ENHANCEMENT OF SPARKS,
A FORTRAN
PREPROCESSOR

BY

JOHN JOSEPH MARTIN JR.
B.S., UNIVERSITY OF SCRANTON, 1973

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

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:

[Signature]
Major Professor
# TABLE OF CONTENTS

**CHAPTER 1** INTRODUCTION  
1.1 PURPOSE OF THE REPORT ............................................. 1  
1.2 GENERAL BACKGROUND OF SPARKS .................................... 1  
1.3 WHAT IS SPARKS ......................................................... 2  
1.4 WHY SPARKS ............................................................. 4  

**CHAPTER 2** COSMETIC MODIFICATIONS  
2.1 INTRODUCTION ............................................................. 5  
2.2 *SPARKS STATEMENT .................................................... 5  
2.3 COMMENT STATEMENT .................................................... 5  
2.4 THE SEMICOLON IN SPARKS OUTPUT ................................. 6  
2.5 EXIT AND CYCLE STATEMENT ........................................... 7  
2.6 SUBROUTINE SCAN1 ..................................................... 8  
2.7 SUMMARY ................................................................. 8  

**CHAPTER 3** MODIFICATIONS CODE RELATED  
3.1 INTRODUCTION ............................................................. 9  
3.2 SINGLE-CHARACTER VARIABLE ......................................... 9  
3.3 LABEL TABLE ............................................................. 10  
3.4 SEMICOLON DETECTION .................................................. 11  
3.5 EXIT CYCLE AND RETURN STATEMENTS ............................... 12  
3.6 DATA AND FORMAT STATEMENTS ...................................... 13  
3.7 SUMMARY ................................................................. 15  

**CHAPTER 4** ADDITION OF NEW FEATURES  
4.1 INTRODUCTION ............................................................. 16  
4.2 COMMENTS ................................................................. 16  
4.3 MULTIPLE ASSIGNMENT STATEMENT ................................... 18  
4.4 CASE STATEMENTS ....................................................... 19  
4.5 MULTILEVEL EXIT AND CYCLE STATEMENTS ......................... 21  
4.6 ERROR RECOVERY ....................................................... 23  
4.7 SUMMARY ................................................................. 23  

**APPENDIX A** KSU SPARKS USER'S MANUAL .......................... A-1  
**APPENDIX B** KSU SPARKS CODE SPECIFICATIONS .................... B-1  
**APPENDIX C** KSU SPARKS CODE .......................................... C-1  
**APPENDIX D** REFERENCES .................................................. D-1
THIS BOOK CONTAINS NUMEROUS PAGES WITH THE PAGE NUMBERS CUT OFF

THIS IS AS RECEIVED FROM THE CUSTOMER
THIS BOOK CONTAINS NUMEROUS PAGES WITH DIAGRAMS THAT ARE CROOKED COMPARED TO THE REST OF THE INFORMATION ON THE PAGE. THIS IS AS RECEIVED FROM CUSTOMER.
### LIST OF FIGURES

<table>
<thead>
<tr>
<th>FIGURE</th>
<th>DESCRIPTION</th>
<th>PAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1</td>
<td>SPARKS DATA FLOW GRAPH</td>
<td>3</td>
</tr>
<tr>
<td>4.1</td>
<td>MULTIPLE ASSIGNMENT STATEMENT GRAPH</td>
<td>18</td>
</tr>
<tr>
<td>4.2</td>
<td>SYNTAX GRAPH FOR CASE STATEMENT</td>
<td>19</td>
</tr>
<tr>
<td>4.3</td>
<td>SYNTAX GRAPH FOR CASE OF STATEMENT</td>
<td>20</td>
</tr>
<tr>
<td>4.4</td>
<td>SYNTAX GRAPH FOR EXIT AND CYCLE STATEMENTS</td>
<td>22</td>
</tr>
</tbody>
</table>
CHAPTER 1
INTRODUCTION

1.1 PURPOSE OF THE REPORT

The purpose of this report is to describe an enhancement of the SPARKS preprocessor and to provide documentation for changes made. Enhancement of SPARKS can be divided into three levels of effort: The first level consists of changes made to the original program which have no effect on the actual code produced, but adds to the readability of it. The second level is the removal of errors which produce improper code or limitations which restrict the user in its use. The third level is the addition of new features. Some features are totally new while others are expansions of features already in SPARKS. These features allow the user more flexibility, add readability, and make some attempt at error recovery. The documentation for all of these changes includes specifications for each module, high level code, data flow, and module access graphs. A User's Manual is also provided to give detailed information on the use of SPARKS including the new features.

1.2 GENERAL BACKGROUND OF SPARKS

The SPARKS language was first designed for describing algorithms in the book, Fundamentals of Data Structures, by Dr. Ellis Horwitz and Dr. Sartaj Sahni, published by Computer Science Press, Woodland Hills, California. Since
that time the implemented language has been evolved to the extent that it no longer matches the book version exactly.

The SPARKS preprocessor was first written in the SPARKS language. In order to produce the first running system it had to be hand translated into FORTRAN. This version is in use at the University of Southern California, and can be obtained from there at a nominal cost. The SPARKS preprocessor has never been proven correct.

A copy of the translator was ported to Kansas State University by Dr. Myron A. Calhoun in 1976. It was provided for use by the students and faculty at Kansas State University. The FORTRAN version has since been translated back into SPARKS by Richard M. Stroud [3]. The new SPARKS translator was implemented by using the FORTRAN version as a bootstrap.

1.3 WHAT IS SPARKS

In simple terms the SPARKS preprocessor is a program which translates code written in the SPARKS language into standard FORTRAN. The main program reads the SPARKS input, then determines the types of statement and branches to the appropriate parts of the program to process the statements. Figure 1.1 contains a flow graph of the SPARKS translator.

The SPARKS translator will recognize defined SPARKS statements and those indicated as pure FORTRAN statements. All others are assumed to be FORTRAN statements and passed to the FORTRAN output file. For each SPARKS statement encountered, the translator will produce FORTRAN statements
which accomplish the equivalent computation. This generated FORTRAN code is then passed to the FORTRAN output file. All input statements are passed to the SPARKS output file for printing.

![SPARKS DATA FLOW GRAPH](image)

**FIGURE 1.1 SPARKS DATA FLOW GRAPH**

A part of the structuring feature of SPARKS is to produce an output which is indented by structure group, as loops, case, etc. This allows the user to enter all input code in relatively free form and receive a neatly indented output. The User's Manual (Appendix A) gives a detailed listing of the features of the SPARKS preprocessor.
1.4 **WHY SPARKS**

The first two decades of high-level programming languages have witnessed the birth and death of many languages. Since FORTRAN was one of the very first high-level languages, many people feel that its death is long overdue. Yet, it remains the language of use in the overwhelming majority of scientific applications of computers and no end to this situation is in sight. Recent advancements in programming techniques have demonstrated that FORTRAN has many deficiencies, but in spite of these deficiencies its usage remains high. Moreover, its evolution through the standardization process is very slow, too slow to offer any hope for major changes in the near future.

The SPARKS preprocessor offers a way into the future without having to wait for the evolutionary changes to take place. It adds many of the new features of good structured programming to present FORTRAN compilers. And, perhaps, the widespread use of SPARKS will somehow speed the evolution change of standard FORTRAN.
CHAPTER 2
COSMETIC MODIFICATIONS

2.1 INTRODUCTION

This chapter describes changes to the original SPARKS program which have no effect on the output code other than to make it more readable. Most of the changes were made by adding, deleting, or making minor modifications to several lines of SPARKS code. Since in some instances, restrictions enforced by SPARKS were relaxed this chapter serves as a record of those changes.

2.2 *SPARKS STATEMENT

According to the original SPARKS translator all programs were required to start with *SPARKS and the asterisk must be located in column 1. If this statement were omitted or if the asterisk were not in column 1, then the translator would continue to search for the statement. If not found, the program would be terminated with no output. It was felt that *SPARKS was not a useful requirement and was thus removed from the code. The SPARKS preprocessor now assumes the first card after the JCL to be part of the SPARKS input. This change was accomplished by the deletion of the test and subsequent statements at the beginning of the main program.

2.3 COMMENT STATEMENT

Within the SPARKS program, the doubleslash is used as a
delimiter for comments. A comment in SPARKS code was originally written as follows:

    // THIS IS A COMMENT //

This comment would appear in both the SPARKS and FORTRAN output files as follows:

    SPARKS:   C  // THIS IS A COMMENT //
    FORTRAN:  C  // THIS IS A COMMENT

In both cases, the 'C' was placed in column 1 and the comment starting with the doubleslash was indented according to the structuring rules. For some unknown reason, the trailing doubleslash was omitted from the FORTRAN version. Since the SPARKS comment was already recognizable by the doubleslash, the 'C' in column 1 served no purpose so it was deleted from the SPARKS output. Furthermore, the FORTRAN code is normally used only by the compiler, and is normally not ever seen by the programmer, therefore comments were deleted from the FORTRAN output.

2.4 THE SEMICOLON IN SPARKS OUTPUT

The semicolon is used as a delimiter to allow more than one statement per line of code. For example:

    A=B+C;C=D+E;X=A

The preprocessor originally expanded this line into three lines for both the the SPARKS and FORTRAN files as follows:

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

    FORTRAN
    A=B+C
    C=D+E
    X=A

-6-
The semicolon remained with the SPARKS version, but is removed from the FORTRAN, since the compiler does not allow semicolons as part of a statement. It was felt that the semicolon distracted from the readability and therefore it was removed from the SPARKS output.

2.5 EXIT AND CYCLE INDENTATION

SPARKS allows two options for transferring control within a loop, the EXIT and CYCLE. The following is an example of the use of the EXIT and CYCLE statements, as they appeared in the indented SPARKS output listing.

OLD

LOOP
  IF X=Y THEN
  EXIT
  ELSE
  CYCLE
  ENDDO
  X=X+1
REPEAT

Since the EXIT and CYCLE are subordinate statements of the IF statement, then they should be indented to show dependency. A change was made to the program so that the output now looks as follows:

NEW

LOOP
  IF X=Y THEN
  EXIT
  ELSE
  CYCLE
  ENDDO
  X=X+1
REPEAT

-7-
2.6 **SUBROUTINE SCAN1**

Within the SPARKS translator the SCAN1 subroutine is used to look for a particular character in a given line. It searches a line from a given starting point to the end of the line, if necessary. If the character is not found the routine returns an end-of-line condition at column 72. However, many times statements occupy only the first 20 columns or so, yet, the rest of the program was required to process the full 72 columns as code. The SCAN1 routine was changed so that when an end condition is encountered, a pointer indicates the last non-blank character.

2.7 **SUMMARY**

The changes covered in this chapter can be classified as cosmetic in nature. They improve the readability of the output or enhance internal processing slightly, but have no effect on the performance of the generated code. Information as to the use of these changes can be found in the User's Manual described in (Appendix A); changes made to the code are contained in Appendix C.
CHAPTER 3
MODIFICATIONS CODE RELATED

3.1 INTRODUCTION
As stated earlier, the SPARKS preprocessor had not been proven correct. Use of the preprocessor at Kansas State University revealed several errors in the code produced. Additional limitations were detected when the translated version of SPARKS was implemented by R. Stroud [3]. This chapter describes modifications made to the program in order to remove the remaining known errors. However, SPARKS has still not been proven correct in the Software Engineering aspect.

3.2 SINGLE-CHARACTER VARIABLES
During normal operation of SPARKS it was noted that for certain single-character variables an error appeared in the output code. For a given input statement such as:

\[ L = X + Y \]

where the variables are surrounded by blanks, the following output would result.

\[
\begin{align*}
\text{SPARKS} & \quad \text{FOTRAN} \\
L & = X + Y \\
= X + Y
\end{align*}
\]

The SPARKS translator processed the single-character variable as a keyword and placed it on a separate line, with the remaining part of the statement on the next line. The FORTRAN output dropped the variable entirely. Further
investigation revealed that this happened only to those variables which had the same first letter as a SPARKS keyword. In order for a keyword to be recognized SPARKS requires that they be surrounded by blanks. However, one of the test cases in subroutine STIYP failed to recognize that just because a single-character variable is surrounded by blanks and starts with the same first letter as a SPARKS keyword, that variable is not necessarily a SPARKS keyword.

This test was rewritten so that it now selects only those statements that are truly SPARKS.

3.3 LABEL TABLE

Once a SPARKS statement has been recognized, and depending on the type of statement, a certain number of internal labels are generated. These labels are then stored in a table until needed by the program. The size of the table is fixed at 40 nodes.

During the translation of a very large SPARKS program the preprocessor abnormally terminated because of a label table overflow. An investigation of the input code indicated that the nesting level was not such to cause the table to overflow. Further investigation revealed that the SPARKS program did not reclaim all of the unused label nodes. This error was cumulative and not detectable in small programs, but a large program would cause the overflow. The cause of this condition was isolated to two statements, the CASE and IF-THEN-ELSE statements. When the normal IF-THEN statement was used the program functioned normally. However when the
ELSE option was used one additional label had to be generated. This label node was never reclaimed. The error was much more severe in the CASE statement. Each time a new conditional statement was added one additional label was generated. Depending on the number of conditional statements used within the CASE statement, that many label nodes were lost.

In order to prevent this overflow, the following modifications were made. Each time a CASE or IF-THEN-ELSE statement is encountered the current label pointer is placed on the stack. Now, when these statements are terminated this stack value is used to reset the table pointer and thus reclaim all unused nodes.

3.4 SEMICOLON DETECTION

SPARKS allows for more than one statement per line with the use of a semicolon as a delimiter. Subroutine SCAN1 is used to detect the semicolon and separate each statement into its proper form. Since FORTRAN does not use the semicolon this should present no problem. However, the semicolon may and often does appear in FORTRAN DATA and FORMAT statements. The SCAN1 routine detected this semicolon and produced two separate statements. This separation of the code without a continuation card then results in a FORTRAN error.

In order to alleviate this condition SCAN1 was changed to make one additional test. Since all SPARKS keywords and symbols are required to be surrounded by blanks, only those
semicollons would be used as delimiters. A further improvement in this area will be covered when DATA and FORMAT statements are discussed later in this chapter.

3.5 **EXIT CYCLE AND RETURN STATEMENTS**

For each SPARKS statement the translator produces FORTRAN statements which accomplish the equivalent computation. The following is an example of a piece of a typical SPARKS program and the equivalent FORTRAN code produced, where S1, S2, S3 represent any executable FORTRAN statement:

```
SPARKS
 LOOP
   S1
   IF COND1 THEN
      RETURN
   ELSE
      CASE
         :COND2:
            S2
            EXIT
         :COND3:
            S3
            CYCLE
      ENDCASE
   ENDFCASE
 REPEAT
```
The last statement generated by the IF-THEN-ELSE and CASE statements is a GO TO label. If an EXIT, CYCLE, or RETURN happen to be the last part of that statement, then an unreachable GO TO is produced as shown at points 1, 2 and 3. The FORTRAN code will usually compile and run correctly, but most compilers will give a warning that an unreachable statement was found.

To eliminate the warning and the unreachable GO TO, the SPARKS preprocessor was modified to set a flag each time an EXIT, CYCLE or RETURN is found. Then, when one of these constructs appears at the end of an IF-THEN-ELSE or CASE statement, the second GO TO statement is not produced.

3.6 DATA AND FORMAT STATEMENTS

SPARKS allows the user to write structured code. As part of this structuring the preprocessor indents all
statements subordinate to a major structure, such as programs, subroutines loops, cases, etc. Each time a new structure is encountered all subordinate code is indented three spaces. This approach offers little problem for the SPARKS output since it is displayed on a 132-character line. However, the FORTRAN output is restricted to a 72-character line. If the output code occupies the entire 72 characters and is part of a structure nested three levels deep then the last nine characters would not fit on the line. The translator recognized this fact and provided a continuation line for the remaining nine characters. This solution was adequate for all statements except DATA and FORMAT statements which contain Hollerith fields. If the break came in the middle of a field, then the number of blanks to which the line is indented after column 6 were taken to be part of the Hollerith field. This produced a mismatch between the actual size of the field and the field specified.

In order to prevent this error, either a new Hollerith field had to be specified or steps taken to insure that Hollerith fields were not broken. The latter option was selected. To insure that DATA and FORMAT statements are not broken they must first be recognized as such. This was accomplished with the addition of a new subroutine FTYP. One of the functions of this routine is to identify DATA and FORMAT statements. Once identified, they are output to SPARKS in the normal fashion, however the FORTRAN output is not indented, but is passed as received and placed in column 7-72. In addition all continuation cards following such
statements, are also not indented.

This isolation of DATA and FCRMAT statements also further resolved the problem of semicolons located within the lines. Since they are now handled separately (SCAN1 is not called) the problem of breaking lines on the wrong semicolon has been eliminated. Additionally, this eliminated the need for semicolons to be surrounded by blanks. Therefore, the SPARKS preprocessor was changed so that semicolons no longer have to be isolated by blanks.

3.7 SUMMARY

The material covered in this chapter has to do with changes to the program which corrected some errors or removed some limitations in the original SPARKS preprocessor. Detailed code for the changes is contained in Appendix C, and information on the use of SPARKS with the new code is contained in Appendix A.
CHAPTER 4

ADDITION OF NEW FEATURES

4.1 INTRODUCTION

This chapter is concerned with additions to the SPARKS preprocessor which expand the language by adding new features. These features give the user more options and flexibility when programming with SPARKS. Some of the added features were suggested as possible extensions for SPARKS in the original text [1].

4.2 COMMENTS

The SPARKS preprocessor recognizes and processes two types of comment statements, the SPARKS and FORTRAN comment. Comments are restricted to one line and in-line comments were not allowed. It was felt that in-line comments would produce more readable and better documented code. Another restriction to the SPARKS comment was that the delimiter, a doubleslash (//), had to be surrounded by blanks. If the user failed to use the blanks the translator would not recognize the statement as a SPARKS comment and, by default, it was passed as a FORTRAN statement. This statement then produced a FORTRAN error. It was decided to change the translator so that comments could appear in-line and so that blanks were not required around the doubleslash. The following is an example of the use of comments before and after the changes were made.
BEFORE

// TEST PTR IF EQUAL TO N THEN //
// THE ITEM IS NOT IN THE LIST //
// SET TEST FLAG EQUAL TO 0 //
// ELSE SET TEST FLAG EQUAL TO 1 //

IF PTR.EQ.N THEN
  TEST=0
ELSE
  TEST=1
ENDIF

AFTER

IF PTR.EQ.N THEN
  TEST=0 //ITEM NOT CONTAINED IN LIST//
ELSE
  TEST=1 //ITEM CONTAINED IN LIST//
ENDIF

Changes to the translator to accept comments which did not have the double slash surrounded by blanks required the removal of the double slash from the keyword table and making a separate test for them. This test is still conducted in subroutine STYP, but is now a test by itself. Allowing comments in-line with code was much more complex, since comments could appear after both SPARKS and FORTRAN statements. This required tests to be made throughout the entire program. An additional subroutine, SCAN2, was supplied. SCAN2 searches the line after each statement to detect the presence of a comment. Once recognized, the routine continues to search the line looking for the end of the comment. The code and comment are then passed to the SPARKS output file, while only the generated code is passed to the FORTRAN file.

For comments following FORTRAN statement SCAN1 had to
be modified. In addition to detecting a semicolon, this subroutine now also detects the doubleslash. Once the presence of a comment is detected then SCAN2 is used to find the end of the comment. Since comments are no longer being passed to the FORTRAN file, this test by SCAN2 is made after the FCRTRAN statements have been processed.

4.3 MULTIPLE ASSIGNMENT STATEMENT

The first feature added to SPARKS was the multiple assignment statement. This feature improves readability, reduces the writing required for initialization, and is easy to understand and use by the programmer. The multiple assignment statement is a modification of the standard FORTRAN assignment statement as shown by the syntax graph in figure 4.1

![Syntax graph of multiple assignment statement](image)

**SEMANTICS**

VAR: any standard FCRTRAN variable
EXP: any legal FCRTRAN expression

**FIGURE 4.1 MULTIPLE ASSIGNMENT STATEMENT SYNTAX GRAPH**

The SPARKS translator recognizes the multiple assignment statement and produces individual statements for the FORTRAN file. The source statements are passed as received to the SPARKS file. The following are examples of
SPARKS input and FORTRAN output.

**SPARKS**

SMALL, LAST, DATA(I) = VALUE

I, J, K, COUNT = 0

**FORTRAN**

SMALL = VALUE
LAST = VALUE
DATA(I) = VALUE

I = 0
J = 0
K = 0
COUNT = 0

START, END = X + Y / 10

Implementation of this feature required the ability to recognize the statement first and then separate it into its component parts. A new subroutine PTYF is used to recognize the statements and identify the individual parts. With some modifications subroutine PHAND was able to output the FORTRAN code.

4.4 CASE STATEMENT

The CASE statement was augmented to allow a new form of the statement. The original CASE statement had the structure as shown in figure 4.2.

![Syntax Graph for CASE Statement](image)

**Semantics**

COND: any FORTRAN Boolean expression
STMT: any FORTRAN/SPARKS statement

**Figure 4.2 Syntax Graph for CASE Statement**
The case statement allowed the user to list any number of test conditions as necessary. Only that group of statements associated with the true condition would be executed. If the ELSE option is used, then if all other conditions were false, the ELSE block is executed. The following example of this statement is given.

```
CASE
  : LENGTH .EQ. 10 :
     SIZE = SMALL
  : LENGTH .EQ. 50 :
     SIZE = MEDIUM
  : LENGTH .EQ. 100 :
     SIZE = LARGE
ENDCASE
```

The added feature of the CASE statement has the structure as shown in figure 4.3.

![Syntax Graph for CASE CF Statement]

**SEMANTICS**

STMT: any FORTRAN/SPARKS statement
EXP & VALUE: any combination of EXP and VALUE so that the concatenation of the two will produce a valid FORTRAN Ecoolan expression.

*FIGURE 4.3 SYNTAX GRAPH FOR CASE CF STATEMENT*

The following two examples will illustrate the use of the CASE OF statement
**EXAMPLE 1**

```
CASE CF Length .EQ. 
  10:
    SIZE = SMALL
  50:
    SIZE = MEDIUM
  100:
    SIZE = LARGE
ENDCASE
```

**EXAMPLE 2**

```
CASE CF Length
  .LT. 50:
    SIZE = SMALL
  .EQ. 50:
    SIZE = MEDIUM
  .GT. 100:
    SIZE = LARGE
ENDCASE
```

The original CASE statement required that the colons be surrounded by blanks. If the blanks were omitted the statement was not recognized as a SPARKS statement. This has now been changed so that the colon is recognized with or without the blanks. This added feature required the addition of two more tests to the old CASE routine.

4.5 **MULTILEVEL EXIT AND CYCLE**

EXIT and CYCLE are SPARKS statements which cause a transfer of control to the first statement outside of the innermost loop, in the case of the EXIT, and to the bottom of the innermost loop in the case of the CYCLE. EXIT and CYCLE can be used within any of the four looping statements: WHILE, LOOP, LOOP UNTIL, and FOR. The EXIT and CYCLE are powerful tools for the user, but sometimes it is desirable to EXIT, not only from the inner loop, but out of several levels of nesting. This feature has now been added to SPARKS and is called the multilevel EXIT/CYCLE. Figure 4.4 show the syntax graph for the new statements.
EXIT

\[ \text{EXIT} \rightarrow \text{EXIT} \rightarrow \text{LEVEL} \]

CYCLE

\[ \text{CYCLE} \rightarrow \text{CYCLE} \rightarrow \text{LEVEL} \]

**SEMATICS**

LEVEL: an integer value which indicates levels to be EXITed/CYCLEd default to one if not specified.

**FIGURE 4.4 SYNTAX GRAPHS FOR EXIT AND CYCLE STATEMENTS**

The user now has the option to indicate the number of levels to be EXITed/CYCLEd. If the user wish only to EXIT/CYCLE one level, then the value can be set to one or omitted, in which case the default will EXIT/CYCLE one level. An example showing the use of the EXIT and CYCLE is listed below.

```plaintext
LOCF
FOR I=1 TO N BY 2 DC
TOKEN=VALUE
IF TOKEN.EQ.BLANK THEN
EXIT 2
ELSE
CYCLE
ENDIF
REPEAT
N=N+100
REPEAT
```

In the above example if the TOKEN is equal to a BLANK then control will be transferred out of the FOR LOOP and the LCOP FOREVER. In the case of the CYCLE, control will be transferred to the bottom of the FOR LCOP, since the level is not specified. The multilevel EXIT/CYCLE now allows the user to transfer control out of several levels of nesting without the use of flags as was necessary in the past.
4.6 ERROR RECOVERY

The original SPARKS preprocessor had no error recovery. Although it did do some error checking at the end of a SPARKS statement. This was accomplished by testing the statement terminator with the current stack top - if they did not match an error message was generated.

To give SPARKS the capability of error recovery a new subroutine ENDFIX was added. This routine is called when a mismatch is found. At this point ENDFIX attempts to unstack and produce statement terminators for those items still remaining on the stack until the desired statement is found. A message indicates the statements supplied by the program. This is not done for all SPARKS statements, but at a few key points such as the end of CASE statements, beginning of a subroutine, and at the end of a subroutine.

4.7 SUMMARY

The material covered in this chapter is concerned with additions to the SPARKS preprocessor. These additions will improve the code produced and give the user more options to use. It is also felt that the code produced will be more readable. Complete information on all features can be found in the User's Manual contained in Appendix A. Specifications and code for the modifications are contained in Appendices B and C.
APPENDIX A

USER'S MANUAL
KSU IMPLEMENTATION OF ENHANCED SPARKS

USER'S MANUAL

BY

JOHN J. MARTIN
RICHARD M. STRCUD
MYRON A. CALHOUN

Department of Computer Science
KANSAS STATE UNIVERSITY
Manhattan, Kansas 66502

December 1977
# TABLE OF CONTENTS

**SECTION 1.** INTRODUCTION ................................................. A-3  
1.1 SPARKS PREPROCESSOR ............................................... A-3  
1.2 INPUT AND OUTPUT .................................................. A-3  
1.3 FORTRAN VS SPARKS STATEMENTS .................................... A-4

**SECTION 2.** KEYWORDS AND SYMBOIS .................................... A-4  
2.1 KEYWORDS ............................................................ A-5  
2.2 SPECIAL SYMBOIS .................................................... A-6

**SECTION 3.** IF-THEN-ELSE STATEMENT ................................. A-6  
3.1 IF-THEN-ELSE STATEMENT SYNTAX GRAPH .......................... A-7  
3.2 EXAMPLE OF AN IF-THEN-ELSE STATEMENT ......................... A-7

**SECTION 4.** CASE STATEMENT ............................................ A-8  
4.1 CASE SYNTAX GRAPH ................................................ A-8  
4.1.1 EXAMPLE OF A CASE STATEMENT ................................ A-9  
4.2 CASE OF SYNTAX GRAPH ............................................. A-10  
4.2.1 EXAMPLE OF CASE OF STATEMENT ................................. A-10

**SECTION 5.** LOOP STATEMENTS .......................................... A-11  
5.1 FOR LCDF ............................................................. A-11  
5.1.1 EXAMPLE OF A FOR LCDF ........................................ A-12  
5.2 WHILE LCDF ........................................................ A-12  
5.2.1 EXAMPLE OF A WHILE LCDF ...................................... A-13  
5.3 LOOP UNTIL ........................................................... A-14  
5.3.1 EXAMPLE OF A LOOP UNTIL ....................................... A-14  
5.4 LOOP ................................................................. A-15  
5.4.1 EXAMPLE OF A LOOP .............................................. A-15

**SECTION 6.** EXIT AND CYCLE STATEMENTS ............................. A-16  
6.1 EXIT STATEMENT ..................................................... A-16  
6.1.1 EXAMPLE OF AN EXIT STATEMENT ............................... A-17  
6.2 CYCLE STATEMENT .................................................. A-17  
6.2.1 EXAMPLE OF A CYCLE STATEMENT ............................... A-18

**SECTION 7.** MULTIPLE ASSIGNMENT STATEMENT ....................... A-18

**SECTION 8.** COMMENT .................................................. A-19

**SECTION 9.** OTHER FEATURES OF SPARKS ............................. A-20  
9.1 LABELS ............................................................... A-20  
9.2 STATEMENT SEPARATICS ............................................. A-20  
9.3 CODE STRUCTURING ................................................ A-21  
9.4 DATA AND FORMAT STATEMENTS .................................... A-22  
9.5 ERROR MESSAGES ................................................... A-22  
9.6 PURE FORTRAN ..................................................... A-23

**SECTION 10.** SAMPLE OF SPARKS PROGRAMS .......................... A-24

A-2
1. INTRODUCTION

FORTRAN has the distinction of being essentially the earliest higher level programming language. Developed in the mid 1950's, it has been revised many times since then, but has changed little since the publication of American Standard Institute FORTRAN IV in 1966. Perhaps because of its early development, FORTRAN lacks many of the features of the newer languages. SPARKS is a preprocessor for FORTRAN which adds some of the missing features.

1.1 SPARKS PREPROCESSOR

A preprocessor is a program which translates statements written in a language X into a target language. A preprocessor is distinguished from a compiler in the following way: the source and target language have many statements in common. In this case language X is SPARKS and the target language is FORTRAN. In addition to the translation the SPARKS preprocessor will produce a structured output.

The SPARKS language is a mix of normal FORTRAN and a few SPARKS statements. Statements recognized by the SPARKS preprocessor include all standard FORTRAN and SPARKS statements defined in this manual.

1.2 INPUT AND OUTPUT

SPARKS accepts standard FORTRAN input with labels in columns 1-5, continuation in column 6 and code in columns 7-72. SPARKS and FORTRAN statements may be intermixed.
SPARKS produces two outputs, a FORTRAN file may be passed to the FORTRAN compiler for compilation and a SPARKS file which may be printed as part of the output. The SPARKS file contains nicely structured statements indented according to the structure to which they are subordinate.

1.3 FORTRAN VS SPARKS STATEMENTS

Within the SPARKS translator, there is a distinction made between FORTRAN and SPARKS statements. This manual describes only the SPARKS statements since it is assumed that the user is familiar with FCBTRAN. Thus only SPARKS statements remain to be defined. All SPARKS statements are recognized by keywords and or symbols. All other statements not recognized are assumed to be FORTRAN and are passed directly to the FORTRAN file without alteration. Each SPARKS statement causes the translator to produce FORTRAN statements which accomplish the equivalent computation. Hence, the local FCBTRAN compiler ultimately defines the semantics of all SPARKS statements.

2. KEYWORD AND SYMBOIS

Before continuing the term 'keyword' must be defined. A keyword is defined as a word which, when used as described in the SPARKS syntax graphs, is recognized by the SPARKS translator. This differs from a 'reserved' word which can be used only in the context defined and cannot appear as a program variable. As an example of this, consider the use of the IF-THEN-ELSE statement. THEN is a keyword when
surrounded by blanks and following an IF. However, it may also be used as a variable as indicated below.

```
IF X.EQ.Y THEN
     A=B
     THEN=B
ENDIF
```

In order to avoid confusion and possible error, in practice the user may find it desirable not to use keywords as variables.

2.1 **KEYWORDS**

The keywords recognized by SPARKS are listed as follows:

```
BLOCK   BY     CASE   CYCLE   DO    ELSE
END     ENDCASE ENDIFF   FIX     EXIT   FOR
FUNCTION IF     INTEGER LCGICAL LCGF   OF
HEAL    REPEAT  RETURN SUBROUTINE UNTIL WHILE
TO      THEN
```

All keywords used as part of a SPARKS statement must be **SURROUNDED BY BLANKS.** Failure to do this will result in the SPARKS statement not being recognized, thus passed as a FORTRAN statement. Keywords can be classified into three types:

**TYPE 1:** Keywords used to indicate the start and end of a SPARKS statement, such as IF and ENDIFF.

**TYPE 2:** Keywords used within a SPARKS statement as a delimiter, such as THEN.
TYPE 3: FORTRAN defined words which must be recognized by the SPARKS translator, such as END.

The use of each type of keyword will become more clear as each statement is defined.

2.2 SPECIAL SYMBOLS

The SPARKS translator has three special symbols which are used as delimiters within a SPARKS program. They are the colon, the semicolon, and the doubleslash. These special symbols need not be surrounded by blanks, but they will be recognized with or without blanks.

: COLON - is used to indicate a CASE condition.
; SEMICOLON - is used as a line separator.
// DOUBLESLASH - is used to indicate a comment.

3. IF-THEN-ELSE STATEMENT

The IF-THEN-ELSE statement consists of a Boolean expression and one or two compound statement pairs. When the IF is encountered by the translator, it then searches for a THEN. If found, the statement is processed as a SPARKS statement; if not, it is passed as FORTRAN. The IF-THEN-ELSE statement functions much like the FCRTRAN IF: if the condition is 'true', do all that follows THEN until an ELSE or ENDIF is found; if 'false', branch to the ELSE or ENDIF and continue.
3.1 IF-THEN-ELSE STATEMENT SYNTAX GRAPH

![Syntax Graph]

**SEMANTICS**

COND: is any legal FORTRAN Boolean expression
STMT: any FORTRAN/SPARKS statement
SEPOR: semicolon or end-of-line

**NOTE**

Comments may be inserted after the THEN, ELSE or ENDF without the use of a semicolon.

3.2 EXAMPLE OF AN IF-THEN-ELSE STATEMENT

Listed below are two examples of IF-THEN-ELSE statements used by SPARKS. The first has the FORTRAN equivalent, while the second demonstrates the use of comments within an IF-THEN-ELSE statement.

**EXAMPLE 1**

**INPUT:** IF A.EQ.B THEN F=X; Q=Y
ELSE F=Y; Q=Z; ENDF

<table>
<thead>
<tr>
<th>SPARKS</th>
<th>FORTRAN</th>
</tr>
</thead>
<tbody>
<tr>
<td>IF A.EQ.B THEN</td>
<td></td>
</tr>
<tr>
<td>F=X</td>
<td></td>
</tr>
<tr>
<td>Q=Y</td>
<td></td>
</tr>
<tr>
<td>ELSE</td>
<td></td>
</tr>
<tr>
<td>F=Y</td>
<td></td>
</tr>
<tr>
<td>Q=Z</td>
<td></td>
</tr>
<tr>
<td>ENDF</td>
<td></td>
</tr>
<tr>
<td>IF (.NOT. (A.EQ.B)) GO TO 99999</td>
<td></td>
</tr>
<tr>
<td>F=X</td>
<td></td>
</tr>
<tr>
<td>Q=Y</td>
<td></td>
</tr>
<tr>
<td>GC TC 99998</td>
<td></td>
</tr>
<tr>
<td>99999 CONTINUE</td>
<td></td>
</tr>
<tr>
<td>F=Y</td>
<td></td>
</tr>
<tr>
<td>G=Z</td>
<td></td>
</tr>
<tr>
<td>99998 CONTINUE</td>
<td></td>
</tr>
</tbody>
</table>
EXAMPLE 2

INPUT: IF PTR.EQ.N THEN TEST=0 //ITEM NOT IN LIST//
ELSE TEST=1 //ITEM IN LIST//
ENDIF

SPARKS OUTPUT
IF PTR.EQ.N THEN
TEST=0 //ITEM NOT IN LIST//
ELSE
TEST=1 //ITEM IN LIST//
ENDIF

As shown by the syntax graph the IF-THEN-ELSE statement can be used with and without the ELSE. However, the ENDIF must always be used. Of course, a FORTRAN IF (without a THEN) cannot be followed by an ENDIF.

4. CASE STATEMENT

The SPARKS translator allows two forms of the CASE statement. Both perform the same function, but go about it differently. In general, a CASE statement can be described as a very powerful FORTRAN computed GO TO statement. It allows the user to list a set of conditions and then execute only that condition which is true.

4.1 CASE SYNTAX GRAPH

![CASE Syntax Graph](image)
**SEMANTICS**

COND: is any FORTRAN Boolean expression
STMT: any FORTRAN/SPARKS statement
SEP: semicolon or end-of-line

**NOTE**

1. Comments may be inserted after the CASE or CCICN, without the use of a semicolon.
2. The colon need not be surrounded by blanks.

4.1.1 **EXAMPLE OF CASE STATEMENT**

**INPUT:**
CASE:LENGTH.EQ.10: SIZE=SMALL
LENGTH.EQ.50: SIZE=MEDIUM
LENGTH.EQ.100: SIZE=LARGE
ELSE: FOUND=.FALSE.; ENDCASE

**SPARKS**

CASE
LENGTH.EQ.10:
SIZE=SMALL
LENGTH.EQ.50:
SIZE=MEDIUM
LENGTH.EQ.100:
SIZE=LARGE
ELSE:
FOUND=.FALSE.
ENDCASE

**FORTRAN**

IF (.NOT. (LENGTH.EQ.10)) GO TO 99998
SIZE=SMALL
GO TO 99999
99998 IF (.NOT. (LENGTH.EQ.50)) GO TO 99997
SIZE=MEDIUM
GO TO 99999
99997 IF (.NOT. (LENGTH.EQ.100)) GO TO 99996
SIZE=LARGE
GO TO 99999
99996 CONTINUE
FOUND=.FALSE.
99999 CONTINUE
CAUTION

Use of the ELSE within the CASE statement can produce invalid results if it is not placed at the very end of the statement. The ELSE condition will always be executed if a true condition is not encountered earlier. If the ELSE were placed in the center of the CASE, then it would be possible for the conditions after the ELSE to be executed if they were true.

4.2 CASE OF SYNTAX GRAPH

CASE STATEMENT

EXP & VALUE: any combination of EXP and VALUE so that the concatenation of the two will produce a valid FORTRAN Boolean expression
STMT: any FORTRAN/SPARKS statement
SEPOR: semicolon or end-of-line

4.2.1 EXAMPLE OF CASE OF STATEMENT

SPARKS

CASE OF LENGTH.EQ.
:10:
    SIZE=SMALL
:50:
    SIZE=MEDIUM
:100:
    SIZE=LARGE
ENDCASE
This form of the CASE statement allows the user a great degree of flexibility.

5. LOOP STATEMENT

Loops are very powerful programming tools, but are also probably the most common sources of error in FORTRAN programs. SPARKS allows four types of LOOP statements, some of which are very safe while others require caution in their use.

* * NOTE * *

ALL LOOP statements are required to end with a REPEAT.

5.1 FOR LOOP

Of the four loop statements in SPARKS, the FOR loop is about the easiest to use. As indicated by the syntax graph it will execute a fixed number of times and then terminate.
SEMANTICS

STM1: any FORTRAN integer assignment statement
EXP: any FORTRAN arithmetic expression
INT: any integer value
SEPOR: semicolon or end-of-line
STMT: any FORTRAN/SPARKS statement

** NOTE **
1. The loop will increment the loop pointer which is stated in STM1.
2. Comments may be inserted after the DO or REPEAT, without the use of a semicolon.

5.1.1 EXAMPLE OF FOR LOOP

INPUT: FCR I=X+Y TO 50 BY 2 DC
A=C/D;E=E*F;REPEAT

SPARKS

FCR I=X+Y TO 50 BY 2 DO
A=C/D
E=E*F
REPEAT

FORTRAN

I=X+Y
GO TO 99999
99998 I=I+(2)
99999 IF (.NOT. ((I-(50))*(2) .LE. 0)) GO TO 99997
A=C/D
B=E*F
99996 GO TO 99998
99997 CONTINUE

5.2 WHILE LOOP

The second form of loop statement is the WHILE loop.
This loop will continue to execute while the stated condition is true. The structure for this statement is given in the following syntax graph.

![Syntax Graph](image)

**SEMANTICS**

EXP: any FORTRAN Boolean expression
SEPOR: semicolon or end-of-line
STMT: any FORTRAN/SPARKS statement

**NOTE**
Comments may be inserted after the DC or REPEAT, without the use of a semicolon.

5.2.1 **EXAMPLE OF WHILE LOOP**

**INPUT:** WHILE TEMP.LE.10 DC  
I=I+1;TEMP=SIZE(I);REPEAT

**SPARKS**

WHILE TEMP.LE.10 DC  
I=I+1  
TEMP=SIZE(I)  
REPEAT

**FORTRAN**

99999 IF (.NOT. (TEMP.LE.10)) GO TO 99998  
I=I+1  
TEMP=SIZE(I)  
99997 GO TO 99999  
99998 CONTINUE

**CAUTION**

When using the WHILE loop the user must insure that the test condition is allowed to change in value. If this is not done the loop will not
terminate. The WHILE loop does not increment any pointers.

5.3 LOOP UNTIL

The third type of loop statement is the LOCP UNTIL. This is much like the WHILE loop except that the test is made at the end of the loop. The syntax graph is as follows:

```
LOOP -----> LOCP -----> STMT -----> SEPCR -----> UNTIL -----> EXP -----> REPEAT
```

**SEMANTICS**

STMT: any FORTRAN/SPARKS statement
SEPCR: semicolon or end-of-line
EXP: any FORTRAN Boolean expression

**NOTE**
Comments may be inserted after the LOCP or REPEAT, without the use of a semicolon.

5.3.1 EXAMPLE OF LOCP UNTIL

**INPUT:** LOCP X=A+5
IF X.GT.100 THEN DCNE=.TRUE.;ENDIF
UNTIL DONE REPEAT

**SPARKS**

```
LOCP
X=A+5
IF X.GT.100 THEN
DONE=.TRUE.
ENDIF
UNTIL DONE REPEAT
```

**FORTRAN**

```
99999 CONTINUE
X=A+5
IF(.NOT.(X.GT.100)) GC TO 99998
    DCNE=.TRUE.
99998 CONTINUE
99997 IF(.NOT.(DCNE)) GO TO 99999
99996 CONTINUE
```
CAUTION

Like the WHILE loop, the UNTIL loop does not increment any pointers or counters. The user must insure that the condition is set inside the loop so that the loop will terminate.

5.4 LCOP

The last of the four loops allowed by SPARKS is the LCOP. This is, as the name implies, an endless loop. In the loop body the user must set conditions to EXIT or GO TO somewhere outside of the loop to cause the loop to terminate. The syntax graph for this loop is given below.

![LCOP Syntax Graph]

SEMANTICS

STMT: any FORTRAN/SPARKS statement
SEPOR: semicolon or end-of-line
* * NOTE * *
Comments may be inserted after the LCOP or REPEAT, without the use of a semicolon.

5.4.1 EXAMPLE OF LCOP

INPUT: LCOP X=X+5
      IF X.GT.100 THEN EXIT;ENDIF;REPEAT

SPARKS

LOOP
  X=X+5
  IF X.GT.100 THEN
    EXIT
  ENDIF
  REPEAT
FORTRAN

99999 CONTINUE
X=X+5
IF(.NOT.(X.GT.100)) GO TO 99998
GO TO 99997
99998 CONTINUE
99996 GO TO 99999
99997 CONTINUE

CAUTION
The LOOP does not increment any pointers or counters
and it will not terminate unless the user does so
explicitly.

6. EXIT AND CYCLE STATEMENTS

When using a statement such as the LOOP, it is
necessary to produce some means to transfer control to a
point outside the loop. EXIT and CYCLE statements may be
used to do this.

6.1 EXIT STATEMENT

EXIT is a SFARKS statement which causes a transfer of
control to the first executable statement outside of the
loop level specified. EXIT can be used with any of the four
SFARKS looping statements: FOR, WHILE, LOOP UNTIL, and LOOP.
The following syntax graph defines the statement structure.

```
EXIT STATEMENT -> EXIT -> LEVEL -> SEFOR
```

SEMANTICS

LEVEL: integer indicating the number of levels to
exit, if omitted the level will be one.

** ** NOTE ** **
If the number of levels specified exceeds
the number of nested levels then the EXIT
will be made to the outermost nesting level.

6.1.1 EXAMPLE OF EXIT

SPARKS OUTPUT

LCCP
FOR I=1 TO N DO
TOKEN=VALUE(I)
IF TOKEN.EQ.BLANK THEN
EXIT
ELSE
  EXIT 2
ENDIF
REPEAT
REPEAT

In the above example the first EXIT transfers control outside of the FOR loop to the second REPEAT. The second one, EXIT 2 transfers control to the first statement following the second REPEAT.

6.2 CYCLE STATEMENT

CYCLE is a SPARKS statement which causes a transfer of control to the bottom of the specified loop. CYCLE can be used with any of the four SPARKS looping statements: FOR, WHILE, LCCP UNTIL, and LOOP. The following syntax graph is given for the CYCLE statement.

CYCLE STATEMENT → CYCLE → LEVEL → SEFOR →

SEMANTICS

LEVEL: integer indicating the number of levels to be cycled.
SEFOR: semicolon or end-of-line
**NOTE**
If the number of levels specified exceeds the nesting level, then the outermost nesting level will be cycled.

6.2.1 EXAMPLE OF CYCLE

**SPARKS OUTPUT**

```
LOOF
 FOR I=N TO M DO
  TOKEN=SPIN(I)
  CASE
    :TOKEN.EQ.BLANK:
      CYCLE
    :TOKEN.EQ.ALPHA:
      CYCLE 2
  ENDCASE
 REPEAT
 REPEAT
```

In the above example the first CYCLE will transfer control to the REPEAT at the bottom of the FOR loop. The second CYCLE 2 will transfer control to the REPEAT at the bottom of the LOOF.

7. MULTIPLE ASSIGNMENT STATEMENT

SPARKS allows the user to assign the same value to more than one variable with the multiple assignment statement. The statement is defined by the following syntax graph.

```
ASSIGNMENT STATEMENT  --->  VAR  --->  =  --->  EXP

SEMANTICS
VAR: any FORTRAN variable
```
EXP: any legal FORTRAN expression

The SPARKS translator recognizes the multiple assignment statement and produces individual statements for the FORTRAN output. The following are examples of its use:

<table>
<thead>
<tr>
<th>SPARKS</th>
<th>FORTRAN</th>
</tr>
</thead>
</table>
| SMALL, LAST, DATA(I) = VALUE | SMALL = VALUE  
LAST = VALUE  
DATA(I) = VALUE |
| I, J, K, COUNT = 0 | I = 0  
J = 0  
K = 0  
COUNT = 0 |
| DONE, CCND = .FALSE. | DONE = .FALSE.  
COND = .FALSE. |

8. COMMENT

SPARKS recognizes and processes two types of comments: standard FORTRAN comments with the 'C' in column 1 and SPARKS comments delimited by the double-slash. The SPARKS comment may be on a separate line or inline after the rest of the statement. All comments are restricted to one line; they must start and terminate in the same line. However, there can be any number of consecutive comments. Comments are not passed to the FORTRAN file, but appear only in the SPARKS output. SPARKS comment which are to appear in-line with the code must not be separated from the preceding statement by a semicolon, since this causes the comment to be displayed on the next line. The following are examples of SPARKS comments.
CALL TEST(TOKEN,COND) //TEST PCB ALPHA CHAR//
IF COND THEN
  FEUF(I)=TOKEN //STORE ALPHA CHAR//
ELSE
  PTR=I //TOKEN NOT ALPHA RESET POINTER//
ENDIF

* * NOTE * *
SPARKS comments must start after column 6.

9. OTHER FEATURES OF SPARKS

This section contains features common to all SPARKS statements and is general in nature.

9.1 LABELS

SPARKS generates its own labels (statements numbers), starting at 99999 and decrementing by one each time a new label is required. If labels are used in the input code, they will be passed to the output. Of course, since most well-structured programs can be written in a go-to-less manner, there is little need for labels. If labels are used, they should be selected so as not to conflict with SPARKS: it is suggested that user-defined labels start with a small number and proceed upward. A typical 600 statement SPARKS program generates labels down to about 99900.

9.2 STATEMENTS SEPARATORS

The SPARKS translator allows more than one statement per line when a semicolon is used to separate such
statements. All statements must use some form of separator to separate them. The following syntax graph defines the type of separators used.

![Syntax Graph]

**NOTE**

End-of-line is by // or last non blank character

If more than one statement appears on a line, they must be separated by a semicolon. However, the last statement on the line is terminated by the end-of-line or column 72, whichever comes first.

**NOTE**

The semicolon is not used between code and comment if they are to appear on the same line.

9.3 **Code Structure**

The SFARKS translator produces a structured looking output. All statements of a program/subroutine are indented three spaces. Each time a new structure such as LOOP, CASE or IF is encountered all subordinate statements are indented an additional three spaces. This is repeated within each structure. When the end of a structure is found the following statements are pulled back three spaces.
This structuring is done automatically and requires no special action by the user. All code is entered columns 7-72 in the normal FORTRAN manner.

9.4 DATA AND FORMAT STATEMENTS

As noted above, the SPARKS translator produces a structured output for both SPARKS and FORTRAN. This approach offers little problem to the SPARKS output which is displayed on a 132-character line. However, the FORTRAN output is restricted to a 72-character line. If input code occupies the entire 72 characters, then the indentation will cause a break in the line and need for a continuation card. If the statement contains a Hollerith field and the break occurs in the middle of the field, then an error will result in the FORTRAN code.

In order to prevent this error, the DATA and FORMAT statements are handled as special cases in the FORTRAN output only. Once the translator recognizes the statement to be a DATA or FORMAT, it is then passed to the FORTRAN output without being indented. Continuation lines of DATA and FORMAT statements are similarly handled.

9.5 ERROR MESSAGE

For each SPARKS statement used there is a defined starting and ending point. When the start of a statement is encountered it is placed on the top of the stack. Later when an ending is found, the corresponding starting element should be on top of the stack. If the starting point given
in the top of the stack and the ending do not match an error message is given to the SPARKS output. The following is an example:

**INPUT**

IF .NOT.DONE THEN
FOR I=1 TO 10 DO
VALUE(I)=EXP(I)*10
ENDIF

**OUTPUT**

IF .NOT.DONE THEN
FOR I=1 TO 10 DO
VALUE(I)=EXP(I)*10
+++++++ OFFENDING CARD: ENDF
+++++++ TCC MANY ENDF
+++++++ ENDF IGNORED

In this example the ENDF is not the problem, but the missing REPEAT at the end of the FOR statement is what causes it. The SPARKS user should be aware of the difficulty of producing meaningful error messages for every situation and should not take error messages at face value.

SPARKS does make some attempt to recover from errors if the ending condition does not match the starting condition on the top of the stack, then missing statements will be supplied until a match is found or the outermost nesting level is reached. Messages indicate the statements added.

9.6 **PURE FORTRAN**

An 'F' in column 1 signals the translator that all subsequent cards contain PURE FORTRAN code. SPARKS allows these statements to pass through without any checking. The
next card with a 'F' in column 1 signals the translator to return to its normal mode.

10. SAMPLE OF A SFARKS PROGRAMS

This section contains two SFARKS programs to give the user some idea as to what SFARKS output looks like. The first program is one which searches for the integer X in the sorted array of integers A(N), N=1 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 binary search algorithm.

```
SUBROUTINE BINS(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 second program is one which uses a best fit algorithm to allocate a block of memory of size N,N>0. P is the address of the block of memory. A head node is used to hold the size of total memory M and the link to the first free block.
SUBROUTINE BESTPT (N,F)
    IMPLICIT INTEGER(A-Z)
    COMMON/MEMORY/AV,SIZE(100),LINK(100)
    P=LINK(AV)
    Q,TEMP=AV
    WHILE SIZE(P).NE.M DO
        IF SIZE(P).GE.N THEN //IF SIZE < N CYCLE//
            IF SIZE(P).GT.N THEN
                IF SIZE(P).LT.SIZE(TEMP) THEN //KEEP SMALLER//
                    TEMP=P
                ENDIF
            ELSE
                LINK(Q)=LINK(P) //EXACT BLOCK SIZE IS FOUND//
                RETURN
            ENDIF
        ENDIF
        Q=P //SET FCINTER TO NEXT FREE BLOCK//
        P=LINK(P)
        REPEAT
        IF TEMP.EQ.AV THEN
            P=0 // NO BLOCK LARGE ENOUGH //
        ELSE
            SIZE(TEMP)=SIZE(TEMP)-N //SET NEW SIZE/
            P=TEMP+SIZE(TEMP) //SET P TO FCINT TO FREE BLOCK//
        ENDIF
        RETURN
    END
APPENDIX E

CODE SPECIFICATIONS
SPECIFICATIONS

FOR

KSU SPARKS FORTRAN PREPROCESSOR

INTRODUCTION

The SPARKS preprocessor is a 1600 line SPARKS program consisting of a main program, a block data, 15 subroutines and 2 integer functions. The preprocessor was bootstrapped with the use of a previous edition written in FORTRAN. This document contains module access matrix, data flow graph, module specifications, and other information pertaining to the SPARKS preprocessor. All information is current and reflects all changes made at KSU through Dec. 1977.
**PROGRAM GLOBAL VARIABLES**

The program's global variables and dimensions, if applicable, are listed below followed by the names of the procedures in which they are used. Those variables local to a single procedure are not listed.

<table>
<thead>
<tr>
<th>VARIABLE</th>
<th>USING PROCEDURE</th>
</tr>
</thead>
<tbody>
<tr>
<td>AEE</td>
<td>BLOCK DATA, FTYP, FOUT</td>
</tr>
<tr>
<td>ELANK</td>
<td>MAIN, BLOCK DATA, DETNX, SCAN, SPRNT, PHAND, FOUT</td>
</tr>
<tr>
<td>ELNKLH</td>
<td>FPRNT, FCUT</td>
</tr>
<tr>
<td>ELNKS(131)</td>
<td>BLOCK DATA, SPRNT</td>
</tr>
<tr>
<td>EKWHC</td>
<td>MAIN, BLOCK DATA</td>
</tr>
<tr>
<td>ELKNC</td>
<td>MAIN, SPRNT</td>
</tr>
<tr>
<td>EYE(2)</td>
<td>MAIN, BLOCK DATA</td>
</tr>
<tr>
<td>CASS(4)</td>
<td>MAIN, BLOCK DATA, CHECK</td>
</tr>
<tr>
<td>CCASE</td>
<td>MAIN, BLOCK DATA, ENDFIX</td>
</tr>
<tr>
<td>CEE</td>
<td>BLOCK DATA, DETNX, SPRNT, FCUT</td>
</tr>
<tr>
<td>CCYCLE(5)</td>
<td>MAIN, BLOCK DATA, CHECK</td>
</tr>
<tr>
<td>CCOLCN(5)</td>
<td>MAIN, BLOCK DATA, CHECK</td>
</tr>
<tr>
<td>CGOTO(9)</td>
<td>MAIN, BLOCK DATA</td>
</tr>
<tr>
<td>CCLN</td>
<td>MAIN, PICCK DATA</td>
</tr>
<tr>
<td>COLCN</td>
<td>MAIN, BLOCK DATA, STTYP, CHECK</td>
</tr>
<tr>
<td>CCMNT</td>
<td>MAIN, PICCK DATA, DETNX, FOUT, SPRNT</td>
</tr>
<tr>
<td>COMMA</td>
<td>BLOCK DATA, FTYP</td>
</tr>
<tr>
<td>CCNT</td>
<td>MAIN, PICCK DATA, DETNX</td>
</tr>
<tr>
<td>CCNTT(8)</td>
<td>MAIN, BLOCK DATA, ENDFIX</td>
</tr>
</tbody>
</table>
CHER  BLOCK DATA, DETNX
CURFLG  MAIN, BLOCK DATA, EXCYL
CEE  BLOCK DATA, FTYP
DELSL  MAIN, BLOCK DATA, STYP
DESLH  MAIN, BLOCK DATA, STYP, SCAN1, SCAN2, PHAND
DOO  MAIN, BLOCK DATA
ECCND  MAIN, DETNX, SCAN, SCAN1, SCAN2, PHAND, FTYP, EXCYL, CHECK
EEEND(3)  BLOCK DATA, CHECK, ENDFIX
EEEXT(4)  BLOCK DATA, CHECK
EEELSE  MAIN, BLOCK DATA, CHECK
EEEND  MAIN, BLOCK DATA, CHECK
EEENDIF  MAIN, BLOCK DATA, CHECK
EEEXIT  MAIN, BLOCK DATA, EXCYL, CHECK
ELZ(4)  MAIN, BLOCK DATA, CHECK
ENCASS(7)  MAIN, BLOCK DATA, CHECK, ENDFIX
ENDCAS  MAIN, BLOCK DATA, CHECK
ENIF(5)  MAIN, BLOCK DATA, CHECK, ENDFIX
ECJ  MAIN, BLOCK DATA
ECL(1)  MAIN, BLOCK DATA, FTYP
FEUP(80)  MAIN, DETNX, PHAND, FPRNT, FOUT, LABFIL
FE  BLOCK DATA, DETNX, FTYP
FFILE  BLOCK DATA, DETNX, FPRNT
FFOR  MAIN, BLOCK DATA, CHECK, ENDFIX
FMAX  MAIN, BLOCK DATA, FOUT, FPRNT
FCTR  MAIN, BLOCK DATA, DETNX, STYP
FTR  MAIN, FERNT, FOUT, PHAND
FRWD  MAIN, BLOCK DATA
FSTRT  MAIN, FPBNT, CHECK
GOTO (6) MAIN, BLOCK DATA, ENDFIX, EXCYL
IIF  MAIN, BLOCK DATA, CHECK
IIIF (2) BLOCK DATA, CHECK
IMAX  MAIN, DETNX, SCAN, SCAN1, SCAN2
INDFLG  MAIN, EXCYL
IPNCT (9) MAIN, BLOCK DATA
ITERAT (9) BLOCK DATA, CHECK
LABEL (200) MAIN, BLOCK DATA, GENLAE
LAST  MAIN, EXCYL, FPBNT
IEFLG  MAIN, DETNX, SPBNT, FHAND, EXCYL
ILEMAX  MAIN, GENLAB
IEPXB  MAIN, GENLAB
LEO (5) MAIN, BLOCK DATA, ENDFIX
LCOP (4) MAIN, BLOCK DATA, CHECK
NCYCLE  MAIN, BLOCK DATA, CHECK
NESTNC  MAIN, SPBNT
NEUTR  MAIN, BLOCK DATA, EXCYL
NUMB  MAIN, BLOCK DATA, GENLAE
NUMCOM  FTYH, FHAND
MINUS (1) MAIN, BLOCK DATA
CLD  MAIN, BLOCK DATA, EXCYL
CNE (1) MAIN, BLOCK DATA
COP  MAIN, BLOCK DATA
FARN  BLOCK DATA, FTYH
FLU (1) MAIN, BLOCK DATA
FULCUT  MAIN, BLOCK DATA
FESRC (200) BLOCK DATA, STYHP

E-4
FEPT (6)  MAIN, BLOCK DATA, CHECK
EINLO    MAIN, BLOCK DATA
FEFPT    MAIN, BLOCK DATA, CHECK
ESTAR (2) MAIN, BLOCK DATA, FTYF
SEMI     EBLOCK DATA, DETNX, SCAN1
SMAX     MAIN, SPRNT
SPIN (80) MAIN, BLOCK DATA, DETNX, STYP, SCAN, SCAN1,
           SCAN2, FPRNT, SPRNT, FHAND, FOUT, FTYF, EXCYL
SSTRT    MAIN, SPRNT, CHECK
STACK (200) MAIN, CHECK, EDNFIX, EXCYL
STBEG    MAIN, SPRNT
STEND    MAIN, SPRNT
STPRM (6) EBLOCK DATA, CHECK, EDNFIX
STMAX    MAIN, BLOCK DATA
STMNC    MAIN, SPRNT, FPRNT
SUBFN    MAIN, BLOCK DATA, CHECK, EDNFIX
THENN (4) MAIN, BLOCK DATA
TOBEG    MAIN, DETNX, STYP, SCAN, FHAND, FTYP
TICEND   MAIN, DETNX, STYP, SCAN, SCAN1, SCAN2, FHAND,
           FTYP, CHECK
TICO (2)  MAIN, EBLOCK DATA
TOP      MAIN, CHECK, EDNFIX
TIFTR    MAIN, CHECK, EDNFIX
TRES (23) EBLOCK DATA, STYP
TYPE     MAIN, DETNX, STYP, FHAND, FTYP, SPRNT, EXCYL
UNTIL    MAIN, BLOCK DATA, CHECK
UNTIL (5) MAIN, EBLOCK DATA, CHECK
WHILE    MAIN, BLOCK DATA, CHECK
VARLCC (10) FHAND, FTYP
SUBROUTINE AND FUNCTION NAMES

NAME                   DEFINITION
CHECK:                Check for a correct statement terminator.
CCNV:                  Converts a digit into its character representation.
                        (CCNVert)
DETNX:                Determine the next statement type.
                        (DETermine Next)
ENDFIX:               Produces missing statement terminator.
                        (END FIX)
ERRCB:                Produce an error message as required.
EXCYL:                Process EXIT and CYCLE statements.
                        (Exit CYcle)
FHANDE:               Process FCRTRAN statements.
                        (Fortran HANDLE)
FOUT:                 Builds a statement in the FCRTRAN output buffer.
                        (Fortran CUT)
FPRTNT:               Writes FCRTRAN statements from buffer to output file.
                        (Fortran FPrNT)
FTYP:                 Determine the type of FCRTRAN statement.
                        (Fortran TYPE)
GENLAB:               Generate a FCRTRAN label.
                        (GENERate LABEL)
LABFIL:               Copies a label into the FCRTRAN output buffer.
                        (LABEL FILL)
SCAN:                 Searches for a given pattern isolated by blanks.
SCAN1:                Searches for a given character.
SCAN2:                Searches for a SPARKS comment after a statement.
SPRTNT:               Writes SPARKS statements to the output file.
                        (Sparks PRINT)
STYP:                 Determine SPARKS statement type.
                        (Statement TYPE)
SPARKS INPUT

DETERMINE STATEMENT TYPE

SPARKS_STMT PRODUCE FORTRAN OUTPUT

FORTRAN PASS TO FORTRAN FILE

EOJ

FORTRAN OUTPUT FILE

STOP

SPARKS OUTPUT FILE

SPARKS DATA FLOW GRAPH
<table>
<thead>
<tr>
<th></th>
<th>CHECK</th>
<th>CONV</th>
<th>DETNX</th>
<th>ENDFIX</th>
<th>ERRCR</th>
<th>EXCYL</th>
<th>FHAND</th>
<th>FOUT</th>
<th>FTP</th>
<th>FPRT</th>
<th>GENTYP</th>
<th>LAB</th>
<th>MAIN</th>
<th>SCANTYP</th>
<th>SCANTYP</th>
<th>SCANTYP</th>
<th>SCANTYP</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALPH</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CENTHL</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LEV1C</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FCRT</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FRTINF</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FRT</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>INDT</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>INPAT</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>INPUT</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>KEYW1</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>KEYW2</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CD2</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CTPAT</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>STRINF</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>STYP1</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>STYP2</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>STYP3</td>
<td>X</td>
<td></td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TABLES</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TYP</td>
<td>X</td>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**COMMON BLOCK ACCESS MATRIX**
<table>
<thead>
<tr>
<th>CALLEE</th>
<th>CHECK</th>
<th>CCNV</th>
<th>DETNX</th>
<th>ENDFIX</th>
<th>ERRCR</th>
<th>EXCYL</th>
<th>FHAND</th>
<th>FOUT</th>
<th>FPRNT</th>
<th>FYP</th>
<th>GENLAB</th>
<th>LABFIL</th>
<th>MAIN</th>
<th>SCAN</th>
<th>SCAN1</th>
<th>SCAN2</th>
<th>SPRINT</th>
<th>STTYP</th>
</tr>
</thead>
<tbody>
<tr>
<td>CHECK</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CCNV</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DETNX</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ENDFIX</td>
<td>X</td>
<td></td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ERRCR</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>EXCYL</td>
<td>X</td>
<td></td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FHAND</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FOUT</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FPRNT</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FYP</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GENLAB</td>
<td>X</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LABFIL</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MAIN</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SCAN</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SCAN1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SCAN2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SPRINT</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>STTYP</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

MODULE ACCESS MATRIX
COMMON BLOCKS

Because of the large number of variables used by each of the SPARKS subroutines, most variables are passed by common statements rather than parameters in the subroutine call. Listed below are all common blocks and the variables associated with each block.

<table>
<thead>
<tr>
<th>NAME</th>
<th>VARIABLES</th>
</tr>
</thead>
<tbody>
<tr>
<td>/ALPH/</td>
<td>AEE,CEE,CCMA,DEE,DBSLH,FEET,PARN</td>
</tr>
<tr>
<td>/CENTRL/</td>
<td>LABELL(200),LBMAX,LFUTH,NUMP,STACK(200),STFRM(6),STMA1,TO TOR,TINT</td>
</tr>
<tr>
<td>/DEVIC/</td>
<td>CRDR,FFILE,PRTR</td>
</tr>
<tr>
<td>/FORT/</td>
<td>BINKIN,ECCND,NUMCOM,PTR,VARLOC(10)</td>
</tr>
<tr>
<td>/FRITNF/</td>
<td>FEUF(80),FMAX,FPTR,FSFRT,INCR</td>
</tr>
<tr>
<td>/FTRT/</td>
<td>DTA(4),FMAT(6),FTYPE,LAST,NLAST</td>
</tr>
<tr>
<td>/INDT/</td>
<td>BKWRT,CURFLG,FRWD,INDFIG,CLD,PULCUT,NEUTR,NEW</td>
</tr>
<tr>
<td>/INPAT/</td>
<td>BYE(2),COIN,DOC(2),ELZ(4),THENN(4),TOO(2)</td>
</tr>
<tr>
<td>/INPUT/</td>
<td>BLANK,I1AX,SPIN(80),STEEG,STEND,TOBEG,TOEND</td>
</tr>
<tr>
<td>/KEYW1/</td>
<td>CASS(4),CCOLON(5),EEEND(3),ENCASS(7),ENIF(5),IIIF(2)</td>
</tr>
<tr>
<td>/KEYW2/</td>
<td>CCYCLE(5),EEExt(4),ITERAT(9),OOF(2),REPT(6),UUNTL(5)</td>
</tr>
<tr>
<td>/ODD/</td>
<td>LABO,LBFLG,N1UMBER(10)</td>
</tr>
<tr>
<td>/OTPFX/</td>
<td>CGCCTO(9),CCNT(8),EQL(1),IFNOT(9),LEO(5),GOTO(6),MINUS(1),ONE(1),PLU(1),RSTAR(2)</td>
</tr>
<tr>
<td>/SPBINF/</td>
<td>BLNKS(131),BLKNO,NESTNO,SMA1,SSFRT,STMNO</td>
</tr>
<tr>
<td>/STYP1/</td>
<td>COLON,DBSLH,EELSE,EEND,ENECAS,LLCOP,RREPT,UUNTL</td>
</tr>
<tr>
<td>/STYP2/</td>
<td>COMT,CONT,EOJ,FORIN,NCYCLE,RINLO,SEMI,WHILE</td>
</tr>
<tr>
<td>/STYP3/</td>
<td>CCASE,EENDIF,EXIT,FPOR,IIIF,RETRN,SUBFN</td>
</tr>
<tr>
<td>/TABLES/</td>
<td>RESRD(200),TRES(23)</td>
</tr>
<tr>
<td>/TYP/</td>
<td>TYPEE</td>
</tr>
</tbody>
</table>
FUNCTION
The main program causes a line of code to be read and determines the statement type. The statement is processed according to type. FORTRAN statements are passed directly to the FORTRAN file; SPARKS statements cause FORTRAN code to be produced which accomplishes the desired actions. The FORTRAN code produced is passed to the FORTRAN file. After proper structuring flags have been set, both SPARKS and FORTRAN statements are passed to the SPARKS file.

COMMON BLOCKS
ALPH, CENTR, DEVIC, FRTINF, FRT, IND, INPT, INPUT, KEYW1, KEYW2, ODD, OTPAT, SRINF, STYF1, STYF2, STYF3, TABLES, TYP

HIGH-LEVEL CODE
PROC: MAIN
  INITIALIZE POINTERS AND FLAGS
  LCOF
    CALL DETEX       DETERMINE NEXT STATEMENT TYPE
    IF TYPE EQUAL REAL INTEGER OR LOGICAL THEN
      IF NEXT WORD EQUAL SUBPROGRAM THEN
        TYPE=SUBFN
      ELSE
        TYPE=FORTRAN
      ENDIF
    ENDIF
  CASE
    :TYPE=FORTRAN:
    PASS TO FORTRAN FILE
    :TYPE=COMMENT:
    TEST FOR END OF COMMENT
    SET INDENTATION FLAGS
:TYPE=SFARKS STATEMENT:
   PRODUCE FORTRAN CCDE
   PASS CCDE TO FORTRAN FILE
:TYPE=EOF:
   CLOSE FILES
   STOP
ENDCASE
   PASS INPUT STATEMENT TO SFARKS FILE
REPEAT
END
FUNCTION
This routine reads a card and determines the statement type. This type is then returned to the main program. If the code is defined as PURE FORTRAN it will be passed directly to the FORTRAN file.

COMMON BLOCKS
ALPH, DEVIC, FRTINF, INPUT, ODD, STYP2

HIGH-LEVEL CODE

PROC: DETNX(TYPE)
LOOP
  IF PTR=72 THEN DROP TO BOTTOM OF LOOP ENDF
  LOOP
  PTR=PTR+1
  IF PTR=72 THEN EXIT ENDF  // END OF CARD//
  TEST FOR FIRST NON BLANK
  IF SEMICOLON THEN CYCLE ENDF
  IF COLUMN 1 = C THEN
    TYPE=COMMENT
ENDIF
  CALL SCAN1  // LOCK FOR A BLANK//
  CASE
  :END-CF-LINE:
    TYPE=FORTRAN
  :COLUMN 6=NON BLANK:
    TYPE=CONTINUATION
  :PTR<6:
    LABEL, STORE IN BUFFER
  :ELSE:
    CALL STYTP  // DETERMINE STATEMENT TYPE//
  ENDCASE
  REPEAT
  READ CARD
  IF PURE FORTRAN THEN
    SEND OUTPUT DIRECTLY TO FORTRAN FILE
  ENDF
  REPEAT
END

B-13
SPECIFICATIONS
FOR
SUBROUTINE STYP

FUNCTION
This routine is called by subroutine DETNX once a word surrounded by blanks has been found. STYP compares this word with those in the keyword table. If a match is found it sets type to indicate the proper type of SPARKS statement. If no match is found the type is set to FORTRAN.

COMMON BLOCKS
ALPH, INPAT, INPUT, STYP1, STYP2, TABLES

HIGH-LEVEL CODE
PROC: STYP (TYPE)
    INITIALIZE POINTERS
    LOOP
        SEARCH KEYWORD TABLE UNTIL A FIRST CHARACTER MATCH IS FOUND
        LOOP
            TEST WORD CHARACTER BY CHARACTER
            IF MATCH IS FOUND THEN
                SET TYPE TO TYPE OF WORD FOUND
                RETURN
            ENDIF
            REPEAT MATCH NOT FOUND INCREMENT POINTERS TO
                LOOK AT NEXT WORD SURROUNDED BY BLANKS
            REPEAT
            TYPE = FORTRAN
    END
FUNCTION
This routine compares the pattern passed to it with all words surrounded by blanks in the input line. If a match is not found the end condition flag is set to true. If a match is found the end condition flag is set to false and a pointer is set to the start of the matching pattern.

COMMON BLOCKS

INPUT

HIGH-LEVEL CODE
FPROC: SCAN(PATTERN,LENGTH,ENDCONDITION)
    INITIALIZE POINTERS
    WHILE FTH .NE. END-OF-LINE DC
        CALL SCAN1 //LOOK FOR A BLANK//
        COMPARE PATTERN WITH WORD FOUND
        IF PATTERN MATCH THEN
            SET ENDCONDITION TO FALSE
            RETURN
        ENDIF
        INDEX POINTER
    REPEAT
    THE PATTERN WAS NOT FOUND
    SET ENDCONDITION TO TRUE
    RETURN
END
FUNCTION
This routine searches the input line for a given character. If the character is not found, an end condition flag is set to true. If the character is found, the end condition flag is set to false and a pointer is set to the character. This routine also reports an end condition for end-of-line (column 72), semicolon, and double-slash.

COMMON BLOCKS
ALPH, INPUT, STYP2

HIGH-LEVEL CODE
PROC: SCAN1 CHARACTER,ENDCONDITION
INITIALIZE POINTERS
WHILE PTR .NE. END-OF-LINE DO
CASE
  :NEXT TOKEN=SEMICOLON:
    SET CONDITION TO TRUE
    RETURN
  :NEXT TOKEN=SLASH:
    IF NEXT TOKEN+1=SLASH THEN
      SET ENDCONDITION TO FALSE
      RETURN
    ENDIF
  :NEXT TOKEN=CHAR:
    SET ENDCONDITION TO FALSE
    RETURN
ENDCASE
INDEX POINTERS
REPEAT
CHARACTER NOT FOUND IN LINE
SET ENDCONDITION TO TRUE
RETURN
END
FUNCTION
This routine is called by the main program after a SPARKS statement has been processed. It searches for a comment delimited by a doubleslash. If found it sets an end condition flag to false and a pointer to the end of the comment. If not found the end condition is set to true.

COMMON BLOCKS
ALPH, INPUT

HIGH-LEVEL CODE
PROC: SCAN2(ENDCONDITION)
   INITIALIZE FCINTER
   WHILE PTR .NE. END-OF-LINE DC
      IF NEXT TOKEN=BLANK THEN
         CYCLE
      ELSE
         IF NEXT TOKEN=S/LASH THEN
            FIND END OF COMMENT
            SET ENDCONDITION TO FALSE
            RETURN
         ENDF
      ENDF
   INDIx POINTER$ REPEAT
      COMMENT NOT FOUND
     SET ENDCONDITION TO TRUE
END
FUNCTION

This routine prepares FORTRAN statements for output to the FORTRAN file. It handles three types of statements: TYPE 1: DATA and FORMAT statements which are not indented; TYPE 2: MULTIPLE assignment statements which must be separated into component parts; TYPE 3: standard FORTRAN statements. It then determines if a SPARKS comment delimited by doubleslashes follows the statement.

COMMON BLOCKS

ALPH, FORT, FRTINF, FRT, INPUT, CDE, STYP2, TYF

HIGH-LEVEL CODE

PROC: FHAND
  IF LABEL FLAG IS FALSE THEN
    FILL FIRST SIX COLUMNS WITH BLANKS
  ENDIF
  CALL FTYP //TEST FOR DATA, FORMAT, OR //
    //MULTIPLE ASSIGNMENT STATEMENT//
  CASE
    :TYPE=DATA OR FORMAT:
      CALL FCUT //DO NOT INDENT//
      CALL FFRNT
    :TYPE=MULTIPLE ASSIGNMENT:
      PRODUCE INDIVIDUAL STATEMENTS
      CALL FCUT
      CALL FFRNT
    :TYPE=FORTRAN:
      CALL FCUT
      CALL FFRNT
  ENDCASE
  CALL SCAN2 //TEST FOR COMMENT/
RETURN
END
FUNCTION

This routine searches the FORTRAN statement to determine if it is a DATA, FORMAT, or MULTIPLE ASSIGNMENT statement. If found it will set type equal to the appropriate type.

COMMON BLOCK

ALPH, FORT, INFUT, CIFAT

HIGH-LEVEL CODE

PROC: FTYP(FORTRAN.TYPES)

    INITIALIZE FINDINTS
    IF FIRST.CHARACTER=F THEN
      TEST FOR FORMAT STATEMENT
      IF TRUE THEN
        FORTRAN.TYPES=FORMAT STATEMENT
        RETURN
      ENDIF
    ENDIF
    IF FIRST.CHARACTER=D THEN
      TEST FOR DATA STATEMENT
      IF TRUE THEN
        FORTRAN.TYPES=DATA STATEMENT
        RETURN
      ENDIF
    ENDIF
    WHILE PTR .NE. END-OF-LINE DO
      IF MULTIPLE ASSIGNMENT STATEMENT THEN
        FORTRAN.TYPES=MULTIPLE-ASSIGNMENT STATEMENT
        RETURN
      ENDIF
    REPEAT
    FORTRAN.TYPES=FORTRAN
    RETURN

END
SPECIFICATIONS
FOR
SUBROUTINE FOUT

FUNCTION
This routine copies FORTRAN statements from the input buffer
to the output buffer according to the structuring rules.

COMMON BLOCKS
ALPH, FORT, FRTINF, INPUT, STYF2, TYF

HIGH-LEVEL CODE

PROC: FOUT(PATTERN, START, END)
    WHILE START .NE. END DC
        IF POINTER=MAX THEN
            CALL FPRNT // PRINT LINE //
            ESTABLISH A CONTINUATION LINE
            RESET POINTERS
        ENDF
        COPY CHARACTER TO BUFFER
        INCREMENT START
    REPEAT
    RETURN
END
FUNCTION

This routine fills all unused spaces in the output buffer with blanks. Then writes PCBTRAN statements to the output file which is passed to the compiler.

COMMON BLOCKS

LEVIC, FORT, FRTINF, FRTT, INPUT, SERINF

HIGH-LEVEL CODE

PROC: PFRNT
  IF START .GT. 7 THEN
    INSERT BLANKS UP TO START OF STATEMENT
  ENDIF
  IF END .LT. 80 THEN
    INSERT BLANKS AT END OF STATEMENT
  ENDIF
  WRITE LINE TO FILE
  RESET POINTERS FOR NEW LINE
  RETURN
END
FUNCTION

This routine writes SPARKS statements to the SPARKS output file. A 132-character line is used for the SPARKS output.

COMMON BLOCKS

ALPH, DEVIC, INPUT, ODL, SPHINE, STYE2, TYP

HIGH-LEVEL CODE

PROC: SPRTN
    IF LABEL THEN
        COPY LABEL INTO BUFFER
    ELSE
        COPY BLANKS INTO BUFFER
    ENDIF
    SET POINTERS TO START AND END OF LINE
    WRITE LINE TO FILE
    RESET POINTERS
    RETURN
END
FUNCTION

This routine handles EXIT and CYCLE statements. It tests for a multilevel EXIT/CYCLE statements. If found, the appropriate EXIT/CYCLE is coded and passed to the output file. If no level is specified it EXIT/CYCLE's one level. It also sets a flag to indicate an EXIT/CYCLE has been processed.

COMMON BLOCKS

CENTERL, PTRI, INDI, INPUT, ODD, KEYW2, CTEAT, STYP3

HIGH-LEVEL CODE

PROC: EXCYL(TYPE)
    SET FLAG=EXIT/CYCLE
    CALL LABEL // FILL LABEL BUFFER //
    CALL SCAN2 // TEST FOR COMMENT //
    IF NOT A COMMENT THEN
        WHILE NOT END-OF-LINE DC
            TEST FOR EXIT/CYCLE LEVEL
            DETERMINE LABEL TO WHICH TO BRANCH
        REPEAT
    ENDF
    CALL FCUT // PRODUCE CODE FOR EXIT/CYCLE //
    CALL FPRNT
    RESET POINTERS
    RETURN
END
FUNCTION
This routine is called by the main program each time a SPARKS statement terminator is encountered. A check of the current stack top is made. If the terminator type matches the stack top, then a flag is set false. If a match is not found subroutine ENDFIX is called to try to find a match. Each time a mismatch is found subroutine ERROR is called to produce an error message.

COMMON BLOCKS
CENTRL, FRTINF, INPAT, INPUT, KEYW1, KEYW2, SPRINF, STYP1, STYP2, STYP3

HIGH-LEVEL CODE
PROC: CHECK(STYPE,BAD)
   IF STACK EMPTY THEN PRINT ERROR ENDIF
CASE
   :STYPE=COLON OR ENDCASE:
      IF STACK TOP=CASE THEN
         BAD=FALSE
      ELSE
         CALL ENDFIX
      ENDIF
   :STYPE=UNTIL:
      IF STACK TOP=LOOP AND NEXT=REPEAT THEN
         BAD=FALSE
      ELSE
         CALL ERROR
         BAD=TRUE
      ENDIF
   :STYPE=REPEAT OR EXIT OR CYCLE :
      IF STACK TOP=LOOP OR WHILE OR FOR THEN
         BAD=FALSE
      ELSE
         CALL ERROR
         BAD=TRUE
      ENDIF
B-24
:TYPE=ENDIF:
    IF STACK TCP=IF THEN
    BAD=FALSE
ELSE
    CALL ERROR
    BAD=TRUE
ENDIF
:TYPE=END:
    IF STACK TCP=PROGRAM THEN
    BAD=FALSE
ELSE
    CALL ENDFIX
ENDIF
ENDCASE
RETURN
END
FUNCTION
This routine is called by subroutine CHECK or the main program when a mismatch condition exists between the stack top and statement terminator. ENDFIX unstacks and produces the statements necessary to terminate the stack top statement. It does this until a match is found or the stack is empty.

COMMON BLOCKS
CENTAL, PRINIF, INEUT, KEYW1, CDE, CIFAT, STYP3

HIGH-LEVEL CODE
PROC: ENDFIX(END.TYPE, STACK.TCP, STATEMENT.TYPE, DUMM)
WHILE END.TYPE NOT EQUAL STACK.TCP DO
CASE
:STACK.TCP=IF:
  PRODUCE ENDF STATEMENT
:STACK.TCP=CASE:
  PRODUCE ENDCASE STATEMENT
:STACK.TCP=SUBROUTINE:
  PRODUCE END STATEMENT
:ELSE:
  PRODUCE REPEAT STATEMENT
ENDCASE
DECREMENT STACK
IF STACK EMPTY THEN
  RETURN
ENDDIF
REPEAT
RETURN
END
FUNCTION

This routine produces an error message based on the input code. ERRCR contains the statements necessary to build the appropriate message.

COMMON BLOCK

LEVIC, INBUF

HIGH-LEVEL CODE

PROC: ERROR(CODE)
     DETERMINE TYPE MESSAGE TO BE PRINTED
     SET POINTER TO STATEMENT TO BE PRINTED
     WRITE MESSAGE
     RETURN
END
FUNCTION
This routine fills the first 5 columns of the FORTRAN buffer with the label passed or with blanks. It also sets column 6 to a blank.

COMMON BLOCKS
CENTRAL, PRTRNF, INUPUT, TYP

HIGH-LEVEL CODE
PROC: LABFIL(LABEL, POINTER)
FOR I=1 TO 5 DO
  CCFY LABEL TO BUFFER
  REPEAT
  COLUMN 6=BLANK
  RETURN
END
FUNCTION

This function generates labels for all FORTRAN statements as required. If the statement already has a label, it copies that label into the label table.

COMMON BLOCKS

CONTROL, INPUT, ODD

HIGH-LEVEL CODE

PROC: GENLAB (GEN)

IF LABEL IS PRESENT THEN
  COPY LABEL INTO LABEL TABLE
  SET LABEL TABLE POINTER
  RETURN
ELSE
  GENERATE A LABEL
  COPY LABEL INTO LABEL TABLE
  SET POINTER
ENDIF
INDEX POINTERS
RETURN
END
SPECIFICATIONS FOR FUNCTION CCNV

FUNCTION
This function converts a digit passed into its character representation.

COMMON BLOCK
CED

ABSTRACT CODE
PROC: CONV(DIGIT)
    CONVERT DIGIT TO ALPHANUMERIC
RETURN
END
APPENDIX C

SPARKS CODE
NESTBLKSTA

THIS IS A SPARKS PROGRAM

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

C 0 0 0
C 0 1
IMPLICIT INTEGER (A-Z)
C 0 2
LOGICAL EENCE1,EENCE2,BADELRFLG
C 0 3
COMMON/ALPH/GCCE,CEE,ACE,FCCE,PARK,COMMON,DESLF
C 0 4
COMMON/CENTPL/STACK(200),LABEL(200),STERM(6),TOP,LIBMAX,
C 0 5
LEFTA,TPITF,NUM2,STMAX
C 0 6
COMMON/DEVIC/PRTFR,FILE,CRDR
C 0 7
COMMON/FRTN/EHLF(8C),FPTR,FMAX,FSTRT,1NCR
C 0 8
COMMON/FRT/IOTA(4),FMAT(6),FAXE,FST,1NCR
C 0 9
COMMON/INOT/INDFLG,CLRFLG,NECTR,BKWD,FWDF,NEW,OLD,PULCUT
C 0 10
COMMON/INPAT/TOO(2),THENN(4),BYE(2),DOO(2),ELZ(4),CCLN
C 0 11
COMMON/INPT/SPIN(3),TREC,TCEN,STREG,TLOG,ILOG,ICLON
C 0 12
COMMON/KEYL/LEED(3),ENCASS(7),ERIF(5),CASS(4),SIF(2),CCOLON(5)
C 0 13
COMMON/KEYB/EEXT(4),CycL(5),COUNT(5),1TER(9),REET(6),DOF(2)
C 0 14
COMMON/CDL/LEN(10),LBF,LABC
C 0 15
COMMON/CIP/CGOT(4),CANN(8),EQL(1),GOTO(6),1FNOT(6),
C 0 16
LEN(5),MLNL(1),FLL(1),RSTAP(2),CNE(1)
C 0 17
COMMON/SPP/STRT,SIMEN,BLKNR,NESTNC,SMAX,DLNK(131)
C 0 18
COMMON/STABILITY/CELCR,OBLSL,ELSE,UNTIL,LLCPP,END,ENDCAS,PREP
C 0 19
COMMON/STMP2/CJ,INLC,NCYCLE,NHE,CCNT,COMT,FOIRN,SEMI
C 0 20
COMMON/STMP3/SIF,CCASE,CONDN,FCCF,EEFIT,ENDT,EENDIF,RETURN
C 0 21
COMMON/TABLE/TRES(23),RESRD(20)
C 0 22
COMMON/TYP/TYPE
C 0 23
DIMENSION EXP(80)
C 0 24
// WRITE TITLES ETC. //
C 0 25
WRITE(PRTFR,3)ISP(1),I=1,80
C 0 26
FORMAT(3X,HREST,2X,2HBLK,2X,4HSTNC,2X,30X,80A1)
C 0 27
WRITE(PRTFR,4)
C 0 28
FORMAT(32X,'SPARKS TRANSLATION BY KSU SPARKS PREPROCESSOR',
C 0 29
AS/32X,'MODIFIED BY J. MARTIN, R. STIEFEL, AND M. CALHOUN.'/
C 0 30
// SET UP OPTIONS //
C 0 31
IMAX=72
C 0 32
SMAX=100
C C 33
C C 34
C C 35
C C 36
C C 37
C C 38
C C 39
C C 40
C C 41
C C 42
C C 43
C C 44
C C 45
C C 46
C C 47
C C 48
C C 49
C C 50
C C 51
C C 52
C C 53
C C 54
C C 55
C C 56
C C 57
C C 58
C C 59
C C 60
C C 61
C C 62
C C 63
C C 64
C C 65
C C 66
C C 67
C C 68
C C 69
C C 70

FMAX=72
INCK=1
FSTRT=7+INCR
SSTRT=7+INCR

// EXTRACT OPTIONS GIVEN ON THE CONTROL CARD AND CHANGE ABOVE //
// DEFAULTS ACCORDINGLY---THIS COULD USE DETNX //
// INITIALISE THE PROGRAM RELATED VARIABLES. //
SIACK(11)=SUBFN
TCP=1
TFITR=0
LPTR=6
TCLEN=IMAX+1
BLKNC=C
TMC=0
MERR=0

// FACCESS STATEMENTS ACCORDING TO TYPE //
// 1. TRANSLATE AND PRINT FORTRAN. //
// 2. SET INDETHATION FLAG--INDFLG. //
// 3. SET CURFLG NEW-CLD--CURRENT STATEMENT TO BE PRINTED IN //
// CLD OR NEW INDETHATION. //
// START MAIN LCCP //
LCCP

FPTR=FSTRT
CALL DETA(C(TYPE)
IF TYPE.EQ.RINLC THEN
A=TIBG
E=TCLEN
CALL DETA(C(TYPE)
// IF THE SECOND TOKEN IS NOT FRACTION THEN PASS AS FORTRAN. //
IF TYPE.AC.SUBFN THEN
TYPE=RETRA
ENDIF
TCFLG=A
TCEND=B
ENDIF
// THE MAIN CASE STATEMENT //
STREG=TCREG
CASE
: TYPE.LG.FCTRNL:
  CALL FHAND
  STEND = TCEND
  CURFLG = CLD
  INCFLG = NEUTR

: TYPE.LG.ODESL:
  TCEND = TCEND + 1
  // LOOK FOR A DOUBLE SLASH--SCAN IS NOT USED SINCE IT STOPS //
  // OR A SEMICOLON. //
  LCP
    IF SPIN(TCEND).EQ.DBSLH.AND.SPIN(TCENC+1).NE.DBSLH THEN
      EXIT
    ENDIF
    IF TCEND.GE.IMAX-1 THEN
      CALL ERRNR(2, DBSLH, 2+1, 0, C)
      // ERROR--MISSING DOUBLE SLASH SUPPLIED AT END //
      EXIT
    ENDIF
    TCEND = TCEND + 1
  REPEAT
    TCENC = TCENC + 1
    STEND = TCEND
    INCFLG = NEUTR
    CURFLG = CLD

: TYPE.LG.IIF:
  CALL SCAN(TEND, 4, ECCNC)
  IF ECCNC THEN
    // ABSENCE OF THEN ASSUMED TO INDICATE FORTRAN IF. //
    TCBEG = TCBEG - 2
    CALL FHAND
  ELSE
    LAE = GENLAE(1)
    // CLTPLT: IF (.NOT.CXRD) GC TO LAB //
    IF .NOT.LBFLG THEN
      CALL LAEFILL(LABJ)
    ENDIF
    CALL FOUT(IFNOT, 1, 9)
    CALL FCLT(SPIN, TCBEG, TCEND)
CALL FOLT(CGCTG,1,9)
CALL FOLT(LABEL,LAB,LAB+4)
CALL FPRT
STACK(TCP+1)=LAB
STACK(TLP+2)=LAB
STACK(TCP+3)=IF
TCP=TCP+3
IF TCP.GT.STMAX THEN
   CALL ERR(6,0,0,5,0,0)
ENDIF
   // STACK OVERFLOWED, PROGRAM TERMINATED. //
TCEND=TCPEND+4
IF FSTRT+INCR.LT.IMAX THEN
   FSTRT=FSTRT+INCR
ENDIF
ENDIF
CALL SCAN2(ECEND)
STEND=TCEND
CLRFGLG=CLD
INCFGLG=FRWrD
: TYPE.TC.ENCOND:
CALL CHECK(ECEND,BAD)
IF BAD THEN
   CYCLE :
   ENDIF
FSTRT=FSTRT+INCR
FPTR=FSTRT
IF LBFLG THEN
   // LABELLED STATEMENT--GENERATE 'LAB CONTINUE' //
   CALL FOLT(CCNT1,1,8)
   CALL FPRT
   ENDIF
LAB=STACK(TCP-1)
LEFT=STACK(TCP-2)
TCP=TCP-3
CALL LAGFIL(LAB)
CALL FCUT(CCNT1,1,8)
CALL FPRT
229147
229148
225149
229150
225151
225152
230153
230154
231155
231156
232157
232158
232159
233160
233161
233162
233163
233164
234165
234166
234167
235168
235169
235170
235171
235172
236173
236174
236175
236176
236177
236178
236179
236180
236181
236182
236183
237184

// GENERATE A LAB CONTINUE. //
CALL SCAN2 (ECN0)
STEAD=ECN0
IND0=IND0
CUR0=NEW
: TYPE=EQ ELSE :
CALL CHECK(ELSE,BAC)
IF ELSE THEN
    CYCLE :
END IF
FSTRT=FSTRT-1NCR
FPTR=FSTRT
IF LBL THEN
    CALL ERCR(1,ELZ,4,2,0,0)
    // ERROR--LABELLING THE ELSE HAMPERS PROGRAM CLEANLINESS-- //
    // DELETE FROM THE FORTRAN OUTPUT. //
    CALL LAE2(LAB0)
END IF
LAB2=GENA(E1)
LAB1=STACK(TCP-2)
IF NLAST.NE.1 THEN
    CALL LAE2(LAB0)
    CALL FOCT(GCTO,1,6)
    CALL FOCT(LABLE,LAB2,LAB2+4)
    CALL FPRNT
END IF
// GC TO LAB2 //
CALL LAE2(LAB1)
CALL FOCT(CONT1,1,8)
CALL FPRNT
// LAB1 CONTINUE //
STACK(TCP-1)=LAB2
226178
226179
226180
226181
226182
226183
227189
227190
227191
227192
227193
227194

// IF FSTRT+1NCR.LT.*MAX THEN
FSTRT=FSTRT+1NCR
ENDIF

: TYPE. EQ. FEQF :
IF LBFGL THEN
// LAB WILL HAVE IDENTIFYING LABEL IF ANY. //
LAB=GENLABIC
CALL FCUT(CTN1,1,8)
CALL FPRTN
ENDIF
TCBG=TCEND+1
CALL SCAN1(FCL, ECCND)
IF ECCND THEN
CALL ERROR(2,FCL,1,3,0,0)
// ERROR--MISSING EQUAL-STATEMENT SKIPPED. //
CYCLE
ENDIF
VBLEA=TCBG
VBLEE=TCEND
TCEND=TCEND+1
CALL SCAN(TCC,2, ECCND)
IF ECCND THEN
CALL ERROR(2, TCC,2,3,0,0)
// ERROR--MISSING 'TO' - STATEMENT SKIPPED. //
TCEND=TCEND+1
CYCLE
ENDIF
TCEND=TCEND+2
IF NOT. LBFGL THEN
CALL LAEFIL(LABJ)
ENDIF
CALL FCUT(SPIN, VBLEA, TCEND-2)
CALL FPRTN
// VARIABLE=EXPRESSIONAL. //
ITMP=TCEND
CALL SCAN(BYE,2, ECCND1)
EX2A=TCBG
EX2B=TCEND
TCEND=TCEND+2
IF ECCND1 THEN
TCEND=ITMF
ENDIF.
CALL SCAN(DCC,2,ECOND)
IF ECCNC1 THEN
  EX2R=TCEND
ENDIF
IF ECCNC THEN
  CALL ERROH(2,DOG,2,1,J,J)
  // ERROR--MISSING 'DO'--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 LAB2 IS THE TARGET OF CYCLES. //
STACK(TCP+1)=LAB2
STACK(TCP+2)=LAB3
STACK(TCP+4)=LAB3
STACK(TCP+5)=LAB4
STACK(TCP+6)=TPR
STACK(TCP+2)=TFTR
TCP=TCP+6
IF TCP.GT.STMAX THEN
  CALL ERROR(0,0,0,0,5,0,0)
ENDIF
// STACK OVERFLOWED----PROGRAM TERMINATED. //
TPR=TCP
CALL LABFIL(LAB0)
CALL FCT(GUTC,1,6)
CALL FFIL(1,LAB1,LAB1+4)
CALL FPNT
// GC TC LAB1. //
CALL LABFIL(LAB2)
CALL FCT(IN3IN1,IN3LEA,IN3LEN)
CALL FCLT(EUL,1,1)
CALL FCLT(SP,FVLEA,VBLE)
CALL FCLT(FL,1,1)
IF ECCN1 THEN
   // THE 'BY' IS MISSING--DEFAULT INCREMENT OF ONE. //
   CALL FCLT(ONE,1,1)
ENDIF
LD
   CALL FCLT(IFNC,5,9)
   CALL FCLT(SP,FVREG,TEND-2)
   CALL FCLT(FSTAR,1,1)
ENDF
CALL FPRNT
   // LAB2--INDICATES VARIABLE=VARIABLE+ EXPRESSION3. //
   CALL LABFIL(LAB1)
   CALL FCLT(IFNC,1,9)
   CALL FCLT(IFNC,9,9)
   CALL FCLT(SP,FVLEA,VBLE)
   CALL FCLT(MINS,1,1)
   CALL FCLT(IFNC,9,9)
   CALL FCLT(SP,EX2A,EX2B)
   CALL FCLT(FSTAR,1,1)
   CALL FCLT(FSTAR,1,1)
IF ACT.ECNC1 THEN
   // IF 'BY' MISSING THEN GENERATE...(EXP2).LE.0.... //
   // IN PLACE OF...(EXP2)*(EXP3).LE.0..... //
   CALL FCLT(FSTAR,2,2)
   CALL FCLT(IFNC,9,9)
   CALL FCLT(SP,FVREG,TEND-2)
   CALL FCLT(FSTAR,1,1)
ENDIF
CALL FCLT(ECG,1,1)
CALL FCLT(CGTC,1,9)
CALL FCLT(LABEL,LAB4,LAB4+4)
CALL FPRNT
   // LAB1...IF(.ACT.(VBLE-(EXP3).LE.0)) GO TO LAB3... //
   CALL SCAN2 (ECCN2)
STND=TCN1
INCLG=FAK1D.
CURRFLG=CLD
IF FSRT+INCR.LT.IMAX THEN
    FSRT=FSRT+INCR
ENDIF

: TYPE. EQ. PREPT :
CALL CHECK(PREPT,PAC)
IF BAD THEN
    CYCLE
ENDIF

IF LDFLG THEN
    // LABELLED STMNT, GENERATED 'LAB CONTINUE' //
    CALL FCLT(CCNDT,1,0)
    CALL FPRTN
ENDIF

STTCP=STACK(TCP)
// LAB3=EXIT TARGET, LAB2=CYCLE TARGET, LAB1=LOOP BACK LABEL //
IF STTCP.EQ.FCRC THEN
    LAB1=STACK(TCP-5)
    LAB2=STACK(TCP-2)
    LAB3=STACK(TCP-1)
    TCP=TCP-6
    LEPTN=LEPTN-20
ELSE
    LAB1=STACK(TCP-3)
    LAB2=STACK(TCP-2)
    LAB3=STACK(TCP-1)
    TCP=TCP-5
    LEPTN=LEPTN-15
ENDIF

TPITR=STACK(TPITR-4)
FSRT=FSRT+INCR
HPTR=FSRT
CALL LABFIL(LAB2)
CALL FCLT(CCNDT,1,0)
CALL FCLT(LABEL,LAB1,LAB1+4)
CALL FPRTN
// GC TC LABEL 1 //
CALL LABFIL(LAB3)
CALL FCLT(CJNT1,1,8)
CALL FPRT
// LAB3 CONTINUE //
CALL SCAN2 (ECNDC)
STEND=TCNDC
CURFLG=NEW
INDFLG=FRWRD

: TYPE EC CCASE :
LAB=GENLAB(1)
// LABEL(1:5)=E LANKS. //
STACK(TCP+1)=1
STACK(TOP+2)=LAB
STACK(TCP+3)=CCASE
TOP=TCP+2
IF TCP.GT.STMAX THEN
    CALL EFFCR(6,0,0,5,0,0)
ENDIF
// STACK OVERFLOWED------PROGRAM TERMINATED. //
EXPTR=C
CALL SCAN(COF,2,ECNDC)
IF *.ACT.ECCNC THEN:
    TCNDC=TCFAD+2
    EXPST=TCNDC
    CALL SCAN1(EBGLH,ECNDC)
    K=0
    FCN(I=EXPST+1 TP TCNDC DO
        K=K+1
        EXP(K)=SPN(I)
    REPEAT
    EXPTR=K
ENDIF
CALL SCAN2 (ECNDC)
STEND=TCNDC
CURFLG=GLC
INDFLG=FRWRD
IF LRFLG THEN
    // LABEL1EC STATEMENT--GENERATE 'LAB CONTINUE'. //
    CALL FCLT(CNT11,1,8)
CALL FPRINT
ENDIF

; SET THE INCREMENT
IF FSTRT+INCR.LT.IMAX THEN
FSTRT=FSTRT+INCR
ENDIF

: TYPE. EQ.COLON :
CALL CHECK(COLON,BAD)
IF BAD THEN
CYCLE
ENDIF

IF LABFLG THEN
CALL ERR(1,CASS,4,2,0)
// ERROR--LABELLING THE CASE HAMPERED PROGRAM CLEANLINESS
// DELETED FROM THE FORTRAN OUTPUT. //
CALL LABFIL(LAB0)
ENDIF

LAB1=STACK(TCP-1)
LAB2=STACK(TCP-2)
FSTRT=FSTRT-1ACA
FPIK=FSTRT
// CASE MATERIAL PRINTED ONE IDENT TO THE RIGHT. //
IF LAB2.NE.1.AND.NLAST.NE.1 THEN
CALL FJUT(GCTO,1,6)
CALL FCLT(LABEL,LAB1,LAB1+4)
CALL LABFIL(LAB0)
CALL FPRINT
// GC TO LAB1. //
ENDIF

CALL SCAN1(COLON,ENDC)
IF ENDCN THEN
CALL ERR(2,CCOLON,1,1,0,0)
ENDIF

// ERROR--MISSING COLON-SUPPLIED AT END OF STATEMENT. //
K=ENDC
TEND=TEND+1
// SPIN(K)=IMAX OR SPIN(K+1)=SEMICOLON OR COLON. //
MAX=IMAX
IMAX = K
CALL SCAN(ELZ, 4, ECCND)
IMAX = IMAX
IF ECCND THEN
  LAB3 = GEALAH(1)
  TCEND = K
  // CPUTL-LAB2-1F ACT CONDITION THE GO TO LAB3. //
  CALL LAFIL(LAB2)
  CALL FCUT(IFNL1, 1, 9)
  IF EXPTR.GT.0 THEN
    CALL FCUT(EMP1, 1, EXPTR)
  ENDIF
  CALL FCUT(SPIN, TCBG, TCEND)
  CALL FCUT(CCCTC, 1, 9)
  CALL FCUT(LABEL, LAB3, LAB3 + 4)
  CALL FPRTK
  TCEND = TCEND + 1
ELSE
  // ELSE Clause Found //
  CALL LAFIL(LAB2)
  CALL FCUT(CCONT1, 1, 8)
  CALL FPRTK
  // LAB2 CONTINUE //
  LAB3 = K + 1
  LAB2 = 1
  // Label(1....5) ARE BLANKS. //
ENDIF
STACK(IVCF-2) = LAB3
CALL SCAN2(ECCND)
STEND = TCEND
INDEX = INDEX
CUR = LG = NEXTP
CUR = LG = PULCUT
// RESTORE THE INCENTATION. //
IF PSTRT + INCR.LT. IMAX THEN
  PSTRT = PSTRT + INCR
ENDIF
: TYPE.EC.ENUCAS :
CALL CHECK(EUCAS, BAD)
IF BAC THEN
  CYCLE:
ENDIF
FSTRT=FSTRT-1NCR
FPTL=FSTRT
IF LBFLG THEN
  // LABELLED STATEMENT--GENERATE "LAB CONTINUE" //
  CALL FCST(CCONTI,1,8)
  CALL FPRT
ENDIF
LAB1=STACK(TCP-1)
LAB2=STACK(TCP-2)
TCP=TCP-3
L3PTL=LAB1
IF LAB2.NE.1 THEN
  CALL LABFIL(LAB2)
  CALL FCST(CCONTI,1,8)
  CALL FPRT
ENDIF
// LAB2 CONTINUE //
CALL LABFIL(LAB2)
CALL FCST(CCONTI,1,8)
CALL FPRT
// LAB1 CONTINUE //
CALL SCAN2 (ECCNO)
STENC=TCENC
INDFLG=FKWPD
CLRFLG=NEW
EXPTH=0
: TYPE.EQ.RETRN :
LAST=1
FTYPE=0
CALL FHAND
STENC=TCENC
CURFLG=CLD
INDFLG=MEUTR
: TYPE.EQ.LLCOP :
// LAB2 IS THE TARGET OF CYCLE AND LAB3 OF EXITS //
LAB1 IS THE IDENTIFYING LABEL.

LAB1=GEMLAB(C)
LAB2=GEMLAB(1)
LAB3=GEMLAB(1)
STACK(TCP+2)=LAB1
STACK(TCP+3)=LAB2
STACK(TCP+4)=LAB3
STACK(TCP+5)=LLGCP
STACK(TCP+1)=TPIR
TCP=TOP+5

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

STACK OVERFLOWED, PROGRAM TERMINATED

CALL LABFIL(LAB1)
CALL FCUTCCNTI,1,8
CALL FPANT
TPIR=TCP
CALL SCAN2 (ECCND)
STEND=ICEND
CURFLG=GLD
INCFLG=FRWC

IF FSTRT+INCR.LT.IMAX THEN
    FSTRT=FSTRT+INCR
ENDIF

TYPE.LEQ.WHILE:

LAB1=GEMLAE(C)
LAB2=GEMLAB(1)
LAB3=GEMLAB(1)

LAB2 IS THE TARGET OF EXITS, LAB3 FOR CYCLES.

CALL SCAN(CCC,2,ECCNC)

IF LCON THEN
    CALL ERPLF(2,CCE,2,1,0,0)
    // CALL ERROR----MISSING 'DC'--SUPPLIED
    TLEN=TLEN+1
ELSE
    TLEN=TLEN+2
ENDIF
IF .NOT. LABLEG THEN
   CALL LABEL1(LAB1)
ENDIF
CALL FCLT(IFCT,1,9)
CALL FCLT(SPIN,TOBEG,TCENC-2)
CALL FCLT(CGCTG,1,9)
CALL FCLT(LABEL,LAB2,LAB2+4)
CALL FRNT
// GENERATE LAB1 AND IF TRUE GND GO TO LAB2. //</br>
STACK(TCP+2)=LAB1
STACK(TCP+3)=LAB3
STACK(TCP+4)=LAB2
STACK(TCP+5)=WHILE
STACK(TCP+1)=TRIFR
TCP=TCP+5
IF TCP GT STMAX THEN
   CALL ERROR(6,0,0,5,0,0)
ENDIF
// STACK OVERFLOWED, PROGRAM TERMINATED. //</br>
TRIT=TCP
CALL SCAN2 (ECND):
STENC=TCENC
CURFLG=CLC
INDFLG=FRAR
IF FSTRT+INCR LT IMAX THEN
   FSTRT=FSTRT+INCR
ENDIF
: TYPE EQ CENT :
FCK I=1 TC 6 DC
   FBUF(I)=SPIN(I)
   REP=AT
TCBE=7
WHILE SPIN(TCBE).EQ.BLANK DC
   TCBE=TCBE+1
IF TCBE.LT.IMAX THEN
   TCENC=IMAX+1
   // BLANK CONTINUATION CAPC, CLEATED. //</br>EXIT
ENDIF
REPEAT
IF TCEND.GT.IMAX THEN
  CYCLE;
ENDIF
// SPIN(TCHEG)=ACNNBLANK; REST OF CARD CAN BE INDENTED. //
CALL FF=NC
STBCG=TCHEG
STEND=TCEND
CURFLG=CLD
INDFLG=DELTR
: TYPE.EQ.EXIT:
CALL CHECK(EXIT,BAD)
IF BAD THEN
  CYCLE;
ENDIF
CALL EXCYL(TYPE)
: TYPE.EQ.NCYCLE:
CALL CHECK(NCYCLE,BAD)
IF BAD THEN
  CYCLE;
ENDIF
CALL EXCYL(TYPE)
: TYPE.EQ.UNTL:
CALL CHECK(UNTL,BAD)
IF BAD THEN
  CYCLE
ENDIF
FSTRT=FSTRT-INCR
FTPRT=FSTRT
LAHI1=STACK(TCP-3)
LAHI2=STACK(TCP-2)
LAHI3=STACK(TCP-1)
TCP=TCP-5
LPPTF=LPPTF-15
TPITR=STACK(TPITR-4)
IF LELFLG THEN
  CALL ERCFI(1,UNTL,5,2,0,0)
// ERROR --- LABELLING OF THE UNTIL HAMPERS PROGRAM //
// CLARITY, DELETED FROM FORTRAN OUTPUT. //
ENDDF
CALL LABFIL( LAB2)
CALL FOUTI(IFACT,1,9)
CALL SCAN(REPT,6,ECCND)
CALL FCUT(SPINTUBEG,TCEND)
IF ECCND THEN
   CALL EPLCF(2,REPT,6,1,0,0)
   // ERROR --- MISSING REPEAT-SUPPLIED //
   TCEND=TCEND+1
ELSE
   TCEND=TCEND+6
ENDDF
CALL FCUT(CGOCT,1,9)
CALL FCUT(LABEL,LAB1,LAB1+4)
CALL FPRI
// GENERATE IF (.NOT.(CCEND)) GO TO LAB1 //
// GENERATE THE EXIT TARGET --- LAB3 CONTINUE. //
CALL LABFIL(LAB3)
CALL FCUT( OCCAT,1,8)
CALL FPRI
CALL SCAN2(ECCND)
STENC=TCEND
INDFLG=8KWD
CURFLG=NEW
: TYPE_EG_SUBLF:
IF TCP.ME. < THEN
   STTCP=STACK(TOP)
   CALL ENDFIX(EEND,STTOP,TYPE,ECCND1)
   CALL ERROR(7,J;G,6,C,J)
   // PREMATURE START //
   PSTART=7
   SSTART=10
ENDIF
WRITE(PTR,2C)
FORMAT(1H1)
LBPTR=6
TCP=0
NUMB=1000000
STACK(TCP+1)=SLB
TCP=TCP+1
CALL FFANG
FSTRT=FSTRT+INCR
STENC=TCENC
CURFLG=CLD
INDEX=FORWD

: TYPE.EQ.EENC :
CALL CHECK(EENC,EAC)
IF BAL THEN
    CYCLE :
ENDIF
FSTRT=FSTRT-INCREMENT
FPTR=FSTRT
IF LABFLG THEN
    // LABELLED STATEMENT--GENERATE 'LAB CONTINUE' //
    CALL FCLT(CCNT1,1,8)
    CALL FPRINT
ENDIF
TCP=TCP-1
CALL LABFIL(LABO)
CALL FCLT(SPIN,TCEG,TCENC)
CALL FPRINT
CALL Scan2(ECCND)
STENC=TCENC
INDEX=FORWD
CURFLG=NEW

: TYPE.EQ.CCMT :
TCENC=IMAX
TCPFG=2
WHILE SPIN(TCEG).EQ.BLANK DC
    TCBE=TCBE+1
REPEAT
STBE=TCBE
STEND=IMAX
CURFLG=CLD
INDFLG=NEILTR
: TYPE/File
    // CLOSE THE FILE. //
WRITE(PTR1,EEGO)
FORMAT(/3F)
END OF SPARKS PROGRAM.....
STOP
ENDCASE
IF CURFLG.EC.FULLLT THEN
    SSTRT=SSTRT-INC
    CALL SPRNT
    SSTRT=SSTRT+INC
    IF SSTRT.GT.SMAX THEN
        SSTRT=SSTRT-INC
    ENDIF
    BLKNC=BLKNC+1
ENDIF
IF CURFLG.EC.CLB THEN
    CALL SPRNT
ENDIF
IF INDFLG.EC.FKWR THEN
    SSTRT=SSTRT+INC
    IF SSTRT.GT.SMAX THEN
        SSTRT=SSTRT-INC
    ENDIF
    IF CURFLG.EC.CLB THEN
        CALL SPRNT
    ENDIF
    BLKNC=BLKNC+1
NEXTC=NEXTC+1
ENDIF
IF INDFLG.EO.EKWR THEN
    SSTRT=SSTRT-INC
    IF CURFLG.EO.CLB THEN
        CALL SPRNT
    ENDIF
    NEXTC=NEXTC-1
    BLKNC=BLKNC+1
ENDIF
-1  175   720
  BLOCK DATA
C   176   721
  IMPLICIT INTEGER(A-Z)
C   176   722
C   176   723
C   176   724
C   176   725
C   176   726
C   176   727
C   176   728
C   176   729
C   176   730
C   176   731
C   176   732
C   176   733
C   176   734
C   176   735
C   176   736
C   176   737
C   176   738
C   176   739
C   176   740
C   176   741
C   176   742
C   176   743
C   176   744
C   176   745
C   176   746
C   176   747
C   176   748
C   176   749
C   176   750
C   176   751
C   176   752
C   176   753
C   176   754
C   176   755
C   176   756
C   176   757
DATA CCASE, CCLCN, LSLSL, SELSE, EEND, E00, EEXIT, FFOR, CONT,
DATA FEXTN, IFL, LLCPP, REPT, UNTL, WHILE, SUBFN, RINLO
DATA CYCLE, ENDCAS, ENDIF, RETRN
DATA RESRD/5, 1H5, 1HL, 1HG, 1HC, 1HK, 5, 4, 1HC, 1HA, 1HS, 1HE, 1,
5, 1HC, 1HY, 1HC, 1HL, 1HE, 19, 4, 1HE, 1FL, 1FS, 1HE, 9,
DATA 3, 1HE, 1HN, 1HS, 17, 1HF, 1HN, 1HC, 1FA, 1FS, 1HE, 20,
5, 1HF, 1HA, 1FC, 1HI, 1HF, 21, 3, 1FL, 1HU, 1HJ, 11,
DATA 4, 1HE, 1AX, 1HI, 1HT, 12, 3, 1HF, 1FC, 1HK, 2,
DATA 9, 1HF, 1HU, 1HA, 1HC, 1HT, 1HI, 1HC, 1HN, 5, 2, 1HL, 1HF, 3,
DATA 7, 1HI, 1HN, 1HT, 1HE, 1FC, 1HE, 1R, 18, 7, 1HL, 1FO, 1HG, 1HI, 1HC,
1HA, 1HL, 1B, 1F, 1HE, 1FC, 1HE, 1R, 18, 7, 1HL, 1FO, 1HG, 1HI, 1HC,
DATA 6, 1HF, 1HE, 1FC, 1HE, 1FO, 1HF, 16, 6, 1HR, 1HE, 1HT, 1HU, 1HK, 1HN,
DATA 22, 1C, 1HS, 1HC, 1F, 1RU, 1HF, 1HT, 1HI, 1HF, 5,
DATA 5, 1HF, 1HN, 1HT, 1HI, 1HL, 1HJ, 17, 5, 1HW, 1HI, 1HL, 1HE, 6, 510VJ,
DATA TRES/1, 1B, 1F, 21, 27, 32, 41, 43, 53, 55, 56, 74, 73, 87, 96, 102, 108,
116, 124, 136, 148,
DATA EINT, CRDR, FFIL/6, 5, 4,
DATA E00ND, ENCASS/1HE, 1HN, 1HC, 1HE, 1FN, 1HF, 1HC, 1HA, 1HS, 1HE,
DATA EEXIT, ENIF/1HE, 1HX, 1HI, 1HT, 1HE, 1FA, 1HC, 1HI, 1HF,
DATA CCYCLE, CASS/1FC, 1HY, 1FC, 1HL, 1HE, 1FC, 1FA, 1HS, 1HE,
DATA UNTL, IIF/1HL, 1HN, 1HT, 1HF, 1HI, 1HL, 1FC, 1HF,
DATA CCLN, CCLCN/1F, 1FC, 1FO, 1HL, 1FC, 1HF,
DATA ITERAT/1HT, 1HE, 1FC, 1FA, 1HT, 1HF, 1HI, 1HE,
DATA BLANK, CEE, SEP/, CCIL/1H, 1HC, 1HF, 1FC,
DATA DBSHL, ELZ/1FL, 1FE, 1HL, 1FS, 1HE,
DATA NUMBER/1HU, 1HI, 1E2, 1H3, 1H4, 1HS, 1HF, 1H7, 1H8, 1H9,
DATA FCL, CEE, ACE, PARN, COMMA/1HF, 1HD, 1FA, 1HT, 1HI,
DATA FMAT, CTA/1HF, 1FC, 1ER, 1FM, 1FA, 1HT, 1HL, 1HA, 1HT, 1HA,
SUBROUTINE DETAX(TYPE)

// THE SUBROUTINE DETERMINES THE TYPE OF CYCLE STATEMENT AND
// SETS TCBEQ AND TCEND ON TOKEN BOUNDARIES.
//
IMPLICIT INTEGER(A-Z)
LOGICAL LBFLOG,ECOND
COMMON/ALPHA/CEE,CEE,CEE,CEE,CEE,CEE,CEE,PARA,COMMON,BDESCH
COMMON/DEVICE/DETF,FFILE,CSLP
COMMON/EFTINE/FNL/FNV(100),FPTR,FMAX,FSTRT,INC
COMMON/INPUT/SPIN(ED),TCBEQ,TCEND,STEND,STEND,STEND,STEND,IMAX,BLANK
COMMON/EDU/NUMENH(IU),LFLG,LABJ
COMMON/STYP2/EEN,FINLC,NCYCLE,WHEL,CCNT,CMPT,FCTRAN,SEMI
LEFLG=.FALSE.

LCCP

LOOP

TCEND= TCEND+1
IF TCEND.GT.IMAX THEN
EXIT
ENDIF
IF SPIN(TCEND).NE.BLANK THEN
IF SPIN(TCEND).EQ.SEMI THEN CYCLE
ENDIF
IF SPIN(1).EQ.CEE THEN
TCBEQ=1
TCEND=1
TYPE=CMPT
RETURN
ENDIF
TCBEQ=TCEND
CALL SCAN1(BLANK,ECOND)
CASE
: SPIN(6).NE.BLANK :
: TYPE=CMPT
: RETURN
: ECOND :
: TYPE=FCTRAN
: RETURN
: TCEND.GT.6 :
CALL ST1YP(TYPE)
RETURN
: TCEND.LE.6 :
FCR I=1 TO 5 CC
FBUF(I)=SPIN(I)
REPEAT
LBFLG=.TRUE.
FBUF(6)=BLANK
ENDCASE
ENDIF
REPEAT
REAC(CRDR,20) SPIN
FCSMAT(PJAI)
IF SPIN(1).EQ.FEE THEN
REAC(CRDR,20)SPIN
WHILE SPIN(1).NE.FEE CC
WRITE(PRTR,33)(SPIN(I),I=1,80)
FCSMAT(2UX,PJAI)
WRITE(FFILE,32)(SPIN(I),I=1,80)
FCSMAT(PJAI)
REAL(CRDR,20) SPIN
REPEAT
REAL(CRDR,20)SPIN
ENDIF
TCEND=)
REPEAT
END
SUBROUTINE STYPY(TYPE)

// THE TOKEN AT SPIN(TCBEG) TO SPIN(TCEND) IS SEARCHED
// FOR IN THE SYMECL TABLE.

IMPLICIT INTEGER(A-Z)

COMMON/ALPH/CCL,CEE,EE,EEE,FEE,FEN,FLH,FLM,FLN
COMMON/INPUT/CCL(2),CSM(4),CTE(2),DCL(4),DLC
COMMON/INPLT/SPELEN,TCBEG,TCEND,TSBEG,TSEND,TSIMAX,TSBLANK
COMMON/STYP1/CCLCL,CELSL,CELSL,UNTSL,LLCOP,LEN,END,ENDCASS,REPT
COMMON/STYP2/CCL,CRCLJ,NCYCLE,WHILE,CONT,CONT1,FCTRN,SEMI
COMMON/TABLES/TRES(24),RESRD(200)

IF SPIN(TCBEG).EQ.CCL THEN
   TCEND=TCBEG
   TYPE=CCL
   RETURN
ENDIF

IF SPIN(TCBEG).EQ.CCQLH AND SPIN(TCBEG+1).EQ.DCLH THEN
   TYPE=CCLSL
   RETURN
ENDIF

TCP=1

BCTTCM=21
LENGTH=TCEND-TCBEG+1

WHILE TCP.LE.BCTTCM DO
   MID=(TCP+BCTTCM)/2
   I=TRES(MID)

   CASE
       : SPIN(TCBEG).LT.RESRD(I+1):
          BCTTCM=MID-1
       : SPIN(TCBEG).GT.RESRD(I+1):
          TCP=MID+1
       : SPIN(TCBEG).EQ.RESRC(I+1):
          PNT=2
          WHILE FAT.LE.RESRC(I).AND.PNT.LE.LENGTH DO
             ITEMP=I+PNT
             JTEMP=TCBEG-1+PNT
          CASE
              : SPIN(JTEMP).LT.RESRC(ITEMP):
                 BCTTCY=BCTTCM-1

       
   END CASE

   END WHILE

   END CASE

   RETURN
4 216 893
4 216 894
4 217 895
4 217 896
4 217 897
4 216 898
4 218 899

ENDCASE

4 215 900
3 220 901
3 221 902
4 222 903
4 223 904
4 223 905
4 223 906
4 223 907
4 223 908
4 223 909
4 223 910
4 223 911
4 225 912
3 226 913
3 226 914
2 227 915
1 228 916
0 229 917
0 225 918

END
SUBROUTINE SCAN(PAT, LEN, ECCND)
C 231 519 // THE SUBROUTINE COMPARES THE PATTERN PAT WITH ALL TOKENS //
C 231 920 // SURROUNDED BY BLANKS FROM TCEND+1 ONWARDS AND //
C 231 921 // RETURNS WITH TCEND AT THE OLD POSITION OF TCEND+1, WITH THE //
C 231 922 // PATTERN STARTING AT TCEND+1. IF NOT FOUND ECCND IS SET TRUE //
C 231 923 IMPLICIT INTEGER(A-Z)
C 231 924 LOGICAL ECCND
O 231 525 COMMON/INPUT/SPIN(80), TCBEG, TCEND, STBEG, STEND, IMAX, BLANK
G 231 926 DIMENSION PAT(6)
C 231 527 TCBEG=TCEND+1
C 231 928 LCCP
1 232 929 K=TCEND+1
1 232 930 WHILE K.LE.IMAX.AND.SPIN(K).EQ.BLANK CC
2 233 931 K=K+1
2 233 932 REPEAT
1 234 933 TCEND=K
1 234 934 CALL SCAN1(BLANK, ECCND)
1 234 935 IF ECCND THEN
2 235 936 RETURN
2 235 937 ENDIF
1 236 938 IF LEN.NE.TCEND-K+1 THEN
2 237 939 CYCLE
2 237 939 ENDIF
1 238 940 // COMPARE THE PATTERN WITH A TOKEN OF THE SAME LENGTH. //
1 238 941 FOR I=1 TO LEN CC
2 239 942 J=K+I-1
2 239 943 IF PAT(J).NE.SPIN(I) THEN
3 240 944 EXIT
3 240 944 ENDIF
3 241 945 IF J.EQ.LEN THEN
3 242 946 // PATTERN FOUND, RETURN WITH ECCND FALSE. //
3 242 947 TCEAL=TCEAL-LEN
3 242 948 RETURN
3 242 948 ENDIF
2 243 952 REPEAT
1 244 953 REPEAT
0 245 954 END
SUBROUTINE SCAN1(CHAR,ECOND)
C 247 556 // LOOKS FOR CHAR AND IF FOUND, SPIN(TEND+1)=CHAR
G 247 557 // AND ECOND=FALSE. IF NOT FOUND UPTO IMAX OR SEMICOLON
C 247 558 // ECOND=TRUE.
C 247 559 IMPLICIT INTEGER(A-Z)
J 247 560 INTEGER ECOND
C 247 561 COMMON/ALPH/CEE,CEE,ACE,CEE,PARN,COMMA,DOSTH
C 247 562 COMMON/INPLT/SPIN(TEND),TCEG,TCENC,STREY,STEND,IMAX,BLANK
J 247 563 COMMON/STYP2/EOJ,INCIC,NCYCLE,WHILE,CCNT,CUMT,FCTR,SEMI
C 247 564 ECOND = .FALSE.
G 247 565 PTR = TEND
C 247 566 WHILE TCENC.LT.IMAX .OR
1 248 567 ITEMP = SPIN(TCENC+1)
1 248 568 CASE
2 249 569 : ITEMP .EQ. SEMI :
2 250 570  ECOND = .TRUE.
2 250 571  RETURN
2 250 572 : ITEMP .EQ. DOSTH :
2 251 573  IF CHAR.EQ.DOSTH THEN
3 252 574    IF SPIN(TEND+2).EQ.DOSTH THEN
3 253 575      RETURN!
4 253 576 ENDIF
3 254 577 ENDIF
2 255 578 : ITEMP .EQ. CHAR :
2 256 579  RETURN
2 256 580 ENDCASE
1 257 581 TCEND = TCENC + 1
1 257 582 IF ITEMP.NE.BLANK THEN
2 258 583 PTR = TCEND
2 258 584 ENDF
1 255 585 REPEAT
0 260 586 ECOND = .TRUE.
C 260 587 TCEND = PTR
0 260 588 RETURN
0 265 589 END
SUBROUTINE SCAN (ECOND)

IMPLICIT INTEGER (A-Z)
LOGICAL ECOND
CCYCN/ALPH/CCE, CLE, ALE, FEE, PRR, CCMMA, DBSLH
CCNEN/INPUT/SPIN(JO), TABEG, TECN, STEC, TEND, IMAX, BLANK
A=TOEND
TEN=TCEND+1
ECOND=.FALSE.

WHILE TOEND.LT.IMAX DO
  TCK=SPIN(TCEND)
  IF TCK NE BLANK THEN
    IF TCK EQ DBSLH THEN
      TCEND=TCEND+1
      WHILE TCEND.LT.IMAX DO
        IF SPIN(TCEND) EQ DBSLH .AND. SPIN(TOEND+1) EQ DBSLH THEN
          TCEND=TCEND+1
          RETURN
        ENDF
        TCEND=TCEND+1
      ENDF
      CALL ERFCF(2, JBSLH, 2, 1, 0, 0)
      RETURN
      ELSE
        TCEND=A
      ENDIF
    ENDIF
  ELSE
    TCEND=TCEND+1
  ENDIF
END
SUBROUTINE FHYAN

// Completes FORTRAN statements and outputs the FORTRAN.
//
IMPLICIT INTEGER(A-Z)

LOGICAL LBFLG, ECCND, ECCND1
COMMON/ALPH/CEE, CEE, AEE, FEE, PAK, COMMA, DBSLH
COMMON/FRC/FPTR, ECCND, BLKLN, VMACR, IMAX, BLANK
COMMON/FRC/FSPH(18), FPTR, FMAX, FSTRT, INCR
COMMON/FRT/FDA(4), FMAT(6), FTYE, LAST, NLAST
COMMON/INPUT/SPIN(80), TBEG, TCEND, STBEG, STEND, IMAX, BLANK
COMMON/COD/NSCHEER(12), LBFLG, LABO
COMMON/SYP2/ECJ, FINE, NCYC, WFLY, CCMT, FOTR, SEMI
COMMON/SC/TYPE
IF .NOT. LTLG, LBFLG THEN
  CALL LAFFIL(LABO)
ENDIF
IF TYPE .NE. RETURN THEN
  IF TYPE .EQ. CNT THEN
    IF FTYEP .EQ. 1 THEN
      TCEND = IMAX
    ENDIF
    FBUP(E) = SPIN(E)
    ECCND = .TRUE.
  ELSE
    CALL FTYEP
  ENDIF
ENDIF
CASE
  : FTYEP.EQ.1 :
    TEMP = FSTRT
    FSTRT = ?
    FPTK = ?
    CALL FOUT(SPIN, TBEG, TCEND)
    CALL FPRNT
    FSTRU = TEMP
    FPTR = FSTRT
    : FTYEP.EQ.2 :
    EPNDE = TCEND
    FCR J = 1 TO RUNCCM CC
TCEND = \text{VAPLCC(J)} - 1
CALL FCUT(SPIN, TOBEG, TCEND)
CALL FCUT(SPIN, PTR, EXPEND)
CALL FPRT
TOBEG = TCEND + 2
REPEAT
CALL FCUT(SPIN, TCEEG, EXPEND)
CALL FPRT
TCLAD = EXPEND
: FTYPE.EQ.J : 
CALL FCUT(SPIN, TOBEG, TCEND)
CALL FPRT
ENDCASE
IF .ALT.ECCNE THEN
CALL SCANZ(ECCNE)
ENDIF
RETURN
END
SUBROUTINE FTYPE

IMPLICIT INTEGER(A-Z)
LOGICAL ECIND
COMMON/ALPH/CEE,CEE,ACE,CEE,PARK,CUMA,DBSHL
COMMON/FORT/PTK,ECIND,KNUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,CUM,
ENDIF
CALL Scanl(0BSLH,ECCAD)
WHILE PTR.LT.TCEND CC
  PTR=PTR+1
  TCK=SPIN(PTR)
CASE
  TCK.=.EQ..FARH :
    PAR=1
    LOOP
      PTR=PTR+1
      TCK=SPIN(PTR)
    CASE
      TCK.=.EQ..FARH :
        PAR=PAR+1
        TCK.EQ.RSTAR(1) :
        PAR=PAR-1
        PTR.EQ.TCEND :
        EXIT
      ENCASE
    UNTIL PAR.EQ.0 REPEAT
      TCK.=.EQ..CLMA :
      NUMCOM=NUMCOM+1
      VAR=LC((NUMCOM)=PTR
      TCK.=.EQ..FCL(1) :
      IF NUMCOM.GT.0 THEN
        TYPE=2
      ENDF
      RETURN
ENCASE
END
SUBROUTINE FOUT(PAT, STRT, END)

// THE VALUE OF FSTRT WILL USUALLY BE 7 AND FMAX WILL BE 72. //
// FPTR SHOULD BE AT FSTRT. BLANKL IS A FLAG TO INDICATE A //
// TO INDICATE A SC FOR BLANK LINE. //
IMPLICIT INTEGER(4-Z)
LOGICAL BLANKL, EFLAGS
COMMON/ALPH/CEE, CLE, AEC, FEE, PARA, CMMA, GBSLH
COMMON/FCRT/PTR, ECCC, NUMCOM, BLANKL, VARLOC(10)
COMMON/FRTINE/FBLF(J), FPTR, FMAX, FSTRT, INCR
COMMON/INPLT/SPINECC, TOBEG, TOSEND, TOEND, IMAX, BLANK
COMMON/STYP2/CLJ, HILC, ARCYLE, HFLC, CONC, CONT, FPTR4, SEMI
COMMON/YYP/ TYPE
DIMENSION PAT(1)
BLANKL = .FALSE.
FOR J = STRT TO END DO
  IF FPTR.GT.FMAX THEN
    CALL FPFRAT
  CALL FPFRAT
  FOR I = 1 TO 5 DO
    FBLF(I) = BLANK
  REPEAT
  FBLF(E) = AEE
  JJ = J
  WHILE PAT(JJ).EQ.BLANK DO
    IF JJ.EQ.END THEN
      BLANKL = .TRUE.
      RETURN
    ENDF.
    JJ = JJ + 1
  REPEAT
    ENDIF.
    JJ = JJ + 1
  REPEAT
  EBLF(FPTR) = PAT(J)
  FPTR = FPTR + 1
  IF FBLF(FPTR).EQ.BLANK THEN
    RETURN
  ENDIF.
  IF FBLF(FPTR).EQ.BLANK THEN
    RETURN
  ENDIF.
END
SUBCUTLINE FPNT

IMPLICIT INTEGER(A-Z)

LOGICAL BLNLK

COMMON/DEVIC/PTR, FFILE, CRDR

COMMON/FORT/PTR, ECND, NUMCOM, BLNLK, VARCC(10)

COMMON/FOINT/BUF(80), FPR, FMAX, FSTRT, INCR

COMMON/FOINT/TA(4), FMAT(6), FTYPE, LAST, ALAST

COMMON/INPLT/SPIN(EC), TOBE, TDEND, TDRE, STRE, TEND, IMAX, BLNK

COMMON/SPLNF/SSTRT, STMNO, BKLNO, NESTNC, SMAX, BLNKS(131)

IF FPR.EQ. FSTRT THEN

RETURN

ENCIF

IF FPR = FSTRT - 1

IF FPR .GE. 7 THEN

FCR I = 7 TO FPR CC

FBUF(I) = BLNK

REPEAT

ENCIF

IF FPR .LE. FMAX THEN

FCR I = FPR TO FMAX CC

FBUF(I) = BLNK

REPEAT

ENCIF

FCR I = 73 TO 75 CC

FBUF(I) = SPIN(1)

REPEAT

IF .NOT. BLNLK THEN

WRITE(FFILE, 10)(FBUF(I), I = 1, 75), STMNO

FCMAT(75A1, 15)

ENCIF

IF FPR = FSTRT

ALAST = LAST

LAST = 0

RETURN

END
SUBROUTINE SPNT

IMPLICIT INTEGER(A-Z)
LOGICAL LVFLG,CVFL
COMMON/ALPH/LCEE,CEE,CEE,CEE,CEE,CEE,CEE,CEE,PARN,COMMA,DASH
COMMON/DEVIC/PPTR,FPFIL,CHR
COMMON/INPUT/SPIN,TOBEG,TOEND,STREG,STEND,IMAX,BLANK
COMMON/COO/ALMAPE(10),LBFLG,LABO
COMMON/SPRINF/SS1ST,STMNC,ALKNC,RESTMNC,SMAX,BLNKS(131)
COMMON/TYP2/EOJ,AIANC,NCYCLE,WHLC,CONT,CUMT,FOTRN,SEMI
FOR I=1 TO 6 DO
    BLNKS(I)=SPIN(I)
REPEAT
LCLP
C
LEN1=STEND-STBEG+1
LEN2=SMAX-SSRT+1
SEND=STENC
IF LEN1.GT.LEN2 THEN
    CVFL=.TRUE.
    SEND=STBEG+LEN2-1
END IF
FIL=LEN2-LEN1+7
WRITE(PPTR,10) NESTNO,ALKNC,STMNC,(BLNKS(I),I=1,SSRT),
1 1=STBEG,SEND),(BLNKS(I),I=7,FIL),(SPIN(I),I=73
,FO)
1 1
FACTU XXX XXX XXX XXX
2 1
FIL=1 TO 6 CC
REPEAT
BLNKS(I)=BLANK
STREG=SEND+1
UNTIL .NOT.CVFL REPEAT
RETURN
END
SERCUTINE EXCYL(TYPE)

IMPLICIT INTEGER(A-Z)
LOGICAL ECCNC, LFLG
COMMON/CONT/STACK(200),LABEL(200),STFRM(6),TOP,LMAX,
1 LFLG,TPTR,NVM,SMAX
COMMON/PTK/ANTA(4),PRT(6),FYTE,LAST,NLAST
COMMON/INDT/INFNLG,LFLG,NECR,AEWML,PRML,NEW,OLD,PULDT
COMMON/INPUT/SPIN(EU),TCNLG,TCNLG,STBEQ,STEND,IMAX,BLANK
COMMON/KYBR2/EXXT(4),CCYCLL(5),UNGRL(5),ITRAT(9),REPT(6),ONF(2)
COMMON/LCD/NUMBER(10),LBFLG,LABG
COMMON/CTPM/CGTIC(9),CENT1(3),ECLL(1),GOTO(6),IFNOJ(9),
1 LEC(9),MINLS(1),PLL(1),RSTAR(2),CNE(1)
COMMON/STYP3/IIF,CASE,SGBN,FFOR,EXIT,EENDIF,RETN
LAST=1
IF .NOT.LBFLG THEN
CALL LABFIL(LBC)
ENDIF
K=0
CALL SCANS(ECCNC)
C=0
1 371 1271
TEMP=IPTR
C 371 1273
IF .ACT.ECCNC.ANC.K.EQ.TCEND THEN
TCEND=K
1 372 1274
WHILE SPIN(TCEND+1).EQ.BLANK DC
2 373 1276
TCEND=TCEND+1
2 372 1277
REPEAT
1 374 1278
PTR=0
2 374 1279
FCF I=1 TO 9 DC
2 375 1280
IF SPIN(TCEND+1).EQ.NUMBER(I+1) THEN
3 376 1281
PTR=I
2 376 1282
ENDIF
3 377 1283
REPEAT
1 378 1284
IF PTR.GT.1 THEN
2 378 1285
TCEND=TCEND+1
2 379 1286
WHILE TEMP.N.E.C DC
3 380 1287
TEMP=STACK(TEMP-4)
3 380 1288
PTR=PTR-1
3 381 1289
IF PTR.LE.1 THEN
EXIT
4 382 1292
3 382 1293
2 382 1294
1 382 1295
4 381 1291
3 384 1294
2 385 1296
1 386 1298
3 387 1299
2 388 1300
1 390 1302
C 390 1302
1 391 1303
1 392 1304
1 392 1305
1 392 1306
C 392 1308
C 393 1307
C 393 1306
C 393 1309
C 393 1310
C 393 1311
C 393 1312
C 393 1313
C 393 1314
C 393 1315
C 393 1315
END
SUBCUTINE LABEL(40,6)

C 395  1317       IPLICIT INTEGER(A-Z)
C 395  1318       COMMON/CENTR/STACK(300),LABEL(200),STKM(6),TOP,LIBMAX,
C 395  1319                   LNPTR,PLPTR,NUMB,STMAX
C 395  1320       COMMON/FRTINF/FBLF(80),FTPTR,FMAX,FSTRT,ICMP
C 395  1321       COMMON/INPUT/SPIN(80),TOBEG,TOEND,STBLG,STEND,IMAX,BLANK
C 395  1322       COMMON/TYP/TYP
C 395  1323       COMMON/STYP(80),RTN,NCYCLE,WHILE,CCNT,CCMT,FCTNR,SEMI
C 395  1324       FOR I=1 TO 5 DO
1 396  1325       L=LAB+1-1
1 396  1326       FBLF(I)=LABEL(L)
1 396  1327       REPEAT
1 397  1328       IF TYPE.NE.CCNT THEN
1 398  1329       FBLF(5)=BLANK
1 398  1330       ENDIF
1 398  1331       RETURN
C 399  1332       END
INTEGER FUNCTION GENLAB(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. NLMR IS INITIALIZED TO 10000). THIS IS AN
// INTEGER FUNCTION RETURNING A POINTER TO THE LABEL.
//
IMPLICIT INTRINSIC(A-Z)
LOGICAL LBFLG
COMMON/GENRFL/STACK(200),LABEL(200),STFMR(6),TDP,LBMAX,
1
LBPR,TPTR,NLMR,STMX
COMMON/INPLT/SPIN(6),TCBEG,TCEND,STBEG,STEND,IMAX,BLANK
COMMON/CPD/NUMBER(10),LBFLG,LAB0

IF .NOT.LBFLG.GR.GEN.EQ.1 THEN
NUMB=NUMB-1
1 NUMB=NUMB
FOR I=1 TO 5 CC
M=N/10
// SHIFT RIGHT
2 N=N-(10*M)
// REMAINDER
3 HLDD=H-1+LBPR
4 LABEL(FCD)=CNV(NR)
5 // LBPT IS LEFT AT FIRST BLANK SPACE. //
6 N=M
REPEAT
1 GENLAB=LBPR
1 NEW LABEL PTR. //
2 LBPR=LBPR+5
// POINT TO NEW BLANK LOCATION. //
1 LBPT=LBPT+5
IF LBPT.GT.LBMAX THEN
2 CALL ERROR(9,0,0,5,0,0)
// NESTING TOO DEEP, TOO MANY LABELS PROGRAM TERMINATED. //
3 ENDIF
1 C
1 ELSE
2 FOR I=1 TO 5 CC
3 HCD=HCD+LBPR+I-1
4 LABEL(FCD)=SPIN(I)
5 REPEAT
2
GENLAB=LOPTR
LOPTR=LOPTR+5
IF LOPTR GT LEMAX THEN
   CALL ERFCR(0,0,0,5,0,0)
ENDIF
END
SUBROUTINE CHECK(STYPE, ECCND)
IMPLICIT INTEGER(0-2)
LOGICAL ECCND, DUMM, UTFLG
COMMON/CENTRL/STACK(200), LABEL(200), STFRM(6), TOP, LBNMAX,
            LUTF, UTF, RFCN, STMAX
COMMON/FRMTNF/FLBLF(80), FTPR, EMAX, FSTRT, INCR
COMMON/INPUT/THCN(4), BYET, DCJ(2), ELZ(4), COLN
COMMON/INPUT/SHINE(J), TCbeg, TCENC, STBEG, STEND, IMAX, BLANK
COMMON/KEYNL/LEND(3), ENCAS(7), ENIF(5), CASS(4), IF1F(2), CCGLEN(5)
COMMON/KEYH2/ELSEXT(4), CECYCLE(5), JUNITL(5), ITERAT(9), REPT(6), QTR(2)
COMMON/SPE/INF/STRT, STANC, BLKNC, AESTNC, SMAX, BLKNS(131)
COMMON/STYPL/CCLEN, BLSL, ELSE, UNTL, LLOOP, SEND, ENDCAS, REPT
COMMON/STYPR/ECJ, RHLC, ACYCDE, WHILE, CCNT, COMT, FCTRL, SEMI
COMMON/STYPS/IIF, CASE, SUFNC, FFCTR, EEXIT, ENDIF, RTEN
ECCND=.FALSE.
IF TCP.EQ.C THEN
   CALL ERR(4, C, 3, 0, 3)
   // ERROR---STACK EMPTY, STATEMENT IGNORED. //
   ECCND=.TRUE.
   RETURN
ENDIF
STOP=STACK(TOP)
CASE:
   STYPE.EQ.CCLCN:
      IF STTOP.NE.CASE THEN
         CALL ENDFIX(CCASE, STTOP, STYPE, DUMM)
         CALL ERRR(2, CASE, 4, 4, CCLCN, 1)
         // ERROR---MISSING CASE STATEMENT, COLON IGNORED. //
         TCENC=TCENC+1
         ECCND=.TRUE.
      ENDIF
      STYPE.EQ.ELSE:
      IF STTOP.NE.IIF THEN
         CALL ERR(2, IIF, 2, 4, ELZ, 4)
         // ERROR---MISSING IF STATEMENT, ELSE IGNORED. //
         ECCND=.TRUE.
      ENDIF
      STYPE.EQ.UNTL:
}
IF STTOP .AE .LLCQP THEN
  CALL ERRCR (5 , IITERAT , 9 , 4 , ULLATL , 5 )
  // ERROR---MISSING LOOP STATEMENT, UNTIL IGNORED. //
  CALL SCAN ( LEFT , 3 , CUMM )
  IF DUMM THEN
    LEND = LEND + 1
  ELSE
    LEND = LEND + 6
  ENDFI:
  LEND = .TFUE.
ENDIF:
:STYFE .EC .EEND : IF STTOP .AE .SUBFN THEN
  // UNSTACK UNTIL A SUBFN IS FOUND ON THE STACK. //
  CALL ENDFIX ( SUEFN , STTOP , STYPE , CUMM )
  IF .ACT .DUMM THEN
    CALL ERRCR ( 3 , LEND , 3 , 4 , ENEND , 3 )
    // ERROR---TOO MANY ENDS, END IGNORED. //
    ECCNZ = .TFLE.
  ELSE
    CALL ERRCR ( 7 , 0 , 0 , 6 , 0 , 0 )
    // PREMATURE END SUBPROGRAM CLOSED. //
  ENDFI:
FSTRT = 10
SSRT = 10
ENDIF:
:STYFE .EC .ENCAS : IF STTOP .AE .CASE THEN
  CALL ENDFIX ( CCASE , STTOP , STYPE , DUMM )
  CALL ERRCR ( 3 , ENCASE , 7 , 4 , ENCASE , 7 )
  // ERROR---TOO MANY ENDCASES, ENDCASE IGNORED. //
  ECCNZ = .TFLE.
ENDIF:
:STYFE .EC .ENDIF : IF STTOP .AE .IF THEN
  CALL ERRCR ( 3 , ENIFF , 5 , 4 , ENIFF , 5 )
  // ERROR---TOO MANY ENDFS, ENDF IGNORED. //
  ECCNZ = .TFLE.
ENDIF
1 441 1457
1 442 1458
2 443 1459
2 444 1460
2 444 1461
2 444 1462
2 444 1463
2 444 1464
1 445 1465
1 446 1466
2 447 1467
2 447 1468
2 447 1469
2 447 1470
1 448 1471
1 449 1472
2 450 1473
2 450 1474
2 450 1475
2 450 1476
1 451 1477
C 452 1478
u 452 1479
END
IF SSCTCP.EQ.FFC1 THEN
  LAB1=STACK(TCP-5)
ELSE
  LAB1=STACK(TCP-3)
ENDIF
LAB2=STACK(TCP-2)
LAB3=STACK(TCP-1)
CALL LABFIL(LAB2)
CALL FCLT(ECCC,1,9)
CALL FCLT(LABEL,LAB1,LAB1+4)
CALL FFNT
CALL LABFIL(LAB3)
CALL FCLT(CONT1,1,8)
CALL FFNT
ENDCASE
TCP=TOP-STM(STMCP)
IF TCP.EQ.U THEN
  RETURN
ENDIF
STTCP=STACK(TCP)
REPEAT
  CUMM=.TRUE.
  RETURN
END
SUBROUTINE ERRGF(CODE1, PARM1, LEN1, CODE2, PARM2, LEN2)
C CODE1 IS CRERR 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(0-Z)
C CCMCN/DEVC/RTR, FFILE, CRDR
C CCMCN/INPLT/SPIN(EU), TCGED, TCGED, STBEG, TENG, IMAX, FLANK
C DIMENSION PARM1(1), PARM2(1), CRPAT(45,5), PSN(9), LEN(9),
C 1 ACPAT(35,7), ALEN(7), APSN(7)
C DATA CRPAT/4HL, 1FA, 1FB, 1HE, 1HL, 1HR, 1HT, 1HM, 1HE, 1H, 1HR, 1HT, 1HM, 1HE, 1H, 1HR, 1HT, 1HM, 1HE, 1H,
C 1 FP, 1FR, 1HC, 1HF, 1HA, 1HM, 1HE, 1H, 1HC, 1HF, 1HA, 1HM, 1HE, 1H, 1HC, 1HF, 1HA, 1HM, 1HE, 1H,
C 1 1FP, 1FR, 1HC, 1HF, 1HA, 1HM, 1HE, 1H, 1HC, 1HF, 1HA, 1HM, 1HE, 1H,
C 1 1FP, 1FR, 1HC, 1HF, 1HA, 1HM, 1HE, 1H,
C 1 1FP, 1FR, 1HC, 1HF, 1HA, 1HM, 1HE, 1H,
C 1 1FP, 1FR, 1HC, 1HF, 1HA, 1HM, 1HE, 1H,
C 1 1FP, 1FR, 1HC, 1HF, 1HA, 1HM, 1HE, 1H,
C 1 1FP, 1FR, 1HC, 1HF, 1HA, 1HM, 1HE, 1H,
C 1 1FP, 1FR, 1HC, 1HF, 1HA, 1HM, 1HE, 1H,
C 1 1FP, 1FR, 1HC, 1HF, 1HA, 1HM, 1HE, 1H,
C 1 1FP, 1FR, 1HC, 1HF, 1HA, 1HM, 1HE, 1H,
C 1 1FP, 1FR, 1HC, 1HF, 1HA, 1HM, 1HE, 1H,
C 1 1FP, 1FR, 1HC, 1HF, 1HA, 1HM, 1HE, 1H,
C 1 1FP, 1FR, 1HC, 1HF, 1HA, 1HM, 1HE, 1H,
C 1 1FP, 1FR, 1HC, 1HF, 1HA, 1HM, 1HE, 1H,
C 1 1FP, 1FR, 1HC, 1HF, 1HA, 1HM, 1HE, 1H,
C 1 1FP, 1FR, 1HC, 1HF, 1HA, 1HM, 1HE, 1H,
C 1 1FP, 1FR, 1HC, 1HF, 1HA, 1HM, 1HE, 1H,
C 1 1FP, 1FR, 1HC, 1HF, 1HA, 1HM, 1HE, 1H,
C 1 1FP, 1FR, 1HC, 1HF, 1HA, 1HM, 1HE, 1H,
C 1 1FP, 1FR, 1HC, 1HF, 1HA, 1HM, 1HE, 1H,
C 1 1FP, 1FR, 1HC, 1HF, 1HA, 1HM, 1HE, 1H,
C 1 1FP, 1FR, 1HC, 1HF, 1HA, 1HM, 1HE, 1H,
C 1 1FP, 1FR, 1HC, 1HF, 1HA, 1HM, 1HE, 1H,
C 1 1FP, 1FR, 1HC, 1HF, 1HA, 1HM, 1HE, 1H,
C 1 1FP, 1FR, 1HC, 1HF, 1HA, 1HM, 1HE, 1H,
C 1 1FP, 1FR, 1HC, 1HF, 1HA, 1HM, 1HE, 1H,
C 1 1FP, 1FR, 1HC, 1HF, 1HA, 1HM, 1HE, 1H,
C 1 1FP, 1FR, 1HC, 1HF, 1HA, 1HM, 1HE, 1H,
C 1 1FP, 1FR, 1HC, 1HF, 1HA, 1HM, 1HE, 1H,
C 1 1FP, 1FR, 1HC, 1HF, 1HA, 1HM, 1HE, 1H,
C 1 1FP, 1FR, 1HC, 1HF, 1HA, 1HM, 1HE, 1H,
C 1 1FP, 1FR, 1HC, 1HF, 1HA, 1HM, 1HE, 1H,
C 1 1FP, 1FR, 1HC, 1HF, 1HA, 1HM, 1HE, 1H,
C 471 1580 3 1FS,F1T,F1A,F1T,F1E,F1V,F1E,F1N,F1T,F1F,
0 471 1581 3 1FS,F1E,F1H,F1F,F1P,F1E,F1D,F1H,F1I,F1I,F1H,F1I,F1H,F1I,
C 471 1582 3 F1F,F1H,F1H,F1H,F1H,F1H,F1H,F1H,F1H,F1H,F1H,F1H,F1H,F1H,F1H,
C 471 1583 4 F1H,F1H,F1H,F1H,F1H,F1H,F1H,F1H,F1H,F1H,F1H,F1H,F1H,F1H,F1H,
C 471 1584 4 F1H,F1H,F1H,F1H,F1H,F1H,F1H,F1H,F1H,F1H,F1H,F1H,F1H,F1H,F1H,
C 471 1585 4 F1H,F1H,F1H,F1H,F1H,F1H,F1H,F1H,F1H,F1H,F1H,F1H,F1H,F1H,F1H,
C 471 1586 4 F1H,F1H,F1H,F1H,F1H,F1H,F1H,F1H,F1H,F1H,F1H,F1H,F1H,F1H,F1H,
C 471 1587 5 F1H,F1H,F1H,F1H,F1H,F1H,F1H,F1H,F1H,F1H,F1H,F1H,F1H,F1H,F1H,
C 471 1588 5 F1H,F1H,F1H,F1H,F1H,F1H,F1H,F1H,F1H,F1H,F1H,F1H,F1H,F1H,F1H,
C 471 1589 6 F1H,F1H,F1H,F1H,F1H,F1H,F1H,F1H,F1H,F1H,F1H,F1H,F1H,F1H,F1H,
C 471 1590 6 F1H,F1H,F1H,F1H,F1H,F1H,F1H,F1H,F1H,F1H,F1H,F1H,F1H,F1H,F1H,
C 471 1591 7 F1H,F1H,F1H,F1H,F1H,F1H,F1H,F1H,F1H,F1H,F1H,F1H,F1H,F1H,F1H,
C 471 1592 7 F1H,F1H,F1H,F1H,F1H,F1H,F1H,F1H,F1H,F1H,F1H,F1H,F1H,F1H,F1H,
C 471 1593 7 F1H,F1H,F1H,F1H,F1H,F1H,F1H,F1H,F1H,F1H,F1H,F1H,F1H,F1H,F1H,
C 471 1594 7 F1H,F1H,F1H,F1H,F1H,F1H,F1H,F1H,F1H,F1H,F1H,F1H,F1H,F1H,F1H,
C 471 1595 7 F1H,F1H,F1H,F1H,F1H,F1H,F1H,F1H,F1H,F1H,F1H,F1H,F1H,F1H,F1H,
C 471 1596 7 F1H,F1H,F1H,F1H,F1H,F1H,F1H,F1H,F1H,F1H,F1H,F1H,F1H,F1H,F1H,
C 471 1597 2222 FORMAT(26H ++OFFENDING CARD: 8040)
C 471 1598 2222 LENGTH=LEN(CODE1)
C 471 1599 2222 ALNTH=#LEN(CODE2)
C 471 1600 2222 POS=POSN(CODE1)
C 471 1601 2222 APCS=APSN(CODE2)
C 471 1602 2222 FCR I=1 TO LEN1 CC
C 471 1603 2222 IT=POS+I-1
C 471 1604 2222 IF POS>LEN1 THEN
C 471 1605 2222 FF=POS+LEN1
C 471 1606 2222 PE=POS+LENGTH-1
C 471 1607 2222 FCR 1=PE TO PE CC
C 471 1608 2222 KPAT(1,CCDE1)=BLANK
C 471 1609 2222 REPEAT
C 471 1610 2222 ENDIF
C 471 1611 2222 FCR I=1 TO LEN2 CC
C 471 1612 2222 IT=APCS+I-1
C 471 1613 2222 KPAT(1,CCDE1)=PARM2(I)
C 471 1614 2222 REPEAT
C 471 1615 2222 IF LENGTH>LEN2 THEN
C 471 1616 2222 PC=APCS+LEN2
1 480 161d          PE=APCS+ALNTH-1
1 4dd 1619          FOR I=PC TO PE DO
2 481 1620         ACPLAT(I,CCCE2)=BLANK
2 481 1621         REPEAT
1 482 1622     ENDIF
C 483 1623      WRITE (PRTR,10) (ERPAT(I,CCCE1),I=1,45)
C 483 1624      WRITE (PRTR,13) (ACPLAT(I,CCCE2),I=1,35)
C 483 1625    10      FORMAT (12H       ++++++++ , 0I1)
C 483 1626         RETURN
C 483 1627        END
INTEGER FUNCTION CONV(DIGIT)

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

IMPLICIT INTEGER(A-Z)

COMMON/COO/ALMEE(10), LBFLG, LAB0

CONV=NUMBER(DIGIT+1)

RETURN

END

END OF SPARKS PROGRAM.....
APPENDIX D

REFERENCES
REFERENCES


ENHANCEMENT OF SPARKS,
A FORTRAN
PREPROCESSOR

by

JOHN JOSEPH MARTIN JR.
B.S., University of Scranton, 1973

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

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
FORTHAN has the distinction of being one of the earliest high-level programming languages. Because of its early development, it lacks the structured features of newer languages. SPARKS is a preprocessor which adds many of the new features of good structured programming to the present FORTHAN compilers.

This report describes an enhancement of the SPARKS preprocessor and provides documentation for additions and modifications to the original program. Enhancement of SPARKS is divided into three levels of effort: The first level consists of changes which have no effect on the actual code produced, but simply adds to its readability. The second level is the removal of errors which produce improper code or limitations in the code. The third level adds new features, some of which are totally new while others are expansions of features already in SPARKS. This report includes a User's Manual, SPARKS code, and specifications to support all modifications made to the original program.