ENCIF
DATA 1/1,7,8,9,10,1,12,2,13,4,14,15,3,4,6,16,17,6,5,18,19,20,21/
DATA RESR/4,1HC,1HA,1HS,1HE,1H,170,
DATA 3,.1HE,1HN,1HQ,14,4,1HE,1HL,1HS,1HE,9,21,
DATA 2,.1HE,1HO,1HJ,11,27,4,1HE,1HX,1FI,1HT,12,152,
DATA 3,.1HF,1HC,1HR,2,133,
DATA 4,.1HI,1HF,3,45,7,1FI,1HN,1HT,1HE,1HG,1HE,1HR,18,9,
DATA 5,.1HL,1HC,1HG,1HI,1HC,1FA,1HL,13,65,4,1HL,1HC,1FO,1HP,4
DATA 6,.10,1HR,1HE,1HA,1FL,18,79,6,1HR,1HE,1HP,1HE,1HA,1HT,
DATA 7,.16,0,5,1HU,1HN,1HT,1FI,1HL,17,0,5,1FW,1HH,1HI,1FL,1HE,6
DATA 8,.1C,4,1HN,1HE,1HX,1HT,0,0,10,1HS,1HU,1HB,1HR,1HC,1HU,1HT
DATA 9,.1HI,1HN,1HE,5,0,1FI,1F,1F,9,0
DATA A,.1HI,7,3,8,1HF,1FL,1HH,1HC,1HT,1HI,1HO,1HN,5,0,
DATA B,.5,1HE,1HL,1HO,1HC,1HK,5,0,7,1HE,1HN,1HC,1HA,1HS,1HE
DATA C,.20,162,5,1HE,1HN,1HC,1HI,1HF,21,0,5,1HC,1HC,1HE,1LC,1HE
DATA D,.19,3,23*0/
DATA // 178TH PLACE AVAILABLE FOR INSERTION. //
DATA // TRES CONTAINS POINTS INTO RESRD WHICH CONTAINS ALL THE //
DATA // RESERVE WORDS IN SPARKS. ***** NOTE ***** THIS ORDER IS //
DATA // PRESENTLY SET UP FOR EBCDIC. //
DATA FRTR,CPCR,FFILE/6,5,4/
DATA BLANK,CCE,SEMI/1H,1HC,1H;
SUBROUTINE DETAX(TYPE)
  // //
  // THE SUBROUTINE DETERMINES THE TYPE OF CYCLE STATEMENT AND //
  // SETS TOBEGIN AND TOEND ON TOKEN BOUNDARIES. //
  //
  IMPLICIT INTEGER(A-Z)
  LOGICAL LBFLG,ECCNC
  COMMON/INPUT/SPIN(80),TOBEGIN,TOEND,STBEGIN,STEND,IMAX,LBFLG
  COMMON/FRTINF/FBUF(80),FPTR,FMAX,FSTRT,BLANKS(131)
  COMMON/STYP2/COMT,CCNT,FCTRL,BLANK,SEMI
  COMMON/NMNIC/CEE,PRTR,FILENAME,CRDR
  DATA EFF/1HF/
  // //
  LBFLG=.FALSE.
  LCCP
    LCCP
      TOBEGIN=TOEND+1
      IF TOEND.GT.IMAX THEN
        EXIT
      ENDIF
      IF SPIN(TOEND).NE.BLANK THEN
        IF SPIN(TOEND).EQ.SEVERAL THEN
          CYCLE
        ENDIF
      ENDIF
      IF SPIN(1).EQ.CEE THEN
        TCBEGIN=1
        TCEND=1
        TYPE=CCMT
        RETURN
      ENDIF
      TCBEG=TCEND
      CALL SCAN1(BLANK,ECEND)
      CASE
        : ECEND :
          TYPE=FCTRL
          RETURN
      CASE
        : SPIN(6).NE.BLANK :

TYPE=CCNT
RETURN
: TCEND.GT.6 :
CALL STYYP(TYPE)
IF .NOT.LBFLG THEN
FCR I=1 TO 6 CO
BUFF(I)=BLNKS(I)
REPEAT
ENDIF
RETURN
: TCEND.LE.6 :
FOR I=1 TO 5 CO
BLNKS(I)=SPIN(I)
BUFF(I)=SPIN(I)
REPEAT
LBFLG=.TRUE.
BUFF(6)=BLANK
ENCASE
ENDIF
REPEAT
READ(CRDR,20) SPIN
FORMAT(80A1)
IF SPIN(I).EQ.EFF THEN
WRITE(PRTR,31)(SPIN(I),I=1,80)
FORMAT(34H -----BEGINNING OF FORTTRAN----- ,80A1)
READ(CRDR,20) SPIN
IF SPIN(I).NE.EFF CC
WRITE(PRTR,33) (SPIN(I),I=1,80)
FORMAT(20X,80A1)
WRITE(FFILE,32)(SPIN(I),I=1,80)
FORMAT(80A1)
READ(CRDR,20) SPIN
REPEAT
WRITE(PRTR,34) (SPIN(I),I=1,80)
FORMAT(28H -----ENC OF FORTRAN----- ,80A1)
READ(CRDR,20) SPIN
ENDIF
TCEND=0
SUBROUTINE STYP(TYPE)
  // THE TOKEN AT SPIN(TCBEG) TO SPIN(TCEND) IS SEARCHED //
  // FOR IN THE SYMGL TABLE. //
  // //
  IMPLICIT INTEGER(A-Z)
COMMON/INPUT/SPIN(80),TCBEG,TCEND,STBEG,STEND,IMAX,LBFLG
COMMON/STYP2/COMT,CCNT,FOTRN,BLANK,SEMI
COMMON/TABLES/TRES(23),RESRD(200)
  // //
  TCP=1
BCTTOM=23
LENGTH=TCEND-TCBEG+1
  WHILE TOP.LE.BCTTOM DO
    MID=(TOP+BCTTOM)/2
    I=TRES(MID)
    CASE
      : SPIN(TCBEG).LT.RESRC(I+1):
        BCTTOM=MID-1
      : SPIN(TCBEG).GT.RESRD(I+1):
        TCP=MID+1
      : SPIN(TCBEG).EQ.RESRC(I+1):
        IF LENGTH.EQ.1 THEN
          TYPE=RESRC(I+2)
          RETURN
        ENDDIF
      PNT=2
      WHILE PNT.LE.RESRD(I).AND.PNT.LE.LENGTH DO
        ITEMP=I+PNT
        JTEMP=TCBEG-1+PNT
        CASE
          : SPIN(JTEMP).LT.RESRC(ITEMP):
            BCTTOM=BCTTOM-1
          : SPIN(JTEMP).GT.RESRC(ITEMP):
            TOP=TOP+1
            PNT=5999
          : SPIN(JTEMP).EQ.RESRC(ITEMP):
4 247 946
4 247 947
3 248 948
2 249 949
3 250 950
4 251 951
4 252 952
4 252 953
4 252 954
4 252 955
4 253 956
4 253 957
4 254 958
4 254 959
3 255 960
2 256 961
1 257 962
1 258 963
0 258 964
0 258 965
FNT=PNT+1
ENDCASE
REPEAT
IF PNT.NE.9999 THEN
    CASE
        : LENGTH.EQ.RESRC(I) :
            JTEMP=LENGTH+1+I
            TYPE=RESRD(JTEMP)
            RETURN
        : LENGTH.GT.RESRC(I) :
            TOP=TOP+1
        : LENGTH.LT.RESRC(I) :
            BOTTM=BOTTM-1
    ENDCASE
ENDIF
ENDCASE
REPEAT
RETURN
END
SUBROUTINE SCAN1(CHAR,ECOND)

// LOOKS FOR CHAR AND IF FOUND, SPIN(TCEND+1)=CHAR //
// AND ECOND=FALSE. IF NOT FOUND UPTO IMAX OR SEMICOLON //
// ECOND=TRUE. //

//

IMPLICIT INTEGER(A-Z)
LOGICAL ECOND

COMMON/INPUT/SPIN(EG),TOBEG,TOEND,STBEG,STEND,IMAX,LBFLG
COMMON/STYP2/COMT,CONT,FOTRN,BLANK,SEMI

//

ECOND = .FALSE.

WHILE TCEND.LT.IMAX DO

ITEM=SPIN(TOEND+1)
KTEMP=SPIN(TOEND)

IF ITEM.EQ.SEMI.AND.KTEMP.EQ.BLANK THEN
   ECOND = .TRUE.
   RETURN
ENDIF

IF CHAR.NE.SEMI THEN
   IF ITEM.EQ.CHAR THEN
      RETURN
   ENDIF

ENDIF

TCEND=TCEND+1
REPEAT

ECOND = .TRUE.
RETURN

END
SUBROUTINE SCAN(PAT,LEN,ECOND)
   // THE SUBROUTINE COMPARES THE PATTERN PAT WITH ALL TOKENS //
   // SURROUNDED BY BLANKS FROM SPIN(TCEND+1) ONWARDS AND //
   // RETURNS WITH TCENC AT THE OLD POSITION OF TCEND; WITH THE //
   // PATTERN STARTING AT TCEND+1. IF NOT FOUND ECOND IS SET TRUE //
   // //
   // IMPLICIT INTEGER(A-Z)
   LOGICAL ECOND
   COMMON/INPUT/SPIN(80),TOBEG,TOEND,STBEG,STEND,IMAX,LBFLG
   COMMON/STYP2/COMT,CCNT,FCTRA,BLANK,SEMI
   DIMENSION PAT(6)
   // //
   TOBEG=TOEND+1
   LOOP
      K=TCEND+1
      WHILE K.LE.IMAX.AND.SPIN(K).EQ.BLANK DO
         K=K+1
      REPEAT
      TOEND=K
      CALL SCAN1(BLANK,ECOND)
      IF ECOND THEN
         RETURN
      ENDIF
      IF LEN.NE.TCEND-K+1 THEN
         CYCLE
      ENDIF
      // COMPARE THE PATTERN WITH A TOKEN OF THE SAME LENGTH. //
      FOR I=1 TO LEN DO
         J=K+1-I
         IF PAT(I).NE.SPIN(J) THEN
            EXIT
         ENDIF
      ENDIF
      IF I.EQ.LEN THEN
         // PATTERN FOUND, RETURN WITH ECOND FALSE. //
         TCEND=TCEND-LEN
         RETURN
      ENDIF
      ENDIF
SUBROUTINE SPRNT

IMPLICIT INTEGER(A-Z)
LOGICAL LBFLG,CVFL
COMMON/INPUT/SPN(80),TOEG,TOEND,STBE,G,STEND,IMAX,LBFLG
COMMON/STYP2/CMT,CCNT,FCTR,BLANK,SEMI
COMMON/SPNINF/SSTR,INCR,STMNO,BLKNO,NESTNO,INDFLG,CURFLG,SMAX
COMMON/FRTINF/BUF(80),FPTR,FMAX,FSTR,BLNKS(131)
COMMON/ANMIC/CEE,PRTR,FFILE,CRDR
COMMON/TYP/TYPE

IF TYPE.EQ.CCNO THEN
  BLNKS(6)=SPN(6)
ENDIF

LCCP
CVFL = .FALSE.
LEN1=STEND-STBE+1
LEN2=SMAX-SSTR+1
SEND=STEND
IF LEN1.GT.LEN2 THEN
  CVFL=.TRUE.
  SEND=STBE+LEN2-1
ENDIF
FILE=LEN2-LEN1+7
WRITE(PRTR,10) NESTNO,BLKNO,STMNO,(BLNKS(I),I=1,5),SSTR,
      (SPN(I),I=STBE,SEND),(BLNKS(I),I=7,FILE),(SPN(I),I=73
      1,80)
  FORMAT(1H,15,15,16,4X,111A1)
  FCR I=1 TC 6 DD
  BLNKS(I)=BLANK
  REPEAT
  STBE=SEND+1
  UNTIL .NOT.CVFL REPEAT
RETUR
CALL SCAN1 (SEMI, ECONC)
IF TYPE .EQ. CONT THEN
FBUF (6) = SPIN (6)
ENDIF
CALL FOUT (SPIN, TCEG, TCEND)
IF EEND THEN
EXIT
ENDIF
TCEND = TCEND + 1
IF SPIN (TCEND + 1) .NE. SEMI THEN
EXIT
ENDIF
TCEG = TCEG + 1
TCEND = TOBEG
REPEAT
CALL FPRNT
RETURN
END
SUBROUTINE FOUT(PAT, STRT, ENC)

// THE VALUE OF FSTART WILL USUALLY BE 7 AND FMAX WILL BE 72. //
// FPTR SHOULD BE AT FSTART. BLANKL IS A FLAG TO INDICATE A //
// TO INDICATE A SC FOR BLANK LINE. //

//
//
IMPLICIT INTEGER(A-Z)
LOGICAL LBFLG, BLANKL
COMMON/CESTR/STACK(200), LABEL(200), TCP, LMAX, LBPTR, STFRM(6),
    TPTR, NEUTR, BKWRC, FRWRD, NEW, CLD, PULCUT, STMAX, NUMB
COMMON/FRTINF/FBLF(80), FPTR, FMAX, FSTART, BLNKS(131)
COMMON/STYP2/COMT, CCNT, FCTR, BLANK, SEMI
COMMON/NMNIC/CEE, PRTR, FFILE, CDR
COMMON/TYP/TYPE
COMMON/FORPRT/BLANKL
DIMENSION PAT(1)
DATA NINE/1+9/

//
// BLANKL = .FALSE.

FOR J=STRT TO ENC DO
  IF FPTR.GT.FMAX THEN
    CALL FPRNT
  ENDIF
  IF FILE.EQ.SEMI THEN
    FBUF(I)=CEE
  ENDIF
  JJ=J
  WHILE PAT(JJ).EQ.BLANK DO
    IF JJ.EQ.ENC THEN
      BLANKL=.TRUE.
      RETURN
    ENDIF
    JJ=JJ+1
  ENDW
ENDFOR
ENDSUB
SUBROUTINE FPRTN
// THE SUBROUTINE PRINTS THE TRANSLATED FORTRAN //
IMPLICIT INTEGER(A-Z)
LOGICAL BLNKLK
COMMON/INPUT/SPK(E0),TUBE,G,TOEND,STBEG,STEND,TMAX,LBFLG
COMMON/AMN/CEE,PRTR,FFILE,CRDR
COMMON/PRINF/SSTRT,INCR,STMNO,BLKNGL,NESTNO,INDFLG,CURFLG,SMAX
COMMON/FRTINF/FBLF(E0),FPTR,FMAX,FSTRT,BLNKS(131)
COMMON/FCRPRT/BLNKLK
IF FPTR.EQ.FSTRT THEN
  RETURN
ENCIF
IFF=FSTRT-1
IF IFF.GE.7 THEN
  FOR I=7 TO IFF DO
    FBUF(I)=BLNKS(I)
  REPEAT
ENCIF
IF FPTR.LE.FMAX THEN
  FOR I=FPTR TO FMAX DO
    FBUF(I)=BLNKS(I)
  REPEAT
ENCIF
FOR I=73 TO 75 DO
  FBUF(I)=SPN(I)
REPEAT
IF .NOT.ELNKLK THEN
  WRITE(FFUL,10)(FBUF(I),I=1,75),STMNO
  FORMAT(75A1,15)
ENCIF
FPTR=FSTRT
RETURN
END
INTEGER FUNCTION ISNUM(C)

// THE FUNCTION CHECKS TO SEE IF C IS THE ALPHANUMERIC
// REPRESENTATION OF A NUMBER AND RETURNS 1 IF IT IS AND 0 IF
// IT IS NOT. //

IMPLICIT INTEGER(A-Z)
DIMENSION NUMBER(10)

DATA NUMBER/1H0,1H1,1H2,1H3,1H4,1H5,1F6,1H7,1H8,1H9/

// //

FCR I=1 TO 10 DO
   IF Q.EQ.NUMBER(I) THEN
      ISNUM=1
      RETURN
   ENDIF
REPEAT
   ISNUM=0
END
INTEGER FUNCTION CONV(DIGIT)

    // CONVERTS THE DIGIT INTO ITS CHARACTER REPRESENTATION. //
    //
    IMPLICIT INTEGER(A-Z)

    DIMENSION CVNUM(10)

    DATA CVNUM/1H0,1H1,1H2,1H3,1H4,1H5,1H6,1H7,1H8,1H9/

    CCNV=CVNUM(DIGIT+1)

    RETURN

END
INTEGER FUNCTION GEALAB(GEN)

// GENERATE A LABEL UNCONDITIONALLY IF GEN IS 1 ELSE //
// GENERATE THE LABEL IF LBFLG=0, IF LABEL FLAG //
// EQUAL 1 AND GEN EQUAL 0 THEN USE THE EXISTING //
// LABEL. Numb IS INITIALIZED TO 100000. THIS IS AN //
// INTEGER FUNCTION RETURNING A POINTER TO THE LABEL. //
// //
IMPLIED INTEGER(A-Z)
LOGICAL LBFLG
COMMON/INPLT/SPIN(80),TCBEG,TCEND,STBEG,STEND,IMAX,LBFLG
COMMON/CENTRL/STACK(200),LABEL(200),TCP,LBMAX,LBPTR,STFRM(6),
TPTR,NEUTR,GBKRC,FRWRT,REW,CLD,PULCUT,STMAX,NUMB
//
//
IF .NOT. LBFLG .AND. GEN.EQ.1 THEN
  Numb=Numb-1
  N=Numb
FOR I=1 TO 5 CJ
  M=N/10
  // SHIFT RIGHT //
  NR=N-(10*M)
  // REMAINDER //
  HCLD=5+I+LBPTR
  LABEL(HCLD)=CCNV(NR)
  // LBPTR IS LEFT AT FIRST BLANK SPACE. //
  N=N
REPEAT
  GENLAB=LBPTR
  // NEW LABEL PTR. //</
  LBPTR=LBPTR+5
  // POINT TO NEW BLANK LOCATION. //</
  IF LBPTR.GT.LBMAX THEN
    CALL ERROR(8,0,0,5,0,0)
    // NESTING TOO DEEP, TOO MANY LABELS PROGRAM TERMINATED. //</
  ENDIF
ELSE
  FOR I=1 TO 5 CO
    HCLD=LBPTR+I-1

A-39
LABEL(FCLD)=SPIN(I)

REPEAT

GENLAB=LBPTR

LBPTR=LBPTR+5

IF LBPTR.GT.LBMAX THEN

CALL ERROR(8,0,0,5,0,0)

ENDIF

// NESTING TOO DEEP, TOO MANY LABELS PROGRAM TERMINATED. //

ENDIF

ENDIF

RETURN

END
SUBROUTINE CHECK(STYPE, ECOND)

IMPLICIT INTEGER(A-Z)
LOGICAL ECCND, CUMM, LBFGLG
COMMON/IPUT/SPIN(80), TOBEG, TOEND, STREG, STEND, IMAX, LBFGLG
CCMCR/FTINF/FBUF(80), FPTR, FMAX, FSTRT, BLNKS(130)
CCMCN/SPRINF/SSTRT, INCR, STMNO, BLKNC, RSTNC, INDFLG, CURFLG, SMAX
CCMCR/OMPAT/CGOTO(9), CONTI(8), EQL(1), GOPTO(6), IFNOT(9), LEOT(5),
MINUS(1), ONE(1), FLU(1), RSTAR(2)
COMMON/STYP1/CCASE, COLON, DBCSL, ELSE, EEND, EOJ, EEEX, FFTOR, IIF,
1 LLCCP, RREPT, UNTL, WHILE, SUBFN, RINLG, NCYCLE, ENDGAGE, EENDIF
CCMCR/CCNTRL/STACK(200), LABEL(200), TCP, LBMAX, LBPTR, SFRM(6),
1 TPITR, NEUTR, AKWRD, FRWRD, NEW, CLN, PULCS, SMAX, NMB
CCMCN/INP/IACLON(1), ELZ(4), CHSLF(2), CDO(2), BYE(2), THENN(4), TOG(2)
CCMCR/TABLES/TRES(23), RESRD(200)
CCMCR/KEYW/EEEND(3), EEEX(4), CCYCLE(5), UNTL(5), CASO(4), IIF(2),
1 ITERAT(9), ENSASS(7), ENSIF(5), REPT(6), CCCLCN(5)
//
ECCND=.FALSE.
IF TOP.EQ.C THEN
1 359 1267
1 355 1268
1 359 1269
1 359 1270
1 359 1271
RETURN
ENDIF
0 360 1272
SSTOP=STACK(TCP)
0 360 1273
CASE
1 361 1274
1 362 1275
IF SSTOP .NE.CCASE THEN
2 363 1276
2 363 1277
2 363 1278
2 363 1279
ENDIF
1 364 1280
: STYPE.EQ.CCASE :
1 365 1281
1 365 1282
IF SSTOP .NE.CCASE THEN
2 366 1283
2 366 1284
// ERRRR----MISSING CASE STATEMENT, COLON IGNORED. //
}
ECOND=.TRUE.
ENDIF

: STYPE.EQ.UNTL :
  IF STTCP.NE.LLOCPT THEN
    CALL ERRCR(5,ITERAT,9,4,UUNTL,5)
    // ERROR---MISSING LOOP STATEMENT, UNTIL IGNORED. //
    CALL SCAN(REPT,3,CUMP)
    IF JUMP THEN
      TCEND=TCEND+1
    ELSE
      TCEND=TCEND+6
    ENDIF
    ECOND=.TRUE.
  ENDIF

: STYPE.EQ.EEND :
  IF STTOP.NE.SUBFN THEN
    // UNSTACK UNTIL A SUBFN IS FOUND ON THE STACK. //
    TTCP=TCP
    LOOP =
      TTOP=TTCP-STFRM(STTCP)
      IF TTOP.LE.0 THEN
        CALL ERRCR(3,EEND,3,4,EEND,3)
        // ERRCR---TOO MANY ENDS, END IGNORED. //
        ECOND=.TRUE.
        RETURN
      ENDIF
      STTCP=STACK(TTOP)
    UNTIL STTOP.EQ.SUBFN REPEAT
  ENDIF

: STYPE.EQ.ENDCAS :
  IF STTOP.NE.GCASE THEN
    CALL ERRCR(3,ENCASS,7,4,ENCASS,7)
    // ERRCR---TCC MANY ENDCASES, ENDCASE IGNORED. //
ECCND=.TRUE.
ENDIF

: STYPE.EQ.EENDIF :
IF STTOP.NE.1IF THEN
CALL ERROR(3,ENIF,5,4,ENIF,5)
// ERRCR---TCC MANY ENIF'S, ENDF IGORRED. //
ECCND=.TRUE.
ENDIF

: STYPE.EQ.REPT :
IF STTOP.EQ.FFOR.CR.STTCP.EQ.LLOOP.CR.STTCP.EQ.WHLE THEN
RETURN
ELSE
CALL ERROR(5,ITERAT,9,4,REPT,6)
// ERRCR---NC ITERATIVE STATEMENT FOUND, REPEAT IGORRED. //
ECOND=.TRUE.
ENDIF

: STYPE.EQ.EXIT :
IF TPITR.EQ.0 THEN
CALL ERPCR(5,ITERAT,9,4,EEEXT,4)
// ERRCR---NC ITERATIVE STATEMENT, EXIT IGORRED. //
ECOND=.TRUE.
ENDIF

: STYPE.EQ.NCYCLE :
IF TPITR.EQ.0 THEN
CALL ERRCR(5,ITERAT,9,4,CCYCLE,5)
// ERRCR---NC ITERATIVE STATEMENT, CYCLE IGORRED. //
ECCND=.TRUE.
ENDIF
ENCASE
RETURN
END
SUBROUTINE ERRCR(CODE1, PARM1, LEN1, CODE2, PARM2, LEN2)

    // CODE1 IS ERRCR MESSAGE CODE //
    // CODE2 IS THE ACTION TAKEN //
    // PARM1 AND PARM2 MODIFY THE TWO //
    // LEN1 AND LEN2 ARE THE LENGTH OF MODIFYING PATTERN //

    IMPLICIT INTEGER(A-Z)

    COMMON/INPUT/SPIN(80), TCBEG, TCEND, STBEG, STEND, IMAX, LBFLG
    COMMON/STYP2/COMT, CONT, FOTRN, BLANK, SEMI
    COMMON/NNMC/CEE, PRTR, PFILE, CDR

    DIMENSION PARM1(1), PARM2(1), ERPAT(45, 6), POSN(8), LEN(8),
    1 ACPAT(35, 6), ALEN(6), APSN(6)

---BEGINNING OF FORTRAN---
1      1H, 1H, 1H, 1H, 1H, 1H, 1H, 1H, 1H, 1H, 1H
2      1H, 1HA, 1H3, 1HE, 1HL, 1F, 1HD, 1HE, 1HL, 1HE, 1HT, 1HE, 1HO,
3      1HF, 1FR, 1HC, 1HM, 1F, 1FF, 1HC, 1HR, 1HT, 1HR, 1FA, 1HM, 1H
4      1HO, 1HU, 1HT, 1HP, 1HU, 1FT, 1H, 1H
5      1HS, 1HT, 1HA, 1HT, 1FE, 1FM, 1HE, 1HN, 1HT, 1H
6      1HS, 1HK, 1HI, 1HP, 1HP, 1FE, 1HO, 1H, 1H, 1H, 1H, 1H
7      1H, 1F, 1H, 1H, 1H, 1F, 1H, 1H, 1H, 1H, 1H
8      1H, 1H, 1H, 1F, 1H, 1H, 1F, 1H, 1H, 1H, 1H, 1H
9      1H, 1H, 1H, 1F, 1H, 1H, 1F, 1H, 1H, 1H, 1H, 1H
10     1H, 1H, 1H, 1F, 1F
11     1HP, 1HR, 1HC, 1HG, 1HR, 1FA, 1HM, 1H, 1HT, 1HE, 1HR, 1HM, 1HI, 1HN, 1HA,
12     1HT, 1HE, 1HD, 17*1H
13     1HS, 1HU, 1HB, 1HP, 1HR, 1FC, 1HG, 1HR, 1HA, 1HM, 1H, 1HC, 1HL, 1HO, 1HS
14     1HE, 1HD, 18*1H

-----END OF FORTRAN-----

0  399  1366  2222
   //   //
   WRITE(PRTR,2222) SPIN
   FORMAT(26H ******CFFENCING CARD: ,80A1)
0  395  1369
0  399  1370
0  399  1371
0  399  1372
0  399  1373
0  399  1374
0  399  1375
0  399  1376
0  400  1377
0  400  1378
0  400  1379
0  401  1380
0  401  1381
0  402  1382
0  403  1383
0  404  1384
0  405  1385
0  406  1386

FOR I=1 TO LEN1 DO
   IT=POS+I-1
   ERPAT(IT,CCDE1)=PARM1(I)
END
FOR I=1 TO LEN2 DO
   IT=APCS+I-1
   ACMPAT(IT,CCDE2)=PARM2(I)
   REPEAT
   IF LENTH.GT.LEN1 THEN
      PO=POS+LEN1
      PE=POS+LENTH-1
      FOR I=PO TO PE CC
         ERPAT(I,CCDE1)=BLANK
         REPEAT
      ENDIF
      FOR I=1 TO LEN2 DO
         IT=APCS+I-1
         ACMPAT(IT,CCDE2)=PARM2(I)
   END
   REPEAT
END

1 406 1387  REPEAT
0 407 1388  IF ALENTH.GT.LEN2 THEN
1 408 1389  PO=APCS+LEN2
1 408 1390  PE=APCS+ALNTH-1
1 408 1391  FOR I=PO TO PE DO CC
2 409 1392    ACPAT(I, CODE2)=BLANK
2 409 1393    REPEAT
1 410 1394  ENDIF
0 411 1395  WRITE(PRTR,10) (ERPAT(I, CCCE1), I=1, 45)
0 411 1396  WRITE(PRTR,10) (ACPAT(I, CODE2), I=1, 35)
0 411 1397  10 FFORMAT(12H ++++++++, 80A1)
0 411 1398  RETURN
0 411 1399  END
SUBROUTINE TYHAND(TYPE, EEXIT, CCYCLE, EEXIT)
    IMPLICIT INTEGER(A-Z)
    LOGICAL LBFLG
    COMMON/INPUT/SPIN(80), TCBEQ, TOEND, STBEG, STEND, IMAX, LBFLG
    COMMON/SPINF/SSTRT, INCR, STMN, RLKNC, RSTNG, INDNLG, CURFLG, SMAX
    COMMON/FATINF/FRUE(80), FPTR, FMAX, FSTRT, BLNKS(131)
    COMMON/OPTPAT/GOTO(5), CONT(8), ECL(1), GOTO(6), IFNOT(9), LED(5),
    COMMON/CENRCL/STACK(200), LABEL(200), TCP, LBM, LBPTR, STFRM(6),
    COMMON/NEUTR, BKREC, FRWRD, NEW, CLD, PULCUT, STMAX, NUMB
    COMMON/STYPER/CONT, CCNT, FOTRN, BLANK, SEMI
    /1
    IF LBFLG THEN
        IF TYPE.EQ.EEXIT THEN
            CALL ERRCR(1, EEXIT, 4, 2, 0, 0)
            // ERROR--LABELLING THE EXIT HAMPERS PROGRAM CLEANLINESS-- //
            // --DELETED FROM THE FORTRAN OUTPUT. //
        ELSE
            CALL ERRCR(1, CCYCLE, 5, 2, 0, 0)
            // ERROR--LABELLING THE CYCLE HAMPERS PROGRAM //
            // CLEANLINESS, DELETED FROM FORTRAN OUTPUT. //
        ENDIF
    ENDIF
    // CHECK IF A NUMERIC LABEL FOLLOWS. //
    K = TCEND + 1
    LCCP
    IF K.GT.IMAX THEN
        I = 0
        EXIT
    ENDIF
    IF SPIN(K).NE.BLANK THEN
        I = ISNUM(SPIN(K))
        EXIT
    ENClF
    K = K + 1
    REPEAT
    IF I.EQ.J THEN
FOR I=1 TO 6 DO
  FBUF(I)=BLANK
  REPEAT
    IF TYPE.EQ.EXIT THEN
      K=STACK(TPITR-1)
    ELSE
      K=STACK(TPITR-2)
    ENDIF
    CALL FCUT(GCTC,1,I)
    CALL FCUT(LABEL,K,K+4)
    CALL FPRNT
    ELSE
      TOEND=K
      CALL SCAK1(BLANK,ECLASS)
      WRITE(PRTR,4999)
      FORMAT('++++++LABELLED EXIT-CYCLE NOT IMPLEMENTED')
    ENDIF
END
APPENDIX B

FORTRAN TRANSLATION

OF THE

SPARKS PREPROCESSOR
C     // MAIN PROGRAM
C
0001   IMPLICIT INTEGER(A-Z)
0002   LOGICAL LBFLG,LCCNC,ECCNC,ECCNC1,BAC
0003   COMMON/IPRINT/SPIN(80),T0REG,TCEND,STREG,STEND,IMAX,LBFLG
0004   COMMON/SPRINTF/SSTRT,INCR,STMNO,BLKNO,ASTEKC,INDFLG,CURFLG,SMAX
0005   COMMON/FPRINTF/FBUF(80),FPRINTF,FMAX,FSTRT,BLKS(131)
0006   COMMON/CENTRL/STACK(200),LABEL(200),TCP,LEMAX,LBPTR,STERM(6),
          TPITP,NEUTR,BKWRD,FRWRL,NEK,OLC,PULOUT,STMAX,NUMR
0007   COMMON/CTPAT/CGUTC(9),CCNTI(8),ECL(1),GOTO(6),IF(NT(9),LEO(5),
          MINUS(1),ONE(1),PLU(1),RSTAR(2)
0008   COMMON/1K PAT/CCLN(1),ELZ(4),OBSLH(2),CDO(2),BYE(2),THENN(4),TOO

9  (2)
0009   COMMON/STTPY1/CCASE,COLCN,CBLSL,EELSE,EENC,EQJ,EEIXT,FFOR,IF,
          LLCCP,RPEPT,UNT1,WHIL,SUBFM,RCINL,NCYCLE,ENCASS,EENDIF
0010   COMMON/STTPY2/COMT,CCNT,FCTR,M,PLNK,SEM
0011   COMMON/NMNIC/CEE,PRTR,FFILE,CRDR
0012   COMMON/KEYWD/EEEND(3),EEEXT(4),CCCYCLE(5),UUATL(5),CASS(4),IIIF
          (2),
          ITERAT(5),ENCASS(7),ENIF(5),REPT(6),CCLCN(5)
0013   COMMON/TABLES/TRES(23),RESRD(200)
0014   COMMON/TYP/TYPE
C
0015   // WRITE TITLES ETC.
0016   WRITE(PRTR,3)SPIN(I),I=1,80)
0017   WRITE(PRTR,4)
0018   FORMAT('/32X,'SPARKS TRANSLATION BY KSU SPARKS PREPROCESSOR','
          'AS'/32X,'MODIFIED BY R. STROUD, M. CALHOUN, AND J. MARTIN.')
C
0019   IMAEC=72
0020   SMAX=100
0021   FMAX=72
0022   INCR=3
FSTRT=7+INCR
SSTRT=7+INCR

C // EXTRACT OPTIONS GIVEN ON THE CCNTRL CARD AND CHANGE ABOVE
C // DEFAULTS ACCORDINGLY---THIS COULD USE DETNX
C // INITIALISE THE PROGRAM RELATED VARIABLES.

STACK(1)=SUBFN
TOP=1
TPTR=0
LBPTR=6
TCEND=IMAX+1
BLKNC=0
STMNO=0
NESTNO=0

C // START MAIN LOOP
CONTINUE

FPTR=FSTRT
CALL DETNX(TYPE)

C // PROCESS STATEMENTS ACCORDING TO TYPE
C // 1. TRANSLATE AND PRINT FORTRAN.
C // 2. SET INDENTATION FLAG---INDFLG.
C // 3. SET CURFLG NEW-CLD--CURRENT STATEMENT TO BE PRINTED IN
C // CLD OR NEW INDENTATION.
C

IF(.NOT.(TYPE.EQ.RINLC)) GC TC 99996
A=TOBEG
B=TOEND
CALL DETNX(TYPE)
C
// IF THE SECOND TOKEN IS NOT FUNCTION THEN PASS AS FORTR
C $ AN.
IF(.NOT.(TYPE.NE.SUBFN)) GO TC 99995
TYPE=FOTRN

CCONTINUE
99995
99996
CONTINUE
STBEC=TOBEG

C
// THE MAIN CASE STATEMENT
IF(.NOT.(TYPE.EQ.FCTRN)) GC TC 99993
CALL FHAND
STEND=TOEND
CURFLG=OLD
INEFLG=NEUTR

GO TC 99994
99993
99992
IF(.NOT.(TYPE.EQ.DBSSL)) GC TC 99994
C
// LOOK FOR A DOUBLE SLASH--SCAN IS NOT USED SINCE IT STD
C $ PS
C
// ON A SEMICOLON.
CONTINUE
99991
99998
IF(.NOT.(TOEND.GE.IMAX-1)) GO TO 99988
CALL ERROR(2,DBSLH,2,1,0,0)
C
// ERROR--MISSING DOUBLE SLASH SUPPLIED AT END OF S
C $ STATEMENT.
GO TO 99989
99988
99989
CONTINUE
99988
99987
IF(.NOT.(SPIN(TOEND+1).EQ.DBSSLH(1)).AND.SPIN(TOEND+2).
EQ.DBSSLH(1))) GC TC 99587
GO TO 99989
99987
CONTINUE
TOEND=TOEND+1
GC TO 99991
CONTINUE
FEUF(1)=CEE
C // THE TYPE IS CHANGED SO THAT FCUT CAN CHECK IT AND PUT
C 9 A C IN
C // COLUMN ONE FOR LANGU COMMENTS.
C
CALL FCUT(SPIN,TCEG,TOEND)
TYPE=CMT
CALL FPRNT
INCFGL=NEUTR
CLRFLG=CLD
TEND=TEND+2
STEND=TEND
GO TC 99994
IF(.NOT. (TYPE.EQ.IIF)) GC TO 99986
LAB=GENLAB(1)
CALL SCANT(THENN,4,ECON
D)
IF (.NOT. (ECON)) GC TO 99985
C // ADSENSE OF THEN ASSUMED TO INDICATE FORTRAN IF.
TCEG=TCEG-2
CALL FHAND
GC TO 99984
CONTINUE

C
// CPUT=IF(11 NOT CCMD) GO TC LAB
CALL FOUT(IFNCT,1,9)
CALL FOUT(SP1N,T08EG,TCEND)
CALL FOUT(CGOTG,1,9)
CALL FOUT(LABEL,LAB,LAB+4)
CALL FPRNT
STACK(TCP+1)=LAB
STACK(TOP+2)=IIF
TCP=TCP+2
IFI(11 NOT ( TCP.G1 .STMAX )) GO TC 99983
CALL ERROR(6,0,0,5,0,0)
CONTINUE

C
// STACK OVERFLOWED PROGRAM TERMINATED.
TCEND=TCEND+4
IFI(11 NOT ( FSTR1+INCR.LT.IMAX )) GO TC 99982
FSTR1=FSTR1+INCR
CONTINUE
CONTINUE
CONTINUE
STEND=TOEND
CLRFLG=OLD
INDFLG=FRWDD
GO TC 99994
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINUE
CONTINU
0113    LAB=STACK(TCP-1)           141
0114    TCP=TOP-2                 142
0115    LBPTR=LBPTR-5            143
0116    I=1                      144
0117    GO TO 99978              144
0118    I= I+1                   144
0119    IF(.NOT.((I-16).LE.0)) GC TC 99975  144
0120    L=LAB+I-1                145
0121    FBUF(I)=LABEL(L)         146
0122    GO TO 99976              147
0123    CCNTINUE                 147
0124    FBUF(6)=BLANK            148
0125    CALL FOUT(CONTI,1,8)     149
0126    CALL FPRNT               150
0127    // GENERATE A LAB CONTINUE.
0128    STEND=TOEND              152
0129    INDFLG=KWRD              153
0129    CURFLG=NEW               154
0130    GO TC 99994              155
0131    IF(.NOT.(TYPE.EQ.EELSE)) GO TC 99974  155
0132    CALL CHECK(EELSE,BAD)    156
0133    IF(.NOT.(BAD)) GC TO 99973  157
GO TO 99998
CONTINUE
FSTRT=FSTRT-INC1
IF(.NOT.(LABLG EQ 1)) GO TO 99972
CALL ERROR(1,ELZ,4,2,0,0)
// ERROR--LABELLING THE ELSE HAMPER PROGRAM CLEANLINE
C 9
SS--
C
// DELETED FROM THE FORTRAN OUTPUT.
CONTINUE
LAB2=GENLAB(1)
LAB1=STACK(TCP-1)
I=1
GC TO 99971
I= I+1
IF(.NOT.*(I-(4)-LE.0)) GO TO 99968
FBUF(1)=BLANK
GO TO 99970
CONTINUE
CALL FCUT(GCTC,1,6)
CALL FOUT(LABEL,LAB2,LAB2+4)
CALL FPRINT
// GO TO LAB2
I=1
GC TO 99967
I= I+1
IF(.NOT.*(I-(4)-LE.0)) GO TO 99964
L=LAB1+I-1
FBUF(I)=LABEL(L)
GO TO 99966
CONTINUE
FBUF(6)=BLANK
CALL FCUT(CCNT1,1,8)
CALL FPRINT
// LABEL CONTINUE
STACK(TOP-1)=LAB2
STEND=TJEND
INDFLG=NEUTR
CURFLG=PULOUT
IF(.NCT.(FSTRT+INCR.LT.IMAX)) GO TO 99963
FSTRT=FSTRT+INCR
99963 CONTINUE
GO TO 99964
IF(.NCT.(TYPE.EQ.FF0R)) GG TO 99962
IF(.NCT.(LPFLG)) GC TO 99961
C // LAB WILL HAVE IDENTIFYING LABEL IF ANY.
LAB=GENLAB(0)
CALL FCUT(CONTI,1,8)
CALL FPRNT
99961 CONTINUE
TCBEG=TCEND+1
CALL SCANI(EQL,ECOND)
IF(.NCT.(ECOND)) GC TO 99960
CALL ERRER(2,EQL,1,3,0,0)
C // ERROR--MISSING EQUAL-STATEMENT SKIPPED.
GC TO 99998
99960 CONTINUE
VLEAE=TCBEG
V8LE8=TCEND
FORTRAN IV G LEVEL 21

0185  TCENC=TOENC+1
0186  CALL SCAN(TCC,2,ECCNC)
0187  IF(.NOT.( ECCND1 )) GC TO 99959
0188  CALL ERRCR(2,TOC,2,3,0,0)
        C // ERROR--MISSING 'TO' - STATEMENT SKIPPED.
0189  TOEND=TCEND+1
0190  GC TC 99958
0191  99959  CCONTINUE
0192  TOENC=TOEND+2
0193  CALL FOLT(SPIN,VELEA,TCEND-2)
0194  CALL FPRNT
        C // VARIABLE=EXPRESSION1.
0195  ITMP=TCEND
0196  CALL SCAN(BYE,2,ECCNC1)
0197  EX2A=TCBG
0198  EX2B=TCEND
0199  TCENC=TOEND+2
0200  IF(.NOT.( ECCND1 )) GO TO 99958
0201  TCEND=ITMP
0202  99958  CCONTINUE
0203  CALL SCAN(DCC,2,ECCNC)
0204  IF(.NOT.( ECCND1 )) GO TO 99957
0205  EX2B=TCENC
0206  99957  CCONTINUE
0207  IF(.NOT.( ECCND1 )) GC TO 99956
0208  CALL ERRCR(2,CCC,2,1,0,0)
        C // ERROR--MISSING 'DO'-- SUPPLIED AT END OF STATEMENT.
0209  TOEND=TCEND+1
0210  GC TO 99955
0211  99956  CCONTINUE
0212  TCEND=TCEND+2
0213  99955  CCONTINUE
0214  LAB1=GENLAB(1)
0215  LAB2= GENLAB(1)
0216  LAB3=GENLAB(1)
0217  LAB4=GENLAB(1)
C // LAB IS THE IDENTIFYING LABEL, LAB4 IS THE TARGET OF EX
C IT,
C // AND LAB3 IS THE TARGET OF CYCLES.
0218 STACK(TOP+1)=LAB2
0219 STACK(TOP+3)=LAB
0220 STACK(TOP+4)=LAB3
0221 STACK(TOP+5)=LAB4
0222 STACK(TOP+6)=FFOR
0223 STACK(TOP+2)=TPITR
0224 TCP=TOP+6
0225 IF(.NOT.(TOP.GT.STMAX)) GO TO 99954
0226 CALL ERRCR(6,0,5,0,0)
0227 99954 CCONTINUE
C // STACK OVERFLOWED-----PROGRAM TERMINATED.
0228 TPITR=TCP
0229 I=1
0230 GC TO 99953
0231 99952 I= I+1
0232 99953 IF(.NOT.((I-(6)).LE.0)) GC TC 99950
0233 FBUF(I)=BLANK
0234 99951 GC TC 99952
0235 99950 CONTINUE
<table>
<thead>
<tr>
<th>Line</th>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0236</td>
<td>CALL FCUT(GCTC,1,6)</td>
<td>Call FCUT subroutine with arguments GCTC, 1, and 6.</td>
</tr>
<tr>
<td>0237</td>
<td>CALL FCUT(LABEL,LAB1,LAB1+4)</td>
<td>Call FCUT subroutine with arguments LABEL, LAB1, and LAB1+4.</td>
</tr>
<tr>
<td>0238</td>
<td>CALL FPRTN</td>
<td>Call FPRTN subroutine.</td>
</tr>
<tr>
<td>0240</td>
<td>I=1</td>
<td>Initialize variable I to 1.</td>
</tr>
<tr>
<td>0241</td>
<td>GO TO 99949</td>
<td>Go to line 99949.</td>
</tr>
<tr>
<td>0242</td>
<td>I= I+1</td>
<td>Increment variable I by 1.</td>
</tr>
<tr>
<td>0244</td>
<td>FBUF(I)=LABEL(L)</td>
<td>Assign LABEL(L) to FBUF(I).</td>
</tr>
<tr>
<td>0245</td>
<td>GO TO 99948</td>
<td>Go to line 99948.</td>
</tr>
<tr>
<td>0246</td>
<td>CONTINUE</td>
<td>Continue to next line.</td>
</tr>
<tr>
<td>0247</td>
<td>FBUF(6)=BLANK</td>
<td>Assign BLANK to FBUF(6).</td>
</tr>
<tr>
<td>0248</td>
<td>CALL FOUT(SPIN,VLEA,VLEB)</td>
<td>Call FOUT subroutine with arguments SPIN, VLEA, and VLEB.</td>
</tr>
<tr>
<td>0249</td>
<td>CALL FOUT(ECL,I,1)</td>
<td>Call FOUT subroutine with arguments ECL, I, and 1.</td>
</tr>
<tr>
<td>0250</td>
<td>CALL FOUT(SPIN,VLEA,VLEB)</td>
<td>Call FOUT subroutine with arguments SPIN, VLEA, and VLEB.</td>
</tr>
<tr>
<td>0251</td>
<td>CALL FOUT(PLU,1,1)</td>
<td>Call FOUT subroutine with arguments PLU, 1, and 1.</td>
</tr>
<tr>
<td>0252</td>
<td>IF(.NOT.(ECCND)) GC TC 99945</td>
<td>If ECCND is not true, go to line 99945.</td>
</tr>
<tr>
<td>0253</td>
<td>C // THE 'BY' IS MISSING--DEFAULT INCREMENT OF EXPRESSION</td>
<td>Comment: By is missing, default increment of expression.</td>
</tr>
<tr>
<td>0254</td>
<td>CALL FOUT(ONE,1,1)</td>
<td>Call FOUT subroutine with arguments ONE, 1, and 1.</td>
</tr>
<tr>
<td>0255</td>
<td>GO TO 99944</td>
<td>Go to line 99944.</td>
</tr>
<tr>
<td>0256</td>
<td>CALL FOUT(IFNCT,9,9)</td>
<td>Call FOUT subroutine with arguments IFNCT, 9, and 9.</td>
</tr>
<tr>
<td>0257</td>
<td>CALL FOUT(SPIN,TBEG,TCEND-2)</td>
<td>Call FOUT subroutine with arguments SPIN, TBEG, and TCEND-2.</td>
</tr>
<tr>
<td>0258</td>
<td>CALL FOUT(RSTAR,1,1)</td>
<td>Call FOUT subroutine with arguments RSTAR, 1, and 1.</td>
</tr>
<tr>
<td>0259</td>
<td>CONTINUE</td>
<td>Continue to next line.</td>
</tr>
<tr>
<td>0260</td>
<td>CALL FPRTN</td>
<td>Call FPRTN subroutine.</td>
</tr>
<tr>
<td>0261</td>
<td>C // LAB2--INDICATES VARIABLE=VARIABLE+EXPRESSION3.</td>
<td>Comment: LAB2 indicates variable = variable + expression3.</td>
</tr>
<tr>
<td>0262</td>
<td>I=1</td>
<td>Initialize variable I to 1.</td>
</tr>
<tr>
<td>0263</td>
<td>GO TO 99943</td>
<td>Go to line 99943.</td>
</tr>
<tr>
<td>0264</td>
<td>I= I+1</td>
<td>Increment variable I by 1.</td>
</tr>
<tr>
<td>0265</td>
<td>GO TO 99940</td>
<td>Go to line 99940.</td>
</tr>
<tr>
<td>0266</td>
<td>FBUF(I)=LABEL(L)</td>
<td>Assign LABEL(L) to FBUF(I).</td>
</tr>
<tr>
<td>0267</td>
<td>GC TO 99942</td>
<td>Go to line 99942.</td>
</tr>
</tbody>
</table>
0268      99940  CONTINUE
0269      99940  F8UF(6)=BLANK
0270      99940  CALL FCUT(IFNOT,1,9)
0271      99940  CALL FOUT(IFNOT,9,9)
0272      99940  CALL FCUT(SPIN,VBLEA,VBLEB)
0273      99940  CALL FOUT(MINUS,1,1)
0274      99940  CALL FOUT(IFNOT,9,9)
0275      99940  CALL FCUT(SPINEX2A,EX2B)
0276      99940  CALL FCUT(RSTAR,1,1)
0277      99940  CALL FCUT(RSTAR,1,1)
0278      99940  IF (.NOT.( .NOT. ECOND1 )) GO TO 99939
      99940  C
      99940  // IF 'BY' MISSING THEN GENERATE...(EXP2).LE.0...
      99940  C
      99940  // IN PLACE CF...(EXP2).LT.(EXP3).LE.0.....
0279      99940  CALL FCUT(RSTAR,2,2)
0280      99940  CALL FOUT(IFNOT,9,9)
0281      99940  CALL FOUT(SPINTCEEG,TCEGC-2)
0282      99940  CALL FOUT(RSTAR,1,1)
0283      99939  CONTINUE
0284      99939  CALL FOUT(LEC,1,5)
0285      99939  CALL FOUT(CGOTO,1,9)
0286      99939  CALL FOUT(LABEL,LAB4,LAB4+4)
0287      99939  CALL FPRNT
C // LAB1...IF(.NOT.((VBLE-(EXP3).LE.0)) GO TO LAB3...
0288 STEND=TOEND
0289 INDFLG=FRWRD
0290 CURFLG=CLD
0291 IF(.NOT.(FSTRT+INCR.LT.IMAX)) GO TO 99938
0292 FSTRT=FSTRT+INCR
0293 99938 CONTINUE
0294 GO TO 9994
0295 99962 IF(.NOT.(TYPE.EQ.RREPT)) GO TO 99937
0296 CALL CHECK(RREPT,BAC)
0297 IF(.NOT.(BAD)) GO TO 99936
0298 GO TO 99958
0299 99936 CONTINUE
0300 IF(.NOT.(LBFLG)) GO TO 99935
C // LABELLED STMT, GENERATED 'LAB CONTINUE'
0301 CALL FOUT(CONT1,1,8)
0302 CALL FPRNT
0303 99935 CONTINUE
0304 STTOP=STACK(TOP)
C // LAB3=EXIT TARGET, LAB2=CYCLE TARGET, LAB1=LOOP BACK LABEL
0305 C 9
L IF(.NOT.(STTOP.EQ.FFOR)) GO TO 99934
0306 LAB1=STACK(TOP-5)
0307 LAB2=STACK(TOP-2)
0308 LAB3=STACK(TOP-1)
0309 TOP=TCP-6
0310 LBPTR=LBPTR-2C
0311 GO TO 99933
0312 99934 CONTINUE
0313 LAB1=STACK(TOP-3)
0314 LAB2=STACK(TCP-2)
0315 LAB3=STACK(TCP-1)
0316 TCP=TCP-5
0317 LBPTR=LBPTR-15
0318 99933 CONTINUE
0319 TPI=STACK(TPITR-4)
0320     FSTRT=FSTRT+INCR
0321     FPTR=FSTRT
0322     I=1
0323     GC TC 99932
0324     I= I+1
0325     99932   IF(.NOT.(( I-( 5 )).LE.0)) GC TC 99925
0326     L=LAB2+I-1
0327     FBUF(I)=LABEL(L)
0328     99930   GO TO 99931
0329     99929   CCONTINUE
0330     C     FBUF(6)=BLANK
0331     CALL FCUT(GCTC,1,6)
0332     CALL FCUT(LABEL,LAB1,LAB1+4)
0333     CALL FPINT
0334     C     // GC TC LABEL 1
0335     99927   I=1
0336     99928   GC TO 99928
0337     99927   I= I+1
0338     99928   IF(.NOT.(( I-( 5 )).LE.0)) GO TO 99925
0339     L=LAB3+I-1
0340     99926   GO TO 99927
0341  99925   CONTINUE
0342     FBUF(6)=BLANK
0343     CALL FOUT(CONTI,1,8)
0344     CALL FPRT
0345     C       // LABEL CONTINUE
0346     STEND=TOEND
0347     CURFLG=NEW
0348     INDFLG=BKWRD
0349     GO TO 99994
0350  99937   IFL(NCT.( TYPE.EQ.CCASE )) GC TC 99924
0351     LAB=GENLAB(1)
0352     C       // LABEL(1:5)=BLANKS.
0353     STACK(TOP+1)=1
0354     STACK(TOP+2)=LAB
0355     STACK(TCP+3)=CCASE
0356     TCP=TCP+3
0357     IFL(NCT.( TCP.GT.STMAX )) GO TO 99923
0358     CALL ERROR(6,0,5,0,0)
0359  99923   CONTINUE
0360     C       // STACK OVERFLOWED---PROGRAM TERMINATED.
0361     STEND=TOEND
0362     CURFLG=OLD
0363     INDFLG=FRWRD
0364  99922   IFL(NCT.( LBFLG )) GC TO 99922
0365     C       // LABELED STATEMENT--GENERATE 'LAB CONTINUE'.
0366     CALL FOUT(CONTI,1,8)
0367  99921   CONTINUE
0368     C       // SET THE INDENTION
0369     IFL(NCT.( FSTR+INCR.LT.IMAX )) GO TO 99921
0370     FSTR=FSTR+INCR
0371  99924   IFL(NCT.( TYPE.EQ.CCASE )) GC TC 99920
0372     CALL CHECK(CCLEN,HAC)
0373     IFL(NCT.( BAC )) GC TO 99919
GO TO 99918
CONTINUE
IF(.NOT.(LBFLG )) GC TO 99918
CALL ERRSR(1,CASS,4,2,0,0)
C        // ERROR--LABELLING THE CASE HAMPERS PROGRAM CLEANLINE
         // DELETED FROM THE FCRTTRAN OUTPUT.
         I=1
GO TO 99917
I= I+1
IF(.NOT.(( I-( 6 )) .LE.0)) GO TO 99914
BUF(I)=BLANK
GO TO 99916
CONTINUE
CONTINUE
LAB1=STACK(TCP-1)
LAB2=STACK(TCP-2)
FSTRT=FSTRT-INCRT
FPTR=FSTRT
C        // CASE MATERIAL PRINTED ONE IDENT TO THE RIGHT.
IF(.NOT.(( LAB2 .NE. 1 )) GO TO 99913
CALL FCUT(GGTC,1,6)
CALL FCLT(LABEL,LAB1,LAB1+4)
CALL FPRNT
C
// GC TC LAB1.
99913 CCNTINUE
CALL SCAN1(COLN,ECCND)
IF(.NOT.(ECCND)) GC TC 99912
IFERROR(2,COLN,1,1,0,0)
99912 CCNTINUE
C
// ERROR--MISSING CCLCN-SUPPLIED AT END OF STATEMENT.
K=TCEND
TCEND=TCREG
C
// SPIN(K)=IMAX CR SPIN(K+1)=SEMICOLON CR COLON.
TMAX=IMAX
IMAX=K
CALL SCAN(ELZ,4,ECCNC)
IMAX=TMAX
If(.NOT.(ECCND)) GC TO 99911
LAB3=GENLAB(1)
TCEND=K
C
// OUTPUT-LAB2-IF NOT CONDITION THE GO TO LAB3.
I=1
GO TC 99910
I= I+1
99910 IFI,.NOT.((I-(5)),LE.0)) GO TO 99907
L=LAB2+I-1
FBUF(I)=LABEL(L)
GO TC 99909
CONTINUE
FBUF(6)=BLANK
CALL FCUT(IFNOT,1,9)
CALL FCUT(SPIN,TCEED,TCEND)
CALL FCUT(CGCTC,1,9)
CALL FCUT(LABEL,LAB3,LAB3+4)
CALL FPRNT
TECED=TECED+1
GO TO 99906
0422   99911   CC\n\nC  
    // ELSE CLAUSE FCUND
    I=1
    GC TO 99905
    I= I+1
0426   99905   IF(.NOT.(( I-( 5 ).LE.0)) GC TO 99902
    L=LAB2+I-1
    FB\n\n0428   99903   GC TO 99904
0430   99902   CC\n\n0431   99902   C
\n0432   99902   CALL FCUT(CC\n\n0433   99902   \n\n0434   99902   C
\n0435   99902   // LABEL(1,...,5) ARE BLANKS.
\n0436   99906   CC\n\n0437   99906   STACK(TJP-2)=LAB3
0438   99906   INCFLG=NEUTR
0439   99906   CURFLG=PUL\n\n0440   99906   ST\n\n108
C // RESTORE THE INCENTATION.
0441 IF(NOT.(FSTRT+INCR.LT.IMAX)) GO TO 99901
0442 FSTRT=FSTRT+INCR
0443 99901 CONTINUE
0444 GO TO 99994
0445 99920 IF(NOT.(TYPE.EQ.ENDCAS)) GO TO 99900
0446 CALL CHECK(ENDCAS,FAC)
0447 IF(NOT.(BAD)) GC TO 99895
0448 GO TO 99998
0449 99899 CONTINUE
0450 FSTRT=FSTRT-INCR
0451 FPTR=FSTRT
0452 IF(NOT.(L3FLG)) GC TO 99898

C // LABELLED STATEMENT--GENERATE 'LAB CONTINUE'
0453 CALL FCUT(CCINT,1,8)
0454 CALL FPRTN
0455 99898 CONTINUE
0456 LBP1=STACK(TCP-1)
0457 LBP2=STACK(TCP-2)
0458 TCP=TCP-3
0459 I=1
0460 GC TO 99897
0461 99896 I=I+1
0462 99897 IF(NOT.((I-15).LE.0)) GO TO 99894
0463 L=LBP2+I-1
0464 FBUF(I)=LABEL(L)
0465 99895 GC TO 99896
0466 99894 CONTINUE
0468 FBUF(6)=BLANK
0469 CALL FCUT(CCINT,1,8)
0470 CALL FPRTN
0471 C // LAB2 CONTINUE
0472 I=1
0473 99892 GC TO 99893
0474 I=I+1
IF(.NOT.((I-(5)).LE.0)) GO TO 9989C
        L=LAB1+I-1
        FBUF(I)=LABEL(L)
        GO TO 99892
479  99890 77  CONTINUE
479  99892 87  FBUF(6)=BLANK
        CALL FCLT(CCNT1,1,8)
        CALL FPRNT
        C  // LAB1 CONTINUE
        STEND=TCEND
        INDFLG=BKWD
        CURFLG=NEW
        GO TO 99994
482  99900 77  IF(.NOT.(TYPE.EQ.LLOC)) GO TO 99889
482  99902 87  C  // LAB2 IS THE TARGET OF CYCLE AND LAB3 OF EXITS
482  99904 7  C  // LAB1 IS THE IDENTIFYING LABEL.
483  99906 97  LAB1=GENLAB(0)
        LAB2=GENLAB(1)
        LAB3=GENLAB(1)
        STACK(TCP+2)=LAB1
        STACK(TOP+3)=LAB2
        STACK(TOP+4)=LAB3
0453  STACK(TOP+5)=LLOCP
0454  STACK(TOP+1)=TPITR
0455  TCP=TOP+5
0456  IF(.NOT.(TOP.GT.STMAX)) GC TO 99888
0457     CALL ERROR(6,C,0,5,0,0)
0458     CCONTINUE
0459     // STACK OVERFLOWED, PROGRAM TERMINATED
0459     I=1
0460     GC TO 99887
0461     I= I+1
0462     99887  IF(.NOT.((I-(5)).LE.0)) GO TO 99884
0463       L=LAB1+I-1
0464       FBUF(I)=LABEL(L)
0465     99884  GO TO 99886
0466     99885  CCONTINUE
0467     FBUF(6)=BLANK
0468     CALL FCUT(CCNT1,1,8)
0469     CALL FPRTN
0470     TPITR=TOP
0471     CLRFLG=CLD
0472     INDFLG=FRWRD
0473     STEND=TCEND
0474     99883  IF(.NOT.((FSTRT+INCR.LT.IMAX)) GO TO 99883
0475       FSTRT=FSTRT+INCR
0476     99882  GO TO 99994
0477     IF(.NOT.((TYPE.EQ.WHE)) GO TO 99882
0478     LAB1=GENLAB(0)
0479     LAB2=GENLAB(1)
0480     LAB3=GENLAB(1)
0481     C     // LAB2 IS THE TARGET OF EXITS, LAB3 FOR CYCLES.
0482     CALL SCAN(DOD,2,ECCND)
0483     IF(.NOT.(ECCND)) GC TO 99881
0484     CALL ERROR(2,CCC,2,1,0,0)
0485     C     // CALL ERROR---MISSING 'DO'--SUPPLIED AT THE END OF S
0485     TMT.
TCEND=TCEND+1
GO TO 99880
CONTINUE
TCEND=TCEND+2
CONTINUE
IF(NOT.(NOT.LBFLG)) GO TO 99879
I=1
GO TO 99878
I= I+1
IF(NOT.(( I-(5)).LE.0)) GO TO 99875
L=LAB1+I-1
FBUF(I)=LABEL(L)
GO TO 99877
CONTINUE
FBUF(6)=BLANK
CONTINUE
CALL FCUT(IFNCT,1,5)
CALL FCUT(SPIN,TCEG,TCEND-2)
CALL FCUT(CGOTO,1,9)
CALL FCUT(LABEL,LA2,LAB2+4)
CALL FPRNT
C  // GENERATE LAB1 AND IF TRUE COND GO TO LAB2.
STACK(TOP+2)=LAB1 548
STACK(TOP+3)=LAB2 549
STACK(TOP+4)=LAB3 550
STACK(TOP+5)=WHILE 551
STACK(TOP+1)=TPITR 552
TCP=TCP+5 553
IF (.NCT. (( TOP .GT. STMAX )) ) GO TO 99874 554
CALL ERRCR(6,0,0,5,0,0) 555
556
CONTINUE 557
C /* STACK OVERFLOWED, PROGRAM TERMINATED. 558
TPITR=TCP 559
CURFLG=CLD 560
INCFLG=FRWRD 561
STPEND=TCEND 562
IF (.NCT. ( FSTRT+INCR .LT. IMAX ) ) GO TO 99873 563
FSTRT=FSTRT+INCR 564
CONTINUE 565
GO TO 99874 566
99873 567
CONTINUE 568
GO TO 99872 569
IF (.NCT. ( TYPE .EQ. CNT ) ) GO TO 99872 570
I=1 571
GO TO 99871 572
I= I+1 573
99870 574
IF (.NCT. ( ( I - ( 6 ) ) .LE. 0 ) ) GC TO 99868 575
FBUF(I)=SPIN(I) 576
99871 577
GC TO 99868 578
CONTINUE 579
99869 580
CONTINUE 581
TCREG=7 582
99867 583
IF (.NCT. ( SPIN(TCBEC).EQ.BLANK ) ) GC TO 99866 584
TOBEG=TOBEG+1 585
TCBEC=TOBEG-1 586
IF (.NCT. ( TCBEC .GT. IMAX ) ) GO TO 99864 587
TCEND=IMAX+1 588
C // BLANK CONTINUATION CARD, DELETED. 589
GO TO 99866 590
99866 591
CONTINUE 592
99864 593
CONTINUE 594
99867 595
GC TO 99867 596
99865 597
CONTINUE 598
0580 IF(.NCT. .EQ. TOEND .GT. IMAX ) GC TC 99863
0581 GO TO 99998
0582 99863 CCNTINUE
0583 C // SPIN(TOBEQ)=NCNELANK; REST OF CARD CAN BE INDENTED.
0584 CALL FNHAND
0585 F8UFL=BLANK
0586 STREG=TOBEQ
0587 STEND=TOEND
0588 CURFL=CLD
0589 INDFL=NEUTR
0590 GO TC 99994
0591 99872 IF(.NCT. .EQ. EEEXIT ) GO TO 99862
0592 CALL CHECK(EEEXIT,BAD)
0593 IF(.NOT. ( BAD ) ) GO TO 99861
0594 GC TO 99998
0595 99861 CCNTINUE
0596 CALL TYHAND(TYPE, EEEXT, CCYCLE)
0597 GO TC 99994
0598 99862 IF(.NOT. ( TYPE .EQ. NCYCLE ) ) GO TO 99860
0599 CALL CHECK(EEEXIT,BAD)
0600 IF(.NCT. ( BAD ) ) GC TO 99859
0601 GO TO 99998
0601   99859          CONTINUE
0602          CALL TYPAND(TYPE,EEEE,CCYCLE)
0603          GC TO 99994
0604   99860          IF(.NOT. (TYPE.EQ.UNL)) GC TO 99858
0605          CALL CHECK(Until,EAC)
0606          IF(.NOT. (EAC)) GC TO 99857
0607          GC TO 99998
0608   99857          CONTINUE
0609          FSTR=FSRT-1NC
0610          FPTR=FSRT
0611          LAB1=STACK(TOP-3)
0612          LAB2=STACK(TOP-2)
0613          LAB3=STACK(TOP-1)
0614          TCP=TCP-5
0615          LBPTR=LBPTR-15
0616          TPITR=STACK(TPITR-4)
0617   99856          IF(.NOT. (LBTFLG)) GC TO 99856
0618          CALL ERROR(1,UNL,1,2,0,0)
0619          // ERROR---LABELLING OF THE UNTIL HAMMERS PROGRAM
0620          // CLARITY, CLEARED FROM FORTRAN OUTPUT.
0621   99856          CONTINUE
0622          I=1
0623   99855          GC TO 99955
0624          I=I+1
0625   99855          IF(.NOT.((I-(5)).LE.0)) GO TO 99952
0626          L=LAB2+I-1
0627          FBUF(I)=LABEL(L)
0628   99853          GO TO 99854
0629   99852          CONTINUE
0630          FBUF(6)=BLANK
0631          CALL FOUT(IFACT,1,5)
0632          CALL SCAN(KEPT,6,ECCMD)
0633          CALL FCUT(SPIN,TCHEG,TOEND)
0634   99851          IF(.NOT. (ECCMD)) GC TO 99851
0635          CALL ERR0R(2,REPT,6,1,0,0)
0636          // ERROR---MISSING REPEAT-SUPPLIED AT THE END OF THE S
C 9
STATEMENT.
TCEND=TCEND+1
GO TO 99850
99851 CONTINUE
TCEND=TCEND+6
99850 CONTINUE
CALL FCUT(I ,CGOTO,1,9)
CALL FCUT(LABEL,LAB1,LAB1+4)
CALL FPRNT
C
// GENERATE IF(.NOT.(CCND)) GC TO LABEL
C
// GENERATE THE EXIT TARGET---LAB3 CONTINUE.
I=1
GO TO 99849
I= I+1
IF(.NOT.((( I-( 5 )).*LE.0))) GC TO 99846
L=LAB3+I-1
FBUF(I)=LABEL(L)
GO TO 99848
CONTINUE
FBUF(6)=BLANK
CALL FCUT(CCNTI,1,8)
CALL FPRNT
STENC=TCEND
INDFLG=8KWRD
CURFLG=NEW
GO TO TC 99994

IF(.NOT.(TYPE.EQ.SUBFN)) GO TO 99845
WRITE(PRTR,5711)
FCMAT(1,H1)
NUMB=100000
STACK(TCP+1)=SUBFN
TCP=TCP+1
IF(.NOT.(TCP.GT.STMAX)) GO TO 99844
CALL ERROR(6,0,0,5,0,0)
CONTINUE

C // STACK OVERFLOWED, PROGRAM TERMINATED.
CALL FHAND
CURFLG=OLD
INDFLG=FRWRD
STEND=TCEND
IF(.NOT.(FSTRT+INCR.LT.IMAX)) GO TO 99843
FSTRT=FSTRT+INCR
CONTINUE
GO TO TC 99994

IF(.NOT.(TYPE.EQ.EENC)) GO TO 99842
CALL CHECK(EENC, EAD)
IF(.NOT.(EAD)) GC TO 99841
GO TO 99998
CONTINUE
FSTRT=FSTRT+INCR
FPTR=FSTRT
IF(.NOT.(LPFLG)) GC TO 99840
C // LABELLED STATEMENT--GENERATE 'LAB CONTINUE'
CALL FOUT(CONTI,1,8)
CALL FPRNT
CONTINUE
TCP=TCP+1
CALL FCUT(SP,W,TCEEG,TCEND)
CALL FPRNT
C     // GIVE ANY END CF SLBRCUTINE MESSAGES AND RESET ANY VARI
C     BLES
C     // IF NECESSARY.
C
0688    STEND=TCEND
0689    INDFLG=3KWRD
0690    CURFLG=NEW
0691    GO TO 99494
0692    99842    IF(*NCT.( TYPE.EG.CCMT )) GO TO 99839
0693    FBUF(I)=CEE
0694    TCBEG=2
0695    TCEND=IMAX
0696    99833    CONTINUE
0697    IF(*NOT.( SPIN(TOBEG).NE.BLANK )) GO TO 99835
0698    I=2
0699    GO TO 99834
0700    99833    = I+1
0701    99834    IF(*NCT.(( I-( 6 ).LE.0)) GO TO 99831
0702    FBUF(I)=BLANK
0703    GO TO 99833
0704    99832    CONTINUE
0705    CALL FCUT(SPIN,TOBEG,IMAX)
CALL FPRNT
STREG=TCBEG
GO TO 99836
CONTINUE
TOBEG=TOBEG+1
IF(.NOT.(TOBEG.GT.IMAX)) GO TO 99830
C // A BLANK COMMENT, OUTPUT A BLANK LINE.
CALL FCUT(  SPIN,FPTR,FMAX)
CALL FPRNT
STREG=SSTRT
GO TO 99836
CONTINUE
GO TO 99838
CONTINUE
STEND=IMAX
CURFLG=OLD
INCFLG=NEUTR
GO TC 99994
IF(.NCT.( TYPE.EQ.EQ)) GO TO 99829
C // CLOSE THE FILE.
WRITE(PRTR,1300)
FORMAT(/35H END OF SPARKS PROGRAM....)
1300
STCP
99829
CONTINUE
99994
CONTINUE
IF(.NCT.(CURFLG.EQ.PULCUT)) GC TC 99828
SSTRT=SSTRT+INCR
CALL SPRNT
SSTRT=SSTRT+INCR
IF(.NOT.(SSTRT.GT.SMAX)) GO TC 99827
SSTRT=SSTRT+INCR
CONTINUE
99827
BLKNO=BLKNO+1
CONTINUE
IF(.NCT.(CURFLG.EQ.OLD)) GO TC 99826
CALL SPRNT
0740  95826  CONTINUE
0741  IF(.NOT.( INDFLG.EQ.FRWRD )) GO TO 99825
0742      SSTRT=SSTRT+INCR
0743      IF(.NOT.( SSTRT.CT.SMAX )) GO TO 99824
0744          SSTRT=SSTRT-INCR
0745  99824  CONTINUE
0746      IF(.NOT.( CURFLG.EQ.NEW )) GO TO 99823
0747          CALL SPRNT
0748  99823  CONTINUE
0749      BLKNC=BLKNC+1
0750      NESTNO=NESTNO+1
0751  99825  CONTINUE
0752      IF(.NOT.( INDFLG.EQ.BKWRD )) GO TO 99822
0753          SSTRT=SSTRT+INCR
0754      IF(.NOT.( CURFLG.EQ.NEW )) GO TO 99821
0755          CALL SPRNT
0756  99821  CONTINUE
0757      NESTAC=NESTNC-1
0758      BLKNO=BLKNO+1
0755  99822  CONTINUE
0760      SMNC=SMNO+1
0761  99998  GO TO 99999
0762  95597  CONTINUE
0763  END
FORTRAN IV GLEVEL 21

BLCK DATA

DATE = 77326
20/47/11

0001
BLCK DATA

0002
IMPLICIT INTEGER(A-Z)

0003
LOGICAL LBFLG

0004
COMMON/INPUT/SPIN(80),IJOEG,GJEN,ST3EG,STEND,SWMAX,LBFLG

0005
COMMON/SPRINT/STRT,INCR,STMC,BLKNC,NESTNC,INDFLG,GURFLG,SMAX

0006
COMMON/FRINT/FBUF(80),FPTR,FMAX,FSTRT,BLNSK(131)

0007
COMMON/CENTRAL/STACK(200),LABEL(200),TCP,LSMAX,BPTR,STFM(6),

0008
1
COMMON/CMPTR/CGOTO(9),CGATT(9),EQL(1),GOTO(6),IFNOT(9),LED(5),

0009
1
COMMON/INPRT/CDL(1),EL2(4),DBSLH(2),DOO(2),BYE(2),THEN(4),TOO

0010
1
COMMON/SSTYP1/CASE,COLNC,CRSL,ELSE,EENC,ECL,EEXT,FFOR,IF,

0011
1
LLCCP,PREP,UNTFL,WITH,SUBFR,INTD,NCYCLE,ENDCAS,EENDIF

0012
COMMON/SSTYP2/CMNT,CMNT,BLANK,SEMI

0013
COMMON/TABLES/TRES(23),RESRD(200)

0014
COMMON/AMNIC/CEE,PTR,FFILE,CRDR

1
COMMON/KEYWD/EEND(3),EEXT(4),CCYCLE(5),UUNTL(5),CASS(4),IIF(2),

1
ITERAT(5),ENCASS(7),ENIF(5),REPT(6),CCCLCN(5)

C
// CRITICAL CODES--STFM IS ACCESED USING THE FOLLOWING:
C
C
// CCASE=1,FFOR=2,IF=3,LLPRINT=4,WHILE=5,SUBFRN=6.

0015
DATA SPIN,/INT,INH,IS,1H,1HI,1HS,1H,1FA,1H,1HS,1HP,1HA,

0016
1HR,1HK,1HS,1H,1HP,1HR,1HC,1HG,1HF,1HA,

1
1T,1H,1H,1H,1T,1H,1T,1H,1H,1H,1H,1H,1H,1H,1H,1H,1H,1H,1H,1H,1H

2
1H,1H,1H,1H,1H,1H,1H,1H,1H,1H,1H,1H,1H,1H,1H,1H,1H,1H,1H,1H,1H

3
1H,1H,1H,1H,1H,1H,1H,1H,1H,1H,1H,1H,1H,1H,1H,1H,1H,1H,1H,1H,1H

4
1H,1H,1H,1H,1H,1H,1H,1H,1H,1H,1H,1H,1H,1H,1H,1H,1H,1H,1H,1H,1H

0016
DATA BLNKS/13141H/

0017
DATA STMAX,LSMAX,NUMB/200,200,100000/

0018
DATA MELT,EBKWRD,FRWSR,NEW,CLD,PUFLAT/1,2,3,1,2,3/

C
// STACK FRAME SIZES:

C
C
// CASE FOR IF LOOP SUBFRN WHILE

0019
DATA STFRM/3,6,2,5,1,5/
0036 C  //  178TH  PLACE  AVAILABLE  FOR  INSERTION.  
0037  DATA  TRES/144,1,17C,148,152,162,21,27,34,133,40,45,55,65,
0038 C  104,72,75,111,88,96,124,125/ 
0039 C  //  TRES  CONTAINS  POINTS  INTO  RESRD  WHICH  CONTAINS  ALL  THE 
0040 C  //  RESERVE  WORDS  IN  SPARKS.  ****  NOTE  ****  THIS  ORDER  IS 
0041 C  //  PRESENTLY  SET  UP  FOR  EBCDIC.  
0042  DATA  PTRA,CRDR,FFILE/ 6, 5, 4/ 
0043  DATA  BLANK,CEE,SEM/1H ,1FC,1H/ 
0044  DATA  EEE,ENCASS/1HE,1HN,1HD,1HE,1HN,1HD,1HC,1HA,1HS,1HE/ 
0045  DATA  EEE,ENIF/1HE,1HX,1HI,1HT,1HE,1HN,1HD,1HI,1HF/ 
0046  DATA  EEE,ECASS/1HC,1HY,1FC,1HL,1HE,1HC,1HA,1HS,1HE/ 
0047  DATA  EEE,UCASS/1HC,1HY,1FC,1HL,1HE,1HC,1HA,1HS,1HE/ 
0048  DATA  UUNTL,1IF/1HU,1HN,1HT,1HI,1HL,1FI,1HF/ 
0049  DATA  IERAT/1HI,1FT,1HE,1HR,1HA,1HT,1FI,1HV,1HE/ 
0050  END
SUBROUTINE CETNX(TYPE)

C    // THE SUBROUTINE DETERMINES THE TYPE OF CYCLE STATEMENT AND
C    // SETS TOBEG AND TOEND CN TOKEN BOUNDARIES.
C
IMPlicit INTEGER(A-Z)
LOGICAL LBFLG, ECCN
COMMON/INPUT/SPIN(80), TOBEG, TOEND, STBEG, STEND, IMAX, LBFLG
COMMON/FRTINF/FBUF(80), FPTR, FMAX, FSTRT, BLNKS(131)
COMMON/STYP2/COMT, CCNT, FCTRN, BLANK, SEMI
COMMON/AMNIOC/CEE, PRTR, FILE, CRDR
DATA EFF/1F1/

C
LBFLG=.FALSE.
CONTINUE
99999
CONTINUE
99996
CONTINUE
TCEND=TOEND+1
IF(NOT.( TCEND.GT.IMAX )) GO TO 99993
GO TO 99994
CONTINUE
99993
CONTINUE
IF(NOT.( SPIN(TCEND).NE.BLANK )) GO TO 99992
IF(NOT.( SPIN(TCEND).EQ.SEMI )) GO TO 99991
GO TO 99995
CONTINUE
99991
CONTINUE
IF(NOT.( SPIN(1).EQ.CEE )) GO TO 99990
TCBEG=1.
TOBEG=1
TOEND=1
TYPE=CGMT
RETURN
CONTINUE
99990
CONTINUE
TOBEG=TOEND
CALL SCANI(BLANK, ECOND)
IF(NOT.( ECOND )) GO TO 99988
TYPE=FCTRN
RETURN
GO TO 99989
**IEY002I LABEL**

```
0032 99988 IF (.NOT. (SPIN(6).NE.BLANK)) GO TC 99987
0033       TYPE=CCNT
0034       RETURN
0035       GC TC 99989

0036 99987 IF (.NOT. (TCEAD.GT.6)) GC TC 99986
0037          CALL STYP(TYPE)
0038          IFL (.NOT. (.NOT..LBFLG)) GC TO 99985
0039          I=1
0040          I= I+1
0041 99983 GO TC 99984
0042 99984 IF (.NOT.((I-6).LE.0)) GC TO 99981
0043          FRUF(I)=BLNKS(I)
0044 99982 GC TO 99983
0045 99981 CCOUNTUE
0046 99985 CONTINUE
0047       RETURN
0048       GO TO 99989
```
**FORTRAN IV G LEVEL 21**

---

**DETNX**

**DATE = 77326**

**20/47/11**

```
0049  99986  IFI(.NOT.( T0END.LE.6 )) GC TO 99980
0050       I=1
0051    GC TO 99979
0052       I= I+1
0053  99978  IFI(.NOT.(( I-( 5 ))*LE.01)) GC TO 99976
0054         BLNKS(I)=SPIN(I)
0055         FBUF(I)=SPIN(I)
0056  99977  GO TO 99978
0057  99976  CCNTINUE
0058       LBFLG=.TRUE.
0059       FBUF(6)=BLANK
0060  99980  CONTINUE
0061  99992  CCNTINUE
0062  99995  GO TO 99996
0063  99996  CONTINUE
0064  99997        READ(CRDR,20) SPIN
0065  99998        FCMAT(80A1)
0066        20
0067  99999        IFI(.NOT.( SPIN(I)*EC.EFF )) GC TO 99975
0068          WRITE(PRTK,31)(SPIN(I),I=1,80)
0069          FCMAT(34H     ----BEGINNING OF FCRTRAN----- ,80A1)
0070          READ(CRDR,20) SPIN
0071  99974        IFI(.NOT.( SPIN(I)*AE.EFF )) GC TO 99973
0072          WRITE(PRTK,33)(SPIN(I),I=1,80)
0073          FCMAT(20X,80A1)
0074          WRITE(FFILE,32)(SPIN(I),I=1,80)
0075          FCMAT(80A1)
0076          READ(CRDR,20) SPIN
0077  99972    GC TO 99974
0078  99973    CONTINUE
0079          WRITE(PRTK,34) (SPIN(I),I=1,80)
0080          FCMAT(28H     ----END OF FORTRAN----- ,80A1)
0081          READ(CRDR,20) SPIN
0082  99975    CCNTINUE
0083          TOEND=0
```
SUBROUTINE STTYP(TYPE)
  // THE TOKEN AT SPIN(TOBEG) TO SPIN(TCEND) IS SEARCHED
  // FCN IN THE SYMBCAL TABLE.
  //
  IMPLICIT INTEGER(A-Z)
  COMMON/INPLT/SPIN(80),TOBEG,TCEND,STBEG,STENC,IMAX,LBFLG
  COMMON/STTYP2/COMT,CONT,FCTRNL,BLANK,SEMI
  COMMON/TABLES/TRES(23),RESRD(200)
  //
  TOP=1
  BCTTCM=23
  LENGTH=TCEND-TOBEG+1
  IF(.NOT.( TCP.LE.BCTTCM )) GO TO 99998
  MID=(TCP+BCTTCM)/2
  I=TRES(MID)
  IF.(.NOT.( SPIN(TOBEG).LT.RESRD(I+1) )) GC TO 99995
  BCTTCM=MID-1
  GO TO 99996
  IF(.NOT.( TCP.GT.RESRD(I+1) )) GC TO 99994
  TCP=MID+1
  GO TO 99996
  IF(.NOT.( SPIN(TCBEQ).EQ.RESRD(I+1) )) GC TO 99993
  IF.(.NOT.( LENGTH.EQ.1 )) GC TO 99992
  TYPE=RESRD(I+2)
  RETURN
C
CONTINUE
PNT=2
IF(.NOT.( PNT.LE.RESRD(I).AND.PNT.LE.LENGTH )) GC TO 99999
  9
   ITEMP=I+PAT
   JTEMP=TCBEG-1+PAT
   IF.(.NOT.( SPIN(JTEMP).LT.RESRD(ITEMP) )) GC TO 99987
   BCTTCM=BCTTCM-1
   PNT=9999
   GC TO 99988
   IF(.NOT.( SPIN(JTEMP).GT.RESRD(ITEMP) )) GC TO 99986
0032  TCP=TCP+1
      PNT=9999
      GO TO 99988
0035  99986  IF(.NOT. ( SPIN(JTEMP).EQ.RESRC(ITEMP) )) GO TO 99985
0036  PNT=PNT+1
0037  99985  CCNTINUE
0038  99988  CCNTINUE
0039  99989  GO TO 99991
0040  99990  CCNTINUE
0041  99991  IF(.NOT. ( PNT. NE. 9999 )) GO TO 99984
0042  99988  IF(.NOT. ( LENGTH.EQ.RESRD(I) )) GO TO 99982
0043  99989  JTEMP=LENGTH+1+I
0044  99990  TYPE=RESRD(JTEMP)
0045  99991  RETURN
0046  99992  GO TO 99983

$******01) IEY002I LABEL************************************************************
SUBROUTINE SCAN1(CHAR,ECOND)
C    // LOOKS FOR CHAR AND IF FOUND, SPIN(TOEND+1)=CHAR
C    // AND ECOND=FALSE. IF NOT FOUND UPTC IMAX OR SEMICOLON
C    // ECOND=TRUE.
C    //
IMPLICIT INTEGER(A-Z)
LOGICAL ECOND
COMMON/INPUT/SPIN(80),TOBEG,TOEND,STBEG,STEND,IMAX,LBFLG
COMMON/OUTP/COMT,CCNT,FCTRL,BLANK,SEMI
C    //
ECOND =.FALSE.
99999 IF(.NOT.( TCEND.LT.IMAX )) GO TO 99998
90010  ITEMP=SPIN(TCEND+1)
90011  KTEMP=SPIN(TOEND)
90012  IF(.NOT.( ITEMP.EQ.SEMI.AND.KTEMP.EQ.BLANK )) GO TO 99996
90013  ECOND =.TRUE.
90014  RETURN
90015  CONTINUE
90016  IF(.NOT.( CHAR.NE.SEMI )) GC TC 99995
90017  IF(.NOT.( ITEMP.EQ.CHAR )) GC TC 99994
90018  RETURN
90019  CONTINUE
90020  TCEND=TCEND+1
90021  GO TC 99999
90022  ECOND = .TRUE.
90023  RETURN
90024  END
SUBROUTINE SCAN(PAT, LEN, ECOND)

    // THE SUBROUTINE COMPARES THE PATTERN PAT WITH ALL TOKENS
    // SURROUNDED BY BLANKS FROM SPINIT(TCEND+1) COWARDS AND
    // RETURNS WITH TCEND AT THE OLD POSITION OF TCEND; WITH THE
    // PATTERN STARTING AT TCEND+1. IF NCT FCUND ECOND IS SET TRUE

    IMPLICIT INTEGER(A-Z)
    LOGICAL ECOND
    COMMON/INPUT/SPIN(80),TOPEG,TOEND,STREG,STEND,IMAX,LBFLG
    COMMON/STYP2/COMT,CONT,FUTRN,BLANK,SEMI
    DIMENSION PAT(6)

    TOBEG=TCEND+1
    CONTINUE

    K=TCEND+1
    IF(.NOT.( K.LE.IMAX.AND.SPIN(K).EQ.BLANK )) GO TO 99995
    K=K+1
    GO TO 99994
    CONTINUE
    TOEND=K
    CALL SCAN1(BLANK,ECOND)
    IF(.NOT.( ECOND )) GO TO 99993
    RETURN
    CONTINUE
    IF(.NOT.( LEN.NE.TOEND-K+1 )) GC TO 99992
    GC TO 99998
    CONTINUE

    // COMPARE THE PATTERN WITH A TOKEN OF THE SAME LENGTH.

    I=1
    GO TO 99991
    I=I+1
    IF(.NOT.( I-( LEN )).LE.0)) GC TO 99988
    J=K+I-1
    IF(.NOT.( PAT(I).NE.SPIN(J) )) GO TO 99987
    GC TO 99988
    CONTINUE
SUBROUTINE SPRNT

IMPLICIT INTEGER(A-Z)
LOGICAL LBFLG,OVFL

COMM/N/INPUT/SPIN(80),TOBEG,TCEND,STBEG,STEAD,IMAX,LBFLG
COMM/N/STYP2/CCNT,CCNT,FCTRN,BLNK,SEMI
COMM/N/SPRINF/STRT,INCR,STMNO,BLNKC,NESTNG,INDFLG,CURFLG,SMAX
COMM/N/FRTINF/PBUF(80),FPTR,FMAX,FSTRT,BLNKS(131)
COMM/N/AMNIE/CEE,PRTR,FFILE,CRDR
COMM/N/TYP/TYPE

C
//

IF(.NOT.(TYPE.EQ.CCAT)) GC TC 99999
BLNKS(6)=SPIN(6)
GO TO 99999
GO TO 99998

OVFL = .FALSE.
LEN1=STEND-STBEG+1
LEN2=SMAX-STRT+1
SEND=STEND
IF(.NOT.(LEN1.GT.LEN2)) GC TC 99995
OVFL = .TRUE.
SEND=STBEG+LEN2-1
CONTINUE

WRITE(PRTR,10) NESTNG,BLNKC,STMNO,(BLNKS(I),I=1,SSSTR1),
1 (SPIN(I),I=STBEG,SEND),(BLNKS(I),I=7,FIL), (SPIN(I),I=73)
2 ,80)
FORMAT(1H ,15,15,16,4X,111A1)
I=1
GC TC 99994
I = I+1
IF(.NOT.(I-(6)).LE.0)) GO TC 99991
BLNKS(I)=BLANK
GO TO 99993
CONTINUE
STBEG=SEND+1
IF(.NOT.(.NOT.OVFL)) GC TO 99998

GO TO 99994
SUBROUTINE FHAND
C // COMPLETES FORTRAN STATEMENTS AND OUTPUTS THE FORTRAN.
C
C IMPLICIT INTEGER(A-Z)
C LOGICAL LPFLG,ECCNC
C COMMON/INPLT/SPIN(80),TOBEG,TCEND,STBEG,STEND,IMAX,LRFLG
C COMMON/STYP2/CMTH,CONT,FRTRN,BLANK,SEMI
C COMMON/FRTINF/FRBUF(80),FPTR,FMAX,FSRTN,BLNKS(131)
C COMMON/TYPE/TYPE
C
C CONTINUE
C
CALL SCAN1(SEMI,ECCNC)
IF (.NOT.(TYPE.EQ.CCNT)) GC TC 99996
FBUF(6)=SPIN(6)
CONTINUE
CALL FGUT(SPIN,TCBEG,TCEND)
IF (.NOT.(ECCNC)) GC TC 99995
GC TC 99997
CONTINUE
TOEND=TCEND+1
IF (.NOT.(SPIN(TCENC+1).NE.SE mi)) GO TO 99994
GC TO 99997
CONTINUE
TOBEG=TCEND+1
TOEND=TOBEG
GO TO 99999
CONTINUE
CALL FPRNT
RETURN
END
SUBROUTINE FCUT(PAT, STRT, ENDC)
C    // THE VALUE OF FSTRT WILL USUALLY BE 7 AND FMAX WILL BE 72.
C    // FPTR SHOULD BE AT FSTRT. BLNKLN IS A FLAG TO INDICATE A
C    // TO INDICATE A SC FAR BLANK LINE.
C
C
IMPLICIT INTEGER(A-Z)
LOGICAL LBFLG, BLNKLN
COMMON/CENTRL, STACK(20C), LABEL(200), TCP, LBMAX, LBPTR, STFRM(6),
1 TPITR, NELTR, BKWRC, FRWRC, HNH, CLD, PULOUT, STMAX, NUMB
COMMON/FRT, INF, FBUF(80), FPTR, FMAX, FSTRT, BLNKLN(131)
COMMON/STYP2, CCNT, FCTRL, BLANK, SEMI
COMMON/MNIC/CEE, PRTR, FFIE, CRDR
COMMON/TYP/TYP
COMMON/FCRPT/BLNKLN
DIMENSION PAT(1)
DATA NINE/1H9/

C
BLNKLN = .FALSE.
J = STRT
GO TO 9999
J = J + 1
99999 IF(.NOT.((J - (END)).LE.0)) GO TO 99996
99997 IF(.NOT.(FPtr.GT.FMAX)) GO TO 99995
99998 CALL FPRNT
I = 1
GO TO 99994
I = I + 1
99993 IF(.NOT.((I - (5)).LE.0)) GC TO 99991
99992 FBUF(I) = LABEL(I)
GO TO 99993
99991 CONTINUE
99994 IF(.NOT.(TYPE.EQ.CCMT)) GC TO 99990
99995 FBUF(1) = CEE
CONTINUE
99996 JJ = J
99989 IF(.NOT.(PAT(JJ).EQ.BLANK)) GC TO 99988
IF (.NOT. (JJ .EQ. END)) GO TO 99986
    BLNKLA = .TRUE.
    RETURN
99986
    CONTINUE
    JJ = JJ + 1
99987
    GO TO 99989
99988
    CONTINUE
    FBUF(6) = NINE
99995
    CONTINUE
    FBUF(FPTR) = PAT(J)
    FPTR = FPTR + 1
99997
    GO TO 99998
99996
    CONTINUE
    RETURN
END
SLBROTHE FPRAT

C // THE SLBROTHE PRINTS THE TRANSLATED FORTRAN

0002 IMPLICIT INTEGER(A-Z)
0003 LOGICAL BLAKLA
0004 COMMON/INPUT/SPIN(J9), TOBEG, TOEND, TBEG, TEND, IMAX, LBFLG
0005 COMMON/AMNIE/CEE, PTRR, FFILE, CRDR
0006 COMMON/SPIRINF/STRT, INCR, STMNO, BLKNC, NESTNO, INDNLG, CURFLG, SMAX
0007 COMMON/RTRINF/FBUF(80), FPTR, FMAX, FSTRT, BLNKS(131)
0008 COMMON/FPRRT/BLAKLA

C //

0009 IF(.NOT.( FPTR .EQ. FSTRT )) GC TC 99999
0010 RETURN
0011 99999 CONTINUE
0012 IFF=FSTRT-1
0013 IF(.NOT.( IFF .GE. 7 )) GO TC 99998
0014 I=7
0015 GO TC 99997
0016 I= I+1
0017 99997 IF(.NOT.(( I-( IFF )) .LE.0)) GO TO 99994
0018 FBLF(I)=BLNKS(I)
0019 99994 GO TC 99996
0020 99993 CONTINUE
0021 99998 CONTINUE
0022 IF(.NOT.( FPTR .LE. FMAX )) GO TO 99993
0023 I=FPTR
0024 GO TC 99992
0025 99992 I= I+1
0026 99991 IF(.NOT.(( I-( FMAX )) .LE.0)) GC TC 99989
0027 FBLF(I)=BLNKS(I)
0028 99990 GC TC 99991
0029 99999 CONTINUE
0030 99993 CONTINUE
0031 I=73
0032 GO TC 99988
0033 99987 I= I+1
0034 99988 IF(.NOT.(( I-( 75 )) .LE.0)) GO TO 99985
FBUF(I)=SPIN(I)
GO TO 99987
CONTINUE
IF(.-NOT.(.-NOT.BLNKLAN)) GC TO 99984
WRITE(FFILE,10)(FBUF(I),I=1,75),STMNO
FORMAT(75A1,15)
CONTINUE
FPTR=FSTRT
RETURN
ENC
INTEGER FUNCTION ISNUM(Q)
C // THE FUNCTION CHECKS TO SEE IF Q IS THE ALPHANUMERIC
C // REPRESENTATION OF A NUMBER AND RETURNS 1 IF IT IS AND 0 IF
C // IT IS NOT:
C
IMPLICIT INTEGER(A-Z)
DIMENSION NUMBER(10)
DATA NUMBER/1H0,1H1,1H2,1H3,1H4,1H5,1H6,1H7,1H8,1H9/

C //
C
.I=1
GO TO 99999
I= I+1
IF(.NOT.(Q.LE.0)) GO TO 99996
GO TO 99995
ISNUM=1
RETURN
CCONTINUE
GO TO 99998
CONTINUE
ISNUM=0
RETURN
END
INTEGER FUNCTION CCNV(DIGIT)

C // CONVERTS THE DIGIT INTO ITS CHARACTER REPRESENTATION.
C

IMPLICIT INTEGER(A-Z)
DIMENSION CVNUM(10)
DATA CVNUM/1H0,1H1,1H2,1H3,1H4,1H5,1H6,1H7,1H8,1H9/
CCNV=CVNUM(DIGIT+1)
RETURN
END
INTEGER FUNCTION GENLAB( Jen)
C // GENERATE A LABEL UNCONDITIONALLY IF GEN IS 1 ELSE
C // GENERATE THE LABEL IF LBFLG=0, IF LABEL FLAG
C // EQUAL 1 AND GEN EQUAL 0 THEN USE THE EXISTING
C // LABEL. NUMB IS INITIALIZED TO 100000. THIS IS AN
C // INTEGER FUNCTION RETURNING A POINTER TO THE LABEL.
C //
IMPLICIT INTEGER(A-Z)
LOGICAL LBFLG
GCMCN/INPUT/SPIN(80),TOPEC,TCEND,STBEG,STEND,IMAX,LBFLG
COMMON/CENTRL/STACK(200),LABEL(200),TCP,LBMAX,LBPTR,STFRM(6),
  TPTTR,NEUTR,BKWR,C,FRWR,C,NEW,OLD,PULOUT,STMAX,NUMB
C //
IF (.NOT.(.NOT.LBFLG.OR.GEN.EQ.1)) GC TO 99999
  NUMB=NUMB-1
  N=NUMB
  I=1
  GO TO 99998
99997
  I= I+1
99998
  IF (.NOT.(( I- (1.5) ) .LE. 0)) GO TO 99995
  M=N/10
  C // SHIFT RIGHT
  NR=N-(10*M)
  C // REMAINDER
  HCLD=5-I+LBPTR
  LABEL(HOLD)=CCVN(NR)
  C // LBPTR IS LEFT AT FIRST BLANK SPACE.
  N=M
99996
  GC TO 99997
99995
  CONTINUE
99994
  GENLAB=LBPTR
  C // NEW LABEL PTR.
  LBPTR=LBPTR+5
  C // POINT TO NEW BLANK LOCATION.
  IF (.NOT.(LBPTR.GT.LBMAX)) GC TO 99994
  CALL ERRCR(9,0,0,5,0,0)
C // NESTING TOO DEEP, TOO MANY LABELS PROGRAM TERMINATED.
0024 99994 CONTINUE
0025 99993 GO TC 99993
0026 99992 CONTINUE
0027 99991 I = 1
0028 99990 GO TC 99992
0029 99991 I = I + 1
0030 99992 IF (.NOT. (( I - (5)) * 1E0)) GO TO 99989
0031 99990 HCID = LBPTR + I - 1
0032 99991 LABEL(HOLD) = SPIN(I)
0033 99990 GO TO 99991
0034 99989 CONTINUE
0035 99990 GENLAB = LBPTR
0036 99989 LBPTR = LBPTR + 5
0037 99988 IF (.NOT. (LBPTR > LBMX)) GO TO 99988
0038 99987 CALL ERRCR(8, 0, 0, 5, 0, 0)
0039 99988 CONTINUE
0040 99993 CONTINUE
0041 99992 RETURN
0042 99991 END

B-55
SUBROUTINE CHECK(STYPE, ECCRND)
  IMPLICIT INTEGER(A-Z)
  LOGICAL ECCRND, DUMP, LBFLG
  COMMON/INPUT/SPIN(80),TOEG,TOEND,STBEG,STEND,IMAX, LBFLG
  COMMON/RTINF/IFBUF(RO),FPTR,FMAX,FSTRT,BLNKS(131)
  COMMON/SRINF/SSTRT,INCR,STMND,BLKN,NESTNO,INDFLG,CURFLG,SMAX
  COMMON/CTPAT/GOTO(5),CNTI(9),EQL(1),GOTO(6),IFNE(9),LEO(5)
  COMMON/STTYPI/CCASE,COLCN,COLSL,EELSE,EEND,ECJ,EXIT,FFOR,IF,
  LLCOP,REPT,UNT1,WHILE,SUBFR,NCASE,ENCCAS,EENDIF
  COMMON/CENTRL/STACK(200),LABEL(200),TCP,LBMAX,LPTR,STFRM(6),
  TPIR,NEUR,PKHRD,FKHRD,NEH,OLD,PLLOUT,STMAX,NUMB
  COMMON/INPAT/CCLN(1),ELZ(4),DBSLH(2),DCO(2),BYE(2),THENN(4),TOO
  COMMON/TABLES/TRES(23),FESRD(200)
  COMMON/KEYWC/EEEND(3),EEEXT(4),CCYCLE(5),UUNTL(5),CASS(4),IIIF(2),
  ITERAT(9),ENCASS(7),ENIF(5),REPT(6),CCLON(5)
  ECCRN = .FALSE.
  IF(.NOT.(TOP.EQ.0)) GO TO 99999
  CALL ERROR(4,0,0,3,0,0)
  C    ERROR---STACK EMPTY, STATEMENT IGNORED.
  ECCRN = .TRUE.
  RETURN
  CONTINUE
  STTOP=STACK(TOP)
  IF(.NOT.(STYPE.EQ.COLCN)) GO TO 99997
  IF(.NOT.(STTOP.NE.ECASE)) GO TO 99996
  CALL ERROR(2,CASS,4,5,CCLON,5)
  C    ERROR---MISSING CASE STATEMENT, CCLON IGNORED.
  TCEND=TCEND+1
  ECCRN = .TRUE.
  CONTINUE
  GO TO 99998
  IF(.NOT.(STYPE.EQ.EELSE)) GO TO 99995
0028 IF (.NOT. (STTCP.NE.IIF )) GO TO 99994
0029 C
0030 CALL ERROR(2,IIIF,2,4,ELZ,4)
0031 // ERROR---MISSING IF STATEMENT, ELSE IGNORED.
0032 ECCND=.TRUE.
0033 CONTINUE
0034 GO TO 99998
0035 99994
0036 IF (.NOT. (STYPE.EQ.UNTL )) GO TO 99993
0037 IF (.NOT. (STTCP.NE.LLCP )) GO TO 99992
0038 CALL ERROR(5,ITERAT,9,4,ULNTL,5)
0039 C
0040 // ERROR---MISSING LCOP STATEMENT, UNTIL IGNORED.
0041 CALL SCAN(REPT,3,DUMM)
0042 IF (.NOT. (DUMM )) GC TO 99991
0043 TOEND=TCEND+1
0044 GO TO 99990
0045 99991
0046 CONTINUE
0047 TCEND=TCEND+6
0048 CONTINUE
0049 ECCND=.TRUE.
0050 CONTINUE
0051 99990
0052 GO TO 99998
0053 99993
0054 IF (.NOT. (STYPE.EQ.EENC )) GO TO 99985
0055 IF (.NOT. (STTCP.NE.SUEFN )) GO TO 99988
0056
0057
C // UNSTACK UNT L A SUBFN IS FOUND ON THE STACK.
0048 TTCP=TCP
0049 CONTINUE
0050 TTCP=TTOP-STRFM(STTOP)
0051 IF(.NOT.( TTOP.LE.0 )) GO TO 9984
0052 CALL ERRCR(3, EEEND, 3, 4, EEEND, 3)
0053 C // ERRCR---TCC MANY ENDS, END IGNORED.
0054 ECCND=.TRUE.
0055 RETURN
0056 CONTINUE
0057 STTCP=STACK(TTOP)
0058 IF(.NOT.( STTCP.EQ.SLBFN )) GO TC 99987
0059 CONTINUE
0060 TCP=TTCP
0061 CALL ERRCR(7, 0, 0, 6, 0)
0062 C // PREMATURE END SUBPROGRAM CLOSED.
0063 FSTRT=7
0064 SSTRT=7
0065 CONTINUE
0066 GO TO 95998
0067 99989 IF(.NOT.( STYPE.EQ.ENDCAS )) GO TO 99983
0068 IF(.ACT. ( STTOP.NE.CCASE )) GO TO 99982
0069 CALL ERRCR(3, ENCASS, 7, 4, ENCASS, 7)
0070 C // ERRCR---TCC MANY ENDCASES, ENDCASE IGNORED.
0071 ECCND=.TRUE.
0072 CONTINUE
0073 GO TO 99998
0074 99982 IF(.NOT.( STYPE.EQ.EENCIF )) GO TO 99981
0075 IF(.ACT. ( STTOP.NE.IIF )) GO TC 99980
0076 CALL ERRCR(3, ENIF, 5, 4, ENIF, 5)
0077 C // ERRCR---TCC MANY ENDIIFS, ENDIF IGNORED.
0078 ECCND=.TRUE.
0079 CONTINUE
0080 GO TO 99998
0081 99983 IF(.NOT.( STYPE.EQ.REPTE )) GO TO 99979
0082 IF(.ACT. ( STTOP.EQ.FFRR.OR. STTCP.EQ.LLCCP.OR. STTOP.EQ.WHLE ) )
0379 ) GC TO 99978
0380 RETURN
0381 GC TO 99977

$ 0382 99978 CONTINUE
0383 CALL ERROR(5,ITERAT,9,4,REPT,6)
0384 C \// ERROR—NO ITERATIVE STATEMENT FOUND, REPEAT IGNORED.
0385 ECCND=.TRUE.
0386 99977 CONTINUE
0387 GO TO 99978
0388 99979 IF(.NOT.( STYPE.EQ.EEXIT )) GO TO 99976
0389 IF(.NCT.( TPIIR.EQ.0 )) GO TO 99975
0390 CALL ERROR(5,ITERAT,9,4,EEEXIT,4)
0391 C \// ERROR—NO ITERATIVE STATEMENT, EXIT IGNORED.
0392 ECCND=.TRUE.
0393 99975 CONTINUE
0394 GO TO 99978
0395 99976 IF(.NOT.( STYPE.EQ.NCYCLE )) GO TO 99974
0396 IF(.NCT.( TPIIR.EQ.0 )) GO TO 99973
0397 CALL ERROR(5,ITERAT,9,4,CCYCLE,5)
0398 C \// ERROR—NO ITERATIVE STATEMENT, CYCLE IGNORED.
FORTRAN IV G LEVEL 21
CHECK
DATE = 77326
20/47/11

0CS5      ECCND=.TRUE.
0096  99973    CONTINUE
0CS7  99974    CONTINUE
0CS98  99993    CONTINUE
0CS59    RETURN
0100      END
SUBROUTINE ERROR(CODE1,PARM1,LEN1,CODE2,PARM2,LEN2)

C      // CODE1 IS ERRCR MESSAGE CODE
C      // CODE2 IS THE ACTION TAKEN
C      // PARM1 AND PARM2 MODIFY THE TWO
C      // LEN1 AND LEN2 ARE THE LENGTH OF MODIFYING PATTERN
C

IMPLICIT INTEGER(A-Z)
COMMON/INPLT/SPIN(80),TOBEG,TCEND,STBEG,STEND,IMAX,LBFLG
COMMON/STYP2/COMT,CENT,FRTR,FFILE,CROR
DIMENSION PARM1(1),PARM2(1),ERPAT(45,8),PCS\(8),LEN(8),
ACPAT(35,6),ALEN(6),APS\(6)

DATA EK\(1HL,1HA,1HE,1HL,1HI,1HN,1HG,1H,1HT,1HH,1HE,
1   1H,1F,1F,1F,1H,1H,1H,1H,1H,1H,1H,1H,1H,1H,
1   1H,1H,1H,1H,1H,1H,1H,1H,1H,1H,1H,1H,1H,1H,
1   1H,1H,1H,1H,1H,1H,1H,1H,1H,1H,1H,1H,1H,1H,
1   1H,1H,1H,1H,1H,1H,1H,1H,1H,1H,1H,1H,1H,1H,
1   1H,1H,1H,1H,1H,1H,1H,1H,1H,1H,1H,1H,1H,1H,
1   1H,1H,1H,1H,1H,1H,1H,1H,1H,1H,1H,1H,1H,1H,
1   1H,1H,1H,1H,1H,1H,1H,1H,1H,1H,1H,1H,1H,1H,
1   1H,1H,1H,1H,1H,1H,1H,1H,1H,1H,1H,1H,1H,1H,
1   1H,1H,1H,1H,1H,1H,1H,1H,1H,1H,1H,1H,1H,1H,
1   1H,1H,1H,1H,1H,1H,1H,1H,1H,1H,1H,1H,1H,1H,
1   1H,1H,1H,1H,1H,1H,1H,1H,1H,1H,1H,1H,1H,1H,
1   1H,1H,1H,1H,1H,1H,1H,1H,1H,1H,1H,1H,1H,1H,
1   1H,1H,1H,1H,1H,1H,1H,1H,1H,1H,1H,1H,1H,1H,
1   1H,1H,1H,1H,1H,1H,1H,1H,1H,1H,1H,1H,1H,1H,
1   1H,1H,1H,1H,1H,1H,1H,1H,1H,1H,1H,1H,1H,1H,
1   1H,1H,1H,1H,1H,1H,1H,1H,1H,1H,1H,1H,1H,1H,
1   1H,1H,1H,1H,1H,1H,1H,1H,1H,1H,1H,1H,1H,1H,
1   1H,1H,1H,1H,1H,1H,1H,1H,1H,1H,1H,1H,1H,1H,
1   1H,1H,1H,1H,1H,1H,1H,1H,1H,1H,1H,1H,1H,1H,
1   1H,1H,1H,1H,1H,1H,1H,1H,1H,1H,1H,1H,1H,1H,
1   1H,1H,1H,1H,1H,1H,1H,1H,1H,1H,1H,1H,1H,1H,
1   1H,1H,1H,1H,1H,1H,1H,1H,1H,1H,1H,1H,1H,1H,
1   1H,1H,1H,1H,1H,1H,1H,1H,1H,1H,1H,1H,1H,1H,
1   1H,1H,1H,1H,1H,1H,1H,1H,1H,1H,1H,1H,1H,1H,
1   1H,1H,1H,1H,1H,1H,1H,1H,1H,1H,1H,1H,1H,1H,
1   1H,1H,1H,1H,1H,1H,1H,1H,1H,1H,1H,1H,1H,1H,
1   1H,1H,1H,1H,1H,1H,1H,1H,1H,1H,1H,1H,1H,1H,
1   1H,1H,1H,1H,1H,1H,1H,1H,1H,1H,1H,1H,1H,1H,
1   1H,1H,1H,1H,1H,1H,1H,1H,1H,1H,1H,1H,1H,1H,
1   1H,1H,1H,1H,1H,1H,1H,1H,1H,1H,1H,1H,1H,1H,
1   1H,1H,1H,1H,1H,1H,1H,1H,1H,1H,1H,1H,1H,1H,
1   1H,1H,1H,1H,1H,1H,1H,1H,1H,1H,1H,1H,1H,1H,
1   1H,1H,1H,1H,1H,1H,1H,1H,1H,1H,1H,1H,1H,1H,
1   1H,1H,1H,1H,1H,1H,1H,1H,1H,1H,1H,1H,1H,1H,
1   1H,1H,1H,1H,1H,1H,1H,1H,1H,1H,1H,1H,1H,1H,
1   1H,1H,1H,1H,1H,1H,1H,1H,1H,1H,1H,1H,1H,1H,
1   1H,1H,1H,1H,1H,1H,1H,1H,1H,1H,1H,1H,1H,1H,
1   1H,1H,1H,1H,1H,1H,1H,1H,1H,1H,1H,1H,1H,1H,
1   1H,1H,1H,1H,1H,1H,1H,1H,1H,1H,1H,1H,1H,1H,
1   1H,1H,1H,1H,1H,1H,1H,1H,1H,1H,1H,1H,1H,1H,
1   1H,1H,1H,1H,1H,1H,1H,1H,1H,1H,1H,1H,1H,1H,
1   1H,1H,1H,1H,1H,1H,1H,1H,1H,1H,1H,1H,1H,1H,
1   1H,1H,1H,1H,1H,1H,1H,1H,1H,1H,1H,1H,1H,1H,
1H, 1H, 1H, 1H, 1H, 1H, 1H, 1H, 1H, 1H, 1H, 1H, 1H, 1H, 1H, 1H, 1H, 1H, 1H
2IHL, 1HA, 1HB, 1HE, 1HL, 1H, 1FC, 1HE, 1FL, 1HE, 1HT, 1HE, 1HD, 2IHF, 1HR, 1HG, 1HM, 1H, 1HF, 1FC, 1HR, 1HT, 1HR, 1HA, 1HN, 1H, 2IHO, 1HU, 1HT, 1HP, 1HU, 1HT, 1F, 1H, 1H
3IH5, 1HT, 1HA, 1HT, 1HE, 1HM, 1FE, 1HN, 1HT, 1H, 3IH5, 1HK, 1HI, 1HP, 1HP, 1HE, 1FD, 1H, 1H, 1H, 1H, 1H, 1H, 1H, 1H, 1H
4IH, 1H, 1H, 1H, 1H, 1H, 1H, 1H, 1H, 1H, 1H, 1H, 1H, 1H, 1H, 1H, 1H, 1H
4IH, 1H, 1H, 1H, 1H, 1H, 1H, 1H, 1H, 1H, 1H, 1H, 1H, 1H, 1H, 1H, 1H, 1H
4IH, 1H, 1F, 1H, 1H, 1H, 1H
5IH, 1HR, 1HC, 1HG, 1HF, 1HA, 1HM, 1H, 1HT, 1HE, 1HR, 1HM, 1HI, 1HN, 1HA, 5IH4, 1HE, 1FD, 17*1H
6IH5, 1HU, 1HB, 1HP, 1HR, 1HC, 1HG, 1HR, 1HA, 1HM, 1H, 1FC, 1FL, 1HC, 1HS, 6IH5, 1HD, 18*1H /
DATA ALEN/0,0,0,0,0/
DATA APSK/0,0,1,0,0/
C
0013 WRITE(PRTR,2222) SPIN
0014 2222 FORMAT(26H "++++OFFENDING CARD: 80A1")
0015 LENTH=LEN(CCCE1)
0016 ALNTH=ALEN(CCDE2)
0017 PCS=POSK(CCCE1)
0018 APCS=APS(A(CD2E))
0019 I=1
0020 GC TO 99999
0021 99998 I= I+1
0022 99999 IF(.NOT.(I-(LEN1)).LE.0)) GO TO 9996
0023 99996 IT=PCS+I-1
0024 ERPAT(IT,CCDE1)=PARM1(I)
0025 GO TO 99998
0026 99997 CONTINUE
0027 IF(.NOT.(Lething.GT.LEN1)) GO TO 9995
0028 PC=PCS+Len
0029 PE=PCS+LENTH-1
0030 I=PC
0031 GC TO 99994
0032 99999 I= I+1
0033 99994 IF(.NOT.(I-(PE)).LE.0)) GO TO 9991
0034 99991 ERPAT(I,CODE1)=BLANK
0035 GO TO 9993
0036 99992 CONTINUE
0037 99995 CONTINUE
0038 I=1
0039 GC TO 99990
0040 99999 I= I+1
0041 99990 IF(.NOT.(I-(LEN2)).LE.0)) GC TO 99987
0042 99987 IT=APC(I-1)
0043 ACPAT(IT,CCDE2)=PARM2(I)
0044 99988 GC TO 99989
0045 99987 CONTINUE
0046 IF(.NOT.(ALENTH.GT.LEN2)) GO TO 99986
0047 PC=APC(I-1)
0048 PE=APC(I-ALNTH-1
0049 I=PC
0050 GC TO 99985
0051 99984 I= I+1
0052 99985 IF(.NOT.(I-(PE)).LE.0)) GC TO 99982
0053 99982 ACPAT(I,CODE2)=BLANK
0054  99983   GO TO C 99984
0055  99982   CONTINUE
0056  99986   CONTINUE
0057       WRITE (PRTR,10)  (ERPAT(I,CODE1), I=1,45)
0058       WRITE (PRTR,10)  (ACPAT(I,CODE2), I=1,35)
0059       10   FCRM1AT (12H  +++++++, 80A1)
0060       RETURN
0061       END
SUBROUTINE TYHAND(TYPE, EEXT, CCYCLE, EEXIT)
IMPLICIT INTEGER(A-Z)
LOGICAL LBFLG
COMMON/INPUT/SPIN(80), TOBEG, TGEND, STBEG, STENC, IMAX, LBFLG
COMMON/SPRIN/STRT, INCR, STMNG, BLKN, NESTNO, INCFLG, CURFLG, SMAX
COMMON/FRIN/FBUF(80), FPTR, FXAT, FSTR, BLNKS(121)
COMMON/CTPRT/CGDT(9), CCATI(8), EQL(1), GOTO(6), IFNOT(9), LEO(5),
1 MINUS(1), ONE(1), PLL(1), RSTAR(2)
COMMON/CENTRL/STACK(20C), LABEL(200), TCP, LMAX, LBPTR, STRFM(6),
1 TPIR, NEUTR, BKWRC, FKWRK, NEW, OLD, PULOUT, STMAX, NUMB
COMMON/STYP2/CCMT, CCNT, FCTR, BLANK, SEMI

//

C IF(.NOT.( LBFLG )) GC TC 99999
0011 IF(.NOT.( TYPE.EQ.EEXIT )) GO TO 99998
0012 CALL ERROR(1, EEXT, 4, 2, 0, 0)
C // ERROR--LABELLING THE EXIT HAMPERS PROGRAM CLEANLINESS--
C 9
C // --DELETED FROM THE FORTRAN OUTPUT.
0013 GO TO 99997
0014 CONTINUE
0015 CALL ERROR(1, CCYCLES, 5, 2, 0, 0)
C // ERROR---LABELLING THE CYCLE HAMPERS PROGRAM
C // CLEANLINESS, DELETED FROM FORTRAN OUTPUT.
0016 CONTINUE
0017 CONTINUE
C // CHECK IF A NUMERIC LABEL FCLLCW.
0018 K=TOEND+1
0019 CONTINUE
0020 IF(.NOT.( K.GT.IMAX )) GC TC 99993
0021 I=0
0022 GC TC 99994
0023 CONTINUE
0024 IF(.NOT.( SPIN(K).NE.BLANK )) GC TC 99992
0025 I=ISKUM(SPIN(K))
0026 GC TC 99994
0027 CONTINUE
K = K + 1
GOTO 99996
CONTINUE
IF (.NOT. (I .EQ. 0)) GO TO 99991
I = 1
GOTO 99990
I = I + 1
IF (.NOT. ((I-(6)) .LE. 0)) GO TO 99987
F8UI(I) = BLANK
GOTO 99989
CONTINUE
IF (.NOT. (TYPE .EQ. EXIT)) GO TO 99986
K = STACK(TPITR-1)
GOTO 99985
CONTINUE
K = STACK(TPITR-2)
CONTINUE
CALL FOUT(GCTC,1,6)
CALL FOUT(LABEL,K,K+4)
CALL FPRTNT
GOTO 99984
CONTINUE
TGENC=K
CALL SCAN1(BLANK, TCEND)
WRITE(PRTR,4999)
FORMAT( '+++++++LABELLED EXIT-CYCLE NOT IMPLEMENTED')
CONTINUE
STEND=TCEND
INDFLG=NEUTR
CURFLG=CLD
RETURN
END
TRANSLATION OF THE SPARKS PREPROCESSOR FROM FORTRAN IC
SPARKS

by

RICHARD MANSON SIBOUT

B.S., UNIVERSITY OF TEXAS, ARLINGTON, 1972

AN ABSTRACT OF A MASTER'S REPORT

submitted in partial fulfillment of the
requirements for the degree

MASTER OF SCIENCE

Department of Computer Science

KANSAS STATE UNIVERSITY
Manhattan, Kansas

1978
This report describes the translation of the SPARKS preprocessor from FORTRAN to SPARKS. The SPARKS preprocessor is a program which translates other programs written in the SPARKS language to a target language of FORTRAN.

The SPARKS language adds additional constructs to FORTRAN so that structured code can be written while maintaining a close tie to FORTRAN. The preprocessor written in SPARKS is a structured program with the necessary modularity so that changes and updates can be made easily. The preprocessor in SPARKS was examined in detail for code that could be improved. Approximately 375 lines of code were found to be unnecessary and were eliminated. The rewritten preprocessor is a 1470 line program versus the 1845 lines of the original FORTRAN version. The report includes updated documentation and user's guide for the SPARKS language.