1026

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:

Myron A. Calhoun

Major Professor

Document
LD
2668
.R4
1978
M37
C.2

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
3.4 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 SYMMARY 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

ILLEGIBLE DOCUMENT

THE FOLLOWING DOCUMENT(S) IS OF POOR LEGIBILITY IN THE ORIGINAL

THIS IS THE BEST COPY AVAILABLE

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

FIGURE		PAGE
1.1	SPARKS DATA FLOW GRAPH	3
4.1	MULTIPLE ASSIGNMENT STATEMENT GRAPH	18
4.2	SYNTAX GRAPH FOR CASE STATEMENT	19
4.3	SYNTAX GRAPH FOR CASE OF STATEMENT	20
4.4	SYNTAX GRAPH FOR EXIT AND CYCLE STATEMENTS	22

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, Califorina. Since

that time the implemented language has been evolved to the extent that it no longer matches the bock versicn exactly.

The SPARKS preprccessor 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 chtained frcm 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 fcr use by the students and faculty at Kansas State University. The FORTRAN versicn 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 bcotstrap.

1.3 WHAT IS SPARKS

In simple terms the SPARKS preprccessor 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 tc prccess the statements. Figure 1.1 ccntains a flow graph of the SPARKS translator.

The SPARKS translator will recognize defined SPARKS statements and thcse indicated as pure FORTRAN statements. All others are assumed to be FORTRAN statements and passed tc the FORTRAN output file. For each SPARKS statement encountered, the translator will produce FORTRAN statements

which acccmplish the equivalent computation. This generated FCRTRAN code is then passed to the FORTRAN output file. All input statements are passed to the SPARKS output file for printing.

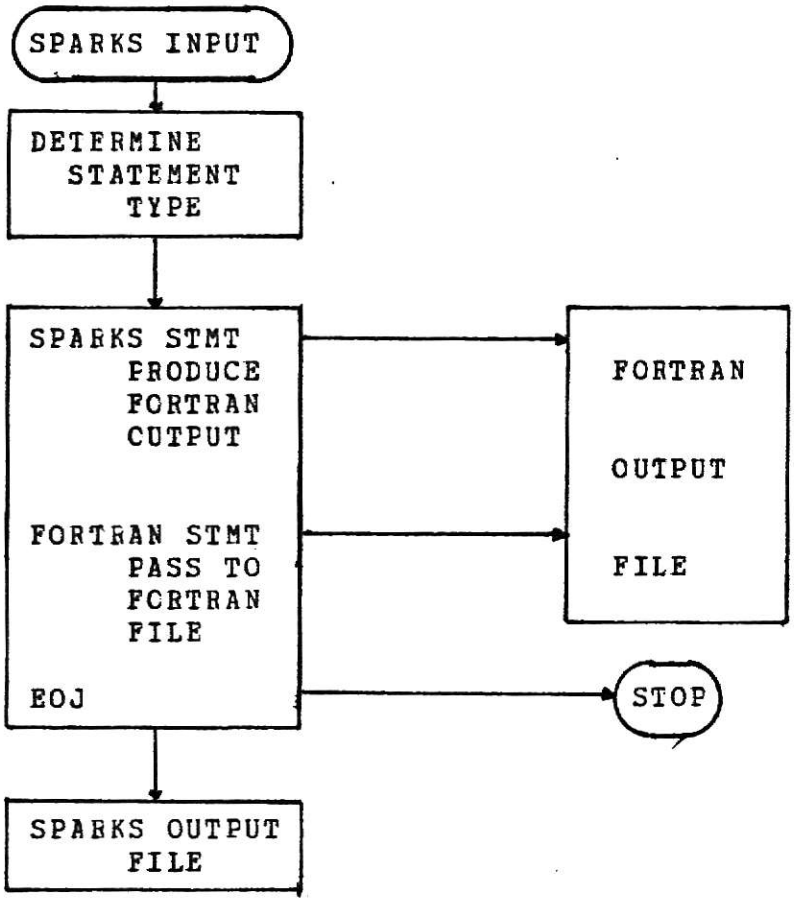


FIGURE 1.1 SPARKS DATA FLOW GRAPH

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 cutput. 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 it 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 evolutional 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 begining 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 criginaly written as follows:

// THIS IS A COMMENT //

This comment would appear in both the SPARKS and FORTRAN cutput 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 scme unknown reason, the trailing doubleslash was omitted from the FORTRAN versicn. Since the SPARKS comment was already recognizable by the doubleslash, the 'C' in column 1 served no purpcse so it was deleted frcm 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 cne 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	FORTRAN
A=B+C;	A=B+C
C=D+E;	C=D+E
X=A	X=A

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
   ENDIF
   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
   ENDIF
   X=X+1
REPEAT
```

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. Additionall 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 Softerware 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.

SPARKS	FORTRAN
L	= X + Y
= X + Y	

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 STTYP 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 additicnal 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 crder tc prevent this overflow, the following mcdifications were made. Each time a CASE or IF-THEN-ELSE statement is encountered the current label pointer is placed cn 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 SEMICCLCN DETECTICN

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 FCRTRAN does not use the semicolon this should present no prcblem. However, the semicclon may and often does appear in FORTRAN DATA and FCRMAT statements. The SCAN1 routine detected this semicolon and produced two separate statements. This separation of the code without a continuaticn card then results in a FORTRAN error.

In order to alleviate this condition SCAN1 was changed tc make one additicnal test. Since all SPARKS keywords and symbols are required to be surrcunded by blanks, only thcse

semicolons would be used as delimiters. A further improvement in this area will be covered when DATA and FORMAT statements are discused 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
   ENDIF
REPEAT
```

FORTRAN

```
99999 CONTINUE
         S1
         IF(.NOT.(CCND1) GO TO 99987
            RETURN
            GO TO 99985 <-------------------------- (1)
99987    CONTINUE
            IF(.NOT.(CCND2)) GO TO 99983
              S2
              GO TO 99989
              GO TO 99984 <-------------------------- (2)
99983       IF(.NOT.(CCND3)) GO TO 99982
                S3
                GO TO 99988
                GO TO 99984 <-------------------------- (3)
99982        CONTINUE
99984        CONTINUE
99985    CONTINUE
99988 CONTINUE
      GO TO 99999
99989 CONTINUE
```

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 FORMAT 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 in 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
```

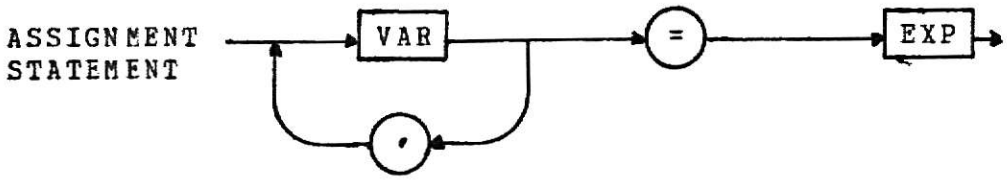
Changes to the translator to accept comments which did not have the doubleslash surrounded by blanks required the removal of the doubleslash from the keyword table and making a separate test for them. This test is still conducted in subroutine STTYP, but is now a test by itself. Allowing comments in-line with code was much more complex, since comments could appear after both SPARKS and FORTRAN statement. 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 continue 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 FORTRAN 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



SEMANTICS

VAR:any standard FORTRAN variable
EXP: any legal FORTRAN 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

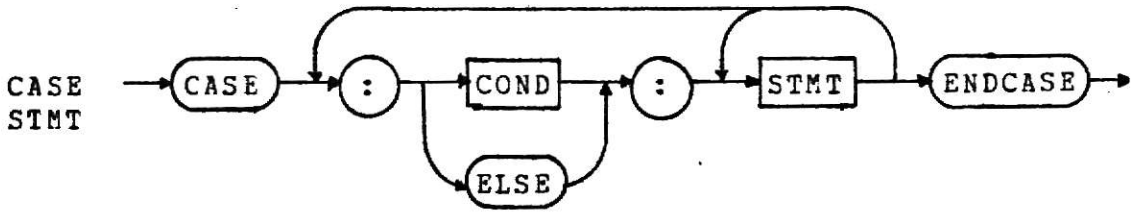
SPARKS input and FORTRAN output.

SPARKS	FORTRAN
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
START,END=X+Y/10	START=X+Y/10 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 FTYP is used to recognize the statements and identify the individual parts. With some modifications subroutine FHAND 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.



SEMANTICS

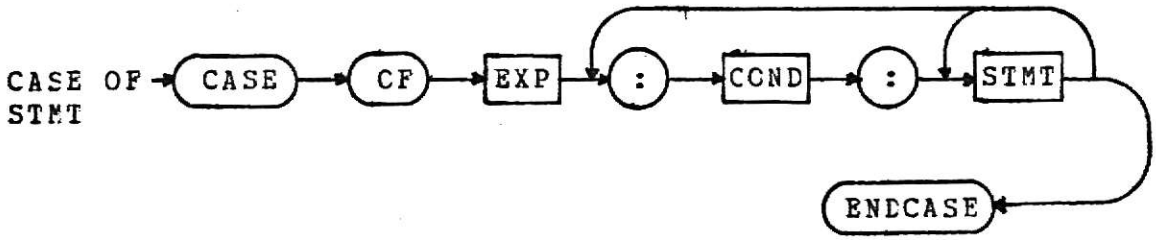
COND: any FORTRAN Boolean expression
STMT: any FORTRAN/SPARKS statement

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.



SEMANTICS

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

FIGURE 4.3 SYNTAX GRAPH FOR CASE OF STATEMENT

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

EXAMPLE 1

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

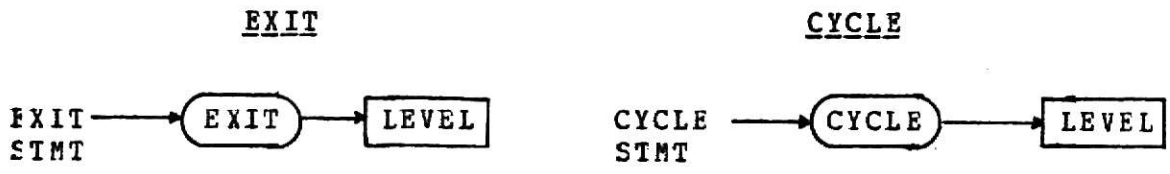
EXAMPLE 2

```
CASE OF 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 desireable 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.



SEMANTICS

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

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.

```
LOOP
   FOR I=1 TO N BY 2 DO
      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 transfered out of the FOR LOOP and the LOOP FOREVER. In the case of the CYCLE, control will be transfered to the bottom of the FOR LOOP, 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. STROUD

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 SYMBOLS A-4
2.1 KEYWORDS A-5
2.2 SPECIAL SYMBOLS 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 STATEMEN . . . 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 LOOP A-11
5.1.1 EXAMPLE OF A FOR LOOP A-12
5.2 WHILE LOOP A-12
5.2.1 EXAMPLE OF A WHILE LOOP 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 SEPARATORS 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

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 FORTRAN. 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 FORTRAN compiler ultimately defines the semantics of all SPARKS statements.

2. KEYWORD AND SYMBOLS

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 can not 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=2
ENDIF
```

In order to avoid confusion and possible error, in practice the user may find it desireable 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	ENDIF	EQJ	EXIT	FOR
FUNCTION	IF	INTEGER	LOGICAL	LOOP	OF
REAL	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 ENDIF.

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.

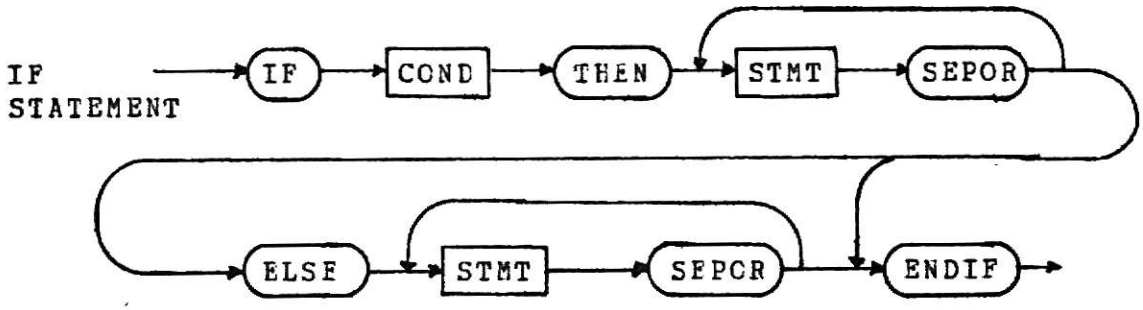
; SEMICOLON - is used as a line separator.

// 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 FORTRAN 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



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 ENDIF 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 P=X;Q=Y
       ELSE P=Y;Q=Z;ENDIF
```

SPARKS

```
IF A.EQ.B THEN
   P=X
   Q=Y
ELSE
   P=Y
   Q=Z
ENDIF
```

FORTRAN

```
      IF(.NOT.(A.EQ.B)) GO TO 99999
         P=X
         Q=Y
         GO TO 99998
99999 CONTINUE
         P=Y
         Q=Z
99998 CONTINUE
```

EXAMPLE 2

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

SPARKS OUTPUT

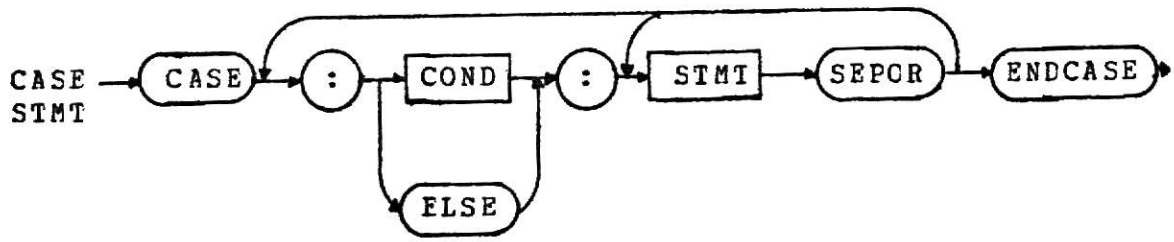
```
IF PTR.EQ.N THEN
   TEST=0 //ITEM NOT IN LIST//
ELSE
   TEST=1 //ITEM IN LIST//
ENDIF
```

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



SEMANTICS

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

* * NOTE * *

1. Comments may be inserted after the CASE or COLON, 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
```

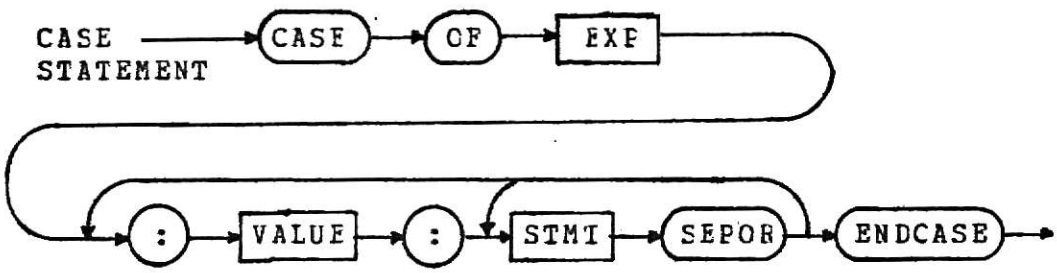
FORTRAN

```
      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



SEMANTICS

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
```

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
99997 IF(.NOT.(LENGTH.EQ.100)) GO TO 99996
         SIZE=LARGE
99996 CONTINUE
99999 CONTINUE
```

This form of the CASE statement allows the user a great degree of flexibility.

5. LOOP STATEMENT

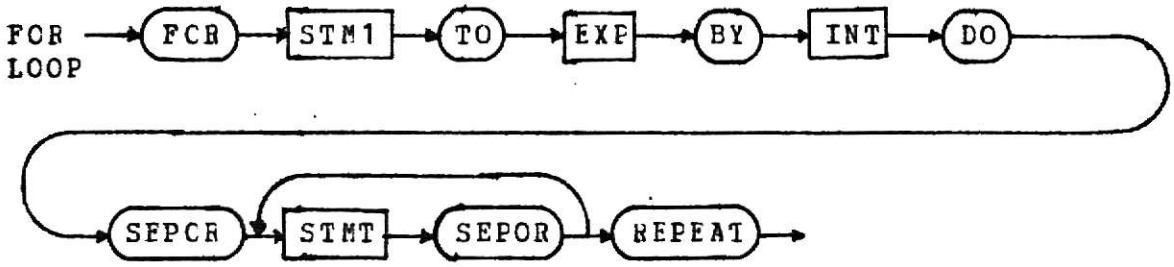
Loops are very powerful programming tools, but are also probably the most common sources of error in FORTRAN programs. SPARKS allows four types of LOOP statements, some of which are very safe while others require caution in their use.

* * 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:
```
FOR I=X+Y TO 50 BY 2 DO
A=C/D;B=E*F;REPEAT
```

SPARKS

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

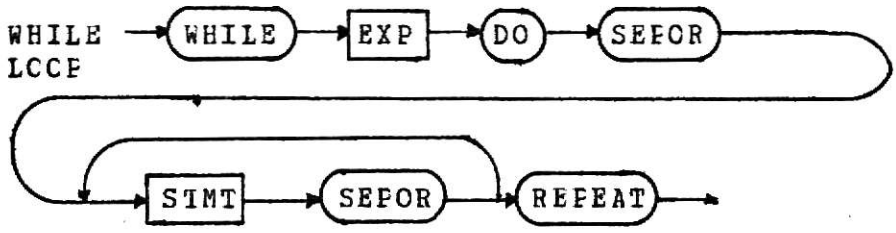
FORTRAN

```
      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.



SEMANTICS

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

5.2.1 EXAMPLE OF WHILE LOOP

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

SPARKS

```
WHILE TEMP.LE.10 DO
  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
```

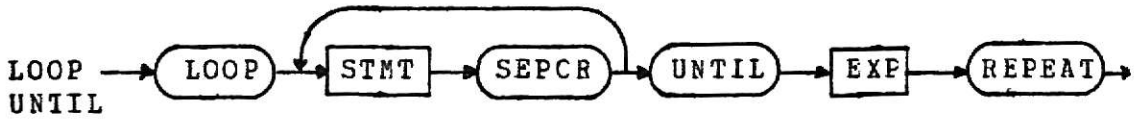
CAUTION

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 LOOP 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:



SEMANTICS

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

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

5.3.1 EXAMPLE OF LOOP UNTIL

```
INPUT: LOOP X=A+5
       IF X.GT.100 THEN DONE=.TRUE.;ENDIF
       UNTIL DONE REPEAT
```

SPARKS

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

FORTRAN

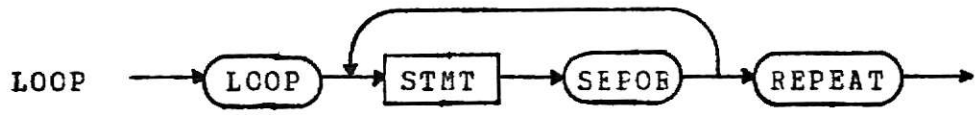
```
99999 CONTINUE
         X=A+5
         IF(.NOT.(X.GT.100)) GO TO 99998
            DONE=.TRUE.
99998    CONTINUE
99997 IF(.NOT.(DONE)) GO TO 99999
99996 CONTINUE
```

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 LOOP

The last of the four loops allowed by SPARKS is the LOOP. 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.



SEMANTICS

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

5.4.1 EXAMPLE OF LOOP

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

SPARKS

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

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

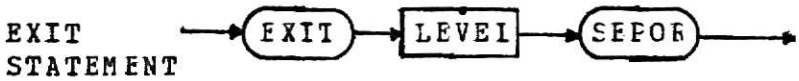
CAUTION
The LOOP does not increment any pointers or counters and it will not terminate unless the user does so explicity.

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 SPARKS 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 SPARKS looping statements: FOR, WHILE, LOOP UNTIL, and LOOP. The following syntax graph defines the statement structure.



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

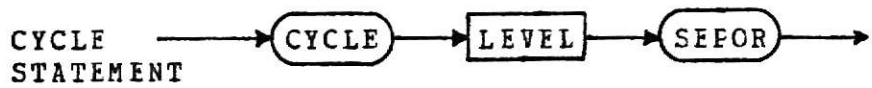
SPARKS OUTPUT

```
LOOP
   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, LOOP UNTIL, and LOOP. The following syntax graph is given for the CYCLE statement.



SEMANTICS

LEVEL: integer indicating the number of levels to be cycled.
SEPOR: 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

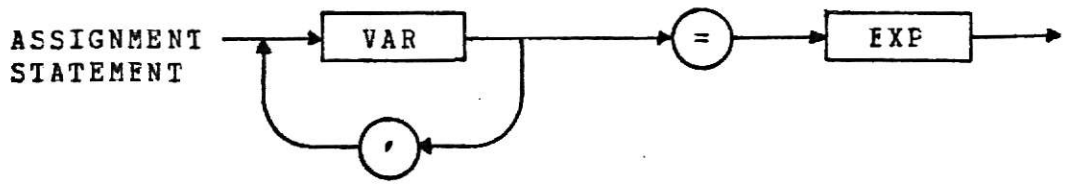
SPARKS OUTPUT

```
LOOP
   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 LOOP.

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.



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:

SPARKS	FORTRAN
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,COND=.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 doubleslash. 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.

```
//A SPARKS COMMENT CAN COVER ANY NUMBER OF LINES//
//EACH LINE OF THE COMMENT MUST END WITH A      //
//DOUBLESLASH                                   //
//    COMMENTS MAY ALSO APPEAR AFTER STATEMENTS //

CALL TEST(TOKEN,COND) //TEST FOR ALPHA CHAR//
IF COND THEN
   FBUF(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.

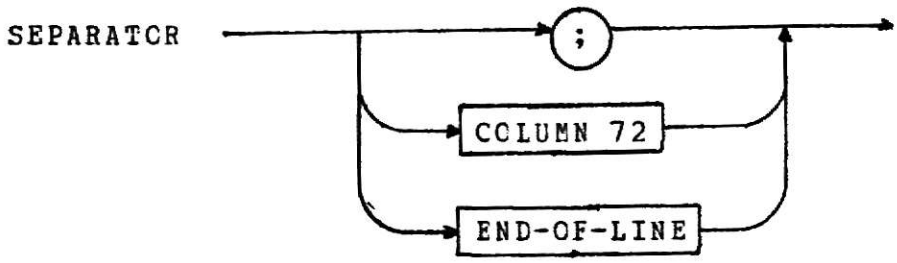
9.1 LABELS

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.



* * 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 SPARKS 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 mammer.

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: ENDIF
++++++++TOO MANY ENDIF
++++++++ ENDIF IGNORED
```

In this example the ENDIF 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 SPARKS PROGRAMS

This section contains two SPARKS programs to give the user some idea as to what SPARKS output looks like. The first program is one which searches for the integer X in the sorted array of integers A(N), N<=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 BESTFT(N,P)
   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 POINTER 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 POINT TO FREE BLOCK//
   ENDIF
   RETURN
END
```

APPENDIX B

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.

VARIABLE	USING PROCEDURE
AEE	BLOCK DATA, FTYP, FOUT
BLANK	MAIN, BLOCK DATA, DETNX, SCAN, SPRNT, PHAND, FOUT
BLNKLN	FPRNT, FOUT
BLNKS(131)	BLOCK DATA, SPRNT
BKWRD	MAIN, BLOCK DATA
BLKNC	MAIN, SPRNT
BYE(2)	MAIN, BLOCK DATA
CASS(4)	MAIN, BLOCK DATA, CHECK
CCASE	MAIN, BLOCK DATA, ENDFIX
CEE	BLOCK DATA, DETNX, SPRNT, FOUT
CCYCLE(5)	MAIN, BLOCK DATA, CHECK
CCOLCN(5)	MAIN, BLOCK DATA, CHECK
CGOTO(9)	MAIN, BLOCK DATA
CCLN	MAIN, BLOCK DATA
COLCN	MAIN, BLOCK DATA, STTYP, CHECK
CCMT	MAIN, BLOCK DATA, DETNX, FOUT, SPRNT
COMMA	BLOCK DATA, FTYP
CCNT	MAIN, BLOCK DATA, DETNX
CCNTI(8)	MAIN, BLOCK DATA, ENDFIX

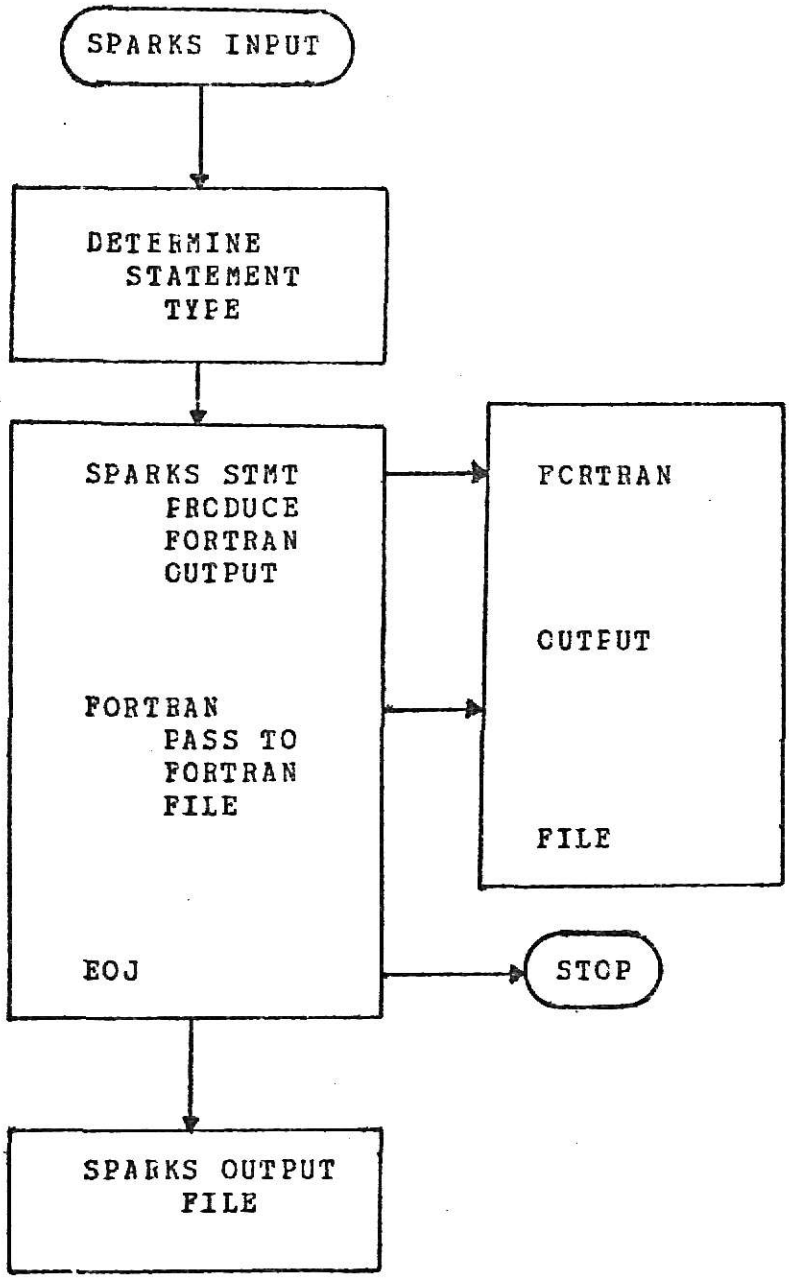
CBDR	BLOCK DATA, DETNX
CURFLG	MAIN, BLOCK DATA, EXCYL
DEE	BLOCK DATA, FTYP
DELSL	MAIN, BLOCK DATA, STTYP
DESLH	MAIN, BLOCK DATA, STTYP, SCAN1, SCAN2, FHAND
DOO	MAIN, BLOCK DATA
ECCND	MAIN, DETNX, SCAN, SCAN1, SCAN2, FHAND, FTYP, EXCYL, CHECK
EEEND(3)	BLOCK DATA, CHECK, ENDFIX
EEEXT(4)	BLOCK DATA, CHECK
EELSE	MAIN, BLOCK DATA, CHECK
EEND	MAIN, BLOCK DATA, CHECK
EENDIF	MAIN, BLOCK DATA, CHECK
EEXIT	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
EQL(1)	MAIN, BLOCK DATA, FTYP
FBUF(80)	MAIN, DETNX, FHAND, FPRNT, FOUT, LABFIL
FEE	BLOCK DATA, DETNX, FTYP
FFILE	BLOCK DATA, DETNX, FPRNT
FFOR	MAIN, BLOCK DATA, CHECK, ENDFIX
FMAX	MAIN, BLOCK DATA, FOUT, FPRNT
FOTRN	MAIN, BLOCK DATA, DETNX, STTYP
FPTR	MAIN, FPRNT, FOUT, FHAND
FRWARD	MAIN, BLOCK DATA

FSTRT	MAIN, FPRNT, 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
IFNCT(9)	MAIN, BLOCK DATA
ITERAT(9)	BLOCK DATA, CHECK
LABEL(200)	MAIN, BLOCK DATA, GENLAB
LAST	MAIN, EXCYL, FPRNT
LBFLG	MAIN, DETNX, SPRNT, FHAND, EXCYL
LBMAX	MAIN, GENLAB
LBPTR	MAIN, GENLAB
LEO(5)	MAIN, BLOCK DATA,ENDFIX
LOOP(4)	MAIN, BLOCK DATA, CHECK
NCYCLE	MAIN, BLOCK DATA, CHECK
NESTNO	MAIN, SPRNT
NEUTR	MAIN, BLOCK DATA, EXCYL
NUMB	MAIN, BLOCK DATA, GENLAB
NUMCOM	FTYP, FHAND
MINUS(1)	MAIN, BLOCK DATA
OLD	MAIN, BLOCK DATA, EXCYL
ONE(1)	MAIN, BLOCK DATA
OOP	MAIN, BLOCK DATA
PARN	BLOCK DATA, FTYP
PLU(1)	MAIN, BLOCK DATA
PULOUT	MAIN, BLOCK DATA
RESRD(200)	BLOCK DATA, STTYP

BEPT(6)	MAIN, BLOCK DATA, CHECK
BINLO	MAIN, BLOCK DATA
BREPT	MAIN, BLOCK DATA, CHECK
BSTAR(2)	MAIN, BLOCK DATA, FTYP
SEMI	BLOCK DATA, DETNX, SCAN1
SMAX	MAIN, SPRNT
SPIN(80)	MAIN, BLOCK DATA, DETNX, STTYP, SCAN, SCAN1, SCAN2, FPRNT, SPRNT, FHAND, FOUT, FTYP, EXCYL
SSTRT	MAIN, SPRNT, CHECK
STACK(200)	MAIN, CHECK, ENDFIX, EXCYL
STBEG	MAIN, SPRNT
STEND	MAIN, SPRNT
STFRM(6)	BLOCK DATA, CHECK, ENDFIX
STMAX	MAIN, BLOCK DATA
STMNC	MAIN, SPRNT, FPRNT
SUBFN	MAIN, BLOCK DATA, CHECK, ENDFIX
THENN(4)	MAIN, BLOCK DATA
TOBEG	MAIN, DETNX, STTYP, SCAN, FHAND, FTYP
TOEND	MAIN, DETNX, STTYP, SCAN, SCAN1, SCAN2, FHAND, FTYP, CHECK
TO(2)	MAIN, BLOCK DATA
TOP	MAIN, CHECK, ENDFIX
TPITR	MAIN, CHECK, ENDFIX
TRES(23)	BLOCK DATA, STTYP
TYPE	MAIN, DETNX, STTYP, FHAND, FTYP, SPRNT, EXCYL
UNTL	MAIN, BLOCK DATA, CHECK
UUNTL(5)	MAIN, BLOCK DATA, CHECK
WHLE	MAIN, BLOCK DATA, CHECK
VARLOC(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)
ERRCR:	Produce an error message as required.
EXCYL:	Process EXIT and CYCLE statements. (EXit CYcLe)
FHAND:	Process FORTRAN statements. (Fortran HANDel)
FOUT:	Builds a statement in the FORTRAN output buffer. (Fortran OUT)
FPRNT:	Writes FORTRAN statements from buffer to output file. (Fortran PRiNT)
FTYP:	Determine the type of FORTRAN statement. (Fortran TYPe)
GENLAB:	Generate a FORTRAN label. (GENerate LABel)
LABFIL:	Copies a label into the FORTRAN 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.
SPRNT:	Writes SPARKS statements to the output file. (Sparks PRiNT)
STTYP:	Determine SPARKS statement type. (STatement TYPe)



SPARKS DATA FLOW GRAPH

	CHECK	CONV	DETNX	ENDFIX	ERRCR	EXCYL	FHAND	FOUT	FPRNT	FTYP	GENLAB	LABFIL	MAIN	SCAN	SCAN1	SCAN2	SPRNT	STTYP
ALPH			X				X	X		X			X		X	X	X	X
CENTRL	X			X		X					X	X	X					
DEVIC			X		X				X				X				X	
FCRT							X	X	X	X								
FRTINF	X		X	X			X	X	X			X	X					
FTRT						X	X		X	X			X					
INDT						X							X					
INPAT	X												X					X
INPUT	X		X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
KEYW1	X			X									X					
KEYW2	X					X							X					
CDD		X	X	X		X	X				X		X				X	
CTPAT				X		X				X			X					
STRINF	X								X				X				X	
STYP1	X												X					X
STYP2	X		X				X	X				X	X		X		X	X
STYP3	X			X		X							X					
TABLES																		X
TYP							X	X				X	X				X	

CCMMCN BLOCK ACCESS MATRIX

CALLEE / CALLER	CHECK	CCNV	DETNX	ENDFIX	ERRCR	EXCYL	FHAND	FOUT	FPRNT	FTYP	GENLAB	LABFIL	SCAN	SCAN1	SCAN2	SPRNT	STTYP
CHECK				X	X								X				
CONV																	
DETNX														X			X
ENDFIX					X			X	X			X					
ERRCR																	
EXCYL					X			X	X			X			X		
FHAND								X	X	X		X		X	X		
FOUT									X								
FPRNT																	
FTYP														X			
GENLAB		X			X												
LABFIL																	
MAIN	X		X	X	X	X	X	X	X		X	X	X	X	X	X	
SCAN														X			
SCAN1																	
SCAN2					X												
SPRNT																	
STTYP																	

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.

NAME	VARIABLES
/ALPH/	AEE, CEE, COMMA, DEE, DBSLH, FEE, PARN
/CENTRL/	LABEL(200), LBMAX, LBPTR, NUMB, STACK(200), STFRM(6), STMAX, TOP, TPTR
/DEVIC/	CRDR, FFILE, PRTR
/FORT/	BLNKLN, ECCND, NUMCOM, PTR, VARLOC(10)
/FRTINF/	FBUF(80), FMAX, FPTR, FSTRT, INCR
/FTRT/	DTA(4), FMAT(6), FTYPE, LAST, NLAST
/INDI/	BKWRD, CURFLG, FRWD, INDFIG, CLD, PULCUT, NEUTR, NEW
/INPAT/	BYE(2), COLN, DOO(2), EIZ(4), THENN(4), TOO(2)
/INPUT/	BLANK, IMAX, SPIN(80), STBEG, STEND, TOBEG, TOEND
/KEYW1/	CASS(4), CCOLON(5), EEEND(3), ENCASS(7), ENIF(5), IIIF(2)
/KEYW2/	CCYCLE(5), EEEXT(4), ITERAT(9), OOF(2), REPT(6), UUNTL(5)
/ODD/	LABO, LBFLG, NUMBER(10)
/OTPAT/	CGCTO(9), CONTI(8), EQL(1), IFNOT(9), LEO(5), GOTO(6), MINUS(1), ONE(1), PLU(1), RSTAR(2)
/SPRINF/	BLNKS(131), BLKNO, NESTNO, SMAX, SSTRT, STMNO
/STYP1/	COLON, DBLSL, EELSE, EEND, ENDCAS, LLCOP, RREPT, UNTL
/STYP2/	COMT, CONT, EOJ, FORTN, NCYCLE, RINLO, SEMI, WHLE
/STYP3/	CCASE, EENDIF, EEXIT, FFOR, IIF, RETRN, SUBFN
/TABLES/	RESRD(200), TRES(23)
/TYP/	TYPE

SPECIFICATIONS
FOR
PROGRAM MAIN

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, CENTRL, DEVIC, FRTINF, FTRT, INDT, INPAT, INPUT, KEYW1, KEYW2, ODD, OTPAT, SRINF, STYP1, STYP2, STYP3, TABLES, TYP

HIGH-LEVEL CODE

```
PROC: MAIN
   INITIALIZE POINTERS AND FLAGS
   LOOP
      CALL DETNX     DETERMINE NEXT STATEMENT TYPE
      IF TYPE EQUAL REAL INTEGER OR LOGICAL THEN
         IF NEXT WORD EQUAL SUBROUTINE 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=SPARKS STATEMENT:
         PRODUCE FORTRAN CODE
         PASS CODE TO FORTRAN FILE
      :TYPE=EOF:
         CLOSE FILES
         STOP
       ENDCASE
       PASS INPUT STATEMENT TO SPARKS FILE
   REPEAT
END
```

SPECIFICATIONS
FOR
SUBROUTINE DETNX

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

SPECIFICATIONS
FOR
SUBROUTINE STTYP

FUNCTION

This routine is called by subroutine DETNX once a word surrounded by blanks has been found. STTYP 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: STTYP(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
```

SPECIFICATIONS
FOR
SUBROUTINE SCAN

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

```
PROC: SCAN(PATTERN,LENGTH,ENDCONDITION)
   INITIALIZE POINTERS
   WHILE PTR .NE. END-OF-LINE DO
      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
```

SPECIFICATION
FOR
SUBROUTINE SCAN1

FUNCTION

This routine searchs 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 doubleslash.

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 ECONDITION 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
```

SPECIFICATIONS
FOR
SUBROUTINE SCAN2

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 POINTER
   WHILE PTR .NE. END-OF-LINE DO
      IF NEXT TOKEN=BLANK THEN
         CYCLE
      ELSE
         IF NEXT TOKEN=SLASH THEN
            FIND END OF COMMENT
            SET ENDCONDITION TO FALSE
            RETURN
         ENDIF
      ENDIF
      INDIX POINTERS
   REPEAT
   COMMENT NOT FOUND
   SET ENDCONDITION TO TRUE
END
```

SPECIFICATIONS
FOR
SUBROUTINE FHAND

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, FIRT, INPUT, ODE, STYP2, TYP

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 FOUT  //DO NOT INDENT//
      CALL FPRNT
   :TYPE=MULTIPLE ASSIGNMENT:
      PRODUCE INDIVIDUAL STATEMENTS
      CALL FOUT
      CALL FPRNT
   :TYPE=FORTRAN:
      CALL FOUT
      CALL FPRNT
   ENDCASE
   CALL SCAN2  //TEST FOR COMMENT//
   RETURN
END
```

SPECIFICATIONS
FOR
SUBROUTINE FTYP

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, INPUT, CTRAT

HIGH-LEVEL CODE

```
PROC: FTYP(FORTRAN.TYPE)
   INITIALIZE POINTERS
   IF FIRST.CHARACTER=F THEN
      TEST FOR FORMAT STATEMENT
      IF TRUE THEN
         FORTRAN.TYPE=FORMAT STATEMENT
         RETURN
      ENDIF
   ENDIF
   IF FIRST.CHARACTER=D THEN
      TEST FOR DATA STATEMENT
      IF TRUE THEN
         FORTRAN.TYPE=DATA STATEMENT
         RETURN
      ENDIF
   ENDIF
   WHILE PTR .NE. END-OF-LINE DO
      IF MULTIPLE ASSIGNMENT STATEMENT THEN
         FORTRAN.TYPE=MULTIPLE-ASSIGNMENT-STATEMENT
         RETURN
      ENDIF
   REPEAT
   FORTRAN.TYPE=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, STYP2, TYP

HIGH-LEVEL CODE

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

SPECIFICATIONS
FOR
SUBROUTINE FPRNT

FUNCTION

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

COMMON BLOCKS

DEVIC, FORT, FRTINF, FTRT, INPUT, SPRINF

HIGH-LEVEL CODE

```
PROC: FPRNT
   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
```

SPECIFICATIONS
FOR
SUBROUTINE SPRNT

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, OUD, SPRINF, STYP2, TYP

HIGH-LEVEL CODE

```
PROC: SPRNT
   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
```

SPECIFICATIONS
FOR
SUBROUTINE EXCYL

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

CENTRL, FIRI, INDI, INPUT, ODD, KEYW2, CTPAT, STYP3

HIGH-LEVEL CODE

```
PROC: EXCYL(TYPE)
   SET FLAG=EXIT/CYCLE
   CALL LABFIL  // FILL LABEL BUFFER //
   CALL SCAN2   // TEST FOR COMMENT //
   IF NOT A COMMENT THEN
      WHILE NOT END-OF-LINE DO
         TEST FOR EXIT/CYCLE LEVEL
         DETERMINE LABEL TO WHICH TO BRANCH
      REPEAT
   ENDIF
   CALL FOUT  // PRODUCE CODE FOR EXIT/CYCLE //
   CALL FPRNT
   RESET POINTERS
   RETURN
END
```

SPECIFICATIONS
FOR
SUBROUTINE CHECK

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
```

```
   :STYPE=ENDIF:
      IF STACK TOP=IF THEN
         BAD=FALSE
      ELSE
         CALL ERROR
         BAD=TRUE
      ENDIF
   :STYPE=END:
      IF STACK TOP=PROGRAM THEN
         BAD=FALSE
      ELSE
         CALL ENDFIX
      ENDIF
   ENDCASE
   RETURN
END
```

SPECIFICATIONS
FOR
SUBROUTINE ENDFIX

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

CENTRL, FRTINF, INPUT, KEYW1, CDD, CTRAT, STYP3

HIGH-LEVEL CODE

```
PROC: ENDFIX(END.TYPE, STACK.TOP, STATEMENT.TYPE, DUMM)
   WHILE END.TYPE NOT EQUAL STACK.TOP DO
      CASE
      :STACK.TOP=IF:
         PRODUCE ENDIF STATEMENT
      :STACK.TOP=CASE:
         PRODUCE ENDCASE STATEMENT
      :STACK.TOP=SUBROUTINE:
         PRODUCE END STATEMENT
      :ELSE:
         PRODUCE REPEAT STATEMENT
      ENDCASE
      DECREMENT STACK
      IF STACK EMPTY THEN
         RETURN
      ENDIF
   REPEAT
   RETURN
END
```

SPECIFICATIONS
FOR
SUBROUTINE ERROR

FUNCTION

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

COMMON BLOCK

DEVIC, INPUT

HIGH-LEVEL CODE

```
PROC: ERROR(CODE)
   DETERMINE TYPE MESSAGE TO BE PRINTED
   SET POINTER TO STATEMENT T BE PRINTED
   WRITE MESSAGE
   RETURN
END
```

SPECIFICATION
FOR
SUBROUTINE LABFIL

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

CENTRL, FRTINF, INPUT, TYP

HIGH-LEVEL CODE

```
PROC: LABFIL(LABEL.POINTER)
   FOR I=1 TO 5 DO
     COPY LABEL TO BUFFER
   REPEAT
   COLUMN 6=BLANK
   RETURN
END
```

SPECIFICATIONS
FOR
FUNCTION GENLAB

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

CNTRL, 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 CONV

FUNCTION

This function converts a digit passed into its character representation.

COMMON BLOCK

CDD

ABSTRACT CODE

```
PROC: CONV(DIGIT)
   CONVERT DIGIT TO ALPHANUMERIC
   RETURN
END
```

APPENDIX C

SPARKS CODE

```
NEST  BLK  STNO                                  THIS IS A SPARKS PROGRAM

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

 C     0      0            // MAIN PROGRAM //
 C     0      1            IMPLICIT INTEGER(A-Z)
 C     0      2            LOGICAL ECONO,ECONO1,BAD,LBFLG
 C     0      3            COMMON/ALPH/CEE,DEE,AEE,FEE,PARN,COMMA,DBSLH
 C     0      4            COMMON/CENTRL/STACK(200),LABEL(200),STERM(6),TOP,LBMAX,
 C     0      5         1  LEFTR,TPITR,NUMB,STMAX
 C     0      6            COMMON/DEVIC/PRTR,FFILE,CRDR
 C     0      7            COMMON/FRTINF/FBUF(80),FPTR,FMAX,FSTRT,INCR
 C     0      8            COMMON/FTRT/CTA(4),FMAT(6),FTYPE,LAST,NLAST
 C     0      9            COMMON/INDT/INDFLG,CLRFLG,NEUTR,BKWRD,FRWRD,NEW,OLD,PULOUT
 C     0     10            COMMON/INPAT/TOO(2),THENN(4),BYE(2),DOO(2),ELZ(4),CCLN
 C     0     11            COMMON/INPUT/SPIN(80),TCBEG,TCEND,STBEG,STEND,IMAX,BLANK
 C     0     12            COMMON/KEYW1/EEEND(3),ENCASS(7),ENIF(5),CASS(4),IIIF(2),CCOLON(5)
 C     0     13            COMMON/KEYW2/EEEXT(4),COYCLE(5),UUNTL(5),ITERAT(9),REPT(6),DOF(2)
 C     0     14            COMMON/ODD/NUMBER(10),LBFLG,LABC
 C     0     15            COMMON/OTPAT/OGOTO(9),OCNTI(8),EQL(1),GOTO(6),IFNOT(9),
 C     0     16         1  LEO(5),MINUS(1),PLU(1),RSTAR(2),ONE(1)
 C     0     17            COMMON/SPRINF/SSTRT,STMNO,BLKNO,NESTNO,SMAX,BLNKS(131)
 C     0     18            COMMON/STYP1/OCLON,DBLSL,EELSE,UNTL,LLOOP,EEND,ENDCAS,PREPT
 C     0     19            COMMON/STYP2/ECJ,RINLO,NCYCLE,WHLE,OONT,OOMT,FOTRN,SEMI
 C     0     20            COMMON/STYP3/IIF,OCASE,SUBFN,FFOR,EEXIT,EENDIF,RETRN
 C     0     21            COMMON/TABLES/TRES(23),RESRD(200)
 C     0     22            COMMON/TYP/TYPE
 C     0     23            DIMENSION EXP(50)
 C     0     24            // WRITE TITLES ETC. //
 C     0     25            WRITE(PRTR,3)(SPIN(I),I=1,80)
 C     0     26       3    FORMAT(4X,4HNEST,2X,3HBLK,2X,4HSTNO,2X,30X,80A1)
 C     0     27            WRITE(PRTR,4)
 C     0     28       4    FORMAT(/32X,'SPARKS TRANSLATION BY KSU SPARKS PREPROCESSOR',
 C     0     29         1  ' AS'/32X,'MODIFIED BY J. MARTIN, R. STROUD, AND M. CALHOUN.'/)
 C     0     30            // SET UP OPTIONS //
 C     0     31            IMAX=72
 C     0     32            SMAX=100
```

```
C     0     33              FMAX=72
C     0     34              INCR=3
C     C     35              FSTRT=7+INCR
C     C     36              SSTRT=7+INCR
C     C     37              // EXTRACT OPTIONS GIVEN ON THE CONTROL CARD AND CHANGE ABOVE //
C     0     38              // DEFAULTS ACCORDINGLY---THIS COULD USE DETNX //
C     0     39              // INITIALISE THE PROGRAM RELATED VARIABLES. //
C     C     40              STACK(1)=SUBFN
C     0     41              TOP=1
C     C     42              TFITR=0
C     0     43              LBPTR=6
C     0     44              TOEND=IMAX+1
C     0     45              BLKNO=0
C     0     46              STMNO=C
C     0     47              NESTNO=0
C     0     48              // PROCESS STATEMENTS ACCORDING TO TYPE //
C     0     49              // 1. TRANSLATE AND PRINT FORTRAN. //
C     0     50              // 2. SET INDENTATION FLAG---INDFLG. //
C     0     51              // 3. SET CURFLG NEW-OLD--CURRENT STATEMENT TO BE PRINTED IN //
C     0     52              // OLD OR NEW INDENTATION. //
C     0     53              // START MAIN LOOP //
C     0     54              LOOP
1     1     55                 FPTR=FSTRT
1     1     56                 CALL DETNX(TYPE)
1     1     57                 IF TYPE.EQ.RIGHT THEN
2     2     58                    A=TOBEG
2     2     59                    B=TOEND
2     2     60                    CALL DETNX(TYPE)
2     2     61                    // IF THE SECOND TOKEN IS NOT FUNCTION THEN PASS AS FORTRAN. /
2     2     62                    IF TYPE.NE.SUBFN THEN
3     3     63                       TYPE=RETRN
3     3     64                    ENDIF
2     4     65                    TOBEG=A
2     4     66                    TOEND=B
2     4     67                 ENDIF
1     5     68                 // THE MAIN CASE STATEMENT //
1     5     69                 STBEG=TOBEG
1     5     70                 CASE
```

```
2     6     71                  : TYPE.EQ.FCTRN :
2     7     72                     CALL FHAND
2     7     73                     STEND=TCEND
2     7     74                     CURFLG=CLD
2     7     75                     INDFLG=NEUTR
2     7     76                  : TYPE.EQ.DBLSL :
2     8     77                     TCEND=TCEND+1
2     8     78                     // LOOK FOR A DOUBLE SLASH--SCAN IS NOT USED SINCE IT STOPS //
2     8     79                     // ON A SEMICOLON. //
2     8     80                     LOOP
3     9     81                        IF SPIN(TCEND).EQ.DBSLH.AND.SPIN(TCEND+1).EQ.DBSLH THEN
4    10     82                           EXIT
4    10     83                        ENDIF
3    11     84                        IF TCEND.GE.IMAX-1 THEN
4    12     85                           CALL ERROR(2,DBSLH,2,1,0,C)
4    12     86                           // ERROR--MISSING DOUBLE SLASH SUPPLIED AT END //
4    12     87                           EXIT
4    12     88                        ENDIF
3    13     89                        TCEND=TCEND+1
3    13     90                     REPEAT
2    14     91                     TCEND=TCEND+1
2    14     92                     STEND=TCEND
2    14     93                     INDFLG=NEUTR
2    14     94                     CURFLG=CLD
2    14     95                  : TYPE.EQ.IIF :
2    15     96                     CALL SCAN(THENN,4,ECOND)
2    15     97                     IF ECOND THEN
3    16     98                        // ABSENSE OF THEN ASSUMED TO INDICATE FORTRAN IF. //
3    16     99                        TOREG=TOREG-2
3    16    100                        CALL FHAND
3    16    101                     ELSE
3    17    102                        LAB=GENLAB(1)
3    17    103                        // OUTPUT: IF(.NOT.COND) GO TO LAB //
3    17    104                        IF .NOT.LBFLG THEN
4    18    105                           CALL LABFIL(LAB0)
4    18    106                        ENDIF
3    19    107                        CALL FOUT(IFNOT,1,9)
3    19    108                        CALL FOUT(SPIN,TOREG,TCEND)
```

```
3   19   109                CALL FOUT(CGOTO,1,9)
3   19   110                CALL FOUT(LABEL,LAB,LAB+4)
3   19   111                CALL FPRNT
3   19   112                STACK(TOP+1)=LAB
3   19   113                STACK(TOP+2)=LAB
3   19   114                STACK(TOP+3)=IIF
3   19   115                TOP=TOP+3
3   19   116                IF TOP.GT.STMAX THEN
4   20   117                   CALL ERROR(6,0,0,5,0,0)
4   20   118                ENDIF
3   21   119                //  STACK OVERFLOWED, PROGRAM TERMINATED. //
3   21   120                TCEND=TCEND+4
3   21   121                IF FSTRT+INCR.LT.IMAX THEN
4   22   122                   FSTRT=FSTRT+INCR
4   22   123                ENDIF
3   23   124             ENDIF
2   24   125             CALL SCAN2 (ECOND)
2   24   126             STEND=TCEND
2   24   127             CURFLG=OLD
2   24   128             INDFLG=FRWRD
2   24   129          : TYPE.EQ.EENDIF :
2   25   130             CALL CHECK(EENDIF,BAD)
2   25   131             IF BAD THEN
3   26   132                CYCLE ;
3   26   133             ENDIF
2   27   134             FSTRT=FSTRT-INCR
2   27   135             FPTR=FSTRT
2   27   136             IF LBFLG THEN
3   28   137                // LABELLED STATEMENT--GENERATE 'LAB CONTINUE' //
3   28   138                CALL FOUT(CONTI,1,8)
3   28   139                CALL FPRNT
3   28   140             ENDIF
2   29   141             LAB=STACK(TOP-1)
2   29   142             LBPTR=STACK(TOP-2)
2   29   143             TOP=TOP-3
2   29   144             CALL LABFIL(LAB)
2   29   145             CALL FOUT(CONTI,1,8)
2   29   146             CALL FPRNT
```

```
2   29   147                    //   GENERATE A LAB CONTINUE.   //
2   29   148                    CALL SCAN2 (ECOND)
2   29   149                    STEND=TCEND
2   29   150                    INDFLG=BKWRD
2   29   151                    CURFLG=NEW
2   29   152                  : TYPE.EQ.ELSE :
2   30   153                    CALL CHECK(ELSE,BAD)
2   30   154                    IF BAD THEN
3   31   155                       CYCLE ;
3   31   156                    ENDIF
2   32   157                    FSTRT=FSTRT-INCR
2   32   158                    FPTR=FSTRT
2   32   159                    IF LBFLG THEN
3   33   160                       CALL ERROR(1,ELZ,4,2,0,0)
3   33   161                       // ERROR--LABELLING THE ELSE HAMPERS PROGRAM CLEANLINESS-- //
3   33   162                       // DELETED FROM THE FORTRAN OUTPUT. //
3   33   163                       CALL LABFIL(LAB0)
3   33   164                    ENDIF
2   34   165                    LAB2=GENLAB(1)
2   34   166                    LAB1=STACK(TOP-2)
2   34   167                    IF NLAST.NE.1 THEN
3   35   168                       CALL LABFIL(LAB0)
3   35   169                       CALL FOUT(GOTO,1,6)
3   35   170                       CALL FOUT(LABEL,LAB2,LAB2+4)
3   35   171                       CALL FPRNT
3   35   172                    ENDIF
2   36   173                    // GO TO LAB2 //
2   36   174                    CALL LABFIL(LAB1)
2   36   175                    CALL FOUT(CONTI,1,8)
2   36   176                    CALL FPRNT
2   36   177                    // LAB1 CONTINUE //
2   36   178                    STACK(TOP-1)=LAB2
2   36   179                    CALL SCAN2 (ECOND)
2   36   180                    STEND=TCEND
2   36   181                    INDFLG=NEUTR
2   36   182                    CURFLG=PLLOUT
2   36   183                    IF FSTRT+INCR.LT.IMAX THEN
3   37   184                       FSTRT=FSTRT+INCR
```

```
3   37   185                ENDIF
2   38   186              : TYPE.EQ.FFOR :
2   39   187                IF LBFLG THEN
3   40   188                   //  LAB WILL HAVE IDENTIFYING LABEL IF ANY. //
3   40   189                   LAB=GENLAB(C)
3   40   190                   CALL FOUT(CONTI,1,8)
3   40   191                   CALL FPRNT
3   40   192                ENDIF
2   41   193                TOBEG=TOEND+1
2   41   194                CALL SCAN1(EQL,ECOND)
2   41   195                IF ECOND THEN
3   42   196                   CALL ERROR(2,EQL,1,3,0,0)
3   42   197                   //  ERROR--MISSING EQUAL-STATEMENT SKIPPED.  //
3   42   198                   CYCLE
3   42   199                ENDIF
2   43   200                VBLEA=TOBEG
2   43   201                VBLEE=TOEND
2   43   202                TOEND=TOEND+1
2   43   203                CALL SCAN(TOO,2,ECOND)
2   43   204                IF ECOND THEN
3   44   205                   CALL ERROR(2,TOO,2,3,0,0)
3   44   206                   // ERROR--MISSING 'TO' - STATEMENT SKIPPED. //
3   44   207                   TOEND=TOEND+1
3   44   208                   CYCLE
3   44   209                ENDIF
2   45   210                TOEND=TOEND+2
2   45   211                IF .NOT.LBFLG THEN
3   46   212                   CALL LABFIL(LAB0)
3   46   213                ENDIF
2   47   214                CALL FOUT(SPIN,VBLEA,TOEND-2)
2   47   215                CALL FPRNT
2   47   216                //  VARIABLE=EXPRESSION1.  //
2   47   217                ITMP=TOEND
2   47   218                CALL SCAN(BYE,2,ECOND1)
2   47   219                EX2A=TOBEG
2   47   220                EX2B=TOEND
2   47   221                TOEND=TOEND+2
2   47   222                IF ECOND1 THEN
```

```
3   48   223                  TCEND=ITMP
3   48   224               ENDIF.
2   49   225               CALL SCAN(DOC,2,ECOND)
2   49   226               IF ECOND1 THEN
3   50   227                  EX2B=TCEND
3   50   228               ENDIF
2   51   229               IF ECOND THEN
3   52   230                  CALL ERROR(2,DOO,2,1,0,0)
3   52   231                  // ERROR--MISSING 'DO'- SUPPLIED AT END OF STATEMENT. //
3   52   232                  TCEND=TCEND+1
3   52   233               ELSE
3   53   234                  TCEND=TCEND+2
3   53   235               ENDIF
2   54   236               LAB1=GENLAB(1)
2   54   237               LAB2= GENLAB(1)
2   54   238               LAB3=GENLAB(1)
2   54   239               LAB4=GENLAB(1)
2   54   240               // LAB IS THE IDENTIFYING LABEL, LAB4 IS THE TARGET OF EXIT, //
2   54   241               // AND LAB3 IS THE TARGET OF CYCLES. //
2   54   242               STACK(TOP+1)=LAB2
2   54   243               STACK(TOP+3)=LAB
2   54   244               STACK(TOP+4)=LAB3
2   54   245               STACK(TOP+5)=LAB4
2   54   246               STACK(TOP+6)=FFOR
2   54   247               STACK(TOP+2)=TPITR
2   54   248               TOP=TOP+6
2   54   249               IF TOP.GT.STMAX THEN
3   55   250                  CALL ERROR(6,0,0,5,0,0)
3   55   251               ENDIF
2   56   252               //  STACK OVERFLOWED----PROGRAM TERMINATED.  //
2   56   253               TPITR=TOP
2   56   254               CALL LABFIL(LAB0)
2   56   255               CALL FOUT(GOTO,1,6)
2   56   256               CALL FOUT(LABEL,LAB1,LAB1+4)
2   56   257               CALL FPRNT
2   56   258               // GO TO LAB1. //
2   56   259               CALL LABFIL(LAB2)
2   56   260               CALL FOUT(SPIN,VBLEA,VBLEB)
```

```
2    55    261          CALL FCUT(EQL,1,1)
2    56    262          CALL FCUT(SPIN,VBLEA,VBLEB)
2    56    263          CALL FCUT(FLL,1,1)
2    56    264          IF ECOND1 THEN
3    57    265             // THE 'BY' IS MISSING--DEFAULT INCREMENT OF ONE. //
3    57    266             CALL FCUT(ONE,1,1)
3    57    267          ELSE
3    58    268             CALL FCUT(IFNOT,9,9)
3    58    269             CALL FCUT(SPIN,TOBEG,TOEND-2)
3    58    270             CALL FCUT (FSTAR,1,1)
3    58    271          ENDIF
2    59    272          CALL FPRNT
2    59    273          //  LAB2--INDICATES VARIABLE=VARIABLE+EXPRESSION3. //
2    59    274          CALL LABFIL(LAB1)
2    59    275          CALL FCUT(IFNOT,1,9)
2    59    276          CALL FCUT(IFNOT,9,9)
2    59    277          CALL FCUT(SPIN,VBLEA,VBLEB)
2    59    278          CALL FCUT(MINUS,1,1)
2    59    279          CALL FCUT(IFNOT,9,9)
2    59    280          CALL FCUT(SPIN,EX2A,EX2B)
2    59    281          CALL FCUT(FSTAR,1,1)
2    59    282          CALL FCUT(FSTAR,1,1)
2    59    283          IF .NOT.ECOND1 THEN
3    60    284             // IF 'BY' MISSING THEN GENERATE...(EXP2)).LE.0... //
3    60    285             //  IN PLACE OF...(EXP2)*(EXP3).LE.0..... //
3    60    286             CALL FCUT(FSTAR,2,2)
3    60    287             CALL FCUT(IFNOT,9,9)
3    60    288             CALL FOUT(SPIN,TOBEG,TOEND-2)
3    60    289             CALL FCUT(FSTAR,1,1)
3    60    290          ENDIF
2    61    291          CALL FCUT(LEQ,1,5)
2    61    292          CALL FCUT(CGOTO,1,9)
2    61    293          CALL FCUT(LABEL,LAB4,LAB4+4)
2    61    294          CALL FPRNT
2    61    295          // LAB1...IF(.NOT.((VBLE-(EXP3).LE.0) GO TO LAB3... //
2    61    296          CALL SCAN2 (ECOND)
2    61    297          STEND=TOEND
2    61    298          INDFLG=FWRD
```

```
2   61   299                  CURFLG=CLD
2   61   300                  IF FSTRT+INCR.LT.IMAX THEN
3   62   301                     FSTRT=FSTRT+INCR
3   62   302                  ENDIF
2   63   303               : TYPE.EQ.FREPT :
2   64   304                  CALL CHECK(FREPT,BAD)
2   64   305                  IF BAD THEN
3   65   306                     CYCLE
3   65   307                  ENDIF
2   66   308                  IF LBFLG THEN
3   67   309                     //  LABELLED STMT, GENERATED 'LAB CONTINUE'  //
3   67   310                     CALL FCLT(CONTI,1,8)
3   67   311                     CALL FPRNT
3   67   312                  ENDIF
2   68   313                  STTOP=STACK(TOP)
2   68   314                  // LAB3=EXIT TARGET,LAB2=CYCLE TARGET,LAB1=LOOP BACK LABEL //
2   68   315                  IF STTOP.EQ.FFOR THEN
3   69   316                     LAB1=STACK(TOP-5)
3   69   317                     LAB2=STACK(TOP-2)
3   69   318                     LAB3=STACK(TOP-1)
3   69   319                     TOP=TOP-6
3   69   320                     LBPTR=LBPTR-20
3   69   321                  ELSE
3   70   322                     LAB1=STACK(TOP-3)
3   70   323                     LAB2=STACK(TOP-2)
3   70   324                     LAB3=STACK(TOP-1)
3   70   325                     TOP=TOP-5
3   70   326                     LBPTR=LBPTR-15
3   70   327                  ENDIF
2   71   328                  TPITR=STACK(TPITR-4)
2   71   329                  FSTRT=FSTRT-INCR
2   71   330                  FPTR=FSTRT
2   71   331                  CALL LABFIL(LAB2)
2   71   332                  CALL FCLT(GOTO,1,6)
2   71   333                  CALL FCUT(LABEL,LAB1,LAB1+4)
2   71   334                  CALL FPRNT
2   71   335                  //     GO TO LABEL 1 //
2   71   336                  CALL LABFIL(LAB3)
```

```
2   71   337                    CALL FCLT(CONTI,1,8)
2   71   338                    CALL FPRNT
2   71   339                    //    LAB3 CONTINUE    //
2   71   340                    CALL SCAN2 (ECOND)
2   71   341                    STEND=TCEND
2   71   342                    CURFLG=NEW
2   71   343                    INDFLG=BKWRD
2   71   344                 : TYPE.EQ.CCASE :
2   72   345                    LAB=GENLAB(1)
2   72   346                    //  LABEL(1:5)=BLANKS. //
2   72   347                    STACK(TOP+1)=1
2   72   348                    STACK(TOP+2)=LAB
2   72   349                    STACK(TOP+3)=CCASE
2   72   350                    TOP=TOP+3
2   72   351                    IF TOP.GT.STMAX THEN
3   73   352                       CALL ERROR(6,0,0,5,0,0)
3   73   353                    ENDIF
2   74   354                    //  STACK OVERFLOWED----PROGRAM TERMINATED. //
2   74   355                    EXPTR=0
2   74   356                    CALL SCAN(COF,2,ECOND)
2   74   357                    IF .NOT.ECOND THEN
3   75   358                       TCEND=TCEND+2
3   75   359                       EXPST=TCEND
3   75   360                       CALL SCAN1(CBSLH,ECOND)
3   75   361                       K=0
3   75   362                       FOR I=EXPST+1 TO TCEND DO
4   76   363                          K=K+1
4   76   364                          EXP(K)=SPIN(I)
4   76   365                       REPEAT
3   77   366                       EXPTR=K
3   77   367                    ENDIF
2   78   368                    CALL SCAN2 (ECOND)
2   78   369                    STEND=TCEND
2   78   370                    CURFLG=OLD
2   78   371                    INDFLG=FRWRD
2   78   372                    IF LBFLG THEN
3   79   373                       //  LABELED STATEMENT--GENERATE 'LAB CONTINUE'. //
3   79   374                       CALL FCLT(CONTI,1,8)
```

```
3   79   375                  CALL FPRNT
3   79   376                ENDIF
2   80   377                //  SET THE INDENTION  //
2   80   378                IF FSTRT+INCR.LT.IMAX THEN
3   81   379                   FSTRT=FSTRT+INCR
3   81   380                ENDIF
2   82   381             : TYPE.EQ.CCLON :
2   83   382                CALL CHECK(CCLON,BAD)
2   83   383                IF BAD THEN
3   84   384                   CYCLE
2   84   385                ENDIF
2   85   386                IF LBFLG THEN
3   86   387                   CALL ERROR(1,CASS,4,2,C,0)
3   86   388                   // ERROR--LABELLING THE CASE HAMPERS PROGRAM CLEANLINESS //
3   86   389                   // DELETED FROM THE FORTRAN OUTPUT. //
3   86   390                   CALL LABFIL(LAB0)
3   86   391                ENDIF
2   87   392                LAB1=STACK(TOP-1)
2   87   393                LAB2=STACK(TOP-2)
2   87   394                FSTRT=FSTRT-INCR
2   87   395                FPTR=FSTRT
2   87   396                // CASE MATERIAL PRINTED ONE IDENT TO THE RIGHT. //
2   87   397                IF LAB2.NE.1.AND.NLAST.NE.1 THEN
3   88   398                   CALL FOUT(GOTO,1,6)
3   88   399                   CALL    FOUT(LABEL,LAB1,LAB1+4)
3   88   400                   CALL LABFIL(LAB0)
3   88   401                   CALL    FPRNT
3   88   402                   //  GO TO LAB1.  //
3   88   403                ENDIF
2   89   404                CALL SCAN1(CCLN,ECEND)
2   89   405                IF ECEND THEN
3   90   406                   CALL ERROR(2,CCOLON,1,1,0,0)
3   90   407                ENDIF
2   91   408                // ERROR--MISSING COLON-SUPPLIED AT END OF STATEMENT. //
2   91   409                K=TOEND
2   91   410                TOEND=TOBEG
2   91   411                // SPIN(K)=IMAX OR SPIN(K+1)=SEMICOLON OR COLON. //
2   91   412                TMAX=IMAX
```

```
2   91   413                    IMAX=K
2   91   414                    CALL SCAN(ELZ,4,ECOND)
2   91   415                    IMAX=IMAX
2   91   416                    IF ECOND THEN
3   92   417                       LAB3=GENLAB(1)
3   92   418                       TCEND=K
3   92   419                       // OUTPUT-LAB2-IF NOT CONDITION THE GO TO LAB3. //
3   92   420                       CALL LABFIL(LAB2)
3   92   421                       CALL    FOUT(IFNOT,1,9)
3   92   422                       IF EXPTR.GT.J THEN
4   93   423                          CALL FOUT(EXP,1,EXPTR)
4   93   424                       ENDIF
3   94   425                       CALL FOUT(SPIN,TCBEG,TCEND)
3   94   426                       CALL    FOUT(CGOTO,1,9)
3   94   427                       CALL FOUT(LABEL,LAB3,LAB3+4)
3   94   428                       CALL    FPRNT
3   94   429                       TCEND=TCEND+1
3   94   430                    ELSE
3   95   431                       //  ELSE CLAUSE FOUND  //
3   95   432                       CALL LABFIL(LAB2)
3   95   433                       CALL FOUT(CONTI,1,8)
3   95   434                       CALL FPRNT
3   95   435                       //  LAB2 CONTINUE  //
3   95   436                       TCEND=K+1
3   95   437                       LAB3=1
3   95   438                       //  LABEL(1....5) ARE BLANKS.  //
3   95   439                    ENDIF
2   96   440                    STACK(TOP-2)=LAB3
2   96   441                    CALL SCAN2 (ECOND)
2   96   442                    STEND=TCEND
2   96   443                    INDFLG=NEUTP
2   96   444                    OUTFLG=PULOUT
2   96   445                    //  RESTORE THE INDENTATION.  //
2   96   446                    IF FSTRT+INCR.LT.IMAX THEN
3   97   447                       FSTRT=FSTRT+INCR
3   97   448                    ENDIF
2   98   449                 : TYPE.EQ.ENDCAS :
2   99   450                    CALL CHECK(ENDCAS,BAD)
```

```
2   99   451                    IF BAC THEN
3  100   452                       CYCLE ;
3  100   453                    ENDIF
2  101   454                    FSTRT=FSTRT-INCR
2  101   455                    FPTR=FSTRT
2  101   456                    IF LBFLG THEN
3  102   457                       // LABELLED STATEMENT--GENERATE 'LAB CONTINUE' //
3  102   458                       CALL FCLT(CONTI,1,8)
3  102   459                       CALL FPRNT
2  102   460                    ENDIF
2  103   461                    LAB1=STACK(TOP-1)
2  103   462                    LAB2=STACK(TOP-2)
2  103   463                    TOP=TOP-3
2  103   464                    LBPTR=LAB1
2  103   465                    IF LAB2.NE.1 THEN
3  104   466                       CALL LABFIL(LAB2)
3  104   467                       CALL FCLT(CONTI,1,8)
3  104   468                       CALL  FPRNT
3  104   469                    ENDIF
2  105   470                    //  LAB2 CONTINUE  //
2  105   471                    CALL LABFIL(LAB1)
2  105   472                    CALL FCLT(CONTI,1,8)
2  105   473                    CALL FPRNT
2  105   474                    //  LAB1 CONTINUE  //
2  105   475                    CALL SCAN2 (ECOND)
2  105   476                    STEND=TOEND
2  105   477                    INDFLG=BKWRD
2  105   478                    CURFLG=NEW
2  105   479                    EXPTR=0
2  105   480                 : TYPE.EQ.RETRN :
2  106   481                    LAST=1
2  106   482                    FTYPE=0
2  106   483                    CALL FHAND
2  106   484                    STEND=TOEND
2  106   485                    CURFLG=OLD
2  106   486                    INDFLG=NEUTR
2  106   487                 : TYPE.EQ.LOOP :
2  107   488                    //  LAB2 IS THE TARGET OF CYCLE AND LAB3 OF EXITS //
```

```
2  107   489                    //  LAB1 IS THE IDENTIFYING LABEL.  //
2  107   490                    LAB1=GENLAB(0)
2  107   491                    LAB2=GENLAB(1)
2  107   492                    LAB3= GENLAB(1)
2  107   493                    STACK(TOP+2)=LAB1
2  107   494                    STACK(TOP+3)=LAB2
2  107   495                    STACK(TOP+4)=LAB3
2  107   496                    STACK(TOP+5)=LLOOP
2  107   497                    STACK(TOP+1)=TPITR
2  107   498                    TOP=TOP+5
2  107   499                    IF TOP.GT.STMAX THEN
3  108   500                       CALL ERROR(6,0,0,5,0,0)
3  108   501                    ENDIF
2  109   502                    //  STACK OVERFLOWED, PROGRAM TERMINATED //
2  109   503                    CALL LABFIL(LAB1)
2  109   504                    CALL FOUT(CONTI,1,8)
2  109   505                    CALL FPRNT
2  109   506                    TPITR=TOP
2  109   507                    CALL SCAN2 (ECOND)
2  109   508                    STEND=TOEND
2  109   509                    CURFLG=OLD
2  109   510                    INDFLG=FRWRD
2  109   511                    IF FSTRT+INCR.LT.IMAX THEN
3  110   512                       FSTRT=FSTRT+INCR
3  110   513                    ENDIF
2  111   514                 : TYPE.EQ.WHLE :
2  112   515                    LAB1=GENLAB(0)
2  112   516                    LAB2=GENLAB(1)
2  112   517                    LAB3=GENLAB(1)
2  112   518                    //  LAB2 IS THE TARGET OF EXITS, LAB3 FOR CYCLES.  //
2  112   519                    CALL SCAN(DOC,2,ECOND)
2  112   520                    IF ECOND THEN
3  113   521                       CALL ERRLP(2,DOC,2,1,0,0)
3  113   522                       //  CALL ERROR---MISSING 'DO'-SUPPLIED //
3  113   523                       TOEND=TOEND+1
3  113   524                    ELSE
3  114   525                       TOEND=TOEND+2
3  114   526                    ENDIF
```

```
2  115   527                IF .NOT.LBFLG THEN
3  116   528                   CALL LABFIL(LAB1)
3  116   529                ENDIF
2  117   530                CALL FCLT(IFNCT,1,9)
2  117   531                CALL FCLT(SPIN,TOBEG,TCENC-2)
2  117   532                CALL FCUT(CGCTO,1,9)
2  117   533                CALL FCLT(LABEL,LAB2,LAB2+4)
2  117   534                CALL FPRNT
2  117   535                //  GENERATE LAB1 AND IF TRUE COND GO TO LAB2.  //
2  117   536                STACK(TCP+2)=LAB1
2  117   537                STACK(TCP+3)=LAB3
2  117   538                STACK(TCP+4)=LAB2
2  117   539                STACK(TCP+5)=WHLE
2  117   540                STACK(TCP+1)=TPITR
2  117   541                TCP=TCP+5
2  117   542                IF TCP.GT.STMAX THEN
3  118   543                   CALL ERRCR(6,0,0,5,0,0)
3  118   544                ENDIF
2  119   545                //  STACK CVERFLOWED, PROGRAM TERMINATED.  //
2  119   546                TPITR=TCP
2  119   547                CALL SCAN2 (ECCND)
2  119   548                STEND=TCENC
2  119   549                CURFLG=CLC
2  119   550                INDFLG=FRWRD
2  119   551                IF FSTRT+INCR.LT.IMAX THEN
3  120   552                   FSTRT=FSTRT+INCR
3  120   553                ENDIF
2  121   554             : TYPE.EQ.CONT :
2  122   555                FOR I=1 TC 6 DC
3  123   556                   FBUF(I)=SPIN(I)
3  123   557                REPEAT
2  124   558                TOBEG=7
2  124   559                WHILE SPIN(TOBEG).EQ.BLANK DC
3  125   560                   TOBEG=TOBEG+1
3  125   561                   IF TOBEG.GT.IMAX THEN
4  126   562                      TCENC=IMAX+1
4  126   563                      // BLANK CONTINUATION CARD, DELETED.  //
4  126   564                      EXIT
```

```
4  126   565                  ENDIF
3  127   566               REPEAT
2  128   567               IF TCEND.GT.IMAX THEN
3  129   568                  CYCLE ;
3  129   569               ENDIF
2  130   570               // SPIN(TCBEG)=NONBLANK; REST OF CARD CAN BE INDENTED.  //
2  130   571               CALL FRANC
2  130   572               STBEG=TCBEG
2  130   573               STEND=TCEND
2  130   574               CURFLG=OLD
2  130   575               INDFLG=NEUTR
2  130   576            : TYPE.EQ.EEXIT :
2  131   577               CALL CHECK(EEXIT,BAD)
2  131   578               IF BAD THEN
3  132   579                  CYCLE ;
3  132   580               ENDIF
2  133   581               CALL EXCYL(TYPE)
2  133   582            : TYPE.EQ.NCYCLE :
2  134   583               CALL CHECK(NCYCLE,BAD)
2  134   584               IF BAD THEN
3  135   585                  CYCLE
3  135   586               ENDIF
2  136   587               CALL EXCYL(TYPE)
2  136   588            : TYPE.EQ.UNTL :
2  137   589               CALL CHECK(UNTL,BAD)
2  137   590               IF BAD THEN
3  138   591                  CYCLE
3  138   592               ENDIF
2  139   593               FSTRT=FSTRT-INCR
2  139   594               FPTR=FSTRT
2  139   595               LAB1=STACK(TOP-3)
2  139   596               LAB2=STACK(TOP-2)
2  139   597               LAB3=STACK(TOP-1)
2  139   598               TOP=TOP-5
2  139   599               LBPTR=LPPTR-15
2  139   600               TPITR=STACK(TPITR-4)
2  139   601               IF LBFLG THEN
3  140   602                  CALL ERROR(1,OUNTL,5,2,0,0)
```

```
3  140  603                    //  ERROR---LABELLING OF THE UNTIL HAMPERS PROGRAM  //
3  140  604                    //  CLARITY, DELETED FROM FORTRAN OUTPUT.  //
3  140  605                 ENDIF
2  141  606                 CALL LABFIL(LAB2)
2  141  607                 CALL FOUTI(FNCT,1,9)
2  141  608                 CALL SCAN(REPT,6,ECOND)
2  141  609                 CALL FOUT(SPIN,TOBEG,TCEND)
2  141  610                 IF ECOND THEN
3  142  611                    CALL ERROR(2,REPT,6,1,0,0)
3  142  612                    // ERROR---MISSING REPEAT-SUPPLIED //
3  142  613                    TCEND=TCEND+1
3  142  614                 ELSE
3  143  615                    TCEND=TCEND+6
3  143  616                 ENDIF
2  144  617                 CALL FOUT( CGOTO,1,9)
2  144  618                 CALL FOUT (LABEL,LAB1,LAB1+4)
2  144  619                 CALL FPRNT
2  144  620                 //  GENERATE IF(.NOT.(COND)) GO TO LAB1  //
2  144  621                 //  GENERATE THE EXIT TARGET---LAB3 CONTINUE.  //
2  144  622                 CALL LABFIL(LAB3)
2  144  623                 CALL FOUT(CONTI,1,8)
2  144  624                 CALL FPRNT
2  144  625                 CALL SCAN2 (ECOND)
2  144  626                 STEND=TCEND
2  144  627                 INDFLG=BKWRD
2  144  628                 CURFLG=NEW
2  144  629              : TYPE.EQ.SUBFN :
2  145  630                 IF TOP.NE.0 THEN
3  146  631                    STTOP=STACK(TOP)
3  146  632                    CALL ENDFIX(EEND,STTOP,TYPE,ECOND1)
3  146  633                    CALL ERROR(7,0,0,6,0,0)
3  146  634                    // PREMATURE START //
3  146  635                    FSTRT=7
3  146  636                    SSTRT=10
3  146  637                 ENDIF
2  147  638                 WRITE(PRTR,20)
2  147  639     20          FORMAT(1H1)
2  147  640                 LBPTR=6
```

```
2  147   641                    TOP=0
2  147   642                    NUMB=100000
2  147   643                    STACK(TOP+1)=SUBEN
2  147   644                    TOP=TOP+1
2  147   645                    CALL FRAND
2  147   646                    FSTRT=FSTRT+INCR
2  147   647                    STEND=TOEND
2  147   648                    CURFLG=OLD
2  147   649                    INDFLG=FRWRD
2  147   650                 : TYPE.EQ.EEND :
2  148   651                    CALL CHECK(EEND,BAD)
2  148   652                    IF BAD THEN
3  149   653                       CYCLE ;
3  149   654                    ENDIF
2  150   655                    FSTRT=FSTRT-INCR
2  150   656                    FPTR=FSTRT
2  150   657                    IF LBFLG THEN
3  151   658                       // LABELLED STATEMENT--GENERATE 'LAB CONTINUE' //
3  151   659                       CALL FCLT(CONTI,1,8)
3  151   660                       CALL FPRNT
3  151   661                    ENDIF
2  152   662                    TOP=TOP-1
2  152   663                    CALL LABFIL(LABO)
2  152   664                    CALL FCLT(SPIN,TOBEG,TOEND)
2  152   665                    CALL FPRNT
2  152   666                    CALL SCAN2 (ECOND)
2  152   667                    STEND=TOEND
2  152   668                    INDFLG=BKWRD
2  152   669                    CURFLG=NEW
2  152   670                 : TYPE.EQ.COMT :
2  153   671                    TOEND=IMAX
2  153   672                    TOBEG=2
2  153   673                    WHILE SPIN(TOBEG).EQ.BLANK DO
3  154   674                       TOBEG=TOBEG+1
3  154   675                    REPEAT
2  155   676                    STBEG=TOBEG
2  155   677                    STEND=IMAX
2  155   678                    CURFLG=OLD
```

```
2  155   679                      INDFLG=NEUTR
2  155   680                    : TYPE.EQ.ECJ :
2  156   681                      //  CLOSE THE FILE.  //
2  156   682                      WRITE(PRTR,1300)
2  156   683    1300              FORMAT(/35H          END OF SPARKS PROGRAM.....)
2  156   684                      STOP
2  156   685                    ENDCASE
1  157   686                    IF CURFLG.EQ.FULLLT THEN
2  158   687                      SSTRT=SSTRT-INCR
2  158   688                      CALL SPRNT
2  158   689                      SSTRT=SSTRT+INCR
2  158   690                      IF SSTRT.GT.SMAX THEN
3  159   691                        SSTRT=SSTRT-INCR
3  159   692                      ENDIF
2  160   693                      BLKNO=BLKNO+1
2  160   694                    ENDIF
1  161   695                    IF CURFLG.EQ.CLD THEN
2  162   696                      CALL SPRNT
2  162   697                    ENDIF
1  163   698                    IF INDFLG.EQ.FRWRD THEN
2  164   699                      SSTRT=SSTRT+INCR
2  164   700                      IF SSTRT.GT.SMAX THEN
3  165   701                        SSTRT=SSTRT-INCR
3  165   702                      ENDIF
2  166   703                      IF CURFLG.EQ.NEW THEN
3  167   704                        CALL SPRNT
3  167   705                      ENDIF
2  168   706                      BLKNO=BLKNO+1
2  168   707                      NESTNO=NESTNO+1
2  168   708                    ENDIF
1  169   709                    IF INDFLG.EQ.BKWRD THEN
2  170   710                      SSTRT=SSTRT-INCR
2  170   711                      IF CURFLG.EQ.NEW THEN
3  171   712                        CALL SPRNT
3  171   713                      ENDIF
2  172   714                      NESTNO=NESTNO-1
2  172   715                      BLKNO=BLKNO+1
2  172   716                    ENDIF
```

```
1   173    717                 STMNO=STMNO+1
1   173    718               REPEAT
C   174    719             END
```

```
-1  175   720            BLOCK DATA
 C  176   721               IMPLICIT INTEGER(A-Z)
 C  176   722               COMMON/ALPH/CEE,DEE,AEE,FEE,PARN,COMMA,DBSLH
 C  176   723               COMMON/CONTRL/STACK(200),LABEL(200),STERM(6),TOP,LBMAX,
 C  176   724         1     LBPTR,TPITP,NUMB,STMAX
 C  176   725               COMMON/DEVIC/PRTR,FFILE,CRDR
 C  176   726               COMMON/FRTINF/FBUF(80),FPTR,FMAX,FSTRT,INCR
 C  176   727               COMMON/FTRT/DTA(4),FMAT(6),FTYPE,LAST,NLAST
 C  176   728               COMMON/INDT/INDFLG,CLRFLG,NEUTR,BKWRD,FRWRD,NEW,CLD,PULOUT
 C  176   729               COMMON/INPAT/TOO(2),THENN(4),BYE(2),DOO(2),ELZ(4),COLN
 C  176   730               COMMON/INPUT/SPIN(80),TCBEG,TCEND,STBEG,STEND,IMAX,BLANK
 C  176   731               COMMON/KEYW1/BEEND(3),ENCASS(7),ENIF(5),CASS(4),IIIF(2),CCOLON(5)
 C  176   732               COMMON/KEYW2/BEEXT(4),CCYCLE(5),OUNTL(5),ITERAT(9),REPT(6),OOF(2)
 C  176   733               COMMON/LOO/NUMBER(10),LBFLG,LABO
 C  176   734               COMMON/OTPAT/GOTO(9),CONTI(8),EQL(1),GOTO(6),IFNOT(9),
 C  176   735         1     LEQ(5),MINUS(1),PLU(1),RSTAR(2),ONE(1)
 C  176   736               COMMON/SPRINF/SSTRT,STMNO,BLKNO,NESTNO,SMAX,BLNKS(131)
 C  176   737               COMMON/STYP1/COLON,DBLSL,DELSE,UNTL,LLOOP,EEND,ENDCAS,RREPT
 C  176   738               COMMON/STYP2/EOJ,RINLE,NCYCLE,WHLE,CONT,COMT,FOTRN,SEMI
 C  176   739               COMMON/STYP3/IIF,CCASE,SUBFN,FFOR,EEXIT,EENDIF,RETRN
 C  176   740               COMMON/TABLES/TRES(23),RESRD(200)
 C  176   741               DATA SPIN/1HT,1HF,1HI,1HS,1F ,1FI,1HS,1H ,1HA,1F ,1HS,1HP,1HA,
 C  176   742         X     1HR,1HK,1HS,1H ,1HP,1HR,1HO,1HG,1HR,1HA,1HM,
 C  176   743         1     1F ,1H ,1H ,1H ,1H ,1F ,1F ,1H ,1H , 1H , 1H ,1H ,1H ,1H ,
 C  176   744         2     1H ,1H ,1H ,1H ,1H ,1H ,1H ,1H ,1H , 1H , 1H ,1H ,1H ,1H ,
 C  176   745         3     1F ,1F ,1H ,1H ,1H ,1H ,1H ,1H ,1H , 1H , 1H ,1H ,1H ,1H ,
 C  176   746         4     1F ,1H ,1H ,1F ,1H ,1H ,1H ,1H ,1H , 1F , 1H ,1F ,1F ,1H /
 C  176   747               DATA STMAX,LBMAX,NUMB,BLNKS/200,200,100000,131*1H /
 C  176   748               DATA NEUTR,BKWRD,FRWRD,NEW,CLD,PULOUT/1,2,3,1,2,3/
 C  176   749               //   STACK FRAME SIZES:                                    //
 C  176   750               //          CASE    FOR      IF      LOOP    SUBFN    WHILE  //
 C  176   751               DATA STERM /  3   ,   6   ,   3   ,   5   ,   1   ,   5   /
 C  176   752               DATA CONTI,MINUS,LABO/1HC,1HC,1FN,1HT,1HI,1HN,1HU,1HE,1H-,1/
 C  176   753               DATA LABEL,ONE,PLU/1F ,1H ,1H ,1H ,1H ,195*1H ,1H1,1H+/
 C  176   754               DATA GOTO,LEQ/1HG,1HO,1H ,1HT,1HO,1F ,1F=,1FL,1F=,1F.,1HO/
 C  176   755               DATA IFNOT,RSTAR/1HI,1HF,1H(,1H.,1HN,1HO,1HT,1H.,1H(,1H),1H*/
 C  176   756               DATA COTO,EQL/1F),1F),1F ,1FG,1HO,1H ,1FT,1HO,1H ,1H=/
 C  176   757               DATA DOO,BYE,THENN,REPT,TOO/1HO,1HO,1FB,1HY,1HT,1HH,1HE,1HN,1HR,
```

```
      1HE,1HP,1HE,1HA,1HT,1HT,1HC/
      DATA     CCASE,CCLCN,CBLSL,CELSE,CEND,CDJ,CEXIT,CFOR,COMT,
     1CONT,CCTRN,1IF,LLCCP,CREPT,CNTL,WHLE,SUBFN,RINLO
     1,NCYCLE,ENDCAS,CENDIF,RETRN
     1/1,7,8,9,10,11,12,2,13,14,15,3,4,16,17,6,5,18,19,20,21,
     122/
      DATA RESRD/5,1HB,1HL,1HO,1HC,1HK,5,4,1HC,1HA,1HS,1HE,1,
     15,1HC,1HY,1HC,1HL,1HE,19,4,1HE,1HL,1HS,1HE,9,
     2 3,1HE,1HN,1HO,10,7,1HE,1HN,1HO,1HC,1HA,1HS,1HE,20,
     3 5,1HE,1HN,1HC,1HI,1HF,21,3,1HE,1HO,1HJ,11,
     4 4,1HE,1HX,1HI,1HT,12,3,1HF,1HC,1HR,2,
     5 8,1HF,1HU,1HN,1HC,1HT,1HI,1HC,1HN,5,2,1HI,1HF,3,
     6 7,1HI,1HN,1HT,1HE,1HC,1HE,1HR,18,7,1HL,1HO,1HG,1HI,1HC,
     7 1HA,1HL,18,4,1HL,1HC,1HO,1HP,4,4,1HR,1HE,1HA,1HL,13,
     8 6,1HR,1HE,1HP,1HC,1HA,1HT,16,6,1HA,1HE,1HT,1HU,1HR,1HN,
     9 22,10,1HS,1HL,1HB,1HR,1HC,1HU,1HT,1HI,1HN,1HE,5,
     A 5,1HO,1HN,1HT,1HI,1HL,17,5,1HW,1HH,1HI,1HL,1HE,6,51*0/
      DATA TBES/1,8,14,21,27,32,41,48,53,59,64,74,78,87,96,102,108,
     1 116,124,136,143/
      DATA FRTR,CRDR,FFILE/ 6, 5, 4/
      DATA BEEND,ENDASS/1HE,1HN,1HC,1HE,1HN,1HD,1HC,1HA,1HS,1HE/
      DATA BEEXT,ENIF/1HE,1HX,1HI,1HT,1HE,1HN,1HD,1HI,1HF/
      DATA CCYCLE,CASS/1HC,1HY,1HC,1HL,1HE,1HC,1HA,1HS,1HE/
      DATA CONTL,IIIF/1HC,1HN,1HT,1HI,1HL,1HI,1HF/
      DATA CCLN,CCCLCN/1H:,1HC,1HO,1HL,1HC,1HN/
      DATA ITERAT/1HI,1HT,1HE,1HR,1HA,1HT,1HI,1HV,1HE/
      DATA BLANK,CEE,SEMI,CCF/1H ,1HC,1H;,1HC,1HF/
      DATA CBSLH,CLZ/1H/,1HE,1HL,1HS,1HE/
      DATA NUMBER/1H0,1H1,1H2,1H3,1H4,1H5,1H6,1H7,1H8,1H9/
      DATA FEE,DEE,AEE,PARN,COMMA/1HF,1HD,1HA,1H(,1H,/
      DATA FMAT,CTA/1HF,1HC,1HR,1HM,1HA,1HT,1HD,1HA,1HT,1HA/
      END
```

```
-1  177   790          SUBROUTINE DETMX(TYPE)
 C  178   791             // THE SUBROUTINE DETERMINES THE TYPE OF CYCLE STATEMENT AND     //
 C  178   792             // SETS TOBEG AND TOEND ON TOKEN BOUNDARIES.                     //
 C  178   793             IMPLICIT INTEGER(A-Z)
 C  178   794             LOGICAL LBFLG,ECOND
 C  178   795             COMMON/ALPH/CEE,DEE,AEE,FEE,PARN,COMMA,DBSLH
 C  178   796             COMMON/DEVIC/PRTR,FFILE,CRDR
 C  178   797             COMMON/FRTINF/FBLF(80),FPTR,FMAX,FSTRT,INCR
 C  178   798             COMMON/INPUT/SPIN(80),TOBEG,TOEND,STBEG,STEND,IMAX,BLANK
 C  178   799             COMMON/COD/NUMBER(10),LBFLG,LABO
 C  178   800             COMMON/STYP2/ECJ,RINLC,NCYCLE,WHLE,CONT,COMT,FCTRN,SEMI
 C  178   801             LBFLG=.FALSE.
 C  178   802             LOOP
 1  179   803                LOOP
 2  180   804                   TOEND=TOEND+1
 2  180   805                   IF TOEND.GT.IMAX THEN
 3  181   806                      EXIT
 3  181   807                   ENDIF
 2  182   808                   IF SPIN(TOEND).NE.BLANK THEN
 3  183   809                      IF SPIN(TOEND).EQ.SEMI THEN
 4  184   810                         CYCLE
 4  184   811                      ENDIF
 3  185   812                      IF SPIN(1).EQ.CEE THEN
 4  186   813                         TOBEG=1
 4  186   814                         TOEND=1
 4  186   815                         TYPE=COMT
 4  186   816                         RETURN
 4  186   817                      ENDIF
 3  187   818                      TOBEG=TOEND
 3  187   819                      CALL SCAN1(BLANK,ECOND)
 3  187   820                      CASE
 4  188   821                      : SPIN(6).NE.BLANK :
 4  189   822                         TYPE=CONT
 4  189   823                         RETURN
 4  189   824                      : ECOND :
 4  190   825                         TYPE=FCTRN
 4  190   826                         RETURN
 4  190   827                      : TOEND.GT.6 :
```

```
4  191  828                          CALL STTYP(TYPE)
4  191  829                          RETURN
4  191  830                       : TCEND.LE.6 :
4  192  831                          FOR I=1 TO 5 DO
5  193  832                             FBUF(I)=SPIN(I)
5  193  833                          REPEAT
4  194  834                          LDFLG=.TRUE.
4  194  835                          FBUF(6)=BLANK
4  194  836                       ENDCASE
3  195  837                    ENDIF
2  196  838                 REPEAT
1  197  839                 READ(CRDR,20) SPIN
1  197  840       20        FORMAT(80A1)
1  197  841                 IF SPIN(1).EQ.FEE THEN
2  198  842                    READ(CRDR,20)SPIN
2  198  843                    WHILE SPIN(1).NE.FEE DO
3  199  844                       WRITE(PRTR,33)(SPIN(I),I=1,80)
3  199  845       33              FORMAT(20X,80A1)
3  199  846                       WRITE(FFILE,32)(SPIN(I),I=1,80)
3  199  847       32              FORMAT(80A1)
3  199  848                       READ(CRDR,20) SPIN
3  199  849                    REPEAT
2  200  850                    READ(CRDR,20)SPIN
2  200  851                 ENDIF
1  201  852                 TCEND=0
1  201  853              REPEAT
0  202  854           END
```

```
-1  203   855          SUBROUTINE STTYP(TYPE)
 C  204   856             //  THE TOKEN AT SPIN(TOBEG) TO SPIN(TOEND) IS SEARCHED          //
 C  204   857             //     FOR IN THE SYMBOL TABLE.                                  //
 C  204   858             IMPLICIT INTEGER(A-Z)
 C  204   859             COMMON/ALPH/CEE,LEE,AEE,FEE,PARN,COMMA,DBSLH
 C  204   860             COMMON/INPAT/TCC(2),THENN(4),BYE(2),DCC(2),ELZ(4),CCLN
 C  204   861             COMMON/INPUT/SPIN(80),TOBEG,TOEND,STBEG,STEND,IMAX,BLANK
 C  204   862             COMMON/STYP1/CCLCN,DBLSL,EELSE,UNTL,LLOOP,FEND,ENDCAS,RREPT
 C  204   863             COMMON/STYP2/ECJ,RINLO,NCYCLE,WHLE,CONT,COMT,FCTRN,SEMI
 C  204   864             COMMON/TABLES/TRES(23),RESRD(200)
 C  204   865             IF SPIN(TOBEG).EQ.CCLN THEN
 1  205   866                TOEND=TOBEG
 1  205   867                TYPE=CCLCN
 1  205   868                RETURN
 1  205   869             ENDIF
 C  206   870             IF SPIN(TOBEG).EQ.DBSLH.AND.SPIN(TOBEG+1).EQ.DBSLH THEN
 1  207   871                TYPE=DBLSL
 1  207   872                RETURN
 1  207   873             ENDIF
 C  208   874             TOP=1
 C  208   875             BOTTOM=21
 C  208   876             LENGTH=TOEND-TOBEG+1
 C  208   877             WHILE TOP.LE.BOTTOM DO
 1  209   878                MID=(TOP+BOTTOM)/2
 1  209   879                I=TRES(MID)
 1  209   880                CASE
 2  210   881                : SPIN(TOBEG).LT.RESRD(I+1) :
 2  211   882                   BOTTOM=MID-1
 2  211   883                : SPIN(TOBEG).GT.RESRD(I+1) :
 2  212   884                   TOP=MID+1
 2  212   885                : SPIN(TOBEG).EQ.RESRD(I+1) :
 2  213   886                   PNT=2
 2  213   887                   WHILE PNT.LE.RESRD(I).AND.PNT.LE.LENGTH DO
 3  214   888                      ITEMP=I+PNT
 3  214   889                      JTEMP=TOBEG-1+PNT
 3  214   890                      CASE
 4  215   891                      : SPIN(JTEMP).LT.RESRD(ITEMP) :
 4  216   892                         BOTTOM=BOTTOM-1
```

```
4  216  893                        PNT=9999
4  216  894                     : SPIN(JTEMP).GT.RESRC(ITEMP) :
4  217  895                        TOP=TOP+1
4  217  896                        PNT=9999
4  217  897                     : SPIN(JTEMP).EQ.RESRC(ITEMP) :
4  218  898                        PNT=PNT+1
4  218  899                     ENDCASE
3  219  900                  REPEAT
2  220  901                  IF PNT.NE.9999 THEN
3  221  902                     CASE
4  222  903                     : LENGTH.EQ.RESRC(I) :
4  223  904                        JTEMP=LENGTH+1+I
4  223  905                        TYPE=RESRC(JTEMP)
4  223  906                        RETURN
4  223  907                     : LENGTH.GT.RESRC(I) :
4  224  908                        TOP=TOP+1
4  224  909                     : LENGTH.LT.RESRC(I) :
4  225  910                        BOTTOM=BOTTOM-1
4  225  911                     ENDCASE
3  226  912                  ENDIF
2  227  913               ENDCASE
1  228  914            REPEAT
0  229  915            TYPE=FOTRN
0  229  916            RETURN
0  229  917      END
```

```
-1  230   918          SUBROUTINE SCAN(PAT,LEN,ECOND)
 C  231   919             // THE SUBROUTINE COMPARES THE PATERN PAT WITH ALL TOKENS          //
 C  231   920             // SURROUNDED BY BLANKS FROM SPIN(TCEND+1) ONWARDS AND            //
 C  231   921             // RETURNS WITH TCEND AT THE OLD POSITION OF TCEND; WITH THE      //
 C  231   922             // PATTERN STARTING AT TCEND+1.  IF NOT FOUND ECOND IS SET TRUE   //
 C  231   923             IMPLICIT INTEGER(A-Z)
 C  231   924             LOGICAL ECOND
 C  231   925             COMMON/INPUT/SPIN(80),TCBEG,TCEND,STBEG,STEND,IMAX,BLANK
 C  231   926             DIMENSION PAT(6)
 C  231   927             TCBEG=TCEND+1
 C  231   928             LOOP
 1  232   929                K=TCEND+1
 1  232   930                WHILE K.LE.IMAX.AND.SPIN(K).EQ.BLANK DO
 2  233   931                   K=K+1
 2  233   932                REPEAT
 1  234   933                TCEND=K
 1  234   934                CALL SCAN1(BLANK,ECOND)
 1  234   935                IF ECOND 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   940                ENDIF
 1  238   941                // COMPARE THE PATTERN WITH A TOKEN OF THE SAME LENGTH. //
 1  238   942                FOR I=1 TO LEN DO
 2  239   943                   J=K+I-1
 2  239   944                   IF PAT(I).NE.SPIN(J) THEN
 3  240   945                      EXIT
 3  240   946                   ENDIF
 2  241   947                   IF I.EQ.LEN THEN
 3  242   948                      // PATTERN FOUND, RETURN WITH ECOND FALSE. //
 3  242   949                      TCEND=TCEND-LEN
 3  242   950                      RETURN
 3  242   951                   ENDIF
 2  243   952                REPEAT
 1  244   953             REPEAT
 C  245   954          END
```

```
-1  246   955          SUBROUTINE SCAN1(CHAR,ECOND)
 C  247   956             //  LOOKS FOR CHAR AND IF FOUND, SPIN(TOEND+1)=CHAR                //
 C  247   957             //  AND ECOND=FALSE. IF NOT FOUND UPTO IMAX OR SEMICOLON           //
 C  247   958             //  ECOND=TRUE.                                                    //
 C  247   959             IMPLICIT INTEGER(A-Z)
 C  247   960             LOGICAL ECOND
 C  247   961             COMMON/ALPH/CEE,DEE,AEE,FEE,PARN,COMMA,DBSLH
 C  247   962             COMMON/INPUT/SPIN(80),TOBEG,TOEND,STBEG,STEND,IMAX,BLANK
 C  247   963             COMMON/STYP2/EOJ,RINLO,NCYCLE,WHLE,CONT,COMT,FCTRN,SEMI
 C  247   964             ECOND =.FALSE.
 C  247   965             PTR=TOEND
 C  247   966             WHILE TOEND.LT.IMAX DO
 1  248   967                ITEMP=SPIN(TOEND+1)
 1  248   968                CASE
 2  249   969                : ITEMP.EQ.SEMI :
 2  250   970                   ECOND=.TRUE.
 2  250   971                   RETURN
 2  250   972                : ITEMP.EQ.DBSLH :
 2  251   973                   IF CHAR.EQ.DBSLH THEN
 3  252   974                      IF SPIN(TOEND+2).EQ.DBSLH THEN
 4  253   975                         RETURN
 4  253   976                      ENDIF
 3  254   977                   ENDIF
 2  255   978                : ITEMP.EQ.CHAR :
 2  256   979                   RETURN
 2  256   980                ENDCASE
 1  257   981                TOEND=TOEND+1
 1  257   982                IF ITEMP.NE.BLANK THEN
 2  258   983                   PTR=TOEND
 2  258   984                ENDIF
 1  259   985             REPEAT
 C  260   986             ECOND=.TRUE.
 C  260   987             TOEND=PTR
 C  260   988             RETURN
 C  260   989          END
```

```
-1  261   990          SUBROUTINE SCAN2 (ECOND)
 C  262   991             IMPLICIT INTEGER (A-Z)
 C  262   992             LOGICAL ECOND
 C  262   993             COMMON/ALPH/CEE,DEE,AEE,FEE,PARN,COMMA,DBSLH
 C  262   994             COMMON/INPUT/SPIN(80),TOBEG,TOEND,STBEG,STEND,IMAX,BLANK
 C  262   995             A=TOEND
 C  262   996             TOEND=TOEND+1
 C  262   997             ECOND=.FALSE.
 C  262   998             WHILE TOEND.LT.IMAX DO
 1  263   999                TOK=SPIN(TOEND)
 1  263  1000                IF TOK.NE.BLANK THEN
 2  264  1001                   IF TOK.EQ.DBSLH THEN
 3  265  1002                      TOEND=TOEND+1
 3  265  1003                      WHILE TOEND.LT.IMAX DO
 4  266  1004                         IF SPIN(TOEND).EQ.DBSLH.AND.SPIN(TOEND+1).EQ.DBSLH THEN
 5  267  1005                            TOEND=TOEND+1
 5  267  1006                            RETURN
 5  267  1007                         ENDIF
 4  268  1008                         TOEND=TOEND+1
 4  268  1009                      REPEAT
 3  269  1010                      TOEND=TOEND+1
 3  269  1011                      CALL ERROR(2,DBSLH,2,1,0,0)
 3  269  1012                      RETURN
 3  269  1013                   ELSE
 3  270  1014                      TOEND=A
 3  270  1015                      RETURN
 3  270  1016                   ENDIF
 2  271  1017                ENDIF
 1  272  1018                TOEND=TOEND+1
 1  272  1019             REPEAT
 C  273  1020             TOEND=A
 C  273  1021             ECOND=.TRUE.
 C  273  1022             RETURN
 C  273  1023          END
```

```
-1  274  1024+          SUBROUTINE FHAND
 C  275  1025             //   COMPLETES FORTRAN STATEMENTS AND OUTPUTS THE FORTRAN.      //
 C  275  1026             IMPLICIT INTEGER(A-Z)
 C  275  1027             LOGICAL LBFLG,FCOND,ECOND1
 C  275  1028             COMMON/ALPH/CEE,DEE,AEE,FEE,PARN,COMMA,DBSLH
 C  275  1029             COMMON/FCRT/PTR,ECOND,NUMCOM,BLNKLN,VARLCC(10)
 C  275  1030             COMMON/FRTINF/FBUF(80),FPTR,FMAX,FSTRT,INCR
 C  275  1031             COMMON/FTRT/DTA(4),FMAT(6),FTYPE,LAST,NLAST
 C  275  1032             COMMON/INPUT/SPIN(80),TOBEG,TOEND,STBEG,STEND,IMAX,BLANK
 C  275  1033             COMMON/COD/NUMBER(10),LBFLG,LABO
 C  275  1034+            COMMON/STYP2/ECJ,PINDC,NCYCLE,WHLE,CONT,COMT,FOTRN,SEMI
 C  275  1035             COMMON/TYP/TYPE
 C  275  1036             IF .NOT.LBFLG THEN
 1  276  1037                CALL LABFIL(LABO)
 1  276  1038             ENDIF
 C  277  1039             IF TYPE.NE.RETRN THEN
 1  278  1040                IF TYPE.EQ.CONT THEN
 2  279  1041                   IF FTYPE.EQ.1 THEN
 3  280  1042                      TOEND=IMAX
 3  280  1043                   ENDIF
 2  281  1044                   FBUF(6)=SPIN(6)
 2  281  1045                   ECOND=.TRUE.
 2  281  1046                ELSE
 2  282  1047                   CALL FTYP
 2  282  1048                ENDIF
 1  283  1049             ENDIF
 C  284  1050             CASE
 1  285  1051             : FTYPE.EQ.1 :
 1  286  1052                TEMP=FSTRT
 1  286  1053                FSTRT=7
 1  286  1054                FPTR=7
 1  286  1055                CALL FOUT(SPIN,TOBEG,TOEND)
 1  286  1056                CALL FPRNT
 1  286  1057                FSTRT=TEMP
 1  286  1058                FPTR=FSTRT
 1  286  1059             : FTYPE.EQ.2 :
 1  287  1060                EXPEND=TOEND
 1  287  1061                FOR J=1 TO NUMCOM DO
```

```
2  288  1062                    TCEND=VARLOC(J)-1
2  288  1063                    CALL FOUT(SPIN,TOBEG,TCEND)
2  288  1064                    CALL FOUT(SPIN,PTR,EXPEND)
2  288  1065                    CALL FPRNT
2  288  1066                    TOBEG=TCEND+2
2  288  1067                 REPEAT
1  289  1068                 CALL FOUT(SPIN,TCBEG,EXPEND)
1  289  1069                 CALL FPRNT
1  289  1070                 TCEND=EXPEND
1  289  1071              : PTYPE.EQ.J :
1  290  1072                 CALL FOUT(SPIN,TOBEG,TCEND)
1  290  1073                 CALL FPRNT
1  290  1074              ENDCASE
C  291  1075              IF .NOT.ECEND THEN
1  292  1076                 CALL SCAN2(ECEND)
1  292  1077              ENDIF
C  293  1078              RETURN
C  293  1079           END
```

```
-1  294  1080            SUBROUTINE FTYP
 C  295  1081               IMPLICIT INTEGER(A-Z)
 C  295  1082               LOGICAL ECCND
 C  295  1083               COMMON/ALPH/CEE,DEE,AEE,FEE,PARN,COMMA,DBSLH
 C  295  1084               COMMON/FCRT/PTR,ECCND,NUMCOM,BLANKN,VARLCC(10)
 C  295  1085               COMMON/FTRT/DTA(4),FMAT(6),FTYPE,LAST,NLAST
 C  295  1086               COMMON/INPUT/SPIN(80),TCBEG,TCEND,STBEG,STEND,IMAX,BLANK
 C  295  1087               COMMON/CTPAT/CGCTC(9),CONTI(8),EQL(1),GOTO(6),IFNOT(9),
 C  295  1088         1     LEC(5),MINUS(1),PLU(1),RSTAR(2),ONE(1)
 C  295  1089               NUMCOM=0
 C  295  1090               FTYPE=0
 C  295  1091               PTR=TCBEG
 C  295  1092               TOK=SPIN(PTR)
 C  295  1093               IF TOK.EQ.FEE THEN
 1  296  1094                  FOR I=2 TO 6 DO
 2  297  1095                     IF SPIN(PTR+I-1).NE.FMAT(I) THEN
 3  298  1096                        EXIT
 3  298  1097                     ELSE
 3  299  1098                        IF I.EQ.6 THEN
 4  300  1099                           TCEND=IMAX
 4  300  1100                           FTYPE=1
 4  300  1101                           RETURN
 4  300  1102                        ENDIF
 3  301  1103                     ENDIF
 2  302  1104                  REPEAT
 1  303  1105               ENDIF
 C  304  1106               IF TOK.EQ.DEE THEN
 1  305  1107                  FOR I=2 TO 4 DO
 2  306  1108                     IF SPIN(PTR+I-1).NE.DTA(I) THEN
 3  307  1109                        EXIT
 3  307  1110                     ELSE
 3  308  1111                        IF I.EQ.4 THEN
 4  309  1112                           TCEND=IMAX
 4  309  1113                           FTYPE=1
 4  309  1114                           RETURN
 4  309  1115                        ENDIF
 3  310  1116                     ENDIF
 2  311  1117                  REPEAT
```

```
1   312  1118              ENDIF
0   313  1119              CALL SCAN1(DBSLH,ECEND)
C   313  1120              WHILE PTR.LT.TCEND DO
1   314  1121                 PTR=PTR+1
1   314  1122                 TOK=SPIN(PTR)
1   314  1123                 CASE
2   315  1124                 : TOK.EQ.PARN :
2   316  1125                    PAR=1
2   316  1126                    LOOP
3   317  1127                       PTR=PTR+1
3   317  1128                       TOK=SPIN(PTR)
3   317  1129                       CASE
4   318  1130                       : TOK.EQ.PARN :
4   319  1131                          PAR=PAR+1
4   319  1132                       : TOK.EQ.RSTAR(1) :
4   320  1133                          PAR=PAR-1
4   320  1134                       : PTR.EQ.TCEND :
4   321  1135                          EXIT
4   321  1136                       ENDCASE
3   322  1137                    UNTIL PAR.EQ.0 REPEAT
2   323  1138                 : TOK.EQ.COMMA :
2   324  1139                    NUMCOM=NUMCOM+1
2   324  1140                    VARLOC(NUMCOM)=PTR
2   324  1141                 : TOK.EQ.ECL(1) :
2   325  1142                    IF NUMCOM.GT.0 THEN
3   326  1143                       FTYPE=2
3   326  1144                    ENDIF
2   327  1145                    RETURN
2   327  1146                 ENDCASE
1   328  1147              REPEAT
C   329  1148           END
```

```
-1  330  1149          SUBROUTINE FOUT(PAT,STRT,END)
 0  331  1150             // THE VALUE OF FSTRT WILL USUALLY BE 7 AND FMAX WILL BE 72.     //
 0  331  1151             //  FPTR SHOULD BE AT FSTRT.  BLNKLN IS A FLAG TO INDICATE A     //
 0  331  1152             // TO INDICATE A SO FAR BLANK LINE.                              //
 0  331  1153             IMPLICIT INTEGER(A-Z)
 0  331  1154             LOGICAL BLNKLN,LBFLG
 0  331  1155             COMMON/ALPH/CEE,DEE,AEE,FEE,PARN,COMMA,DBSLH
 0  331  1156             COMMON/FCRT/PTR,ECOND,NUMCOM,BLNKLN,VARLOC(10)
 0  331  1157             COMMON/FRTINF/FBUF(80),FPTR,FMAX,FSTRT,INCR
 0  331  1158             COMMON/INPUT/SPIN(80),TOBEG,TOEND,STBEG,STEND,IMAX,BLANK
 0  331  1159             COMMON/STYP2/ECJ,RINLC,NCYCLE,WHLE,CONT,CONT,FOTRN,SEMI
 0  331  1160             COMMON /TYP/ TYPE
 0  331  1161             DIMENSION PAT(1)
 0  331  1162             BLNKLN = .FALSE.
 0  331  1163             FOR J=STRT TO END DO
 1  332  1164                IF FPTR.GT.FMAX THEN
 2  333  1165                   CALL FPRNT
 2  333  1166                   FOR I=1 TO 5 DO
 3  334  1167                      FBUF(I)=BLANK
 3  334  1168                   REPEAT
 2  335  1169                   FBUF(6)=AEE
 2  335  1170                   JJ=J
 2  335  1171                   WHILE PAT(JJ).EQ.BLANK DO
 3  336  1172                      IF JJ.EQ.END THEN
 4  337  1173                         BLNKLN=.TRUE.
 4  337  1174                         RETURN
 4  337  1175                      ENDIF
 3  338  1176                      JJ=JJ+1
 3  338  1177                   REPEAT
 2  339  1178                ENDIF
 1  340  1179                FBUF(FPTR)=PAT(J)
 1  340  1180                FPTR=FPTR+1
 1  340  1181             REPEAT
 0  341  1182             RETURN
 0  341  1183          END
```

```
-1  342  1184              SUBROUTINE FPRNT
 0  343  1185                 IMPLICIT INTEGER(A-Z)
 0  343  1186                 LOGICAL BLNKLN
 0  343  1187                 COMMON/DEVIC/PRTR,FFILE,CRDR
 0  343  1188                 COMMON/FORT/PTR,ECEND,NUMCOM,BLNKLN,VARLOC(10)
 0  343  1189                 COMMON/FRTINF/FBUF(80),FPTR,FMAX,FSTRT,INCR
 0  343  1190                 COMMON/FTRT/DTA(4),FMAT(6),FTYPE,LAST,NLAST
 0  343  1191                 COMMON/INPUT/SPIN(80),TOBEG,TOEND,STBEG,STEND,IMAX,BLANK
 0  343  1192                 COMMON/SPRINF/SSTRT,STMNO,BLKNO,NESTNO,SMAX,BLNKS(131)
 0  343  1193                 IF FPTR.EQ.FSTRT THEN
 1  344  1194                    RETURN
 1  344  1195                 ENDIF
 0  345  1196                 IFF=FSTRT-1
 0  345  1197                 IF IFF.GE.7 THEN
 1  346  1198                    FOR I=7 TO IFF DO
 2  347  1199                       FBUF(I)=BLANK
 2  347  1200                    REPEAT
 1  348  1201                 ENDIF
 0  349  1202                 IF FPTR.LE.FMAX THEN
 1  350  1203                    FOR I=FPTR TO FMAX DO
 2  351  1204                       FBUF(I)=BLANK
 2  351  1205                    REPEAT
 1  352  1206                 ENDIF
 0  353  1207                 FOR I=73 TO 75 DO
 1  354  1208                    FBUF(I)=SPIN(I)
 1  354  1209                 REPEAT
 0  355  1210                 IF .NOT.BLNKLN THEN
 1  356  1211                    WRITE(FFILE,10)(FBUF(I),I=1,75),STMNO
 1  356  1212           10       FORMAT(75A1,I5)
 1  356  1213                 ENDIF
 0  357  1214                 FPTR=FSTRT
 0  357  1215                 NLAST=LAST
 0  357  1216                 LAST=0
 0  357  1217                 RETURN
 0  357  1218              END
```

```
-1  358  1219              SUBROUTINE SPRNT
 0  359  1220                 IMPLICIT INTEGER(A-Z)
 C  359  1221                 LOGICAL LBFLG,CVFL
 C  359  1222                 COMMON/ALPH/CEE,DEE,AEE,FEE,PARN,COMMA,DBSLH
 C  359  1223                 COMMON/DEVIC/PRTR,FFILE,CRDR
 C  359  1224                 COMMON/INPUT/SPIN(80),TOBEG,TOEND,STBEG,STEND,IMAX,BLANK
 C  359  1225                 COMMON/COD/NUMBER(10),LBFLG,LABO
 C  359  1226                 COMMON/SPRINF/SSTRT,STMNO,BLKNO,NESTNO,SMAX,BLNKS(131)
 C  359  1227                 COMMON/STYP2/ECJ,RTNLO,NCYCLE,WHLE,CONT,COMT,FOTRN,SEMI
 C  359  1228                 COMMON/TYP/TYPE
 C  359  1229                 FOR I=1 TO 6 DO
 1  360  1230                    BLNKS(I)=SPIN(I)
 1  360  1231                 REPEAT
 C  361  1232                 LOOP
 1  362  1233                    CVFL = .FALSE.
 1  362  1234                    LEN1=STEND-STBEG+1
 1  362  1235                    LEN2=SMAX-SSTRT+1
 1  362  1236                    SEND=STEND
 1  362  1237                    IF LEN1.GT.LEN2 THEN
 2  363  1238                       CVFL=.TRUE.
 2  363  1239                       SEND=STBEG+LEN2-1
 2  363  1240                    ENDIF
 1  364  1241                    FIL=LEN2-LEN1+7
 1  364  1242                    WRITE(PRTR,10) NESTNO,BLKNO,STMNO,(BLNKS(I),I=1,SSTRT),
 1  364  1243           1        I=STBEG,SEND),(BLNKS(I),I=7,FIL),(SPIN(I),I=73
 1  364  1244           2        ,80)
 1  364  1245     10            FORMAT(1H ,I5,I5,I6,4X,111A1)
 1  364  1246                    FOR I=1 TO 6 DO
 2  365  1247                       BLNKS(I)=BLANK
 2  365  1248                    REPEAT
 1  366  1249                    STBEG=SEND+1
 1  366  1250                 UNTIL .NOT.CVFL REPEAT
 C  367  1251                 RETURN
 C  367  1252              END
```

```
-1  368  1253              SUBROUTINE EXCYL(TYPE)
 0  369  1254                 IMPLICIT INTEGER(A-Z)
 C  369  1255                 LOGICAL ECOND, LBFLG
 0  369  1256                 COMMON/CENTRL/STACK(200),LABEL(200),STERM(6),TOP,LBMAX,
 C  369  1257           1     LBPTR,TPITR,NUMB,STMAX
 C  369  1258                 COMMON/FTRT/CTA(4),FMAT(6),FTYPE,LAST,NLAST
 0  369  1259                 COMMON/INDT/INDFLG,CLRFLG,NELTR,BKWRD,FRWRD,NEW,OLD,PULOUT
 C  369  1260                 COMMON/INPUT/SPIN(80),TOBEG,TOEND,STBEG,STEND,IMAX,BLANK
 0  369  1261                 COMMON/KEYW2/EEEXT(4),CCYCLE(5),COUNTL(5),ITERAT(9),REPT(6),ODF(2)
 C  369  1262                 COMMON/LOD/NUMBER(10),LBFLG,LABC
 C  369  1263                 COMMON/CTPAT/CGOTO(9),CONTI(8),EQL(1),GOTO(6),IFNOT(9),
 0  369  1264           1     LEC(5),MINUS(1),PLL(1),RSTAR(2),ONE(1)
 C  369  1265                 COMMON/STYP3/IIF,CCASE,SUBFN,FFOR,EEXIT,EENDIF,RETRN
 0  369  1266                 LAST=1
 0  369  1267                 IF .NOT.LBFLG THEN
 1  370  1268                    CALL LABFIL(LABC)
 1  370  1269                 ENDIF
 0  371  1270                 K=TOEND
 C  371  1271                 CALL SCAN2(ECOND)
 0  371  1272                 TEMP=TPITR
 C  371  1273                 IF .NOT.ECOND.AND.K.EQ.TOEND THEN
 1  372  1274                    TOEND=K
 1  372  1275                    WHILE SPIN(TOEND+1).EQ.BLANK DO
 2  373  1276                       TOEND=TOEND+1
 2  373  1277                    REPEAT
 1  374  1278                    PTR=0
 1  374  1279                    FOR I=1 TO 9 DO
 2  375  1280                       IF SPIN(TOEND+1).EQ.NUMBER(I+1) THEN
 3  376  1281                          PTR=I
 3  376  1282                       ENDIF
 2  377  1283                    REPEAT
 1  378  1284                    IF PTR.GT.1 THEN
 2  379  1285                       TOEND=TOEND+1
 2  379  1286                       WHILE TEMP.NE.0 DO
 3  380  1287                          TEMP=STACK(TEMP-4)
 3  380  1288                          PTR=PTR-1
 3  380  1289                          IF PTR.EQ.1 THEN
 4  381  1290                             EXIT
```

```
4  381  1291                        ENDIF
3  382  1292                     REPEAT
2  383  1293                     IF PTR.GT.1 THEN
3  384  1294                        IF TYPE.EQ.EEXIT THEN
4  385  1295                           CALL ERROR(9,EEEXT,4,7,EEEXT,4)
4  385  1296                        ELSE
4  386  1297                           CALL ERROR(9,CCYCLE,5,7,CCYCLE,5)
4  386  1298                        ENDIF
3  387  1299                     ENDIF
2  388  1300                  ENDIF
1  389  1301               ENDIF
0  390  1302               IF TYPE.EQ.EEXIT THEN
1  391  1303                  TEMP=TEMP-1
1  391  1304               ELSE
1  392  1305                  TEMP=TEMP-2
1  392  1306               ENDIF
0  393  1307               K=STACK(TEMP)
0  393  1308               CALL FOUT(GOTO,1,6)
0  393  1309               CALL FOUT(LABEL,K,K+4)
0  393  1310               CALL FPRNT
0  393  1311               STEND=TCEND
0  393  1312               INDFLG=NEUTR
0  393  1313               CLRFLG=CLD
0  393  1314               RETURN
0  393  1315            END
```

```
-1   394   1316            SUBROUTINE LABFIL(LAB)
 C   395   1317               IMPLICIT INTEGER(A-Z)
 C   395   1318               COMMON/CONTRL/STACK(200),LABEL(200),STFRM(6),TOP,LBMAX,
 C   395   1319           1   LBPTR,IPITR,NUMB,STMAX
 C   395   1320               COMMON/FRTINF/FBUF(80),FPTR,FMAX,FSTRT,INCR
 C   395   1321               COMMON/INPUT/SPIN(80),TOBEG,TOEND,STBEG,STEND,IMAX,BLANK
 C   395   1322               COMMON/TYP/TYPE
 C   395   1323               COMMON/STYP2/EOJ,RINLO,NCYCLE,WHLE,CONT,COMT,FOTRN,SEMI
 C   395   1324               FOR I=1 TO 5 DO
 1   396   1325                  L=LAB+I-1
 1   396   1326                  FBUF(I)=LABEL(L)
 1   396   1327               REPEAT
 C   397   1328               IF TYPE.NE.CONT THEN
 1   398   1329                  FBUF(6)=BLANK
 1   398   1330               ENDIF
 C   399   1331               RETURN
 C   399   1332            END
```

```
-1  400  1333              INTEGER FUNCTION GENLAB(GEN)
 C  401  1334                 //  GENERATE A LABEL UNCONDITIONALLY IF GEN IS 1 ELSE                //
 C  401  1335                 //  GENERATE THE LABEL IF LBFLG=0, IF LABEL FLAG                     //
 C  401  1336                 //  EQUAL 1 AND GEN EQUAL 0 THEN USE THE EXISTING                    //
 C  401  1337                 //  LABEL. NUMB IS INITIALIZED TO 100000. THIS IS AN                 //
 C  401  1338                 //  INTEGER FUNCTION RETURNING A POINTER TO THE LABEL.               //
 C  401  1339                 IMPLICIT INTEGER(A-Z)
 C  401  1340                 LOGICAL LBFLG
 C  401  1341                 COMMON/CONTRL/STACK(200),LABEL(200),STERM(6),TOP,LBMAX,
 C  401  1342           1     LBPTR,TPTR,NUMB,SMAX
 C  401  1343                 COMMON/INPUT/SPIN(80),TOBEG,TOEND,STBEG,STEND,IMAX,BLANK
 C  401  1344                 COMMON/COD/NUMBER(10),LBFLG,LABO
 C  401  1345                 IF .NOT.LBFLG.OR.GEN.EQ.1 THEN
 1  402  1346                    NUMB=NUMB-1
 1  402  1347                    N=NUMB
 1  402  1348                    FOR I=1 TO 5 DO
 2  403  1349                       M=N/10
 2  403  1350                       //  SHIFT RIGHT  //
 2  403  1351                       NR=N-(10*M)
 2  403  1352                       //  REMAINDER  //
 2  403  1353                       HOLD=5-I+LBPTR
 2  403  1354                       LABEL(HOLD)=CONV(NR)
 2  403  1355                       //  LBPTR IS LEFT AT FIRST BLANK SPACE.  //
 2  403  1356                       N=M
 2  403  1357                    REPEAT
 1  404  1358                    GENLAB=LBPTR
 1  404  1359                    //  NEW LABEL PTR.  //
 1  404  1360                    LBPTR=LBPTR+5
 1  404  1361                    //  POINT TO NEW BLANK LOCATION.  //
 1  404  1362                    IF LBPTR.GT.LBMAX THEN
 2  405  1363                       CALL ERROR(8,0,0,5,0,0)
 2  405  1364                       //  NESTING TOO DEEP, TOO MANY LABELS PROGRAM TERMINATED. //
 2  405  1365                    ENDIF
 1  406  1366                 ELSE
 1  407  1367                    FOR I=1 TO 5 DO
 2  408  1368                       HOLD=LBPTR+I-1
 2  408  1369                       LABEL(HOLD)=SPIN(I)
 2  408  1370                    REPEAT
```

```
1  409  1371                GENLAB=LBPTR
1  409  1372                LBPTR=LBPTR+5
1  409  1373                IF LBPTR.GT.LEMAX THEN
2  410  1374                   CALL ERROR(8,0,0,5,0,0)
2  410  1375                ENDIF
1  411  1376                //  NESTING TOO DEEP, TOO MANY LABELS PROGRAM TERMINATED.      //
1  411  1377             ENDIF
0  412  1378             RETURN
0  412  1379          END
```

```
-1  413  1380              SUBROUTINE CHECK(STYPE,ECOND)
 C  414  1381                 IMPLICIT INTEGER(A-Z)
 0  414  1382                 LOGICAL ECOND,DUMM,LBFLG
 C  414  1383                 COMMON/CENTRL/STACK(200),LABEL(200),STERM(6),TOP,LBMAX,
 0  414  1384           1     LBPTR,TPTR,NUMB,STMAX
 0  414  1385                 COMMON/FRTINF/FBLE(80),FPTR,FMAX,FSTRT,INCR
 C  414  1386                 COMMON/INPAT/TCC(2),THENN(4),BYE(2),DCO(2),ELZ(4),COLN
 0  414  1387                 COMMON/INPUT/SPIN(80),TCBEG,TCEND,STBEG,STEND,IMAX,BLANK
 C  414  1388                 COMMON/KEYW1/EEND(3),ENCASS(7),ENIF(5),CASS(4),IIIF(2),CCOLON(5)
 C  414  1389                 COMMON/KEYW2/EEXT(4),CCYCLE(5),UUNTL(5),ITERAT(9),REPT(6),DDF(2)
 0  414  1390                 COMMON/SPFINF/SSTRT,STMNC,BLKNC,NESTNC,SMAX,BLNKS(131)
 C  414  1391                 COMMON/STYP1/CCLON,DELSE,EELSE,UNTL,LLOOP,EEND,ENDCAS,RREPT
 0  414  1392                 COMMON/STYP2/ECO,RINLC,NCYCLE,WHLE,CONT,COMT,FCTRN,SEMI
 C  414  1393                 COMMON/STYP3/IIF,CCASE,SUBFN,FFOR,EEXIT,EENDIF,RETRN
 C  414  1394                 ECOND=.FALSE.
 C  414  1395                 IF TOP.EQ.C THEN
 1  415  1396                    CALL ERROR(4,C,C,3,0,0)
 1  415  1397                    // ERROR---STACK EMPTY, STATEMENT IGNORED. //
 1  415  1398                    ECOND=.TRUE.
 1  415  1399                    RETURN
 1  415  1400                 ENDIF
 0  416  1401                 STTOP=STACK(TOP)
 C  416  1402                 CASE
 1  417  1403                 : STYPE.EQ.CCLON :
 1  418  1404                    IF STTOP.NE.CCASE THEN
 2  419  1405                       CALL ENDFIX(CCASE,STTOP,STYPE,DUMM)
 2  419  1406                       CALL ERROR(2,CASS,4,4,CCLON,1)
 2  419  1407                       // ERROR---MISSING CASE STATEMENT, COLON IGNORED. //
 2  419  1408                       TCEND=TCEND+1
 2  419  1409                       ECOND=.TRUE.
 2  419  1410                    ENDIF
 1  420  1411                 : STYPE.EQ.EELSE :
 1  421  1412                    IF STTOP.NE.IIF THEN
 2  422  1413                       CALL ERROR(2,IIF,2,4,ELZ,4)
 2  422  1414                       // ERROR---MISSING IF STATEMENT, ELSE IGNORED. //
 2  422  1415                       ECOND=.TRUE.
 2  422  1416                    ENDIF
 1  423  1417                 : STYPE.EQ.UNTL :
```

```
1  424  1418                IF STTOP.NE.LLOOP THEN
2  425  1419                   CALL ERROR(5,ITERAT,9,4,UNTIL,5)
2  425  1420                   // ERROR---MISSING LOOP STATEMENT, UNTIL IGNORED. //
2  425  1421                   CALL SCAN(REPT,3,DUMM)
2  425  1422                   IF DUMM THEN
3  426  1423                      TOEND=TOEND+1
3  426  1424                   ELSE
3  427  1425                      TOEND=TOEND+6
3  427  1426                   ENDIF
2  428  1427                   ECOND=.TRUE.
2  428  1428                ENDIF
1  429  1429             : STYPE.EQ.EEND :
1  430  1430                IF STTOP.NE.SUBFN THEN
2  431  1431                   // UNSTACK UNTIL A SUBFN IS FOUND ON THE STACK. //
2  431  1432                   CALL ENDFIX(SUBFN,STTOP,STYPE,DUMM)
2  431  1433                   IF .NOT.DUMM THEN
3  432  1434                      CALL ERROR(3,EEEND,3,4,EEND,3)
3  432  1435                      // ERROR---TOO MANY ENDS, END IGNORED. //
3  432  1436                      ECOND=.TRUE.
3  432  1437                   ELSE
3  433  1438                      CALL ERROR(7,0,0,6,0,0)
3  433  1439                      // PREMATURE END SUBPROGRAM CLOSED. //
3  433  1440                   ENDIF
2  434  1441                   FSTRT=10
2  434  1442                   SSTRT=10
2  434  1443                ENDIF
1  435  1444             : STYPE.EQ.ENDCAS :
1  436  1445                IF STTOP.NE.CCASE THEN
2  437  1446                   CALL ENDFIX(CCASE,STTOP,STYPE,DUMM)
2  437  1447                   CALL ERROR(3,ENDCASS,7,4,ENDCASS,7)
2  437  1448                   // ERROR---TOO MANY ENDCASES, ENDCASE IGNORED. //
2  437  1449                   ECOND=.TRUE.
2  437  1450                ENDIF
1  438  1451             : STYPE.EQ.EENDIF :
1  439  1452                IF STTOP.NE.IIF THEN
2  440  1453                   CALL ERROR(3,ENIF,5,4,ENIF,5)
2  440  1454                   // ERROR---TOO MANY ENDIFS, ENDIF IGNORED. //
2  440  1455                   ECOND=.TRUE.
```

```
2  440  1456                ENDIF
1  441  1457              : STYPE.EQ.RREPT :
1  442  1458                IF STTOP.EQ.FFOR.OR.STTOP.EQ.LLOOP.OR.STTOP.EQ.WHLE THEN
2  443  1459                   RETURN
2  443  1460                ELSE
2  444  1461                   CALL ERROR(5,ITERAT,9,4,REPT,6)
2  444  1462                   // ERROR---NO ITERATIVE STATEMENT FOUND, REPEAT IGNORED. //
2  444  1463                   ECOND=.TRUE.
2  444  1464                ENDIF
1  445  1465              : STYPE.EQ.EEXIT :
1  446  1466                IF IPITR.EQ.0 THEN
2  447  1467                   CALL ERROR(5,ITERAT,9,4,EEXT,4)
2  447  1468                   // ERROR---NO ITERATIVE STATEMENT, EXIT IGNORED. //
2  447  1469                   ECOND=.TRUE.
2  447  1470                ENDIF
1  448  1471              : STYPE.EQ.NCYCLE :
1  449  1472                IF IPITR.EQ.0 THEN
2  450  1473                   CALL ERROR(5,ITERAT,9,4,CCYCLE,4)
2  450  1474                   // ERROR---NO ITERATIVE STATEMENT, CYCLE IGNORED. //
2  450  1475                   ECOND=.TRUE.
2  450  1476                ENDIF
1  451  1477             ENDCASE
C  452  1478             RETURN
0  452  1479          END
```

```
-1  453  1480            SUBROUTINE ENDFIX(ETYP,STTOP,STYPE,DUMM)
 C  454  1481               IMPLICIT INTEGER(A-Z)
 C  454  1482               LOGICAL DUMM
 C  454  1483               COMMON/CONTRL/STACK(200),LABEL(200),STERM(6),TOP,LBMAX,
 C  454  1484         1     LBPTR,TPITR,NUMB,STMAX
 C  454  1485               COMMON/FRTINF/FBUF(80),FPTR,FMAX,FSTRT,INCR
 C  454  1486               COMMON/INPUT/SPIN(80),TOBEG,TOEND,STBEG,STEND,IMAX,BLANK
 C  454  1487               COMMON/KEYWI/EEEND(3),ENCASS(7),ENIF(5),CASS(4),IIIF(2),CCOLON(5)
 C  454  1488               COMMON/CDD/NUMBER(10),LBFLG,LABO
 C  454  1489               COMMON/CTPAT/CGOTO(9),CONTI(8),EQL(1),GOTO(6),IFNOT(9),
 C  454  1490         1     LEO(5),MINUS(1),PLL(1),RSTAR(2),ONE(1)
 C  454  1491               COMMON/STYPB/IIF,CCASE,SUBFN,FFOR,EEXIT,EENDIF,RETRN
 C  454  1492               DUMM=.FALSE.
 C  454  1493               WHILE STTOP.NE.ETYP DO
 1  455  1494                  CASE
 2  456  1495                  : STTOP.EQ.IIF :
 2  457  1496                     LAB1=STACK(TOP-2)
 2  457  1497                     CALL LABFIL(LAB1)
 2  457  1498                     CALL FOUT(CONTI,1,8)
 2  457  1499                     CALL FPRNT
 2  457  1500                     CALL ERROR(2,ENIF,5,1,0,0)
 2  457  1501                  : STTOP.EQ.CCASE :
 2  458  1502                     LAB1=STACK(TOP-1)
 2  458  1503                     LAB2=STACK(TOP-2)
 2  458  1504                     CALL LABFIL(LAB2)
 2  458  1505                     CALL FOUT(CONTI,1,8)
 2  458  1506                     CALL FPRNT
 2  458  1507                     CALL LABFIL(LAB1)
 2  458  1508                     CALL FOUT(CONTI,1,8)
 2  458  1509                     CALL FPRNT
 2  458  1510                     CALL ERROR(2,ENCASS,7,1,0,0)
 2  458  1511                  : STTOP.EQ.SUBFN :
 2  459  1512                     IF STYPE.EQ.SUBFN THEN
 3  460  1513                        CALL LABFIL(LABO)
 3  460  1514                        CALL FOUT(EEEND,1,3)
 3  460  1515                        CALL FPRNT
 3  460  1516                     ENDIF
 2  461  1517                  : ELSE :
```

```
2  462  1518                    IF STTCP.EQ.FFCP THEN
3  463  1519                       LAB1=STACK(TCP-5)
3  463  1520                    ELSE
3  464  1521                       LAB1=STACK(TCP-3)
3  464  1522                    ENDIF
2  465  1523                    LAB2=STACK(TCP-2)
2  465  1524                    LAB3=STACK(TCP-1)
2  465  1525                    CALL LABFIL(LAB2)
2  465  1526                    CALL FCLT(GCTC,1,6)
2  465  1527                    CALL FCLT(LABEL,LAB1,LAB1+4)
2  465  1528                    CALL FPRNT
2  465  1529                    CALL LABFIL(LAB3)
2  465  1530                    CALL FCLT(CONTI,1,8)
2  465  1531                    CALL FPRNT
2  465  1532                 ENDCASE
1  466  1533                 TCP=TOP-STERM(STTCP)
1  466  1534                 IF TCP.EQ.0 THEN
2  467  1535                    RETURN
2  467  1536                 ENDIF
1  468  1537                 STTCP=STACK(TCP)
1  468  1538              REPEAT
C  469  1539              DUMM=.TRUE.
C  469  1540              RETURN
C  469  1541           END
```

```
-1  470  1542            SUBROUTINE ERROR(CODE1,PARM1,LEN1,CODE2,PARM2,LEN2)
 C  471  1543               //  CODE1 IS ERROR MESSAGE CODE                                  //
 C  471  1544               //  CODE2 IS THE ACTION TAKEN                                    //
 C  471  1545               //  PARM1 AND PARM2 MODIFY THE TWO                               //
 C  471  1546               //  LEN1 AND LEN2 ARE THE LENGTH OF MODIFYING PATTERN            //
 C  471  1547               IMPLICIT INTEGER(A-Z)
 C  471  1548               COMMON/DEVIC/PRTR,PFILE,CRDR
 C  471  1549               COMMON/INPUT/SPIN(80),TCBEG,TCEND,STBEG,STEND,IMAX,BLANK
 C  471  1550               DIMENSION PARM1(1),PARM2(1),ERPAT(45,9),POSN(9),LEN(9),
 C  471  1551         1     ACPAT(35,7),ALEN(7),APSN(7)
 C  471  1552               DATA ERPAT/1HL,1HA,1HB,1HE,1HL,1HL,1HI,1HN,1HG,1H ,1HT,1HH,1HE,
 C  471  1553         1     1H ,1H ,1H ,1H ,1H ,1H ,1H ,1HF,1HA,1HM,1HP,1HE,1HR,1HS,
 C  471  1554         1     1HP,1HR,1HC,1HG,1HR,1HA,1HM,1H ,1HC,1HL,1HA,1HR,1HI,1HT,1HY,1H.,
 C  471  1555         1     1H ,1H ,
 C  471  1556         2     1HM,1HI,1HS,1HS,1HI,1HN,1HG,1H ,1H ,1H ,1H ,1H ,1H ,1H ,1H ,
 C  471  1557         2     1H ,1H ,1H ,1H ,1H ,1H ,1H ,1H ,1H ,1H ,1H ,1H ,1H ,17*1H ,
 C  471  1558         3     1HC,1HA,1HN,1HG,1HL,1HI,1HN,1HG,1H ,1H ,1H ,1H ,1H ,1H ,1H ,
 C  471  1559         3     1H ,1H ,1H ,1H ,1H ,1H ,1H ,1H ,1H ,1H ,1H ,1H ,1H ,17*1H ,
 C  471  1560         4     1HS,1HT,1HA,1HC,1HK,1H ,1HE,1HM,1HP,1HT,1HY,1H.,1H ,1H ,1H ,
 C  471  1561         4     1H ,1H ,1H ,1H ,1H ,1H ,1H ,1H ,1H ,1H ,1H ,1H ,1H ,17*1H ,
 C  471  1562         5     1HN,1HO,1H ,1HC,1HC,1HR,1HR,1HE,1HS,1HP,1HC,1HN,1HD,1HI,1HN,1HG,
 C  471  1563         5     1H ,1H ,1H ,1H ,1H ,1H ,1H ,1H ,1H ,1H ,1H ,1HS,1HT,1HM,1HT,1H.,
 C  471  1564         5     1H ,1H ,1H ,1H ,1H ,1H ,1H ,1H ,1H ,1H ,1H ,1H ,1H ,
 C  471  1565         6     1HS,1HT,1HA,1HC,1HK,1H ,1HF,1HC,1HL,1HL,
 C  471  1566         7     35*1H ,1HP,1HR,1HE,1HM,1HA,1HT,1HU,1HR,1HE,1H ,1HE,1HN,1HD,
 C  471  1567         8     32*1H ,1HT,1HC,1HG,1H ,1HM,1HA,1HN,1HY,1H ,1HL,1HA,1HB,1HE,1HL,
 C  471  1568         8     1HS,30*1H ,
 C  471  1569         9     1H ,1H ,1H ,1H ,1H ,1H ,1HL,1HE,1HV,1HE,1HL,1H ,1HE,1HX,1HC,
 C  471  1570         9     1HE,1HE,1HD,1HS,1H ,1HN,1HE,1HS,1HT,1HI,1HN,1HG,1H ,1HD,1HE,
 C  471  1571         9     1HP,1HT,1HF,1H.,11*1H /
 C  471  1572               DATA LEN/5,6,6,0,9,0,0,0,6/
 C  471  1573               DATA POSN/15,9,10,0,18,0,0,0,1/
 C  471  1574               DATA ACPAT/1HS,1HU,1HP,1HP,1HL,1HI,1HE,1HD,1H ,1HA,1HT,
 C  471  1575         1     1HE,1HN,1HD,1H ,1HC,1HF,1H ,1HS,1HT,1HM,1HT,1H.,
 C  471  1576         1     1H ,1H ,1H ,1H ,1H ,1H ,1H ,1H ,1H ,1H ,1H ,1H ,
 C  471  1577         2     1HL,1HA,1HC,1HE,1HL,1H ,1HD,1HE,1HL,1HE,1HT,1HE,1HD,
 C  471  1578         2     1HF,1HR,1HC,1HM,1H ,1HF,1HC,1HR,1HT,1HR,1HA,1HN,1H ,
 C  471  1579         2     1HC,1HO,1HT,1HP,1HO,1HT,1H.,1H ,1H ,
```

```
C  471  1580          3     1HS,1HT,1HA,1HT,1HE,1HM,1HE,1HN,1HT,1H ,
0  471  1581          3     1HS,1HK,1HI,1HP,1HP,1HE,1HD,1H.,1H ,1H ,1H ,1H ,
C  471  1582          3     1H ,1H ,1H ,1H ,1H ,1H ,1H ,1H ,1H ,1H ,1H ,1H ,
0  471  1583          4     1H ,1H ,1H ,1H ,1H ,1H ,1H ,1H1,1HG,1HN,1HC,1HR,1HE,1HD,
C  471  1584          4     1H ,1H ,1H ,1H ,1H ,1H ,1H ,1H ,1H ,1H ,1H ,1H ,1H ,1H ,
C  471  1585          4     1H ,1H ,1H ,1H ,1H ,1H ,
0  471  1586          5     1HP,1HR,1HO,1HG,1HR,1HA,1HM,1H ,1HT,1HE,1HR,1HM,1HI,1HN,1HA,
C  471  1587          5     1HT,1HE,1HD,17*1H ,
0  471  1588          6     1HS,1HC,1HB,1HP,1HR,1HC,1HG,1HR,1HA,1HM,1H ,1HC,1HL,1HO,1HS,
0  471  1589          6     1HE,1HD,18*1H ,
C  471  1590          7     1H ,1H ,1H ,1H ,1H ,1H ,1HC,1HO,1HM,1HP,1HL,1HE,1HT,1HE,1H ,
0  471  1591          7     1HT,1HO,1H ,1HC,1HL,1HT,1HE,1HR,1H ,1HM,1HC,1HS,1HT,1H ,1HL,
C  471  1592          7     1HE,1HV,1HE,1HL,1/
C  471  1593                DATA ALEN/6,6,0,6,6,0,6/
0  471  1594                DATA APSN/0,6,6,1,6,6,1/
C  471  1595                WRITE(PRTR,2222) SPIN
0  471  1596     2222       FORMAT(26H    ++++++OFFENDING CARD: ,80A1)
C  471  1597                LENTH=LEN(CODE1)
C  471  1598                ALNTH=ALEN(CODE2)
0  471  1599                POS=POSN(CODE1)
C  471  1600                APOS=APSN(CODE2)
0  471  1601                FOR I=1 TO LEN1 DO
1  472  1602                   IT=POS+I-1
1  472  1603                   ERRAT(IT,CODE1)=PARM1(I)
1  472  1604                REPEAT
C  473  1605                IF LENTH.GT.LEN1 THEN
1  474  1606                   PC=POS+LEN1
1  474  1607                   PE=POS+LENTH-1
1  474  1608                   FOR I=PC TO PE DO
2  475  1609                      ERRAT(I,CODE1)=BLANK
2  475  1610                   REPEAT
1  476  1611                ENDIF
0  477  1612                FOR I=1 TO LEN2 DO
1  478  1613                   IT=APOS+I-1
1  478  1614                   ACRAT(IT,CODE2)=PARM2(I)
1  478  1615                REPEAT
C  479  1616                IF ALENTH.GT.LEN2 THEN
1  480  1617                   PC=APOS+LEN2
```

```
1   480   1618                  PE=APOS+ALNTH-1
1   480   1619                  FOR I=PC TO PE DO
2   481   1620                     ACPAT(I,CODE2)=BLANK
2   481   1621                  REPEAT
1   482   1622               ENDIF
C   483   1623               WRITE(PRTR,10)    (ERPAT(I,CODE1),I=1,45)
C   483   1624               WRITE (PRTR,10)      (ACPAT(I,CODE2),I=1,35)
C   483   1625         10    FORMAT(12H      +++++++,80A1)
C   483   1626               RETURN
C   483   1627            END
```

```
-1  484  1628            INTEGER FUNCTION CONV(DIGIT)
 0  485  1629               // CONVERTS THE DIGIT INTO ITS CHARACTER REPRESENTATION.        //
 0  485  1630               IMPLICIT INTEGER(A-Z)
 0  485  1631               COMMON/COD/NUMBER(10),LBFLG,LABO
 0  485  1632               CONV=NUMBER(DIGIT+1)
 0  485  1633               RETURN
 0  485  1634            END

     END OF SPARKS PROGRAM.....
```

APPENDIX D

REFERENCES

REFERENCES

[1] Horowitz, E. and Sahni, S. Fundamentals of Data Structures. Woodland Hills, Califorina: Computer Science Press, 1976.

[2] Ralston, A. and Wagener, J. 'Structured FORTRAN An Evolution of Standard FORTRAN'. IEEE Transactions on Software Engineering, 3,Sep 1976, Vol. SE - 2, No. 3, pp. 154-176.

[3] Stroud, R. 'Translation of the SPARKS Preprocessor from FORTRAN to SPARKS'. Master Report Kansas State University, 1978.

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

FORTRAN 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 FORTRAN 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 reports includes a User's Manual, SPARKS code, and specifications to support all modifications made to the original program.