1076

USER'S GUIDE FOR THE RATIONAL FORTRAN PREPROCESSOR SOFTWARE PACKAGE

by

BENZELL FLOYD

B.S., University of Southern Mississippi, 1974

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:

Major Professor

Document
LD
2668
.R4
1978
F62
c. 2

ACKNOWLEDGEMENTS

This report describes the implementation and operation procedures necessary to execute the Rational Fortran preprocessor and RatFor programs in the IBM 370's CMS environment. For their technical support in preparation of this report, I would like tc thank the Computing Center consultants. For his patience and time in providing guidance and support to me during the preparation of this report, I would like to thank Doctor William J. Hankley.

ILLEGIBLE DOCUMENT

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

THIS IS THE BEST COPY AVAILABLE

THIS BOOK CONTAINS NUMEROUS PAGE NUMBERS THAT ARE ILLEGIBLE

THIS IS AS RECEIVED FROM THE CUSTOMER

TABLE OF CONTENTS

LIST OF FIGURES

Figure

CHAPTER 1

INTRODUCTION

1.1 Purpose.

The purpose of this guide is:

- to provide the user with the procedures necessary to implement and to execute the Rational Fortran (RatFor) preprocessor and RatFor programs in the IBM 370's CMS environment.

- to present the general format of each RatFor statement and examples of its use.

- to provide information about changes made to the preprocessor so that it executes properly in Kansas State University's IBM 370's CMS environment.

- to provide documentation of the preprocessor by using module access graphs and a module access matrix.

1.2 Explanation of the RatFor Software Package.

The RatFor software package is one of many software tools used to teach Software Engineering Courses. The package will aid the user who is learning how to write programs that make good tools and how to program well in the process. The package contains a RatFor preprocessor and most of the software tools that are outlined and presented in the book, Software Tools, by Brian W. Kernighan and P.J. Plauger.

The RatFor preprocessor is a preprocessor which allows for a simple extension of the Fortran computer language.

The extension, Rational Fortran, provides modern flow control statements like those in PL/I, COBOL, and ALGOL so that structured programming can be achieved properly. Preprocessing into Fortran is a convenient way to achieve structured programming. By preprocessing into Fortran, the user retains the advantages of Fortran, i.e. a language that is universal, portable, and relatively efficient, while at the same time concealing its worst drawback, unstructured code. Except for a few new statements like while, repeat-until, if-else, and the fact that it's easy to read and easy to write, RatFor is Fortran. The preprocessor simply translates each of the RatFor statements into equivalent Fortran code.

The other software tools in the package are programs to help the user develop other programs. They are tools that are generally used during the life cycle of a program. Included in the package are Ratfor pattern finders, a RatFor text editor, a RatFor text formatter, and many other useful software tools. All of these tools are directed toward mechanizing as much as possible those activities that are at the heart of programming. Program development is the place where these tools will have the most impact.

1.3 Background.

The RatFor preprocessor software package was developed by Brian W. Kernighan and P.J. Plauger. The package was

developed to enhance and demonstrate software engineering teaching points presented in their book, Software Tools. Their book and package present a comprehensive set of programs which provides lessons in program design and implementation.

The RatFor package and Software Tools book was obtained from the Addison-Wesley Publishing Company in Reading, Massachusetts. Addison-Wesley Publishing Company distributes the software package by way of a 9-track non-labeled magnetic tape having the following characteristics:

Density - 800 bits per inch

Record length - 80 bytes

Number of blocks - 888

Block size - 10 records per block

Number of files - 1

Character set - EBCDIC (using the mapping described in IBM 370 Principles of Operation Manual, GA22-7000-4 and Reference Summary GX20-1850-2.)

Both upper and lower case alphabetic characters are used in the package. Questions about the package that are not covered in this guide or the book, Software Tools, should be address to Addison-Wesley Publishing Company. Fortran used in the package is standard (ANSI) Fortran.

CHAPTER 2

USER'S GUIDE

2.1 RatFor Programming.

This chapter serves as an introduction to the RatFor language. Additionally, it acquaints the user with the appropriate IBM 370 CMS commands that are necessary to perform certain operations. The information pertaining to the RatFor statements and symbols is a consolidation of information extracted from chapters 1, 3, 8, and 9 of the book, Software Tools, from the RatFor version of the preprocessor listing, and from the Fortran preprocessor listing. The procedures outlined in this chapter have been tested and they work. The user is cautioned against any deviation from the outlined procedures as the outcome is unpredictable.

2.1.1 RatFor Statements Formats.

Basically, RatFor is Fortran, except that several new statements have been added to make it easier to control the program flow. RatFor also provides some notational conveniences so programs can be made more readable. Any statement that is encountered by the RatFor preprocessor and is not recognized as being a RatFor statement is considered to be a regular Fortran statement. All regular Fortran statements are simply passed through the RatFor preprocessor untouched except for blanks being removed. In the following

syntax graphs, a statement is any legal Fortran statement or RatFor statement. A group of RatFor and/or Fortran statements can be enclosed in "$(", "$)", which are RatFor notation for brackets, to make the group a compound statement which is equivalent to a single statement and usable anywhere a single statement can be used. RatFor source statements may begin anywhere on a line. The preprocessor scans all 80 columns for source statements. Therefore, columns 73 - 80 should not be used for line numbers, comments, etc. The 80 column scanning can be changed by adjusting the read statement in the preprocessor subroutine, Getch. Multiple statements on a line must be separated by a semicolon. The user must indent systematically so he or she can easily see the scope of control statements. A statement ending with a comma is automatically continued. A statement that contains a "$(" [left bracket], is automatically continued until the "$)" [right bracket], is encountered. Automatic continuation also occurs for statements containing a "(" until a ")" is encountered. Consequently, the regular Fortran convention of statement continuation is not needed nor allowed.

The RatFor statements keywords are: while, repeat, until, for, if, else, do, break, next, define, and include. In RatFor these keywords are reserved words and must not be used as variable names. Keywords cannot contain blanks or they will not be recognized. The following syntax graphs

and examples serve to present the formats of the RatFor statements. All keywcrds are underlined.

2.1.1.1 The while statement: Designed to produce repetitive operations.

General format:

--> while --> (Boolean condition) --> statement -->

Procedure: Test Boolean condition; if true, do statement cnce then test again. If the Boclean condition is ever false, go to the first executable statement following the body of the while.

Example: This segment of code produces the sum of 10 numbers.

```
total = 0
while(total '= 10.)$(
     sum = sum + number
     total = total + 1
     $)
stop
end
```

Execution stops when the Boolean condition becomes false. Ncte the "¦" is an alternative symbol for the logical, "not".

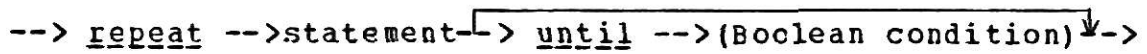
Translation:

```
      total = 0
      continue
23000 if(.not.(total.ne.10.))go to 23001
      sum = sum + number
      total = total + 1
      go to 23000
23001 continue
      stop
      end
```

If the while statement has a label then during translation that label is placed on the continue statement that precedes the if statement. Labels 23000 and 23001 are supplied by the preprocessor. Preprocessor supplied labels begin arbitrarily at 23000 and are numbered upward consecutively.

2.1.1.2 The repeat-until statement: Also designed to produce repetitive operation. However, the main difference between the repeat-until statement and the while statement is the point where the Boolean condition is tested. The Boolean condition is at the end of the repeat-until loop and at the beginning of the while loop.

General format:



Procedure: The statement is executed one or more times until the Boolean condition becomes true, at which time the loop is exited. The until part is optional. However, if it is

cmitted, the result is an infinite loop, which must be broken some other way.

Example: This segment of code sums the variable A and the square root of the variable A until the variable Count is greater than 5.

```
Count = 0
repeat $(
     A = A + sqrt(A)
     Count = Count + 1
     $)
until ( Count > 5 )
```

Translation:

```
      count = 0
      continue
23000 continue
      a = a + sqrt(a)
      count = count + 1
23001 if(.not.(count.gt.5))go to 23000
23002 continue
```

Another form of the repeat-until is the form excluding the until clause.

Example:

```
Count = 0
repeat $(
     A = A + sqrt(A)
     Count = Count + 1
     $)
continue
---
---
```

Translation:

```
      continue
23000 continue
      a = a + sqrt(a)
      count = count + 1
      continue
      go to 23000
      continue
      ---
      ---
```

This second form of the repeat statement represents an infinite loop and executes theoretically forever. The break statement could be placed at a likely breaking point within the loop to cause an exit out of the loop.

2.1.1.3 The break statement: Designed to enable one to get out of an infinite loop.

General format:

--> break -->

Procedure: It causes whatever loop, i.e., repeat, while, for,etc., it is contained in to be exited immediately. Only one loop is terminated by a break statement, even if it is contained inside several nested loops. Program control resumes with the statement following the loop which is exited.

Example: Using the break statement, the previously shown infinite loop can be exited.

```
Count = 0
repeat $(
     A = A + sqrt(A)
     Count = Count + 1
       if( Ccunt > 5 )
          break
     $)
continue
---
---
```

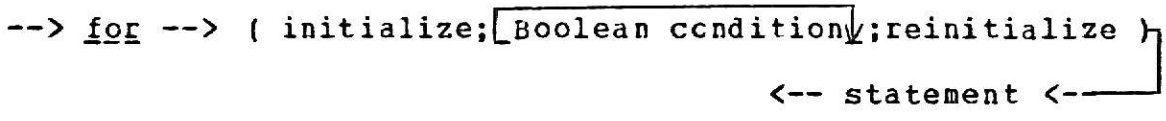
Translation:

```
      count = 0
      continue
23000 continue
      a = a + sqrt(a)
      count = count + 1
      if(.not.(count.gt.5))go to 23003
      go to 23002
23003 continue
23001 go to 23000
23002 continue
      continue
```

The break statement causes an exit out of the loop and program execution resumes with the continue statement. In this example, the break statement translates to "go to 23002".

2.1.1.4 The for statement: Designed to produce repetitive operations that are based on the iterations of a variable.

General format:



Procedure: Initialize and reinitialize parts are single Fortran statements, i.e., J = 1 and J = J + 1, respectively. The value, true or false, of the Boolean condition is dependent upon the value of the initialize and reinitialize variable. The statement is executed as long as the Boolean condition remains true. If the condition is false, program execution resumes at the first executable statement following the for statement. If the Boolean condition is omitted, it is taken to be true, resulting in an infinite loop which must be broken in some other way.

Example: This segment of code prints the value of the variable inum until the variable i is greater than number.

```
number = 10
for ( i = 1; i < number; i = i + 1 )
     $(
     inum = number - i
     print 9, inum
     next
     $)
```

Translation:

```
      number = 10
      i = 1
23000 if(.not.(i.lt.number))go to 23002
      inum = number - i
      print 9, inum
      go to 23001
23001 i = i + 1
      go to 23000
23002 continue
```

This for statement executes as long as the Boolean condition

"i < number" is true. The next statement causes the variable, i, to be incremented. The Boolean condition can be left out.

Example:

```
number = 10
for ( i = 1; ; i = i + 1 )
      $(
      inum = number - i
      print 9, inum
      next
      $)
```

Translation:

```
      number = 10
      i = 1
23000 continue
      inum = number - i
      print 9, inum
23001 i = i + 1
      go to 23000
```

A <u>for</u> that does not have a Boolean condition produces an infinite loop. Note that although the Boolean condition is missing, the semicolon that follows the Boolean condition must appear in the <u>for</u> statement. The <u>break</u> statement can be used to cause an exit out of an infinite <u>for</u> loop.

Example:

```
number = 10
for ( i = 1; ; i = i + 1 )
     $(
     if ( i >= number ) break
     inum = number - i
     print 9, inum
     next
     $)
```

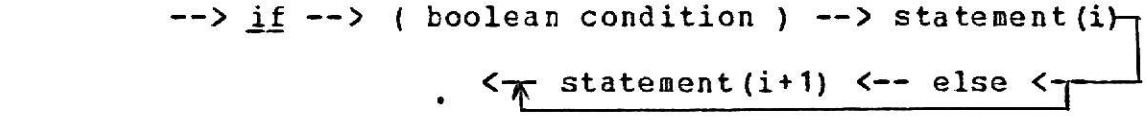
Translation:

```
      number = 10
      continue
      i = 1
23000 continue
      if(.not.(i.ge.number))go to 23003
      go to 23002
23003 continue
      inum = number - i
      print 9, inum
      go to 23001
23001 i = i + 1
      go to 23000
23002 continue
```

When the Boolean condition, i >= number, becomes true, the break statement is executed and the loop is exited.

2.1.1.5 The if statement: Designed for conditional control of program flow.

General format:



Procedure: If the Boolean condition is true, statement(i) is executed. If the Boolean condition is false, statement(i+1)

is executed. The else clause is optional. In the case where there are nested if's, the else always goes with the most recent un-elsed if.

Example: This segment of code reads a file until the end of the file is reached.

```
      read 9, input
      if ( input != eof )
           print 10, input
      else
         print 20
          20 format ( ' end of data' )
continue
```

Translation:

```
       read 9, input
       if(.not.(input.ne.eof))go to 23000
       print 10, input
       go to 23001
23000  continue
       print 20
   20  format( ' end of data' )
23001  continue
       continue
```

The execution of the if or else part of the if-else statement is dependent upon the logical value of the Boolean condition, "input '= eof" . As stated previously, the else clause of the if statement can be excluded.

Example:

```
      read 9, input
       if ( input == eof )
           $(
            print 20
```

```
          20 format (' end of data' )
        break
        $)
    print 10, input
```

Translation:

```
        read 9, input
        if(.not.(input .eq.ecf))go to 23000
        print 20
     20 format( ' end of data' )
        go to 23001
23000 continue
        print 10, input
23001 continue
```

In this previous if statement example, the Boolean condition is changed to compensate for the excluded else clause. In this example, there are no logical inconveniences caused by the exclusion of the else clause. In other situations there may be logical inconveniences. Therefore, the user must select the form of the if statement that is compatible with his program's logic.

2.1.1.6 The do statement: Designed to set up standard Fortran do loops.

General format:

--> do --> limits --> statement -->

Procedure: The limits must be a legal Fortran do statement specification, i.e., "i = 1,n". Using the limits, the

RatFor preprocessor will build a legal Fortran do loop around the statement. Do not attempt to form a regular Fortran do loop in a RatFor program. The RatFor preprocessor preprocesses all dc statements and if the do statements are not in the RatFor form then the preprocessed output is erroneous.

Example: This segment of code initializes an array.

```
dimension ibuf(80)
do k = 1,80
     ibuf(k) = 0
call grader( ibuf)
---
---
```

Translation:

```
      dimension ibuf(80)
      do 23000 k = 1,80
      ibuf(k) = 0
23000 continue
23001 continue
      call grader( ibuf)
      ---
      ---
```

The second continue statement is produced in case the loop contains a break statement. If the statement is a compound statement (a group of statements enclosed in brackets or braces) then a standard Fortran do loop is built around the group of statements.

2.1.1.7 The next statement: Designed to cause whatever loop that contains it to go immediately to the next iteration, skipping the rest of the loop body.

General format:

--> next -->

Procedure: The execution of a next statement causes a jump to the conditional part of a while, do, or until statement; causes a jump to the top of an infinite repeat statement loop; and causes a jump to the reinitialize part of a for statement.

Example: This segment of code determines an average grade point.

```
grade = 0
studen = 30
for ( i = 1; i <= studen; i = i + 1 )
     $(
     grade = grade + score(i)
     next
     $)
aver = grade / studen
```

Translation:

```
      grade = 0
      studen = 30
      continue
      i = 1
23000 if(.not.(i.le.studen))go to 23002
      grade = grade + score(i)
      go to 23001
23001 i = i + 1
      go to 23000
23002 continue
      aver = grade / studen
```

After each grade is added, the next is executed. Execution cf the next statement causes iterations of the for loop.

2.1.1.8 The define statement: Designed to allow one to place symbolic constants in a program. Symbolic constants are readable names placed on certain values.

General format:

--> define --> (symbolic constant name, value) -->

Procedure: The first character of the symbolic constant name must be alphabetic. The value can be any integer value. The define statement must be placed in a position where it will be encountered by the preprocessor before the symbolic constant that it defines is encountered.

Example: This example allows 80 card columns to be read.

```
define( MAXCARD, 80 )
read(9,1)(ibuf(i), i = 1, MAXCARD )
   1 format( MAXCARD a1 )
---
---
```

There should only be one define statement for each symbolic constant used. During preprocessor execution, the value 80 is inserted wherever the symbolic constant "MAXCARD" is encountered. For example, the preprocessor output of the format statement would be, "1 format(80a1)", not that which appears in the example

2.1.1.9 The include statement: Designed to allow replacement of the include statement with the contents of an external Fortran file.

General format:

--> include --> filename -->

Procedure: Filename should be less than or equal to 8 alphanumeric characters. The first character must be alphabetic.

Example: This segment of code demonstrates the inclusion of common blocks.

```
dimension name(40), ssn(40)
include comfil
data a,b / 1.0, 2.3 /
---
---
```

A file called comfil must be established prior to execution of the preprocessor or an error message will be printed. Of course, the contents of the file must be standard Fortran common block statements.

2.1.1.10 The comment statement: Designed to allow English-like explanations of the source code to be placed in the program.
General format:

--> % --> comment -->

Procedure: The comment can begin anywhere on a line. The comment can be on the same line with a RatFor or Fortran statement. However, once the comment is encountered on a line, the rest of the line is assumed to be part of the comment. Note, the programs in Software Tools show a pound sign to signify a comment. The pound sign has a special meaning on the K.S.U. IBM 370 computer so the "%" character was chosen to denote a comment statement. The comment statement appears only in the RatFor program. The preprocessor discards all comment statements during the preprocessing of the RatFor program.

Example: This segment of code demonstrates the use of the RatFor comment statement.

```
define(MAXCARD, 80)   % maximum size of input
dimension ibuf(MAXCARD)    %  input array
 %  read data into buffer
read(9,1)(ibuf(i), i = 1,MAXCARD )
     1 format (MAXCARD a1 )
---
---
```

The comment statements in the preceding example are used to describe a symbolic constant, an array variable, and a segment of code. Other examples of all of the RatFor statements can be found in Appendix B.

2.1.2 RatFor Symbolic Notation.

The following symbols are recognized by the RatFor preprocessor and are converted to the following Fortran equivalents.

symbol	definition	Fortran
==	logical equal	.eq.
!=	logical not equal	.ne.
¬=	logical not equal	.ne.
&	logical and	.and.
<	less than	.lt.
<=	less than or equal to	.le.
>	greater than	.gt.
>=	greater than or equal to	.ge.
%	comment statement	no output

Note, blanks are significant in RatFor programming. Symbols like ">=" should not contain blanks, i.e., "> =". Symbols with blanks will not be recognized by the preprocessor.

2.2 RatFor Interactive Execution Procedures.

The procedures outlined in this section should be used to create and execute RatFor programs. Explanations of the commands copyfile, fmretr, fmsave, osjob, osprint, osretr, and ossave can be found in the K.S.U. CF/CMS Guide. Explanations of the commands runfort, watfiv,read, and type can be found in the K.S.U. CMS Cookbook. Explanations of the commands edit, input, getfile, and file can found in the IBM CMS Command and Macro Reference Manual. There are probably other ways to achieve the same results. However, if other ways are used, proceed with caution.

2.2.1 RatFor Program Creation.

The following is the minimal procedure for creating a RatFor program in an IBM 370 CMS environment. It is assumed that the user knows how to logon the K.S.U. computer system utilizing an interactive terminal. All user responses are underlined. Explanation lines are enclosed in parenthesis. The other lines are system responses.

```
(logon onto system)
R;
edit filename filetype
NEWFILE:
EDIT:
case m
input
INPUT:
--
--
```

(Now, enter your RatFor program. Press the carriage return (CR) twice when all of the code has been entered to get out of input mode. Statements can begin anywhere on a line.)

```
EDIT:
file
R;
```

At this point, your program has been created.

2.2.2 RatFor Preprocessor Storage, Retrieval, and Execution.

The RatFor Preprocessor Software Package was delivered to K.S.U. via a 9-track nonstandard magnetic tape. Since this tape was not created using standard IBM labeling conventions, it was exceedingly difficult to work from the tape. Consequently, the entire contents of the tape were off-loaded to cards. The RatFor preprocessor source cards can be read into the system via a card reader using the following job cards:

```
//VMRDR JOB ( JOB CARD PARAMETERS)
:READ FILENAME FILETYPE
$JOB
---
(source cards)
$ENTRY
```

Note: there is an upper limit on the number of cards per job input at a K.S.U. remote job entry terminal. At a remote location, the RatFor preprocessor source deck has to be divided into two jobs, with each job having a different filename. Next, at an interactive terminal, logon the CMS system and then type READ *, for each file that was read in. The READ * command brings the file(s) to your interactive terminal. If more than one job is used to read the source then combine the files first by using a copyfile command or

a getfile command. For example, the copyfile command to combine two files is:

```
copyfile filename2 filetype a1 filename1 filetype a1(append)
```

To combined two files using the getfile command, edit filename1, go to the bottom of filename1 by issuing the edit command, bot, then issue the getfile command. An example getfile command is:

```
getfile filename2 filetype a1 1 *
```

After the files are combined, store the RatFor preprocessor source code using the ossave or fmsave command. Note, before using the commands, ossave or fmsave, contact the K.S.U. Computing Center consultants for details surrounding their usage. For example, one might type:

```
cssave ratfor fortran
```

The system will inform you when the file called RatFor has been stored.

To retrieve the file, use the csretr or fmretr command. For example, type:

```
csretr filename filetype
```

Cnce retrieved, the system will respond with messages such as ""PUN FILE XXXX FROM OS"", where XXXX are four digits. Then you type, read *, to bring the file to your terminal. Ecfore you can execute the preprccessor, you must first create a text file, if the text file does not exist. To create a text file, you must ccmpile the program. To compile the program, type:

```
csjob ratfor fortran a1 (proc fortgc parm deck punch print
  time ,45)
```

The system will respond with:
month/day/year -- YOUR SEQUENCE NUMBER IS VMXXXXXX , where XXXXXX are all digits. Once the RatFor program has been compiled, the system will respcnd:

```
PRT FILE XXXX FROM CS
PUN FILE XXXX FROM OS
```

where XXXX are all digits. Your response should be

```
read ratfor listing
```

The listing will contain a listing of the program, diagonostics (if any), and statistics of the file's compilatien. In the statistics section, the return code

should be zero. If the return code is not zero then an error has occurred. To get the return code, type the first page of the listing by issuing this command:

```
type ratfor listing
```

When the first page has been printed, press the 'attn' button and type ht to stop the printing. If the return code is not zero then get a copy of the listing by issuing this command:

```
osprint ratfor listing
```

The system will respond with a listing number. Record the number and your listing can be obtained at the Computing Center. Your listing will be distinguished by the listing number. If the return code is zero, type:

```
read ratfor text  .
```

This command brings the RatFor text file to your terminal. Before you execute the text file, there are two preliminary commands that must be issued. These preliminary commands define the input file (05) and the output file (01), respectively. The preliminary commands are:

```
filedef 05 disk (user's filename) (user's filetype) a1 (perm
filedef 01 disk (ofilename) (ofiletype) a1 (perm recfm fb
  block 800 lrecl 80)
```

whereas, the user's filename and filetype is the name of the file that the user created and the filetype given to the file. Ofilename and ofiletype is the name and type of the file that the preprocessor will output. After the preliminary commands have been issued, type:

```
runfort ratfor text
```

and the system will respond:

```
EXECUTION BEGINS ...
```

If the input, RatFor program, is syntactically correct, the preprocessor will print the following:

```
*** BEGIN RATFOR PREPROCESSOR
*** WAIT ***
*** END RATFOR PREPROCESSOR ***
R;
```

The output is in the file that was defined as 01. If the RatFor program is not syntactically correct, then an error message(s) will be printed. After the errors have been corrected, restart with the preliminary filedef commands.

2.2.3 Preprocessed Program Execution.

The execution procedure of a program that has been preprocessed by the RatFor preprocessor is no different from any other Fortran program's execution procedure. You now have a standard Fortran program. Compilation of your preprocessed program can be accomplished through interactive compilation procedures, i.e., Watfiv (user's filename) or through submission to OS (osjob) for a batch compilation. Program compilation and execution procedures, and Fortran compilers' constraints are outlined in the K.S.U.'s Computing Center's CMS Cookbook and CP/CMS Guide. A sample execution is listed in Appendix C.

CHAPTER 3

RATFOR PREPROCESSOR DOCUMENTATION

3.1 Detailed Description of Changes Made.

This section contains the specifications for each of the changed preprocessor's subroutines, modules, and other changes. Each of the preprocessor's modules is described in the book, Software Tools. Therefore, only explanations of the changes will be given. The parameters passed and the names of the common blocks used in each module are listed. For the contents of a common block, reference the block data subroutine found in the preprocessor listing in Appendix A of this report. For the contents of a subprogram, reference the listing at Appendix A also. The main program calls the subroutine, Parse, and Parse does the rest of the work. Specifications for changed modules are as follows:

3.1.1 Module Name - Main Program

Function - Initiate preprocessing.

Parameters Passed - none

Changes Made - Write statements were inserted to inform the user of the preprocessor's status, i.e., when preprocessing is beginning, is occurring, and has ended.

3.1.2 Module Name - Initkw

Function - Install the key word, define, in the key word table so that recognition of symbolic constant definitions

can occur.

Parameters Passed - none

Changes Made - Previously, this module installed only the lower case character representation of the key word, define, in the key word table. Consequently, when the upper case representation of define was encountered by the preprocessor, it was not recognized as a key word. Fortran data statements which reflects the upper case character representation of define and statements which inserts the upper case representation into the key word table were added to this module.

3.1.3 Module Name - Gettok

Function - Breaks the input into alphanumeric strings, quoted string, and single non-alphanumeric characters. Additionally, it strips out the blanks, tabs, and comment statements that separate tokens; it handles file inclusions and line numbers.

Parameters Passed - token, token size, and uses the common block, Cchar.

Changes Made - To allow recognition of the upper case character representation of the key word "include", Fortran data statements which reflect upper case representation of include and statements which determine whether or not a token is the upper case representation of include were added.

3.1.4 Module Name - Putch

Function - Put a character into the output buffer until the last character on a line has been reached, then transmit the contents of the output buffer to the output file.

Parameters Passed - character (c), output device number (f).

Changes Made - Fortran statements were inserted to insure that the preprocessor would output fixed length records, i.e. 80 byte records. Variable length records cause problems during compilation.

3.1.5 Other Changes

The output file number for the preprocessor was arbitrarily changed to 01. This change was made in the preprocessor subroutine called Outdcn. In subroutine Outdcn, there is a call to subroutine Putlin with arguments that contain the number of the output file. The number in that call statement was changed to 01.

The error message file number is 09. A file number of 09 causes the error messages to be printed at the CMS terminal. If desired, one can have the error messages printed on a file by changing the error message file number. To change the error message file number, locate the preprocessor subroutine, Remark. In subroutine Remark, change the output device number from 09 to whatever is desired. Before executing the preprocessor, a filedef command must be issued to define the new error message file.

In the block data subroutine, the array, Extlet, now contains the hexidecimal representation of lower case alphabets to allow recognition of lower case alphabets. Before this change, the preprocessor would ignore RatFor statements written in lower case letters.

3.2 Module Access Graphs.

This section contains the module access graphs of all the major subroutines of the RatFor preprocessor. The graphs are arranged in functional order. The directional flow of the graphs is downward. Other directions are denoted by lines capped with arrowheads. Each box constitutes a module. A box that has a thick horizontal bottom line is a terminal node. A terminal node is one where the directional flow stops, (that is, no other branching occurs).

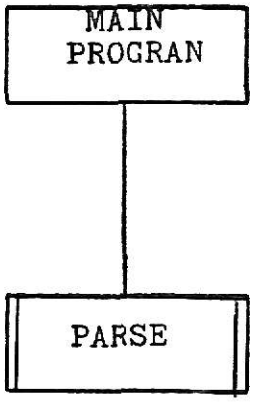


Figure 3.2.1
Module Access Graph

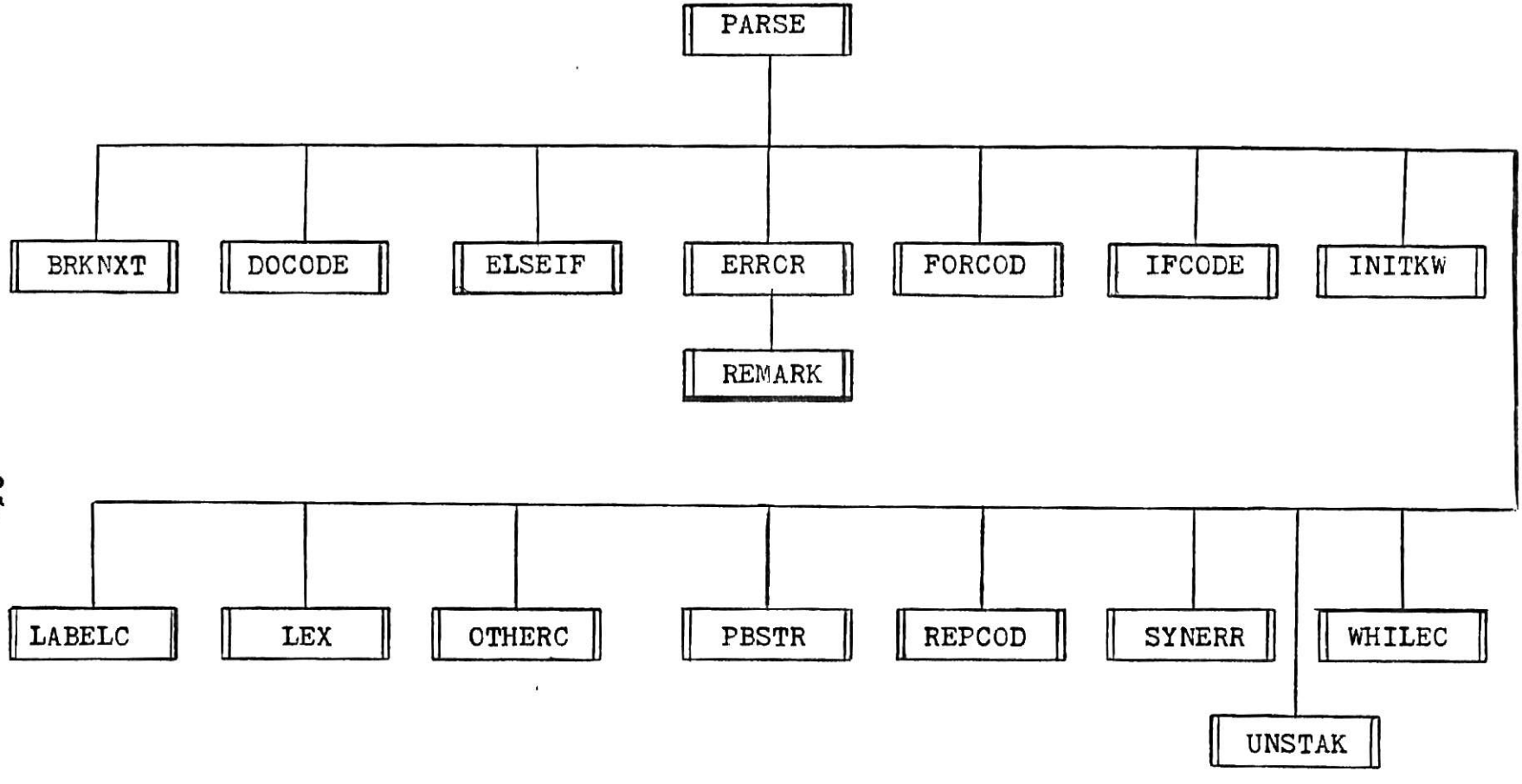


Figure 3.2.2
Module Access Graph

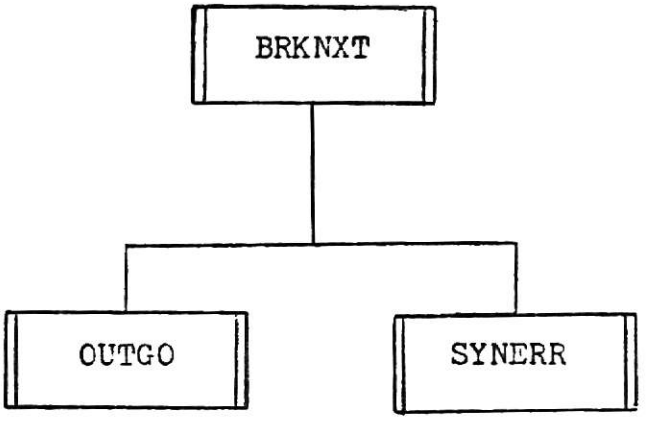


Figure 3.2.3
Module Access Graph

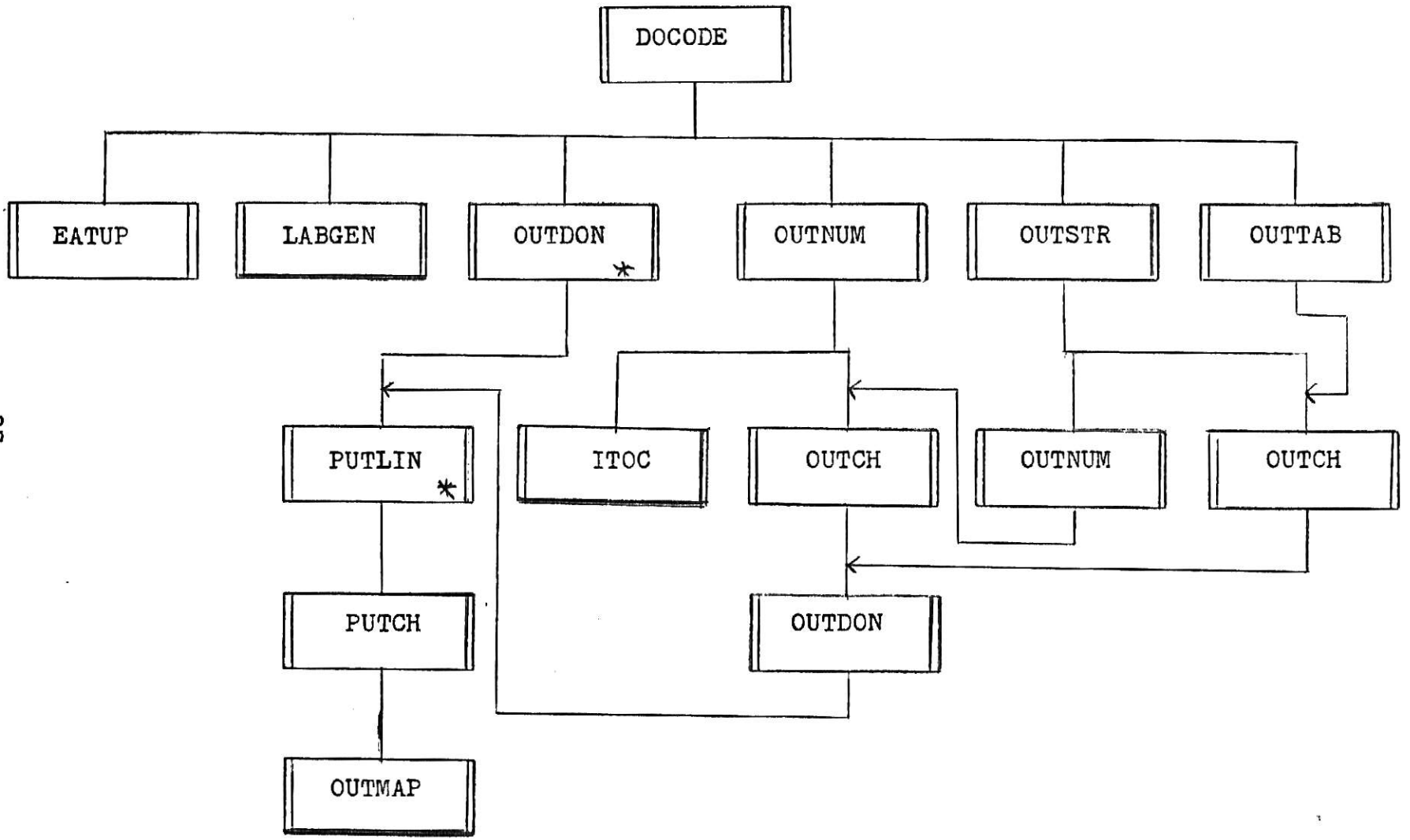


Figure 3.2.4
Module Access Graph

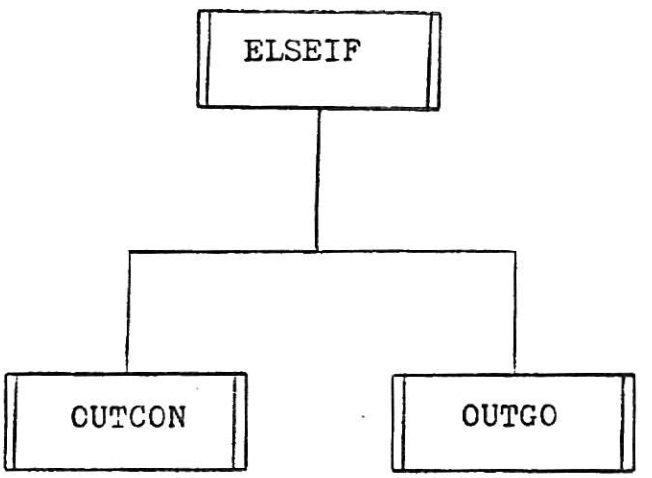


Figure 3.2.5
Module Access Graph

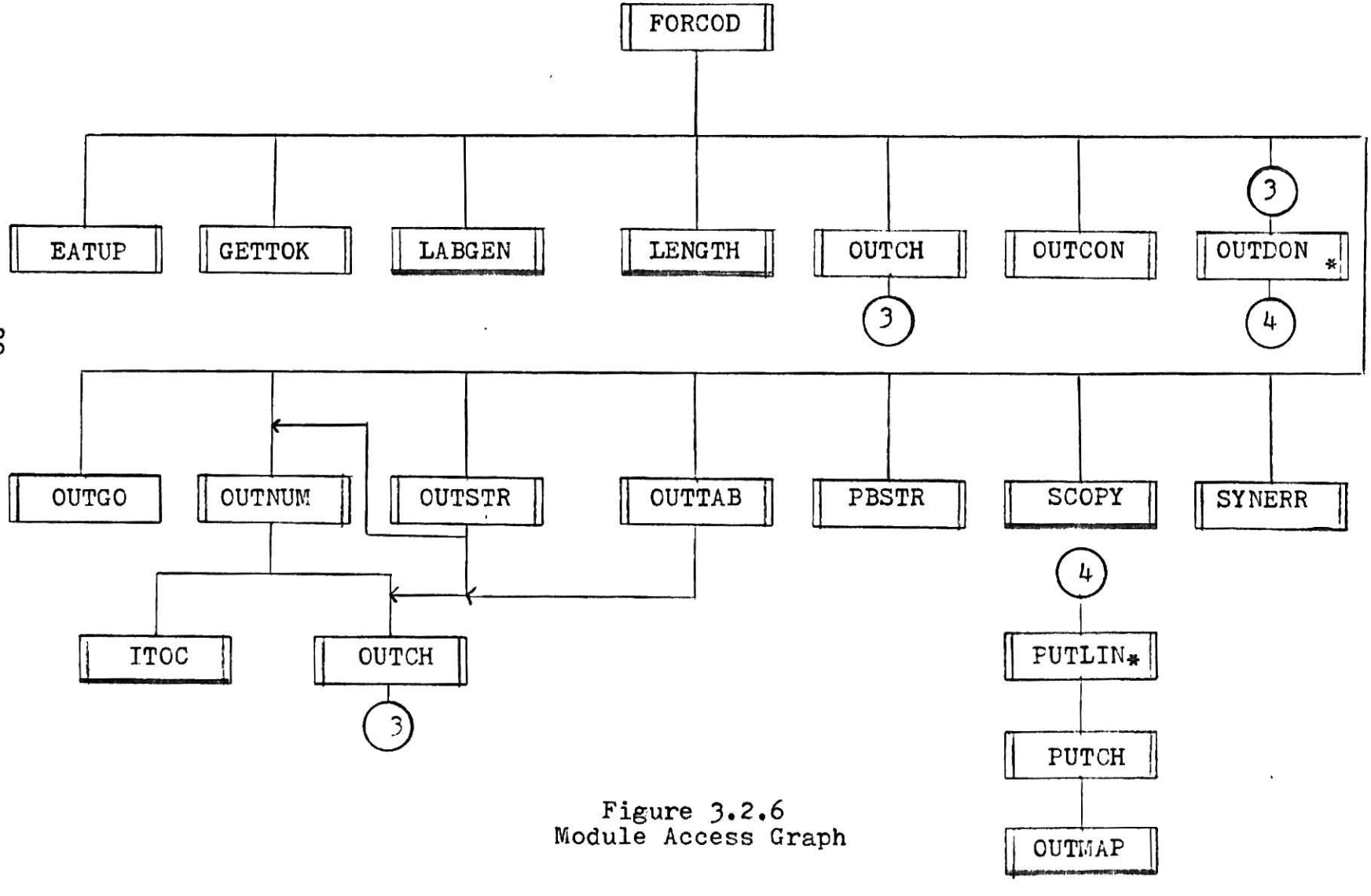


Figure 3.2.6
Module Access Graph

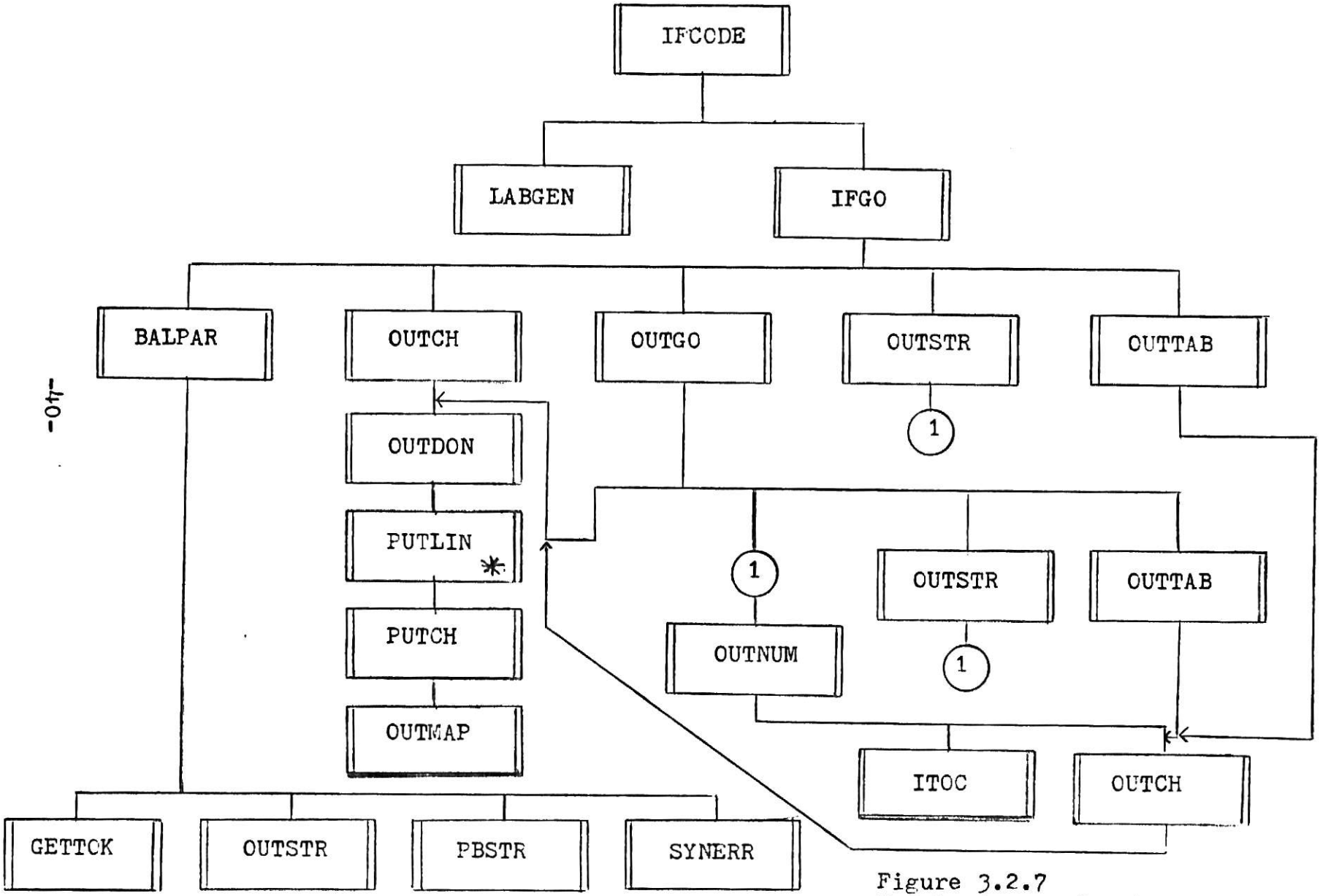


Figure 3.2.7
Module Access Graph

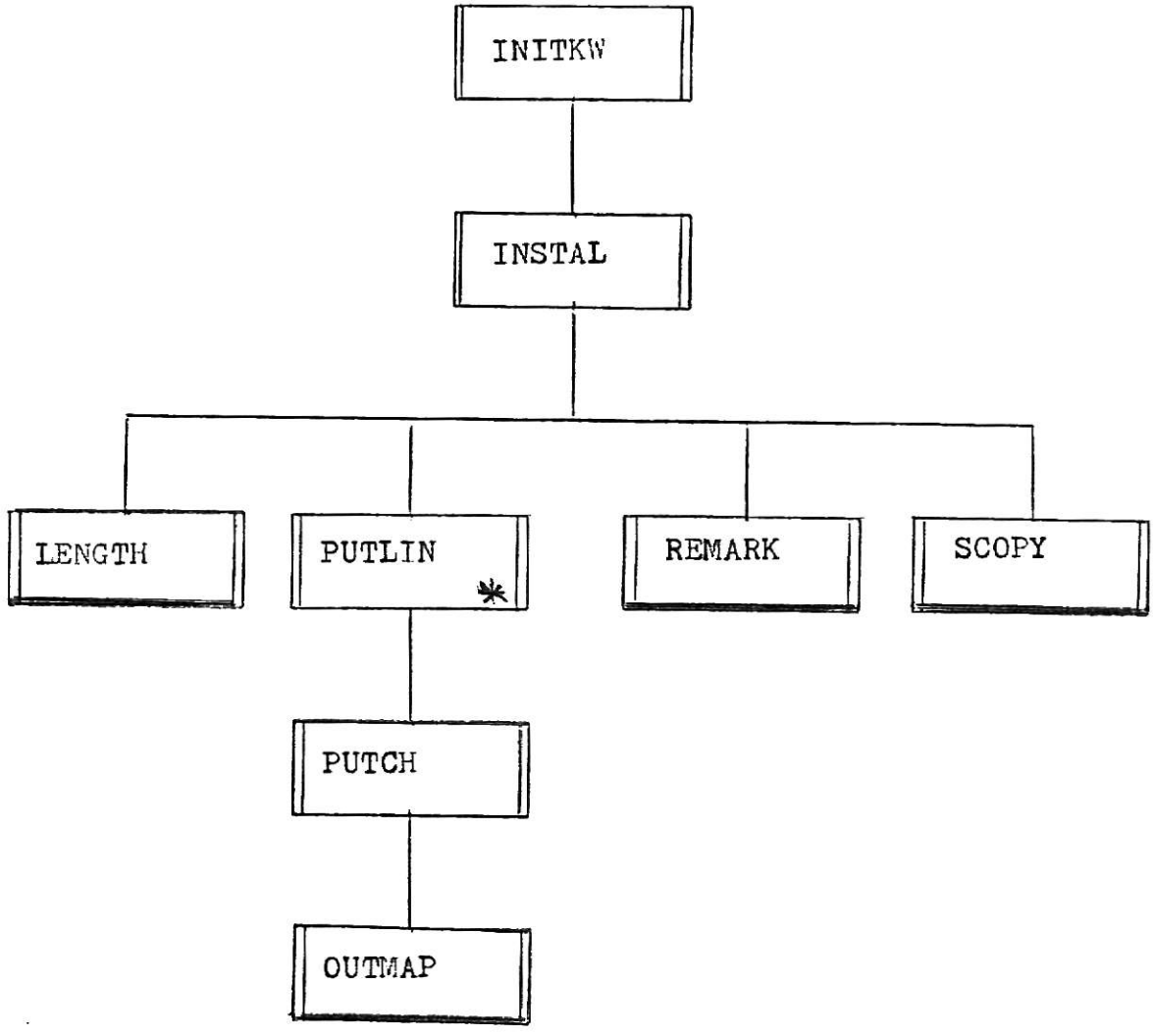


Figure 3.2.8
Module Access Graph

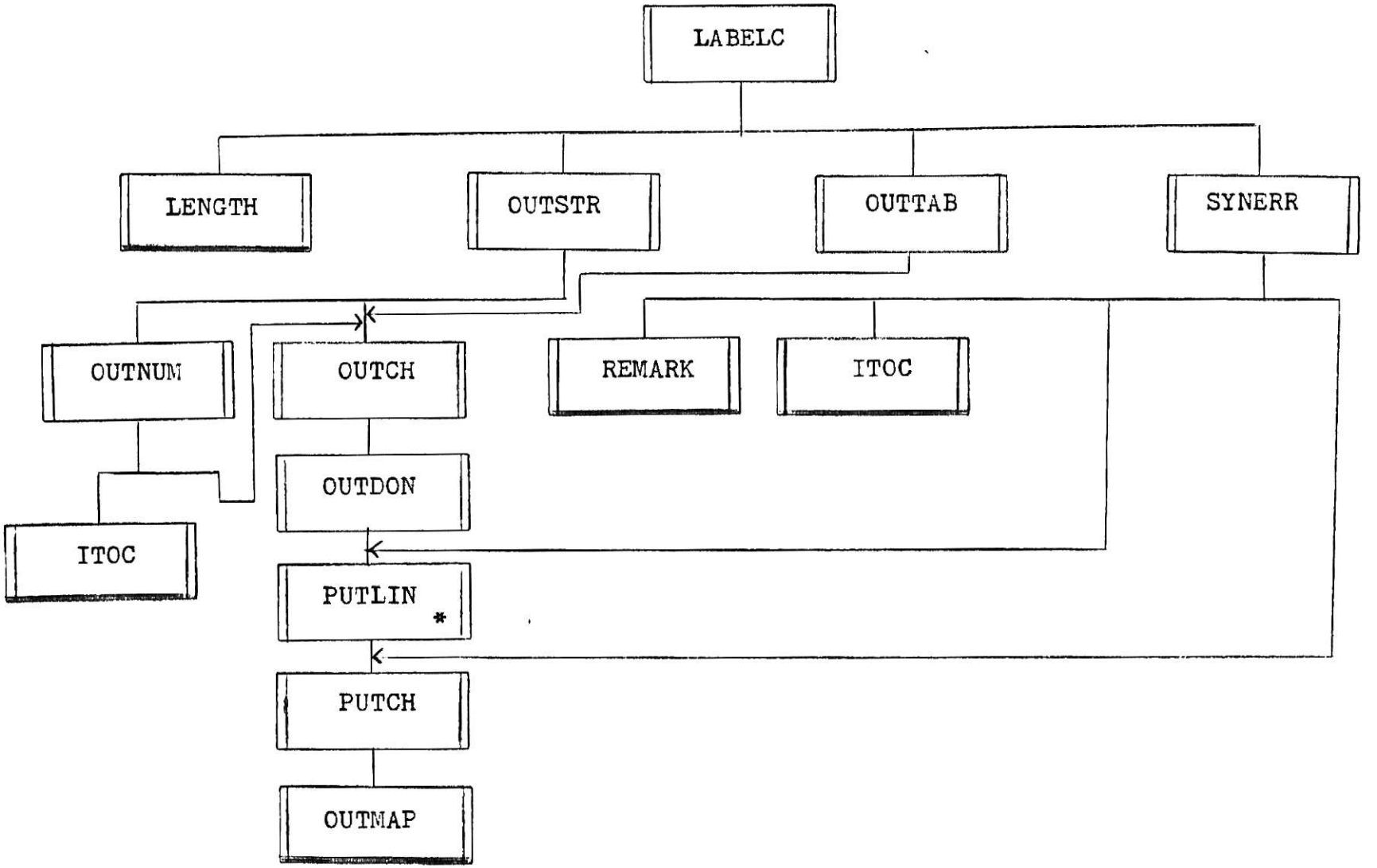


Figure 3.2.9
Module Access Graph

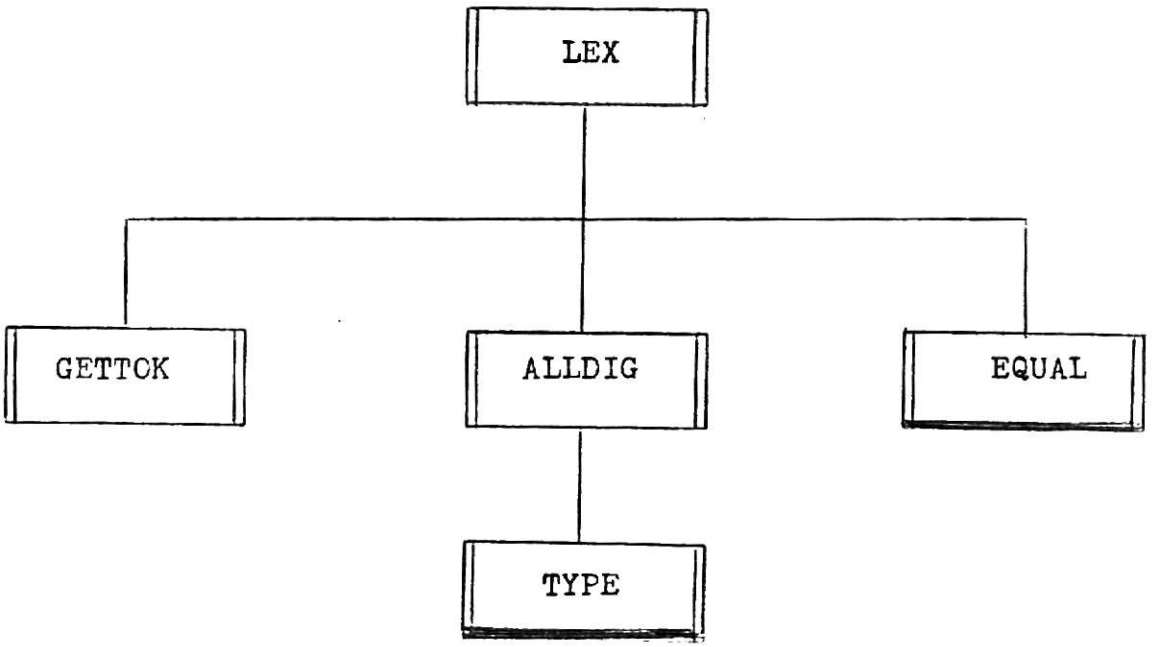


Figure 3.2.10
Module Access Graph

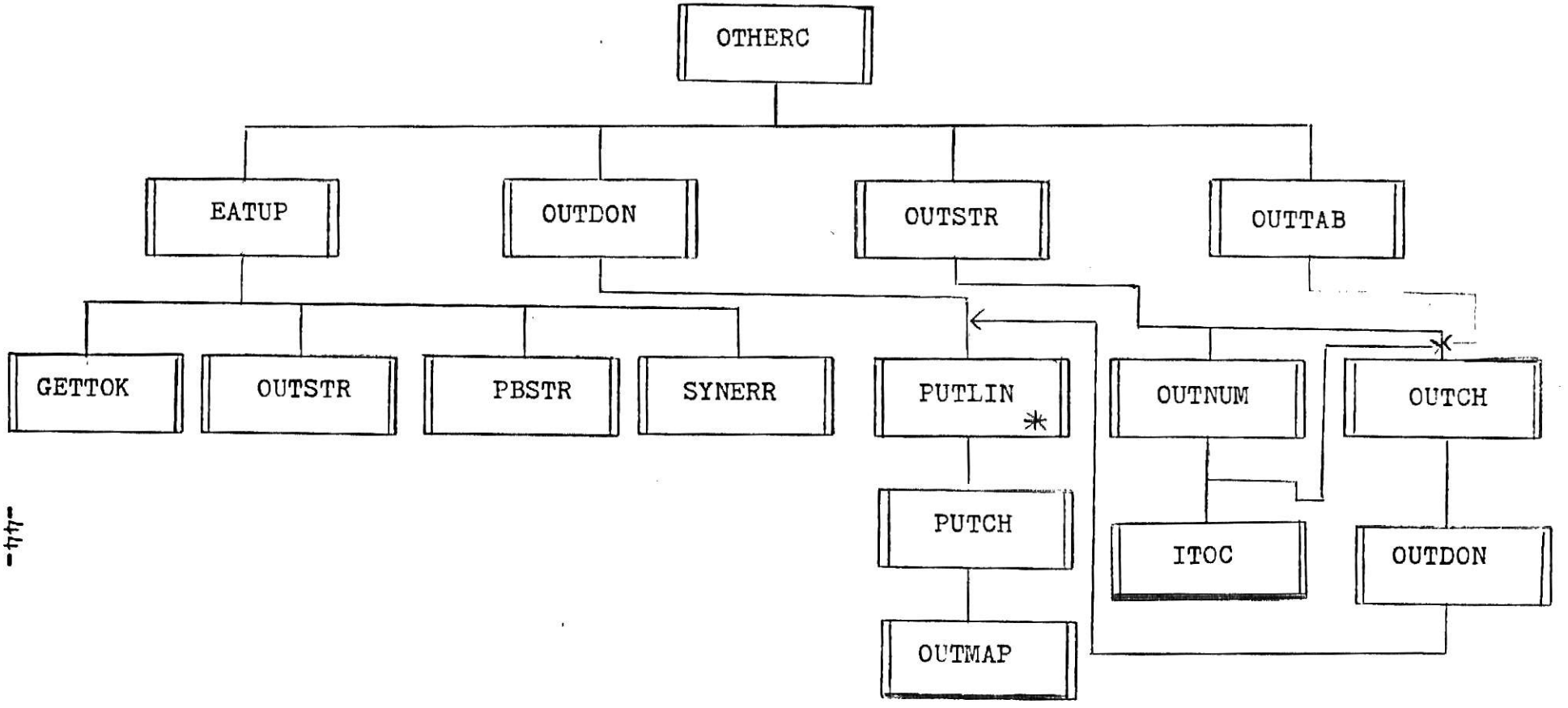


Figure 3.2.11
Module Access Graph

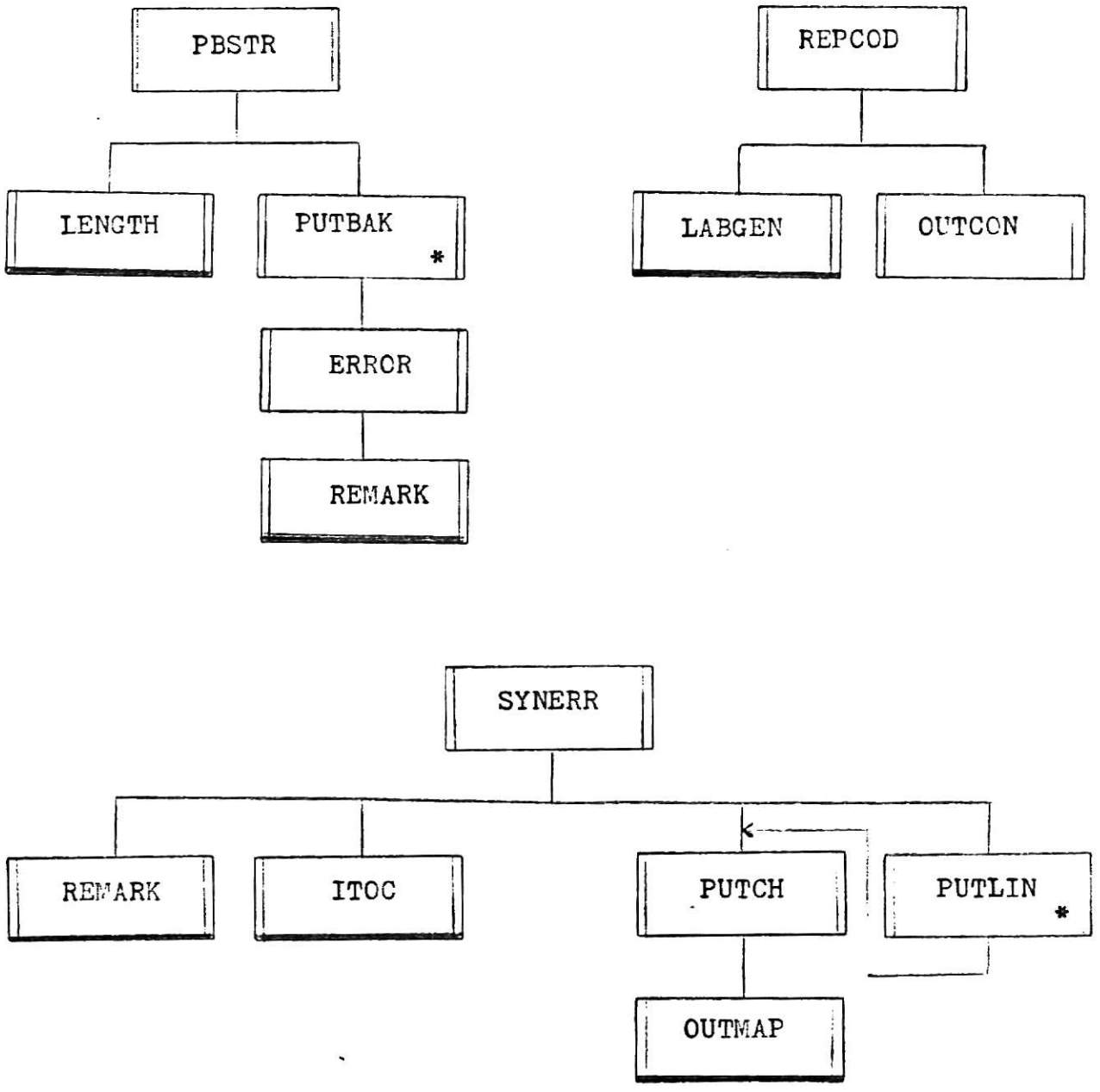


Figure 3.2.12
Module Access Graphs

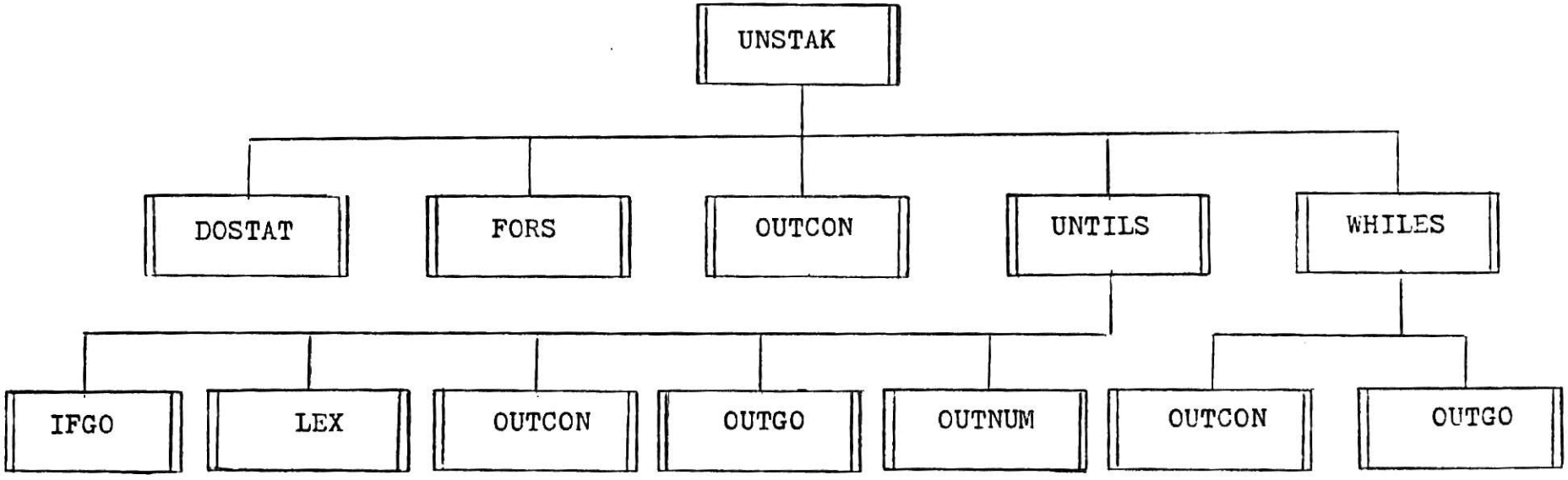


Figure 3.2.13
Module Access Graph

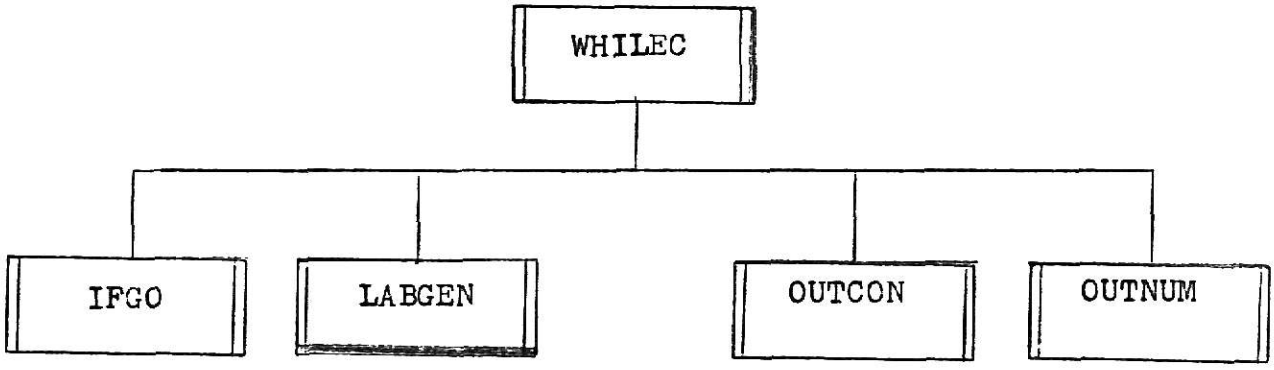


Figure 3.2.14
Module Access Graph

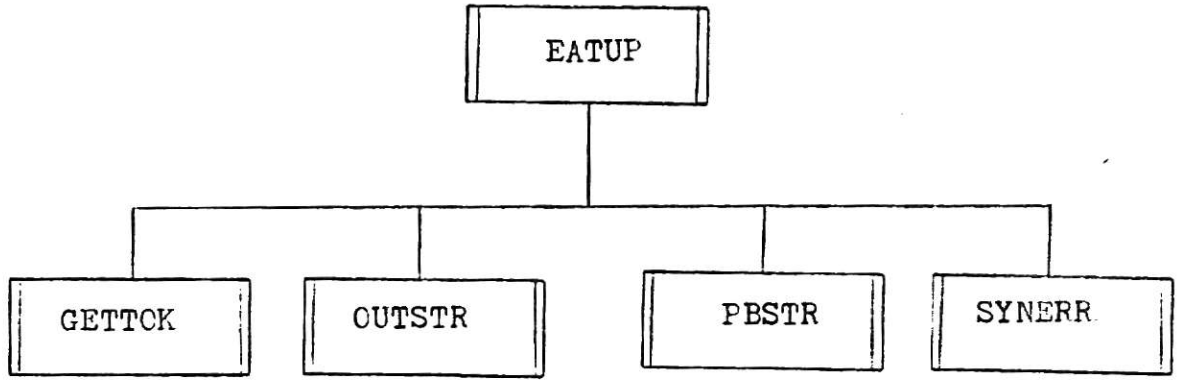


Figure 3.2.15
Module Access Graph

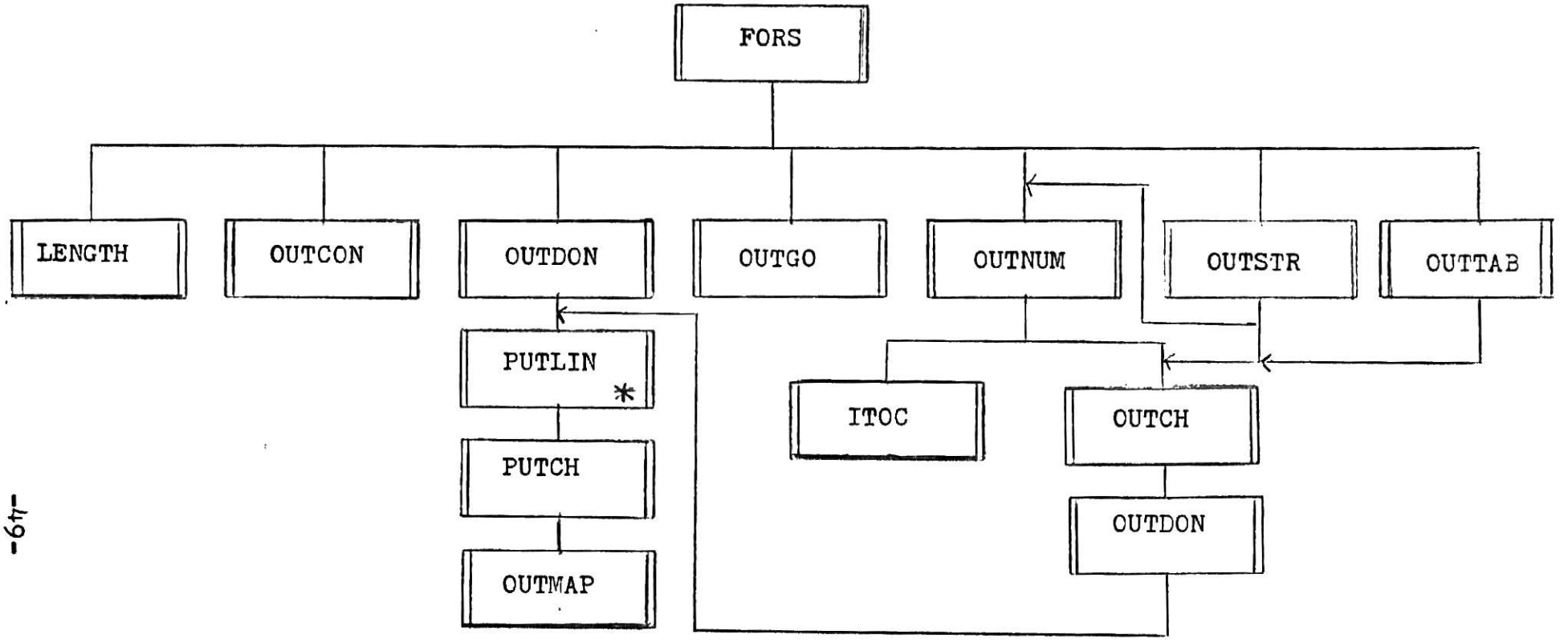


Figure 3.2.16
Module Access Graph

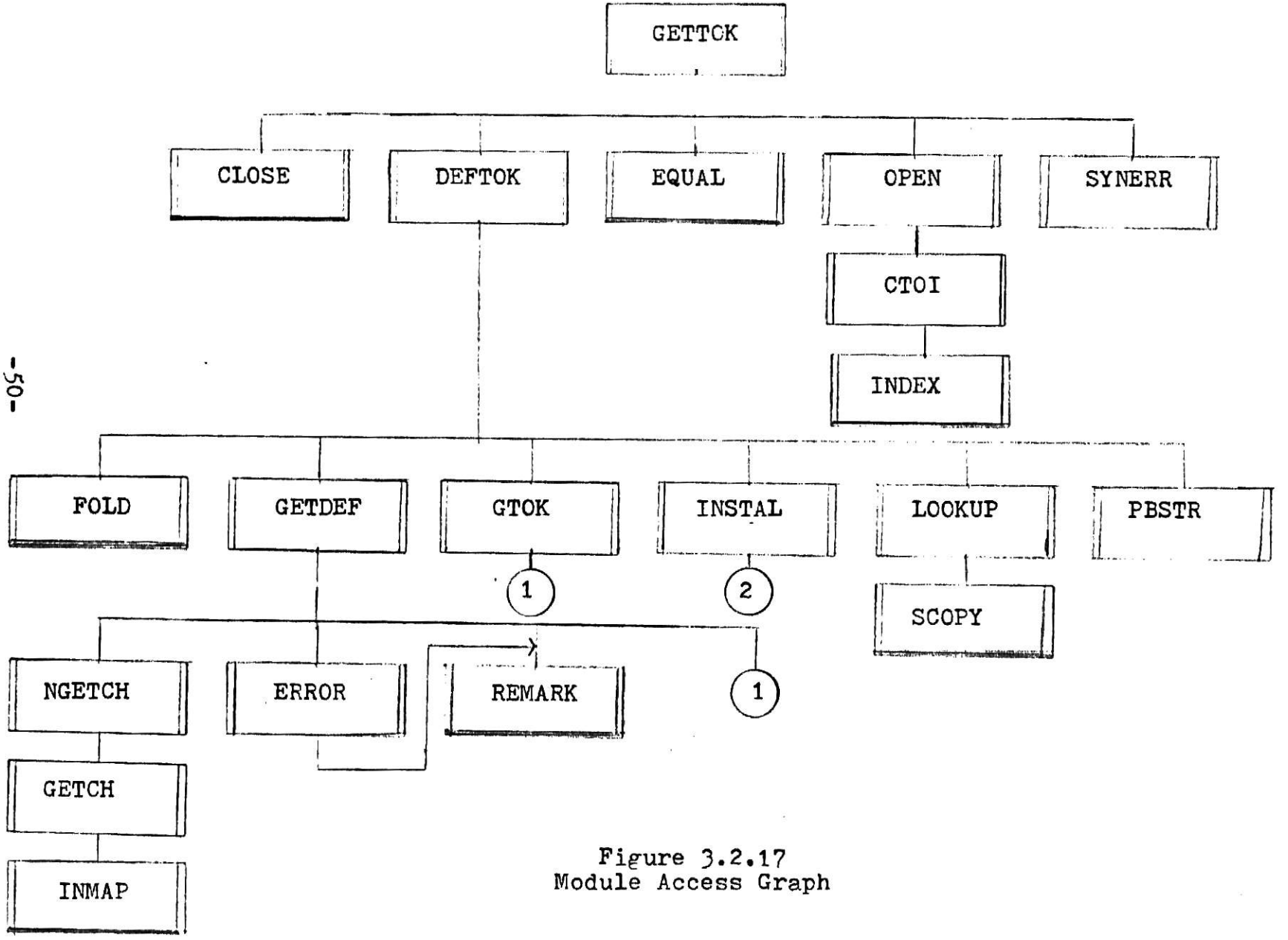


Figure 3.2.17
Module Access Graph

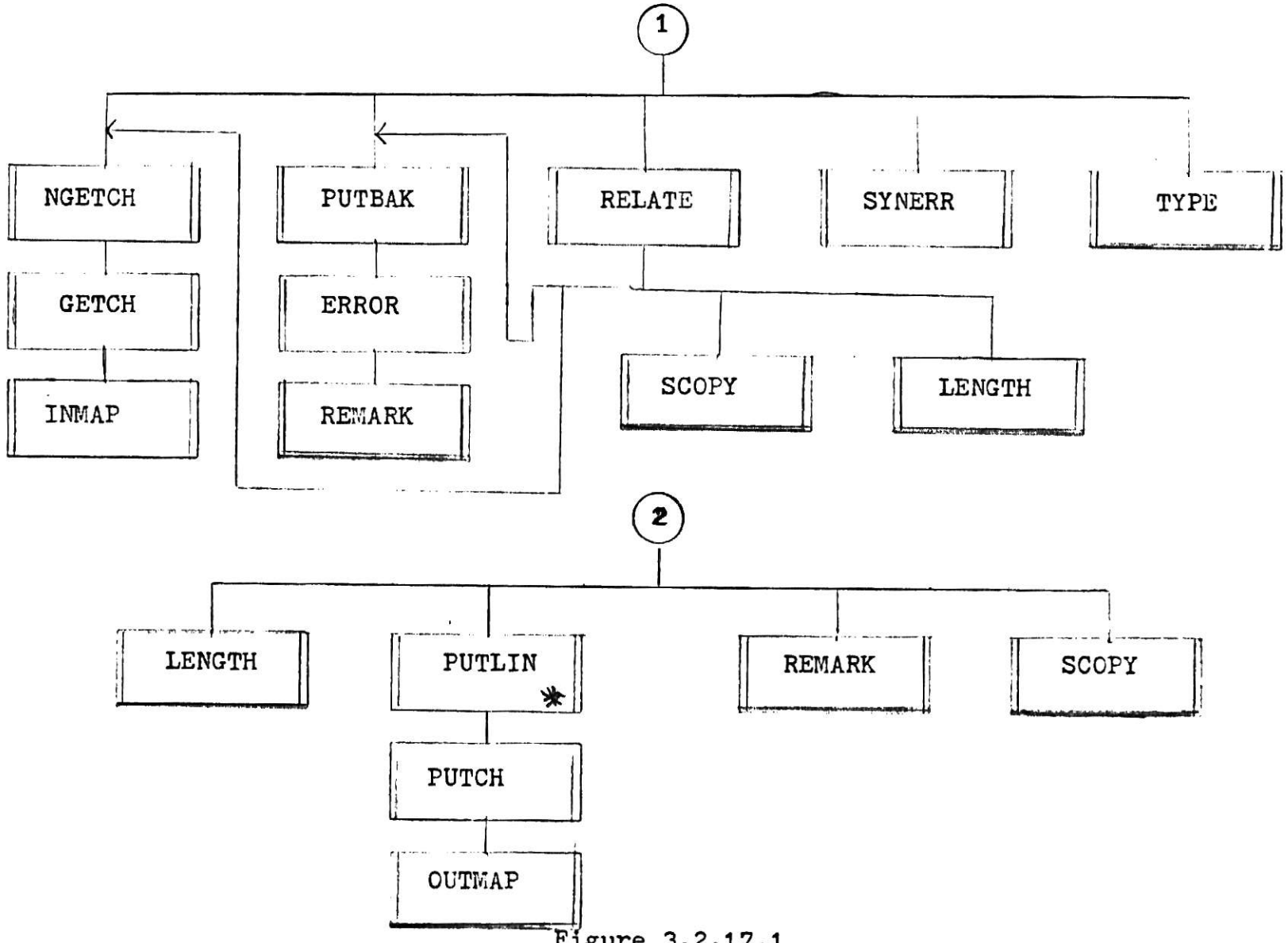


Figure 3.2.17.1
Module Access Graph

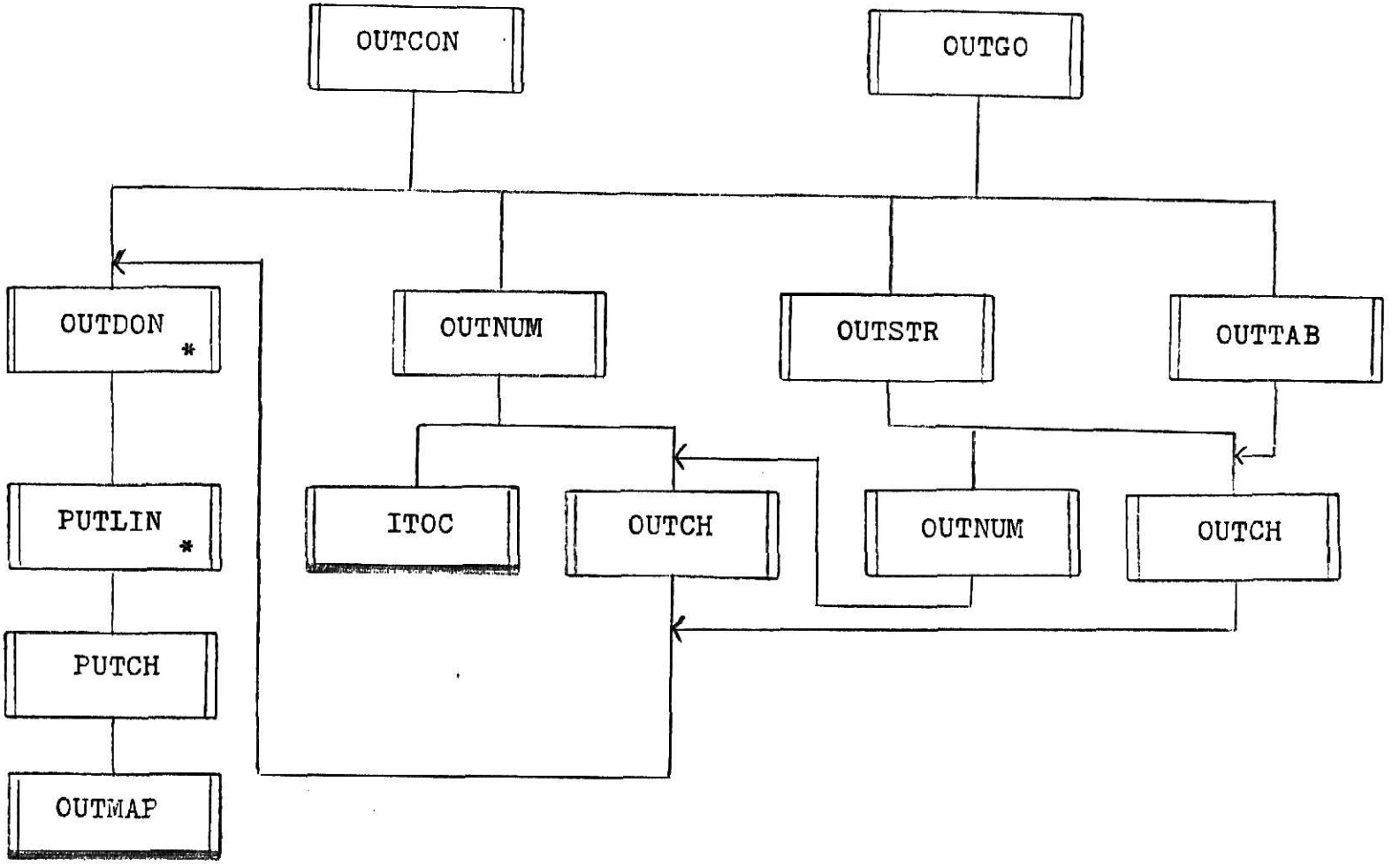


Figure 3.2.18
Module Access Graphs

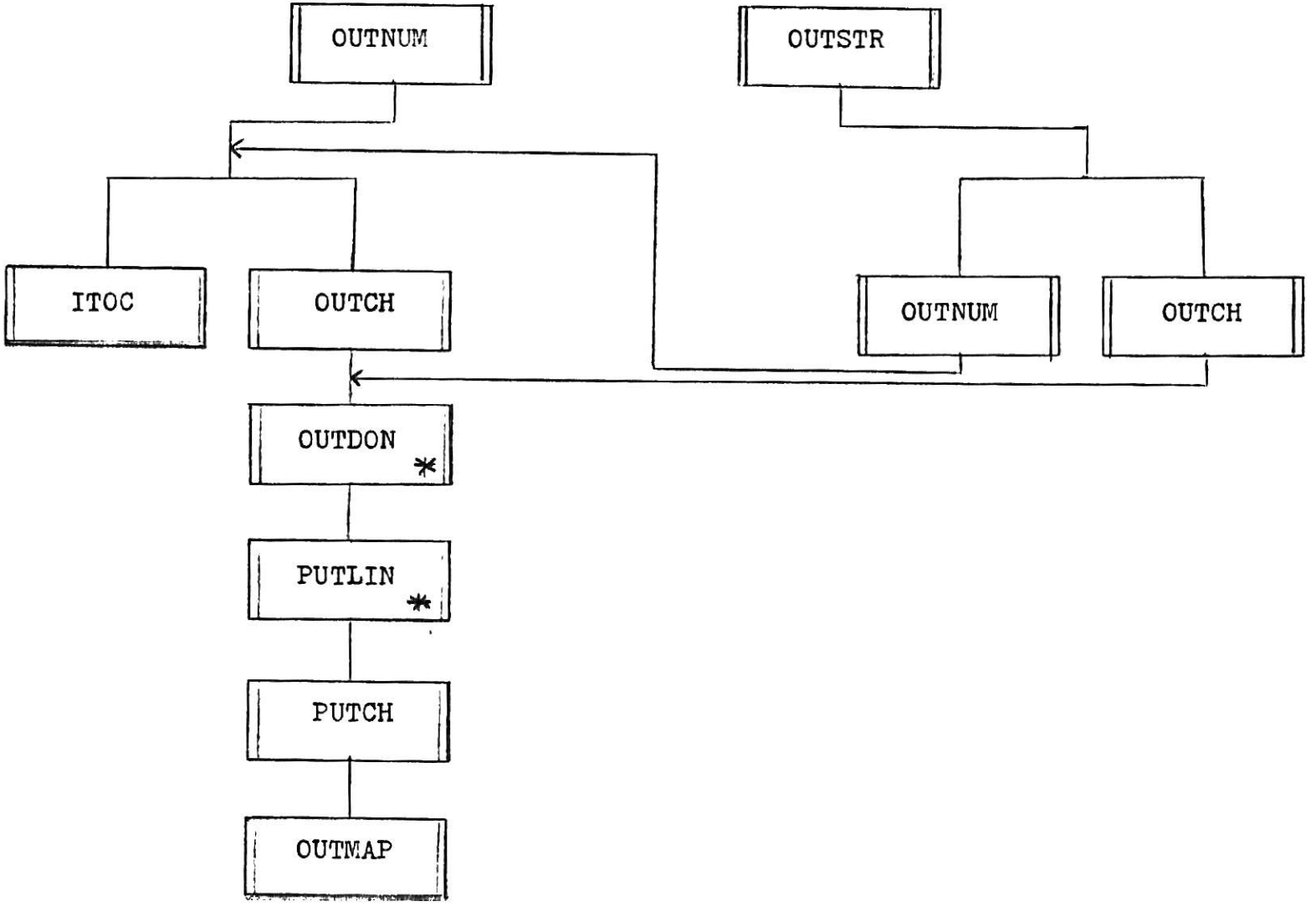


Figure 3.2.19
Module Access Graphs

3.3 Module Access Matrix.

This section contains the access matrix for the RatFor preprocessor. This matrix is designed to show the modules that a particular module calls and the modules that call a particular module. To determine what modules a particular module calls, locate the name of the calling module on the left side of the matrix. From the name of the module, scan horizontally until "1's" are encountered. Scan vertically from the "1's" to find the names of the modules called. To determine the modules that call a particular module, locate the name of the called module at the top of the matrix. From the name of the module, scan vertically downward until "1's" are encountered. From the "1's", scan horizontally and to the left to find the names of the modules doing the calling. There are 58 modules including the main program, thereby producing a 58 by 57 matrix.

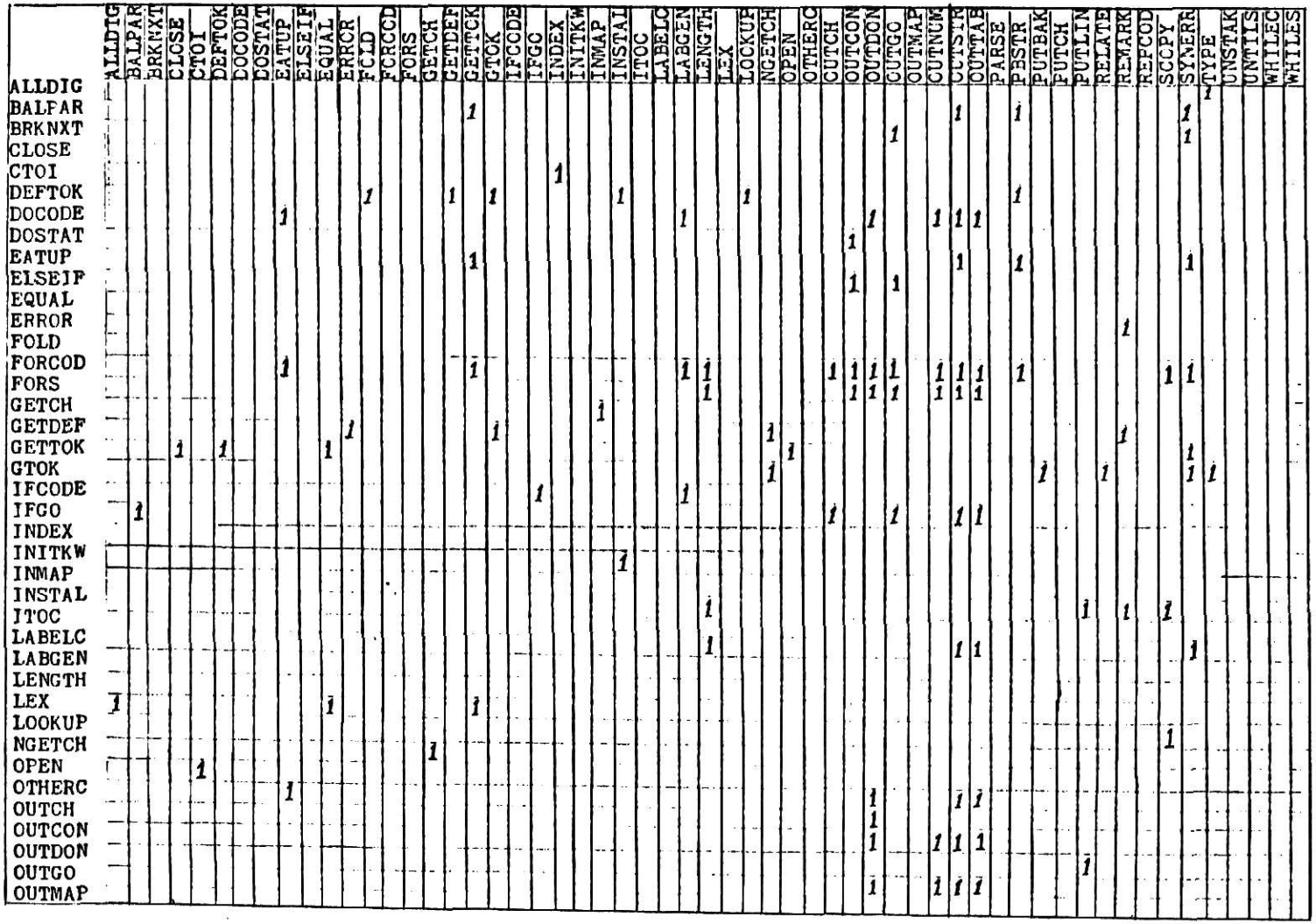


Figure 3.3
Matrix Access graph

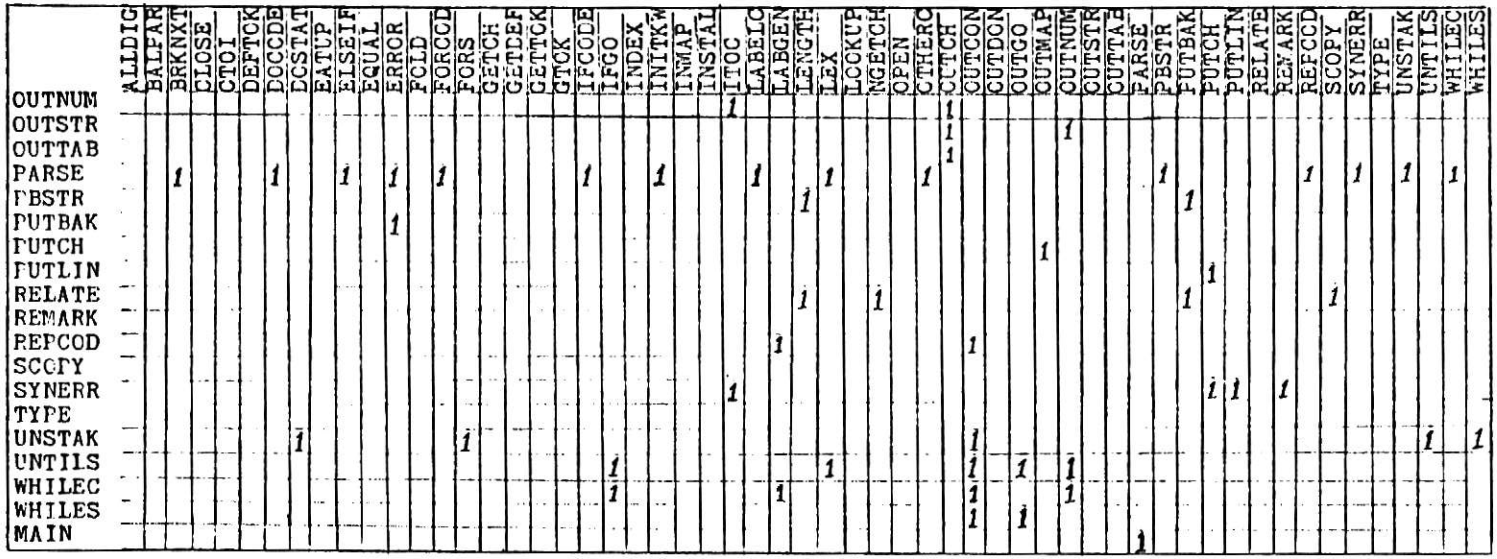


Figure 3.3.1
Matrix Access Graph

3.4 RatFor Statistics.

The RatFor preprocessor has 2318 lines of source code, including comments. It takes an average of 36 seconds to compile the preprocessor using the IBM 370's OS Batch System. Maximum core used during compilation is 102K.

The response time of the preprocessor is dependent upon the workload of the IBM 370. Additionally, as the complexity of a program increases, the preprocessing time also increases. For example, a program that has numerous nested if-else statements will take longer to preprocess than a program with the same number of statements but fewer if-else statements. The preprocessing (CPU) time for different size programs are listed below. These times were taken during a weekday afternoon and complex RatFor programs were used.

Size of Program	Preprocessing CPU Time
50 lines	2.82 seconds
100 lines	7.14 seconds
500 lines	26.58 seconds
1000 lines	57 seconds

CHAPTER 4

THE IMPLEMENTATION OF A RATFOR TOOL

4.1 Implementation Procedures.

Each of the RatFor tools in the RatFor Software Package contains the modules that are necessary to provide the function for which the tool was intended. However, these modules alone may not be sufficient to execute the tool. In the software package, the tools are arranged according to the sequence in which they are presented in the book, Software Tools. Consequently, the user should be aware that the tools often call on modules, including block data modules, that were written in earlier sections or chapters of the book, Software Tools and these called-on modules may or may not be a part of the tool that is going to be used.

Generally, the names of the modules of a tool are listed at the end of the section or the chapter of Software Tools in which the tool is presented. To determine if any called-on modules are missing, the listed names should be compared with the names of the modules that are actually in the tool. If the names of the modules of a tool are not listed at the end of the section or the chapter, one can preprocess the tool, compile the tool, attempt to execute it and the missing module (s) if any, will be listed. Another method of determining if modules are missing is to search the source code of the tool manually for module calls and then search for the module being called.

When a tool calls on a previously written module that is not a part of its set of modules then the previously written module must be located in the previous section's source cards, extracted from the previous section's cards, and inserted into the appropriate place in the tool that is being constructed. A tool might also call a module which was supposed to be written as part of an exercise in a preceding chapter of Software Tools. In this case, the user will have to create the module and then insert it into the appropriate place before attempting to execute the tool.

Some tools may have more than one version of the same module in them. The primary purpose here is to demonstrate that one can achieve greater software productivity and performance by refining what has already been done, instead of endlessly reinventing the same things with minor variations. In such cases, the user has to select one version of the module and remove the other(s) before executing the tool.

Each RatFor tool employs the use of symbolic constants. Therefore, the user must insure that all necessary define statements are present in the tool before attempting to preprocess it. The symbolic constants are listed throughout the sections or chapters of Software Tools in which the tool is presented and are typed in upper case letters in the tool. The rest of the tool's code is in lower case letters. Once a tool is pieced together, it can be preprocessed and

compiled using the procedures outlined in section 2.2.2 and section 2.2.3 of this report. Section 4.2 of this report will explain how one of the RatFor tools, the text formatter, was implemented.

4.2 Introduction to the Text Formatter.

The text formatter is a tool similar to SCRIPT, but unlike SCRIPT, the text formatter is portable. The text formatter is used to format a document on a line printer or interactive terminal. It can produce output which is paginated, titled, centered, indented, etc. The text formatter accepts input which is interspersed with text formatter commands and produces the command described output.

The text formatter is a very useful tool for anyone who is revising a draft or any other document. Once a draft is correct, it will never have to be completely retyped again. If the draft is revised, all one has to do is type in the revisions, submit the draft to the text formatter and obtain the desired output. Machine formatting eases the typing job, since margin alignment, centering, underlining, etc., are handled by the computer, not by a typist. Using machine formatting, drastic format changes in a document can be made without altering any of the text material. But most important, machine formatting seems to encourage writers to improve their product, since the overhead of making an

improvement is small.

4.2.1 Implementation of the Text Formatter.

A list of modules that are used by the text formatter is given in the back of chapter 7 of Software Tools. The listed names are:

main program	init	getlin	comand
comtyp	getval	set	gettl
space	brk	put	text
leadbl	underl	center	width
phead	pfoot	skip	putc
puttl	putdec	putlin	getwrd
putwrd	spread		

This list of names was compared with the names of the modules that were actually in the set of text formatter modules. Initially, only the modules- getlin, putc, putlin, putdec were found to be missing. However, after locating these missing modules in the previous sections and scanning their code, it was noticed that these modules also called on modules which were not in the text formatter's set of modules. Consequently, the following modules were found to be missing:

getlin	putc	putlin	putdec
ctoi	itoc	index	length
scopy	inmap	outmap	

With the exception of the module "getlin", all of the missing modules were found in the previous sections' cards,

extracted from these sections and inserted into the appropriate place in the text formatter. The module getlin had to be written. The function of the module, getlin is to read in the document to be formatted a line at time. After the module, getlin, was written, it was inserted into the text formatter's source code.

The text formatter also used symbolic constants which were not included in the text formatter's set of define statements. Note that all of the symbolic constants are typed with upper case letters while the rest of the text formatter's code is in lower case letters. The symbolic constants were located by manual scanning of the text formatter's code. The missing symbolic constants were found in a previous section's cards, extracted from that section, and inserted into the text formatter's cards.

Last, it was noticed that the text formatter used common blocks and array variables found in the block data module of the RatFor preprocessor. These common blocks and array variables cards were extracted from the preprocessor's source cards and used to form a block data module for the text formatter. The common block's name is Cchar. Cchar contains array variables for upper and lower case letters, numbers, and special characters. See the text formatter listing in Appendix B for the actual content of Cchar.

After all of the missing parts had been located and placed in the text formatter, the cards were read into the

system under the filename, Textfm, and the filetype, Watfiv. At this point, the text formatter was ready to be preprocessed. It was preprocessed by using the procedures outlined in section 2.2.2 of this report.

4.2.2 Text Formatter Commands.

The text formatter commands will be briefly explained in this section. For a more detailed explanation of a command, reference chapter 7 of Software Tools. All formatter commands begins with a period. There should only be one command on a line and that command should precede the line(s) of text material that it is suppose to affect. The formatter commands are:

Command	Break?	Default	function
.bp n	yes	n= +1	begin page numbered n
.br	yes		cause break
.ce n	yes	n= 1	center next n lines
.fi	yes		start filling
.fo	no	empty	footer title
.he	no	empty	header title
.in n	no	n= 0	indent n spaces
.ls n	no	n= 1	line spacing is n
.nf	yes		stop filling
.pl n	no	n= 66	set page length to n
.rm n	no	n= 60	set right margin to n
.sp n	yes	n= 1	space down n lines
.ti n	yes	n= 0	temporary indent of n
.ul n	no	n= 1	underline words from next n lines

The value of the variable, n, is an integer. If the integer is preceded by a "+" or "-", the previous value is changed

by this amount; otherwise the argument represents the new value. If no argument is given, the default value is used. A line that consists of blanks causes a break and produces a number of blank lines equal to the current line spacing. If a line begins with n blanks followed by text, it causes a break and a temporary indentation of +n spaces. The listed defaults and the special blank line actions help ensure that a document containing no formatting commands or only basic formatting commands will still be formatted reasonably.

4.2.3 Execution of the Text Formatter.

Before executing the text formatter, an input file containing the document to be formatted has to be created with the text formatter commands interspersed. The input file can be created using the procedures outlined in section 2.2.1 of this report. An example input file is:

```
.bp
.rm 60
.ce
.ul
A Format Test
This file is a test for the RatFor text formatter.
The text formatter will neatly format the contents
of this file.  The title will be centered and
underlined.  The text material will be arranged in
60 character lines.
.br
The text formatter uses the computer to do work
that a typist normally does.
```

Once the input file has been created, the input file and the output file have to be defined as file 05 and 01,

respectively, before executing the text formatter. This can be done by issuing a filedef command for each file. The command to invoke execution of the text formatter depends on the method used to compiled the text formatter. After the text formatter has been executed, its output can be printed using the osprint or type commands. If the previously described input file was submitted to the text formatter, the output would start on a new page and look as follows:

A Format Test

This file is a test for the RatFor text formatter. The text formatter will neatly format the contents of this file. The title will be centered and underlined. The text material will be arranged in 60 character lines.
The text formatter uses the computer to do work that a typist normally does.

The output of the text formatter reflects the commands issued. The text formatter takes 2.37 and 4.07 CPU seconds to format 50 and 100 lines of input respectively. The listing of the RatFor version of the text formatter can be found in Appendix B of this report.

CHAPTER 5

CONCLUSIONS

The RatFor Software Package contains the preprocessor and many tools. Detailed explanations of the preprocessor and the tools can be found in Software Tools. Using the book Software Tools and this report, a user can implement the RatFor preprocessor and tools in the IBM 370's CMS environment.

The IBM 370's job control language was a major obstacle to the preparation of this report. It was an obstacle in that there was no single source to be referenced. The procedures outlined in this report contains information derived from IBM manuals, computing center consultants, instructors, and my personal knowledge.

The RatFor code is easy to read and write. Therefore, necessary changes could be made without fear of introducing perplexing errors intc the code. Reading and revising are keywords in programming and the RatFor programs are well suited for both. Each of the modules that make up a RatFor tool rarely spreads over more than one page. The modules are cohesive in that they can be taken out of one tool and used in another tool without any modification being made to them.

Recommendations for further improvement of the RatFor Software Package are:

1. Create a magnetic tape that is compatible with the IBM 370.

2. Write the preprocessor and each tool on the tape as individual files. However, insure that each tool is complete before being written to tape. By this I mean, all modules, define statements, and block data information of a tool should be combined before the tool is written to tape.

3. Make a table of contents for the tape so that the user will know what file to access when a specific tool is desired. List the commands that are necessary to read the tape and to bring the file up at an interactive terminal. Once the file is brought to a terminal, the procedures outlined in this report can be used.

BIBLIOGRAPHY

1. Conrow, Kenneth. The CMS Cookbook. K.S.U. Computing Center, Manhattan,Kansas (1976).

2. The CP/CMS Guide, K.S.U. Computing Center, Manhattan,Kansas (1977).

3. B. W. Kernighan and P.J. Plauger. Software Tools. Addison-Wesley Publishing Company. Reading, Massachusetts.

4. IBM System/360 OS, Fortran IV Programmer's Guide. Publication No. GC28-6817-4. IBM Corp. New York, New York (1973).

5. IBM System/360 OS: Messages and Codes; Publication No. GC28-6631-13. IBM Corp. New york, New York (1973).

6. IBM Virtual Machine Facility/370. CMS Command and Macro Reference. Publication No. GC20-1818-0. IBM Corp. White Plains, New York (1976).

APPENDIX A

RatFor Preprocessor Listing

```
C========= RATIONAL FORTRAN PREPROCESSOR =========
C
C RATFOR - MAIN PROGRAM FOR RATFOR
C
      WRITE(9,10)
   10 FORMAT('0    *** BEGIN RATFOR PREPROCESSOR ***'/'0',
     *'    *** WAIT ***')
      CALL PARSE
      WRITE(9,20)
   20 FORMAT('0    *** END RATFOR PREPROCESSOR ***')
      STOP
      END
C
C BLOCK DATA - INITIALIZE GLOBAL VARIABLES
C
      BLOCK DATA
      COMMON /CCHAR/ EXTDIG(10), INTDIG(10), EXTLET(26), INTLET(26),  E
     *XTBIG(26), INTBIG(26), EXTCHR(33), INTCHR(33),  EXTBLK, INTBLK
      INTEGER EXTDIG
      INTEGER INTDIG
      INTEGER EXTLET
      INTEGER INTLET
      INTEGER EXTBIG
      INTEGER INTBIG
      INTEGER EXTCHR
      INTEGER INTCHR
      INTEGER EXTBLK
      INTEGER INTBLK
      COMMON /CDEFIO/ BP, BUF(300)
      INTEGER BP
      INTEGER BUF
      COMMON /CFOR/ FORDEP, FORSTK(200)
      INTEGER FORDEP
      INTEGER FORSTK
      COMMON /CKEYWD/ SDO, SIF, SELSE, SWHILE, SBREAK, SNEXT, SFOR, SRE
     *PT, SUNTIL, VDO, VIF, VELSE, VWHILE, VBREAK, VNEXT, VFOR, VREPT, V
     *UNTIL
      INTEGER SDO(3), SIF(3), SELSE(5), SWHILE(6), SBREAK(6), SNEXT(5)
```

```
      INTEGER SFOR(4), SREPT(7), SUNTIL(6)
      INTEGER VDO(2), VIF(2), VELSE(2), VWHILE(2), VBREAK(2), VNEXT(2)
      INTEGER VFOR(2), VREPT(2), VUNTIL(2)
      COMMON /CLINE/ LEVEL, LINECT(5), INFILE(5)
      INTEGER LEVEL
      INTEGER LINECT
      INTEGER INFILE
      COMMON /CLOOK/ LASTP, LASTT, NAMPTR(200), TABLE(1500)
      INTEGER LASTP
      INTEGER LASTT
      INTEGER NAMPTR
      INTEGER TABLE
      COMMON /COUTLN/ OUTP, OUTBUF(81)
      INTEGER OUTP
      INTEGER OUTBUF
      DATA OUTP /0/
      DATA LEVEL /1/
      DATA LINECT(1) /1/
      DATA INFILE(1) /5/
      DATA BP /0/
      DATA FORDEP /0/
      DATA LASTP /0/
      DATA LASTT /0/
      DATA SDO(1), SDO(2), SDO(3) /100, 111, 10002/
      DATA VDO(1), VDO(2) /10266, 10002/
      DATA SIF(1), SIF(2), SIF(3) /105, 102, 10002/
      DATA VIF(1), VIF(2) /10261, 10002/
      DATA SELSE(1), SELSE(2), SELSE(3), SELSE(4), SELSE(5) /101,  108,
     * 115, 101, 10002/
      DATA VELSE(1), VELSE(2) /10262, 10002/
      DATA SWHILE(1), SWHILE(2), SWHILE(3), SWHILE(4), SWHILE(5), SWHIL
     *E(6) /119, 104, 105, 108, 101, 10002/
      DATA VWHILE(1), VWHILE(2) /10263, 10002/
      DATA SBREAK(1), SBREAK(2), SBREAK(3), SBREAK(4), SBREAK(5), SBREA
     *K(6) /98, 114, 101, 97, 107, 10002/
      DATA VBREAK(1), VBREAK(2) /10264, 10002/
      DATA SNEXT(1), SNEXT(2), SNEXT(3), SNEXT(4), SNEXT(5) /110,  101,
     * 120, 116, 10002/
```

```
      DATA VNEXT(1), VNEXT(2) /10265, 10002/
      DATA SFOR(1), SFOR(2), SFOR(3), SFOR(4) /102,  111, 114, 10002/
      DATA VFOR(1), VFOR(2) /10268, 10002/
      DATA SREPT(1), SREPT(2), SREPT(3), SREPT(4), SREPT(5), SREPT(6),
     * SREPT(7) /114, 101, 112, 101, 97, 116, 10002/
      DATA VREPT(1), VREPT(2) /10269, 10002/
      DATA SUNTIL(1), SUNTIL(2), SUNTIL(3), SUNTIL(4), SUNTIL(5), SUNTI
     *L(6) /117, 110, 116, 105, 108, 10002/
      DATA VUNTIL(1), VUNTIL(2) /10270, 10002/
      DATA EXTBLK /1H /, INTBLK /32/
      DATA EXTDIG(1) /1H0/, INTDIG(1) /48/
      DATA EXTDIG(2) /1H1/, INTDIG(2) /49/
      DATA EXTDIG(3) /1H2/, INTDIG(3) /50/
      DATA EXTDIG(4) /1H3/, INTDIG(4) /51/
      DATA EXTDIG(5) /1H4/, INTDIG(5) /52/
      DATA EXTDIG(6) /1H5/, INTDIG(6) /53/
      DATA EXTDIG(7) /1H6/, INTDIG(7) /54/
      DATA EXTDIG(8) /1H7/, INTDIG(8) /55/
      DATA EXTDIG(9) /1H8/, INTDIG(9) /56/
      DATA EXTDIG(10) /1H9/, INTDIG(10) /57/
C     EXTLET IS THE LOWER CASE REPRESENTATION OF THE ALPHABETS.  EACH
C     LOWER CASE ALPHABET IS REPRESENTED BY ITS HEXIDECIMAL VALUE.
C     THE HEXIDECIMAL VALUE IS REPRESENTED BY Z81404040, ETC.
      DATA EXTLET(1) /Z81404040/, INTLET(1) /97/
      DATA EXTLET(2) /Z82404040/, INTLET(2) /98/
      DATA EXTLET(3) /Z83404040/, INTLET(3) /99/
      DATA EXTLET(4) /Z84404040/, INTLET(4) /100/
      DATA EXTLET(5) /Z85404040/, INTLET(5) /101/
      DATA EXTLET(6) /Z86404040/, INTLET(6) /102/
      DATA EXTLET(7) /Z87404040/, INTLET(7) /103/
      DATA EXTLET(8) /Z88404040/, INTLET(8) /104/
      DATA EXTLET(9) /Z89404040/, INTLET(9) /105/
      DATA EXTLET(10) /Z91404040/, INTLET(10) /106/
      DATA EXTLET(11) /Z92404040/, INTLET(11) /107/
      DATA EXTLET(12) /Z93404040/, INTLET(12) /108/
      DATA EXTLET(13) /Z94404040/, INTLET(13) /109/
      DATA EXTLET(14) /Z95404040/, INTLET(14) /110/
      DATA EXTLET(15) /Z96404040/, INTLET(15) /111/
```

```
DATA EXTLET(16) /Z97404040/, INTLET(16) /112/
DATA EXTLET(17) /Z98404040/, INTLET(17) /113/
DATA EXTLET(18) /Z99404040/, INTLET(18) /114/
DATA EXTLET(19) /ZA2404040/, INTLET(19) /115/
DATA EXTLET(20) /ZA3404040/, INTLET(20) /116/
DATA EXTLET(21) /ZA4404040/, INTLET(21) /117/
DATA EXTLET(22) /ZA5404040/, INTLET(22) /118/
DATA EXTLET(23) /ZA6404040/, INTLET(23) /119/
DATA EXTLET(24) /ZA7404040/, INTLET(24) /120/
DATA EXTLET(25) /ZA8404040/, INTLET(25) /121/
DATA EXTLET(26) /ZA9404040/, INTLET(26) /122/
DATA EXTBIG(1) /1HA/, INTBIG(1) /65/
DATA EXTBIG(2) /1HB/, INTBIG(2) /66/
DATA EXTBIG(3) /1HC/, INTBIG(3) /67/
DATA EXTBIG(4) /1HD/, INTBIG(4) /68/
DATA EXTBIG(5) /1HE/, INTBIG(5) /69/
DATA EXTBIG(6) /1HF/, INTBIG(6) /70/
DATA EXTBIG(7) /1HG/, INTBIG(7) /71/
DATA EXTBIG(8) /1HH/, INTBIG(8) /72/
DATA EXTBIG(9) /1HI/, INTBIG(9) /73/
DATA EXTBIG(10) /1HJ/, INTBIG(10) /74/
DATA EXTBIG(11) /1HK/, INTBIG(11) /75/
DATA EXTBIG(12) /1HL/, INTBIG(12) /76/
DATA EXTBIG(13) /1HM/, INTBIG(13) /77/
DATA EXTBIG(14) /1HN/, INTBIG(14) /78/
DATA EXTBIG(15) /1HO/, INTBIG(15) /79/
DATA EXTBIG(16) /1HP/, INTBIG(16) /80/
DATA EXTBIG(17) /1HQ/, INTBIG(17) /81/
DATA EXTBIG(18) /1HR/, INTBIG(18) /82/
DATA EXTBIG(19) /1HS/, INTBIG(19) /83/
DATA EXTBIG(20) /1HT/, INTBIG(20) /84/
DATA EXTBIG(21) /1HU/, INTBIG(21) /85/
DATA EXTBIG(22) /1HV/, INTBIG(22) /86/
DATA EXTBIG(23) /1HW/, INTBIG(23) /87/
DATA EXTBIG(24) /1HX/, INTBIG(24) /88/
DATA EXTBIG(25) /1HY/, INTBIG(25) /89/
DATA EXTBIG(26) /1HZ/, INTBIG(26) /90/
DATA EXTCHR(1) /1H!/, INTCHR(1) /33/
```

```
      DATA EXTCHR(2) /1H"/, INTCHR(2) /34/
      DATA EXTCHR(3) /1H%/, INTCHR(3) /37/
      DATA EXTCHR(4) /1H$/, INTCHR(4) /36/
      DATA EXTCHR(5) /1H%/, INTCHR(5) /37/
      DATA EXTCHR(6) /1H&/, INTCHR(6) /38/
      DATA EXTCHR(7) /1H'/, INTCHR(7) /39/
      DATA EXTCHR(8) /1H(/, INTCHR(8) /40/
      DATA EXTCHR(9) /1H)/, INTCHR(9) /41/
      DATA EXTCHR(10) /1H*/, INTCHR(10) /42/
      DATA EXTCHR(11) /1H+/, INTCHR(11) /43/
      DATA EXTCHR(12) /1H,/, INTCHR(12) /44/
      DATA EXTCHR(13) /1H-/, INTCHR(13) /45/
      DATA EXTCHR(14) /1H./, INTCHR(14) /46/
      DATA EXTCHR(15) /1H//, INTCHR(15) /47/
      DATA EXTCHR(16) /1H:/, INTCHR(16) /58/
      DATA EXTCHR(17) /1H;/, INTCHR(17) /59/
      DATA EXTCHR(18) /1H</, INTCHR(18) /60/
      DATA EXTCHR(19) /1H=/, INTCHR(19) /61/
      DATA EXTCHR(20) /1H>/, INTCHR(20) /62/
      DATA EXTCHR(21) /1H?/, INTCHR(21) /63/
      DATA EXTCHR(22) /1H@/, INTCHR(22) /64/
      DATA EXTCHR(23) /1H[/, INTCHR(23) /91/
      DATA EXTCHR(24) /1H /, INTCHR(24) /92/
      DATA EXTCHR(25) /1H ]/, INTCHR(25) /93/
      DATA EXTCHR(26) /1H_/, INTCHR(26) /95/
      DATA EXTCHR(27) /1H /, INTCHR(27) /123/
      DATA EXTCHR(28) /1H|/, INTCHR(28) /124/
      DATA EXTCHR(29) /1H /, INTCHR(29) /125/
      DATA EXTCHR(30) /1H /, INTCHR(30) /8/
      DATA EXTCHR(31) /1H /, INTCHR(31) /9/
      DATA EXTCHR(32) /1H¬/, INTCHR(32) /33/
      DATA EXTCHR(33) /1H°/, INTCHR(33) /33/
      END
C
C ALLDIG - RETURN YES IF STR IS ALL DIGITS
C
      INTEGER FUNCTION ALLDIG(STR)
      INTEGER TYPE
```

```
      INTEGER STR(100)
      INTEGER I
      ALLDIG = 0
      IF(.NOT.(STR(1) .EQ. 10002))  GOTO 23000
      RETURN
23000 CONTINUE
      CONTINUE
      I = 1
23002 IF(.NOT.( STR(I) .NE. 10002)) GOTO 23004
      IF(.NOT.(TYPE(STR(I)) .NE. 2))   GOTO 23005
      RETURN
23005 CONTINUE
23003  I = I + 1
      GOTO 23002
23004 CONTINUE
      ALLDIG = 1
      RETURN
      END
C
C BALPAR - COPY BALANCED PAREN STRING
C
      SUBROUTINE BALPAR
      INTEGER GETTOK
      INTEGER T, TOKEN(200)
      INTEGER NLPAR
      IF(.NOT.(GETTOK(TOKEN, 200) .NE. 40))  GOTO 23007
      CALL SYNERR(19HMISSING LEFT PAREN.)
      RETURN
23007 CONTINUE
      CALL OUTSTR(TOKEN)
      NLPAR = 1
      CONTINUE
23009 CONTINUE
      T = GETTOK(TOKEN, 200)
      IF(.NOT.(T.EQ.59 .OR. T.EQ.123 .OR. T.EQ.125 .OR. T.EQ.10003)) GO
     *TO 23012
      CALL PBSTR(TOKEN)
      GOTO 23011
```

```
23012  CONTINUE
       IF(.NOT.(T .EQ. 10)) GOTO 23014
       TOKEN(1) = 10002
       GOTO 23015
23014  CONTINUE
       IF(.NOT.(T .EQ. 40)) GOTO 23016
       NLPAR = NLPAR + 1
       GOTO 23017
23016  CONTINUE
       IF(.NOT.(T .EQ. 41)) GOTO 23018
       NLPAR = NLPAR - 1
23018  CONTINUE
23017  CONTINUE
23015  CONTINUE
       CALL OUTSTR(TOKEN)
23010  IF(.NOT.(NLPAR .LE. 0)) GOTO 23009
23011  CONTINUE
       IF(.NOT.(NLPAR .NE. 0)) GOTO 23020
       CALL SYNERR(33HMISSING PARENTHESIS IN CONDITION.)
23020  CONTINUE
       RETURN
       END
C
C BRKNXT - GENERATE CODE FOR BREAK AND NEXT
C
       SUBROUTINE BRKNXT(SP, LEXTYP, LABVAL, TOKEN)
       INTEGER I, LABVAL(100), LEXTYP(100), SP, TOKEN
       CONTINUE
       I = SP
23022  IF(.NOT.( I .GT. 0)) GOTO 23024
       IF(.NOT.(LEXTYP(I) .EQ. 10263 .OR. LEXTYP(I) .EQ. 10266       .OR
     *. LEXTYP(I) .EQ. 10268 .OR. LEXTYP(I) .EQ. 10269))  GOTO 23025
       IF(.NOT.(TOKEN .EQ. 10264))   GOTO 23027
       CALL OUTGO(LABVAL(I)+1)
       GOTO 23028
23027  CONTINUE
       CALL OUTGO(LABVAL(I))
23028  CONTINUE
```

```
      RETURN
23025 CONTINUE
23023  I = I - 1
      GOTO 23022
23024 CONTINUE
      IF(.NOT.(TOKEN .EQ. 10264))   GOTO 23029
      CALL SYNERR(14HILLEGAL BREAK.)
      GOTO 23030
23029 CONTINUE
      CALL SYNERR(13HILLEGAL NEXT.)
23030 CONTINUE
      RETURN
      END
C
C CLOSE - EXCEEDINGLY TEMPORARY VERSION FOR GETTOK
C
      SUBROUTINE CLOSE(FD)
      INTEGER FD
      REWIND FD
      RETURN
      END
C
C CTOI - CONVERT STRING AT IN(I) TO INTEGER, INCREMENT I
C
      INTEGER FUNCTION CTOI(IN, I)
      INTEGER IN(100)
      INTEGER INDEX
      INTEGER D, I
      INTEGER DIGITS(11)
      DATA DIGITS(1) /48/
      DATA DIGITS(2) /49/
      DATA DIGITS(3) /50/
      DATA DIGITS(4) /51/
      DATA DIGITS(5) /52/
      DATA DIGITS(6) /53/
      DATA DIGITS(7) /54/
      DATA DIGITS(8) /55/
      DATA DIGITS(9) /56/
```

```
      DATA DIGITS(10) /57/
      DATA DIGITS(11) /10002/
      CONTINUE
23031 IF(.NOT.(IN(I) .EQ. 32 .OR. IN(I) .EQ. 9))   GOTO 23032
      I = I + 1
      GOTO 23031
23032 CONTINUE
      CONTINUE
      CTOI = 0
23033 IF(.NOT.( IN(I) .NE. 10002))  GOTO 23035
      D = INDEX(DIGITS, IN(I))
      IF(.NOT.(D .EQ. 0))  GOTO 23036
      GOTO 23035
23036 CONTINUE
      CTOI = 10 * CTOI + D - 1
23034  I = I + 1
      GOTO 23033
23035 CONTINUE
      RETURN
      END
C
C DEFTOK - GET TOKEN; PROCESS MACRO CALLS AND INVOCATIONS
C
      INTEGER FUNCTION DEFTOK(TOKEN, TOKSIZ, FD)
      INTEGER GTOK
      INTEGER FD, TOKSIZ
      INTEGER DEFN(200), T, TOKEN(TOKSIZ)
      INTEGER LOOKUP
      CONTINUE
      T=GTOK(TOKEN, TOKSIZ, FD)
23038 IF(.NOT.( T.NE.10003))  GOTO 23040
      IF(.NOT.(T .NE. 10100)) GOTO 23041
      GOTO 23040
23041 CONTINUE
      IF(.NOT.(LOOKUP(TOKEN, DEFN) .EQ. 0))  GOTO 23043
      GOTO 23040
23043 CONTINUE
      IF(.NOT.(DEFN(1) .EQ. 10010)) GOTO 23045
```

```
      CALL GETDEF(TOKEN, TOKSIZ, DEFN, 200, FD)
      CALL INSTAL(TOKEN, DEFN)
      GOTO 23046
23045 CONTINUE
      CALL PBSTR(DEFN)
23046 CONTINUE
23039  T=GTOK(TOKEN, TOKSIZ, FD)
      GOTO 23038
23040 CONTINUE
      DEFTOK = T
      IF(.NOT.(DEFTOK .EQ. 10100))  GOTO 23047
      CALL FOLD(TOKEN)
23047 CONTINUE
      RETURN
      END
C
C FOLD - CONVERT ALPHABETIC TOKEN TO SINGLE CASE
C
      SUBROUTINE FOLD(TOKEN)
      INTEGER TOKEN(100)
      INTEGER I
      CONTINUE
      I = 1
23049 IF(.NOT.( TOKEN(I) .NE. 10002))  GOTO 23051
      IF(.NOT.(TOKEN(I) .GE. 65 .AND. TOKEN(I) .LE. 90)) GOTO 23052
      TOKEN(I) = TOKEN(I) - 65 + 97
23052 CONTINUE
23050  I = I + 1
      GOTO 23049
23051 CONTINUE
      RETURN
      END
C
C DOCODE - GENERATE CODE FOR BEGINNING OF DO
C
      SUBROUTINE DOCODE(LAB)
      INTEGER LABGEN
      INTEGER LAB
```

```
      INTEGER DOSTR(4)
      DATA DOSTR(1), DOSTR(2), DOSTR(3), DOSTR(4)/100, 111, 32, 10002/
      CALL OUTTAB
      CALL OUTSTR(DOSTR)
      LAB = LABGEN(2)
      CALL OUTNUM(LAB)
      CALL EATUP
      CALL OUTDON
      RETURN
      END
C
C DOSTAT - GENERATE CODE FOR END OF DO STATEMENT
C
      SUBROUTINE DOSTAT(LAB)
      INTEGER LAB
      CALL OUTCON(LAB)
      CALL OUTCON(LAB+1)
      RETURN
      END
C
C EATUP - PROCESS REST OF STATEMENT; INTERPRET CONTINUATIONS
C
      SUBROUTINE EATUP
      INTEGER GETTOK
      INTEGER PTOKEN(200), T, TOKEN(200)
      INTEGER NLPAR
      NLPAR = 0
      CONTINUE
23054 CONTINUE
      T = GETTOK(TOKEN, 200)
      IF(.NOT.(T .EQ. 59 .OR. T .EQ. 10)) GOTO 23057
      GOTO 23056
23057 CONTINUE
      IF(.NOT.(T .EQ. 125))   GOTO 23059
      CALL PBSTR(TOKEN)
      GOTO 23056
23059 CONTINUE
      IF(.NOT.(T .EQ. 123 .OR. T .EQ. 10003))   GOTO 23061
```

```
      CALL SYNERR(24HUNEXPECTED BRACE OR EOF.)
      CALL PBSTR(TOKEN)
      GOTO 23056
23061 CONTINUE
      IF(.NOT.(T .EQ. 44 .OR. T .EQ. 95)) GOTO 23063
      IF(.NOT.(GETTOK(PTOKEN, 200) .NE. 10)) GOTO 23065
      CALL PBSTR(PTOKEN)
23065 CONTINUE
      IF(.NOT.(T .EQ. 95)) GOTO 23067
      TOKEN(1) = 10002
23067 CONTINUE
      GOTO 23064
23063 CONTINUE
      IF(.NOT.(T .EQ. 40)) GOTO 23069
      NLPAR = NLPAR + 1
      GOTO 23070
23069 CONTINUE
      IF(.NOT.(T .EQ. 41)) GOTO 23071
      NLPAR = NLPAR - 1
23071 CONTINUE
23070 CONTINUE
23064 CONTINUE
      CALL OUTSTR(TOKEN)
23055 IF(.NOT.(NLPAR .LT. 0)) GOTO 23054
23056 CONTINUE
      IF(.NOT.(NLPAR .NE. 0)) GOTO 23073
      CALL SYNERR(23HUNBALANCED PARENTHESES.)
23073 CONTINUE
      RETURN
      END
C
C ELSEIF - GENERATE CODE FOR END OF IF BEFORE ELSE
C
      SUBROUTINE ELSEIF(LAB)
      INTEGER LAB
      CALL OUTGO(LAB+1)
      CALL OUTCON(LAB)
      RETURN
```

```
      END
C
C EQUAL - COMPARE STR1 TO STR2; RETURN YES IF EQUAL, NO IF NOT
C
      INTEGER FUNCTION EQUAL(STR1, STR2)
      INTEGER STR1(100), STR2(100)
      INTEGER I
      CONTINUE
      I = 1
23075 IF(.NOT.( STR1(I) .EQ. STR2(I))) GOTO 23077
      IF(.NOT.(STR1(I) .EQ. 10002)) GOTO 23078
      EQUAL = 1
      RETURN
23078 CONTINUE
23076  I = I + 1
      GOTO 23075
23077 CONTINUE
      EQUAL = 0
      RETURN
      END
C
C ERROR - PRINT FATAL ERROR MESSAGE, THEN DIE
C
      SUBROUTINE ERROR(BUF)
      INTEGER BUF(100)
      CALL REMARK(BUF)
      STOP
      END
C
C FORCOD - BEGINNING OF FOR STATEMENT
C
      SUBROUTINE FORCOD(LAB)
      INTEGER GETTOK
      INTEGER T, TOKEN(200)
      INTEGER LENGTH, LABGEN
      INTEGER I, J, LAB, NLPAR
      COMMON /CCHAR/ EXTDIG(10), INTDIG(10), EXTLET(26), INTLET(26),  E
     *XTBIG(26), INTBIG(26), EXTCHR(33), INTCHR(33),  EXTBLK, INTBLK
```

```
      INTEGER EXTDIG
      INTEGER INTDIG
      INTEGER EXTLET
      INTEGER INTLET
      INTEGER EXTBIG
      INTEGER INTBIG
      INTEGER EXTCHR
      INTEGER INTCHR
      INTEGER EXTBLK
      INTEGER INTBLK
      COMMON /CDEFIO/ BP, BUF(300)
      INTEGER BP
      INTEGER BUF
      COMMON /CFOR/ FORDEP, FORSTK(200)
      INTEGER FORDEP
      INTEGER FORSTK
      COMMON /CKEYWD/ SDO, SIF, SELSE, SWHILE, SBREAK, SNEXT, SFOR, SRE
     *PT, SUNTIL, VDO, VIF, VELSE, VWHILE, VBREAK, VNEXT, VFOR, VREPT, V
     *UNTIL
      INTEGER SDO(3), SIF(3), SELSE(5), SWHILE(6), SBREAK(6), SNEXT(5)
      INTEGER SFOR(4), SREPT(7), SUNTIL(6)
      INTEGER VDO(2), VIF(2), VELSE(2), VWHILE(2), VBREAK(2), VNEXT(2)
      INTEGER VFOR(2), VREPT(2), VUNTIL(2)
      COMMON /CLINE/ LEVEL, LINECT(5), INFILE(5)
      INTEGER LEVEL
      INTEGER LINECT
      INTEGER INFILE
      COMMON /CLOOK/ LASTP, LASTT, NAMPTR(200), TABLE(1500)
      INTEGER LASTP
      INTEGER LASTT
      INTEGER NAMPTR
      INTEGER TABLE
      COMMON /COUTLN/ OUTP, OUTBUF(81)
      INTEGER OUTP
      INTEGER OUTBUF
      INTEGER IFNOT(9)
      DATA IFNOT(1) /105/
      DATA IFNOT(2) /102/
```

```
      DATA IFNOT(3) /40/
      DATA IFNOT(4) /46/
      DATA IFNOT(5) /110/
      DATA IFNOT(6) /111/
      DATA IFNOT(7) /116/
      DATA IFNOT(8) /46/
      DATA IFNOT(9) /10002/
      LAB = LABGEN(3)
      CALL OUTCON(0)
      IF(.NOT.(GETTOK(TOKEN, 200) .NE. 40))  GOTO 23080
      CALL SYNERR(19HMISSING LEFT PAREN.)
      RETURN
23080 CONTINUE
      IF(.NOT.(GETTOK(TOKEN, 200) .NE. 59))  GOTO 23082
      CALL PBSTR(TOKEN)
      CALL OUTTAB
      CALL EATUP
      CALL OUTDON
23082 CONTINUE
      IF(.NOT.(GETTOK(TOKEN, 200) .EQ. 59))  GOTO 23084
      CALL OUTCON(LAB)
      GOTO 23085
23084 CONTINUE
      CALL PBSTR(TOKEN)
      CALL OUTNUM(LAB)
      CALL OUTTAB
      CALL OUTSTR(IFNOT)
      CALL OUTCH(40)
      NLPAR = 0
      CONTINUE
23086 IF(.NOT.(NLPAR .GE. 0)) GOTO 23087
      T = GETTOK(TOKEN, 200)
      IF(.NOT.(T .EQ. 59)) GOTO 23088
      GOTO 23087
23088 CONTINUE
      IF(.NOT.(T .EQ. 40)) GOTO 23090
      NLPAR = NLPAR + 1
      GOTO 23091
```

```
23090  CONTINUE
       IF(.NOT.(T .EQ. 41)) GOTO 23092
       NLPAR = NLPAR - 1
23092  CONTINUE
23091  CONTINUE
       IF(.NOT.(T .NE. 10 .AND. T .NE. 95))    GOTO 23094
       CALL OUTSTR(TOKEN)
23094  CONTINUE
       GOTO 23086
23087  CONTINUE
       CALL OUTCH(41)
       CALL OUTCH(41)
       CALL OUTGO(LAB+2)
       IF(.NOT.(NLPAR .LT. 0)) GOTO 23096
       CALL SYNERR(19HINVALID FOR CLAUSE.)
23096  CONTINUE
23085  CONTINUE
       FORDEP = FORDEP + 1
       J = 1
       CONTINUE
       I = 1
23098  IF(.NOT.( I .LT. FORDEP))  GOTO 23100
       J = J + LENGTH(FORSTK(J)) + 1
23099   I = I + 1
       GOTO 23098
23100  CONTINUE
       FORSTK(J) = 10002
       NLPAR = 0
       CONTINUE
23101  IF(.NOT.(NLPAR .GE. 0)) GOTO 23102
       T = GETTOK(TOKEN, 200)
       IF(.NOT.(T .EQ. 40)) GOTO 23103
       NLPAR = NLPAR + 1
       GOTO 23104
23103  CONTINUE
       IF(.NOT.(T .EQ. 41)) GOTO 23105
       NLPAR = NLPAR - 1
23105  CONTINUE
```

```
23104  CONTINUE
       IF(.NOT.(NLPAR .GE. 0 .AND. T .NE. 10 .AND. T .NE. 95))  GOTO 231
     *07
       CALL SCOPY(TOKEN, 1, FORSTK, J)
       J = J + LENGTH(TOKEN)
23107  CONTINUE
       GOTO 23101
23102  CONTINUE
       LAB = LAB + 1
       RETURN
       END
C
C FORS - PROCESS END OF FOR STATEMENT
C
       SUBROUTINE FORS(LAB)
       INTEGER LENGTH
       INTEGER I, J, LAB
       COMMON /CCHAR/ EXTDIG(10), INTDIG(10), EXTLET(26), INTLET(26),  E
     *XTBIG(26), INTBIG(26), EXTCHR(33), INTCHR(33),  EXTBLK, INTBLK
       INTEGER EXTDIG
       INTEGER INTDIG
       INTEGER EXTLET
       INTEGER INTLET
       INTEGER EXTBIG
       INTEGER INTBIG
       INTEGER EXTCHR
       INTEGER INTCHR
       INTEGER EXTBLK
       INTEGER INTBLK
       COMMON /CDEFIO/ BP, BUF(300)
       INTEGER BP
       INTEGER BUF
       COMMON /CFOR/ FORDEP, FORSTK(200)
       INTEGER FORDEP
       INTEGER FORSTK
       COMMON /CKEYWD/ SDO, SIF, SELSE, SWHILE, SBREAK, SNEXT, SFOR, SRE
     *PT, SUNTIL, VDO, VIF, VELSE, VWHILE, VBREAK, VNEXT, VFOR, VREPT, V
     *UNTIL
```

```
      INTEGER SDO(3), SIF(3), SELSE(5), SWHILE(6), SBREAK(6), SNEXT(5)
      INTEGER SFOR(4), SREPT(7), SUNTIL(6)
      INTEGER VDO(2), VIF(2), VELSE(2), VWHILE(2), VBREAK(2), VNEXT(2)
      INTEGER VFOR(2), VREPT(2), VUNTIL(2)
      COMMON /CLINE/ LEVEL, LINECT(5), INFILE(5)
      INTEGER LEVEL
      INTEGER LINECT
      INTEGER INFILE
      COMMON /CLOOK/ LASTP, LASTT, NAMPTR(200), TABLE(1500)
      INTEGER LASTP
      INTEGER LASTT
      INTEGER NAMPTR
      INTEGER TABLE
      COMMON /COUTLN/ OUTP, OUTBUF(81)
      INTEGER OUTP
      INTEGER OUTBUF
      CALL OUTNUM(LAB)
      J = 1
      CONTINUE
      I = 1
23109 IF(.NOT.( I .LT. FORDEP))  GOTO 23111
      J = J + LENGTH(FORSTK(J)) + 1
23110  I = I + 1
      GOTO 23109
23111 CONTINUE
      IF(.NOT.(LENGTH(FORSTK(J)) .GT. 0)) GOTO 23112
      CALL OUTTAB
      CALL OUTSTR(FORSTK(J))
      CALL OUTDON
23112 CONTINUE
      CALL OUTGO(LAB-1)
      CALL OUTCON(LAB+1)
      FORDEP = FORDEP - 1
      RETURN
      END
C
C GETCH - GET CHARACTERS FROM FILE
C
```

```
      INTEGER FUNCTION GETCH(C, F)
      INTEGER INMAP
      INTEGER BUF(81), C
      INTEGER F, I, LASTC
      DATA LASTC /81/, BUF(81) /10/
      IF(.NOT.(BUF(LASTC) .EQ. 10 .OR. LASTC .GE. 81))   GOTO 23114
      READ(F, 1, END=10) (BUF(I), I = 1, 80)
1        FORMAT(80 A1)
      CONTINUE
      I = 1
23116 IF(.NOT.( I .LE. 80))   GOTO 23118
      BUF(I) = INMAP(BUF(I))
23117  I = I + 1
      GOTO 23116
23118 CONTINUE
      CONTINUE
      I = 80
23119 IF(.NOT.( I .GT. 0)) GOTO 23121
      IF(.NOT.(BUF(I) .NE. 32))  GOTO 23122
      GOTO 23121
23122 CONTINUE
23120  I = I - 1
      GOTO 23119
23121 CONTINUE
      BUF(I+1) = 10
      LASTC = 0
23114 CONTINUE
      LASTC = LASTC + 1
      C = BUF(LASTC)
      GETCH = C
      RETURN
10       C = 10003
      GETCH = 10003
      RETURN
      END
C
C GETDEF (FOR NO ARGUMENTS) - GET NAME AND DEFINITION
C
```

```
      SUBROUTINE GETDEF(TOKEN, TOKSIZ, DEFN, DEFSIZ, FD)
      INTEGER GTOK, NGETCH
      INTEGER DEFSIZ, FD, I, NLPAR, TOKSIZ
      INTEGER C, DEFN(DEFSIZ), TOKEN(TOKSIZ)
      IF(.NOT.(NGETCH(C, FD) .NE. 40)) GOTO 23124
      CALL REMARK(19HMISSING LEFT PAREN.)
23124 CONTINUE
      IF(.NOT.(GTOK(TOKEN, TOKSIZ, FD) .NE. 10100))   GOTO 23126
      CALL REMARK(22HNON-ALPHANUMERIC NAME.)
      GOTO 23127
23126 CONTINUE
      IF(.NOT.(NGETCH(C, FD) .NE. 44)) GOTO 23128
      CALL REMARK(24HMISSING COMMA IN DEFINE.)
23128 CONTINUE
23127 CONTINUE
      NLPAR = 0
      CONTINUE
      I = 1
23130 IF(.NOT.( NLPAR .GE. 0))   GOTO 23132
      IF(.NOT.(I .GT. DEFSIZ))   GOTO 23133
      CALL ERROR(20HDEFINITION TOO LONG.)
      GOTO 23134
23133 CONTINUE
      IF(.NOT.(NGETCH(DEFN(I), FD) .EQ. 10003)) GOTO 23135
      CALL ERROR(20HMISSING RIGHT PAREN.)
      GOTO 23136
23135 CONTINUE
      IF(.NOT.(DEFN(I) .EQ. 40)) GOTO 23137
      NLPAR = NLPAR + 1
      GOTO 23138
23137 CONTINUE
      IF(.NOT.(DEFN(I) .EQ. 41)) GOTO 23139
      NLPAR = NLPAR - 1
23139 CONTINUE
23138 CONTINUE
23136 CONTINUE
23134 CONTINUE
23131  I = I + 1
```

```
      GOTO 23130
23132 CONTINUE
      DEFN(I-1) = 10002
      RETURN
      END
C
C GETTOK - GET TOKEN. HANDLES FILE INCLUSION AND LINE NUMBERS
C
      INTEGER FUNCTION GETTOK(TOKEN, TOKSIZ)
      INTEGER EQUAL, OPEN
      INTEGER JUNK, TOKSIZ
      INTEGER DEFTOK
      INTEGER NAME(30), TOKEN(TOKSIZ)
      COMMON /CCHAR/ EXTDIG(10), INTDIG(10), EXTLET(26), INTLET(26),  E
     *XTBIG(26), INTBIG(26), EXTCHR(33), INTCHR(33),  EXTBLK, INTBLK
      INTEGER EXTDIG
      INTEGER INTDIG
      INTEGER EXTLET
      INTEGER INTLET
      INTEGER EXTBIG
      INTEGER INTBIG
      INTEGER EXTCHR
      INTEGER INTCHR
      INTEGER EXTBLK
      INTEGER INTBLK
      COMMON /CDEFIO/ BP, BUF(300)
      INTEGER BP
      INTEGER BUF
      COMMON /CFOR/ FORDEP, FORSTK(200)
      INTEGER FORDEP
      INTEGER FORSTK
      COMMON /CKEYWD/ SDO, SIF, SELSE, SWHILE, SBREAK, SNEXT, SFOR, SRE
     *PT, SUNTIL, VDO, VIF, VELSE, VWHILE, VBREAK, VNEXT, VFOR, VREPT, V
     *UNTIL
      INTEGER SDO(3), SIF(3), SELSE(5), SWHILE(6), SBREAK(6), SNEXT(5)
      INTEGER SFOR(4), SREPT(7), SUNTIL(6)
      INTEGER VDO(2), VIF(2), VELSE(2), VWHILE(2), VBREAK(2), VNEXT(2)
      INTEGER VFOR(2), VREPT(2), VUNTIL(2)
```

```
      COMMON /CLINE/ LEVEL, LINECT(5), INFILE(5)
      INTEGER LEVEL
      INTEGER LINECT
      INTEGER INFILE
      COMMON /CLOOK/ LASTP, LASTT, NAMPTR(200), TABLE(1500)
      INTEGER LASTP
      INTEGER LASTT
      INTEGER NAMPTR
      INTEGER TABLE
      COMMON /COUTLN/ OUTP, OUTBUF(81)
      INTEGER OUTP
      INTEGER OUTBUF
      INTEGER INCL(8), INC(8)
      DATA INCL(1) /105/, INC(1) /73/
      DATA INCL(2) /110/, INC(2) /78/
      DATA INCL(3) /99/, INC(3) /67/
      DATA INCL(4) /108/, INC(4) /76/
      DATA INCL(5) /117/, INC(5) /85/
      DATA INCL(6) /100/, INC(6) /68/
      DATA INCL(7) /101/, INC(7) /69/
      DATA INCL(8) /10002/, INC(8) /10002/
      CONTINUE
23141 IF(.NOT.( LEVEL .GT. 0))   GOTO 23143
      CONTINUE
      GETTOK = DEFTOK(TOKEN, TOKSIZ, INFILE(LEVEL))
23144 IF(.NOT.( GETTOK .NE. 10003)) GOTO 23146
      IF(.NOT.(EQUAL(TOKEN, INCL) .EQ. 0))   GOTO 23147
      IF(.NOT.(EQUAL(TOKEN, INC) .EQ. 0)) GO TO 23147
      RETURN
23147 CONTINUE
      JUNK = DEFTOK(NAME, 30, INFILE(LEVEL))
      IF(.NOT.(LEVEL .GE. 5)) GOTO 23149
      CALL SYNERR(27HINCLUDES NESTED TOO DEEPLY.)
      GOTO 23150
23149 CONTINUE
      INFILE(LEVEL+1) = OPEN(NAME, 0)
      LINECT(LEVEL+1) = 1
      IF(.NOT.(INFILE(LEVEL+1) .EQ. 10001))  GOTO 23151
```

```
      CALL SYNERR(19HCAN'T OPEN INCLUDE.)
      GOTO 23152
23151 CONTINUE
      LEVEL = LEVEL + 1
23152 CONTINUE
23150 CONTINUE
23145          GETTOK = DEFTOK(TOKEN, TOKSIZ, INFILE(LEVEL))
      GOTO 23144
23146 CONTINUE
      IF(.NOT.(LEVEL .GT. 1)) GOTO 23153
      CALL CLOSE(INFILE(LEVEL))
23153 CONTINUE
23142  LEVEL = LEVEL - 1
      GOTO 23141
23143 CONTINUE
      GETTOK = 10003
      RETURN
      END
C
C GTOK - GET TOKEN FOR RATFOR
C
      INTEGER FUNCTION GTOK(LEXSTR, TOKSIZ, FD)
      INTEGER NGETCH, TYPE
      INTEGER FD, I, TOKSIZ
      INTEGER C, LEXSTR(TOKSIZ)
      COMMON /CCHAR/ EXTDIG(10), INTDIG(10), EXTLET(26), INTLET(26),  E
     *XTBIG(26), INTBIG(26), EXTCHR(33), INTCHR(33),  EXTBLK, INTBLK
      INTEGER EXTDIG
      INTEGER INTDIG
      INTEGER EXTLET
      INTEGER INTLET
      INTEGER EXTBIG
      INTEGER INTBIG
      INTEGER EXTCHR
      INTEGER INTCHR
      INTEGER EXTBLK
      INTEGER INTBLK
      COMMON /CDEFIO/ BP, BUF(300)
```

```
      INTEGER BP
      INTEGER BUF
      COMMON /CFOR/ FORDEP, FORSTK(200)
      INTEGER FORDEP
      INTEGER FORSTK
      COMMON /CKEYWD/ SDO, SIF, SELSE, SWHILE, SBREAK, SNEXT, SFOR, SRE
     *PT, SUNTIL, VDO, VIF, VELSE, VWHILE, VBREAK, VNEXT, VFOR, VREPT, V
     *UNTIL
      INTEGER SDO(3), SIF(3), SELSE(5), SWHILE(6), SBREAK(6), SNEXT(5)
      INTEGER SFOR(4), SREPT(7), SUNTIL(6)
      INTEGER VDO(2), VIF(2), VELSE(2), VWHILE(2), VBREAK(2), VNEXT(2)
      INTEGER VFOR(2), VREPT(2), VUNTIL(2)
      COMMON /CLINE/ LEVEL, LINECT(5), INFILE(5)
      INTEGER LEVEL
      INTEGER LINECT
      INTEGER INFILE
      COMMON /CLOCK/ LASTP, LASTT, NAMPTR(200), TABLE(1500)
      INTEGER LASTP
      INTEGER LASTT
      INTEGER NAMPTR
      INTEGER TABLE
      COMMON /COUTLN/ OUTP, OUTBUF(81)
      INTEGER OUTP
      INTEGER OUTBUF
      CONTINUE
23155 IF(.NOT.(NGETCH(C, FD) .NE. 10003)) GOTO 23156
      IF(.NOT.(C .NE. 32 .AND. C .NE. 9)) GOTO 23157
      GOTO 23156
23157 CONTINUE
      GOTO 23155
23156 CONTINUE
      CALL PUTBAK(C)
      CONTINUE
      I = 1
23159 IF(.NOT.( I .LT. TOKSIZ-1))   GOTO 23161
      GTOK = TYPE(NGETCH(LEXSTR(I), FD))
      IF(.NOT.(GTOK .NE. 1 .AND. GTOK .NE. 2))  GOTO 23162
      GOTO 23161
```

```
23162  CONTINUE
23160   I = I + 1
       GOTO 23159
23161  CONTINUE
       IF(.NOT.(I .GE. TOKSIZ-1)) GOTO 23164
       CALL SYNERR(15HTOKEN TOO LONG.)
23164  CONTINUE
       IF(.NOT.(I .GT. 1))  GOTO 23166
       CALL PUTBAK(LEXSTR(I))
       LEXSTR(I) = 10002
       GTOK = 10100
       GOTO 23167
23166  CONTINUE
       IF(.NOT.(LEXSTR(1) .EQ. 36))  GOTO 23168
       IF(.NOT.(NGETCH(LEXSTR(2), FD) .EQ. 40))  GOTO 23170
       LEXSTR(1) = 123
       GTOK = 123
       GOTO 23171
23170  CONTINUE
       IF(.NOT.(LEXSTR(2) .EQ. 41))  GOTO 23172
       LEXSTR(1) = 125
       GTOK = 125
       GOTO 23173
23172  CONTINUE
       CALL PUTBAK(LEXSTR(2))
23173  CONTINUE
23171  CONTINUE
       GOTO 23169
23168  CONTINUE
       IF(.NOT.(LEXSTR(1) .EQ. 39 .OR. LEXSTR(1) .EQ. 34))   GOTO 23174
       CONTINUE
       I = 2
23176  IF(.NOT.( NGETCH(LEXSTR(I), FD) .NE. LEXSTR(1)))   GOTO 23178
       IF(.NOT.(LEXSTR(I) .EQ. 10 .OR. I .GE. TOKSIZ-1))  GOTO 23179
       CALL SYNERR(14HMISSING QUOTE.)
       LEXSTR(I) = LEXSTR(1)
       CALL PUTBAK(10)
       GOTO 23178
```

```
23179  CONTINUE
23177   I = I + 1
       GOTO 23176
23178  CONTINUE
       GOTO 23175
23174  CONTINUE
       IF(.NOT.(LEXSTR(1) .EQ. 37))  GOTO 23181
       CONTINUE
23183  IF(.NOT.(NGETCH(LEXSTR(1), FD) .NE. 10))  GOTO 23184
       GOTO 23183
23184  CONTINUE
       GTOK = 10
       GOTO 23182
23181  CONTINUE
       IF(.NOT.(LEXSTR(1) .EQ. 62 .OR. LEXSTR(1) .EQ. 60 .OR. LEXSTR(1)
     *.EQ. 33       .OR. LEXSTR(1) .EQ. 61 .OR. LEXSTR(1) .EQ. 38 .OR. LE
     *XSTR(1) .EQ. 124)) GOTO 23185
       CALL RELATE(LEXSTR, I, FD)
23185  CONTINUE
23182  CONTINUE
23175  CONTINUE
23169  CONTINUE
23167  CONTINUE
       LEXSTR(I+1) = 10002
       IF(.NOT.(LEXSTR(1) .EQ. 10))  GOTO 23187
       LINECT(LEVEL) = LINECT(LEVEL) + 1
23187  CONTINUE
       RETURN
       END
C
C IFCODE - GENERATE INITIAL CODE FOR IF
C
       SUBROUTINE IFCODE(LAB)
       INTEGER LABGEN
       INTEGER LAB
       LAB = LABGEN(2)
       CALL IFGO(LAB)
       RETURN
```

```
      END
C
C IFGO - GENERATE "IF(.NOT.(...))GOTO LAB"
C
      SUBROUTINE IFGO(LAB)
      INTEGER LAB
      INTEGER IFNOT(9)
      DATA IFNOT(1) /105/
      DATA IFNOT(2) /102/
      DATA IFNOT(3) /40/
      DATA IFNOT(4) /46/
      DATA IFNOT(5) /110/
      DATA IFNOT(6) /111/
      DATA IFNOT(7) /116/
      DATA IFNOT(8) /46/
      DATA IFNOT(9) /10002/
      CALL OUTTAB
      CALL OUTSTR(IFNOT)
      CALL BALPAR
      CALL OUTCH(41)
      CALL OUTGO(LAB)
      RETURN
      END
C
C INDEX - FIND CHARACTER  C  IN STRING  STR
C
      INTEGER FUNCTION INDEX(STR, C)
      INTEGER C, STR(100)
      CONTINUE
      INDEX = 1
23189 IF(.NOT.( STR(INDEX) .NE. 10002))   GOTO 23191
      IF(.NOT.(STR(INDEX) .EQ. C))  GOTO 23192
      RETURN
23192 CONTINUE
23190  INDEX = INDEX + 1
      GOTO 23189
23191 CONTINUE
      INDEX = 0
```

```
      RETURN
      END
C
C INITKW - INSTALL KEYWORD "DEFINE" IN TABLE
C
      SUBROUTINE INITKW
      INTEGER DEFNAM(7), DEFTYP(2), UDEFNA(7)
      DATA UDEFNA /68,69,70,73,78,69,10002/
      DATA DEFNAM(1) /100/, DEFNAM(2) /101/, DEFNAM(3) /102/
      DATA DEFNAM(4) /105/, DEFNAM(5) /110/, DEFNAM(6) /101/
      DATA DEFNAM(7) /10002/
      DATA DEFTYP(1), DEFTYP(2) /10010, 10002/
      CALL INSTAL(DEFNAM, DEFTYP)
      CALL INSTAL(UDEFNA,DEFTYP)
      RETURN
      END
C
C INMAP - CONVERT LEFT ADJUSTED EXTERNAL REP TO RIGHT ADJ ASCII
C
      INTEGER FUNCTION INMAP(INCHAR)
      INTEGER I, INCHAR
      COMMON /CCHAR/ EXTDIG(10), INTDIG(10), EXTLET(26), INTLET(26),  E
     *XTBIG(26), INTBIG(26), EXTCHR(33), INTCHR(33),  EXTBLK, INTBLK
      INTEGER EXTDIG
      INTEGER INTDIG
      INTEGER EXTLET
      INTEGER INTLET
      INTEGER EXTBIG
      INTEGER INTBIG
      INTEGER EXTCHR
      INTEGER INTCHR
      INTEGER EXTBLK
      INTEGER INTBLK
      COMMON /CDEFIO/ BP, BUF(300)
      INTEGER BP
      INTEGER BUF
      COMMON /CFOR/ FORDEP, FORSTK(200)
      INTEGER FORDEP
```

```
      INTEGER FORSTK
      COMMON /CKEYWD/ SDO, SIF, SELSE, SWHILE, SBREAK, SNEXT, SFOR, SRE
     *PT, SUNTIL, VDO, VIF, VELSE, VWHILE, VBREAK, VNEXT, VFOR, VREPT, V
     *UNTIL
      INTEGER SDO(3), SIF(3), SELSE(5), SWHILE(6), SBREAK(6), SNEXT(5)
      INTEGER SFOR(4), SREPT(7), SUNTIL(6)
      INTEGER VDO(2), VIF(2), VELSE(2), VWHILE(2), VBREAK(2), VNEXT(2)
      INTEGER VFOR(2), VREPT(2), VUNTIL(2)
      COMMON /CLINE/ LEVEL, LINECT(5), INFILE(5)
      INTEGER LEVEL
      INTEGER LINECT
      INTEGER INFILE
      COMMON /CLOOK/ LASTP, LASTT, NAMPTR(200), TABLE(1500)
      INTEGER LASTP
      INTEGER LASTT
      INTEGER NAMPTR
      INTEGER TABLE
      COMMON /COUTLN/ OUTP, OUTBUF(81)
      INTEGER OUTP
      INTEGER OUTBUF
      IF(.NOT.(INCHAR .EQ. EXTBLK)) GOTO 23194
      INMAP = INTBLK
      RETURN
23194 CONTINUE
      DO23196I = 1, 10
      IF(.NOT.(INCHAR .EQ. EXTDIG(I))) GOTO 23198
      INMAP = INTDIG(I)
      RETURN
23198 CONTINUE
23196 CONTINUE
23197 CONTINUE
      DO23200I = 1, 26
      IF(.NOT.(INCHAR .EQ. EXTLET(I))) GOTO 23202
      INMAP = INTLET(I)
      RETURN
23202 CONTINUE
23200 CONTINUE
23201 CONTINUE
```

```
      DO23204I = 1, 26
      IF(.NOT.(INCHAR .EQ. EXTBIG(I))) GOTO 23206
      INMAP = INTBIG(I)
      RETURN
23206 CONTINUE
23204 CONTINUE
23205 CONTINUE
      DO23208I = 1, 33
      IF(.NOT.(INCHAR .EQ. EXTCHR(I))) GOTO 23210
      INMAP = INTCHR(I)
      RETURN
23210 CONTINUE
23208 CONTINUE
23209 CONTINUE
      INMAP = INCHAR
      RETURN
      END
C
C INSTAL - ADD NAME AND DEFINITION TO TABLE
C
      SUBROUTINE INSTAL(NAME, DEFN)
      INTEGER DEFN(200), NAME(200)
      INTEGER LENGTH
      INTEGER DLEN, NLEN
      COMMON /CCHAR/ EXTDIG(10), INTDIG(10), EXTLET(26), INTLET(26),  E
     *XTBIG(26), INTBIG(26), EXTCHR(33), INTCHR(33),  EXTBLK, INTBLK
      INTEGER EXTDIG
      INTEGER INTDIG
      INTEGER EXTLET
      INTEGER INTLET
      INTEGER EXTBIG
      INTEGER INTBIG
      INTEGER EXTCHR
      INTEGER INTCHR
      INTEGER EXTBLK
      INTEGER INTBLK
      COMMON /CDEFIO/ BP, BUF(300)
      INTEGER BP
```

```
      INTEGER BUF
      COMMON /CFOR/ FORDEP, FORSTK(200)
      INTEGER FORDEP
      INTEGER FORSTK
      COMMON /CKEYWD/ SDO, SIF, SELSE, SWHILE, SBREAK, SNEXT, SFOR, SRE
     *PT, SUNTIL, VDO, VIF, VELSE, VWHILE, VBREAK, VNEXT, VFOR, VREPT, V
     *UNTIL
      INTEGER SDO(3), SIF(3), SELSE(5), SWHILE(6), SBREAK(6), SNEXT(5)
      INTEGER SFOR(4), SREPT(7), SUNTIL(6)
      INTEGER VDO(2), VIF(2), VELSE(2), VWHILE(2), VBREAK(2), VNEXT(2)
      INTEGER VFOR(2), VREPT(2), VUNTIL(2)
      COMMON /CLINE/ LEVEL, LINECT(5), INFILE(5)
      INTEGER LEVEL
      INTEGER LINECT
      INTEGER INFILE
      COMMON /CLOOK/ LASTP, LASTT, NAMPTR(200), TABLE(1500)
      INTEGER LASTP
      INTEGER LASTT
      INTEGER NAMPTR
      INTEGER TABLE
      COMMON /COUTLN/ OUTP, OUTBUF(81)
      INTEGER OUTP
      INTEGER OUTBUF
      NLEN = LENGTH(NAME) + 1
      DLEN = LENGTH(DEFN) + 1
      IF(.NOT.(LASTT + NLEN + DLEN .GT. 1500 .OR. LASTP .GE. 200))   GO
     *TO 23212
      CALL PUTLIN(NAME, 6)
      CALL REMARK(23H: TOO MANY DEFINITIONS.)
23212 CONTINUE
      LASTP = LASTP + 1
      NAMPTR(LASTP) = LASTT + 1
      CALL SCOPY(NAME, 1, TABLE, LASTT + 1)
      CALL SCOPY(DEFN, 1, TABLE, LASTT + NLEN + 1)
      LASTT = LASTT + NLEN + DLEN
      RETURN
      END
C
```

```
C ITOC - CONVERT INTEGER  INT  TO CHAR STRING IN  STR
C
      INTEGER FUNCTION ITOC(INT, STR, SIZE)
      INTEGER IABS, MOD
      INTEGER D, I, INT, INTVAL, J, K, SIZE
      INTEGER STR(SIZE)
      INTEGER DIGITS(11)
      DATA DIGITS(1) /48/
      DATA DIGITS(2) /49/
      DATA DIGITS(3) /50/
      DATA DIGITS(4) /51/
      DATA DIGITS(5) /52/
      DATA DIGITS(6) /53/
      DATA DIGITS(7) /54/
      DATA DIGITS(8) /55/
      DATA DIGITS(9) /56/
      DATA DIGITS(10) /57/
      DATA DIGITS(11) /10002/
      INTVAL = IABS(INT)
      STR(1) = 10002
      I = 1
      CONTINUE
23214 CONTINUE
      I = I + 1
      D = MOD(INTVAL, 10)
      STR(I) = DIGITS(D+1)
      INTVAL = INTVAL / 10
23215 IF(.NOT.(INTVAL .EQ. 0 .OR. I .GE. SIZE)) GOTO 23214
23216 CONTINUE
      IF(.NOT.(INT .LT. 0 .AND. I .LT. SIZE))   GOTO 23217
      I = I + 1
      STR(I) = 45
23217 CONTINUE
      ITOC = I - 1
      CONTINUE
      J = 1
23219 IF(.NOT.( J .LT. I)) GOTO 23221
      K = STR(I)
```

```
        STR(I) = STR(J)
        STR(J) = K
        I = I - 1
23220    J = J + 1
        GOTO 23219
23221   CONTINUE
        RETURN
        END
C
C LABELC - OUTPUT STATEMENT NUMBER
C
        SUBROUTINE LABELC(LEXSTR)
        INTEGER LEXSTR(100)
        INTEGER LENGTH
        IF(.NOT.(LENGTH(LEXSTR) .EQ. 5)) GOTO 23222
        IF(.NOT.(LEXSTR(1) .EQ. 50 .AND. LEXSTR(2) .EQ. 51))  GOTO 23224
        CALL SYNERR(33HWARNING: POSSIBLE LABEL CONFLICT.)
23224   CONTINUE
23222   CONTINUE
        CALL OUTSTR(LEXSTR)
        CALL OUTTAB
        RETURN
        END
C
C LABGEN - GENERATE  N  CONSECUTIVE LABELS, RETURN FIRST ONE
C
        INTEGER FUNCTION LABGEN(N)
        INTEGER LABEL, N
        DATA LABEL /23000/
        LABGEN = LABEL
        LABEL = LABEL + N
        RETURN
        END
C
C LENGTH - COMPUTE LENGTH OF STRING
C
        INTEGER FUNCTION LENGTH(STR)
        INTEGER STR(100)
```

```
      CONTINUE
      LENGTH = 0
23226 IF(.NOT.( STR(LENGTH+1) .NE. 10002))   GOTO 23228
23227  LENGTH = LENGTH + 1
      GOTO 23226
23228 CONTINUE
      RETURN
      END
C
C LEX - RETURN LEXICAL TYPE OF TOKEN
C
      INTEGER FUNCTION LEX(LEXSTR)
      INTEGER GETTOK
      INTEGER LEXSTR(200)
      INTEGER ALLDIG, EQUAL
      COMMON /CCHAR/ EXTDIG(10), INTDIG(10), EXTLET(26), INTLET(26),  E
     *XTBIG(26), INTBIG(26), EXTCHR(33), INTCHR(33),  EXTBLK, INTBLK
      INTEGER EXTDIG
      INTEGER INTDIG
      INTEGER EXTLET
      INTEGER INTLET
      INTEGER EXTBIG
      INTEGER INTBIG
      INTEGER EXTCHR
      INTEGER INTCHR
      INTEGER EXTBLK
      INTEGER INTBLK
      COMMON /CDEFIO/ BP, BUF(300)
      INTEGER BP
      INTEGER BUF
      COMMON /CFOR/ FORDEP, FORSTK(200)
      INTEGER FORDEP
      INTEGER FORSTK
      COMMON /CKEYWD/ SDO, SIF, SELSE, SWHILE, SBREAK, SNEXT, SFOR, SRE
     *PT, SUNTIL, VDO, VIF, VELSE, VWHILE, VBREAK, VNEXT, VFOR, VREPT, V
     *UNTIL
      INTEGER SDO(3), SIF(3), SELSE(5), SWHILE(6), SBREAK(6), SNEXT(5)
      INTEGER SFOR(4), SREPT(7), SUNTIL(6)
```

```
      INTEGER VDO(2), VIF(2), VELSE(2), VWHILE(2), VBREAK(2), VNEXT(2)
      INTEGER VFOR(2), VREPT(2), VUNTIL(2)
      COMMON /CLINE/ LEVEL, LINECT(5), INFILE(5)
      INTEGER LEVEL
      INTEGER LINECT
      INTEGER INFILE
      COMMON /CLOOK/ LASTP, LASTT, NAMPTR(200), TABLE(1500)
      INTEGER LASTP
      INTEGER LASTT
      INTEGER NAMPTR
      INTEGER TABLE
      COMMON /COUTLN/ OUTP, OUTBUF(81)
      INTEGER OUTP
      INTEGER OUTBUF
      CONTINUE
23229 IF(.NOT.(GETTOK(LEXSTR, 200) .EQ. 10)) GOTO 23230
      GOTO 23229
23230 CONTINUE
      LEX = LEXSTR(1)
      IF(.NOT.(LEX.EQ.10003 .OR. LEX.EQ.59 .OR. LEX.EQ.123 .OR. LEX.EQ.
     *125))  GOTO 23231
      RETURN
23231 CONTINUE
      IF(.NOT.(ALLDIG(LEXSTR) .EQ. 1)) GOTO 23233
      LEX = 10260
      GOTO 23234
23233 CONTINUE
      IF(.NOT.(EQUAL(LEXSTR, SIF) .EQ. 1))   GOTO 23235
      LEX = VIF(1)
      GOTO 23236
23235 CONTINUE
      IF(.NOT.(EQUAL(LEXSTR, SELSE) .EQ. 1)) GOTO 23237
      LEX = VELSE(1)
      GOTO 23238
23237 CONTINUE
      IF(.NOT.(EQUAL(LEXSTR, SWHILE) .EQ. 1))   GOTO 23239
      LEX = VWHILE(1)
      GOTO 23240
```

```
23239  CONTINUE
       IF(.NOT.(EQUAL(LEXSTR, SDO) .EQ. 1))   GOTO 23241
       LEX = VDO(1)
       GOTO 23242
23241  CONTINUE
       IF(.NOT.(EQUAL(LEXSTR, SBREAK) .EQ. 1))   GOTO 23243
       LEX = VBREAK(1)
       GOTO 23244
23243  CONTINUE
       IF(.NOT.(EQUAL(LEXSTR, SNEXT) .EQ. 1)) GOTO 23245
       LEX = VNEXT(1)
       GOTO 23246
23245  CONTINUE
       IF(.NOT.(EQUAL(LEXSTR, SFOR) .EQ. 1))  GOTO 23247
       LEX = VFOR(1)
       GOTO 23248
23247  CONTINUE
       IF(.NOT.(EQUAL(LEXSTR, SREPT) .EQ. 1)) GOTO 23249
       LEX = VREPT(1)
       GOTO 23250
23249  CONTINUE
       IF(.NOT.(EQUAL(LEXSTR, SUNTIL) .EQ. 1))   GOTO 23251
       LEX = VUNTIL(1)
       GOTO 23252
23251  CONTINUE
       LEX = 10267
23252  CONTINUE
23250  CONTINUE
23248  CONTINUE
23246  CONTINUE
23244  CONTINUE
23242  CONTINUE
23240  CONTINUE
23238  CONTINUE
23236  CONTINUE
23234  CONTINUE
       RETURN
       END
```

```
C
C LOOKUP - LOCATE NAME, EXTRACT DEFINITION FROM TABLE
C
      INTEGER FUNCTION LOOKUP(NAME, DEFN)
      INTEGER DEFN(200), NAME(200)
      INTEGER I, J, K
      COMMON /CCHAR/ EXTDIG(10), INTDIG(10), EXTLET(26), INTLET(26),  E
     *XTBIG(26), INTBIG(26), EXTCHR(33), INTCHR(33),  EXTBLK, INTBLK
      INTEGER EXTDIG
      INTEGER INTDIG
      INTEGER EXTLET
      INTEGER INTLET
      INTEGER EXTBIG
      INTEGER INTBIG
      INTEGER EXTCHR
      INTEGER INTCHR
      INTEGER EXTBLK
      INTEGER INTBLK
      COMMON /CDEFIO/ BP, BUF(300)
      INTEGER BP
      INTEGER BUF
      COMMON /CFOR/ FORDEP, FORSTK(200)
      INTEGER FORDEP
      INTEGER FORSTK
      COMMON /CKEYWD/ SDO, SIF, SELSE, SWHILE, SBREAK, SNEXT, SFOR, SRE
     *PT, SUNTIL, VDO, VIF, VELSE, VWHILE, VBREAK, VNEXT, VFOR, VREPT, V
     *UNTIL
      INTEGER SDO(3), SIF(3), SELSE(5), SWHILE(6), SBREAK(6), SNEXT(5)
      INTEGER SFOR(4), SREPT(7), SUNTIL(6)
      INTEGER VDO(2), VIF(2), VELSE(2), VWHILE(2), VBREAK(2), VNEXT(2)
      INTEGER VFOR(2), VREPT(2), VUNTIL(2)
      COMMON /CLINE/ LEVEL, LINECT(5), INFILE(5)
      INTEGER LEVEL
      INTEGER LINECT
      INTEGER INFILE
      COMMON /CLOOK/ LASTP, LASTT, NAMPTR(200), TABLE(1500)
      INTEGER LASTP
      INTEGER LASTT
```

```
      INTEGER NAMPTR
      INTEGER TABLE
      COMMON /COUTLN/ OUTP, OUTBUF(81)
      INTEGER OUTP
      INTEGER OUTBUF
      CONTINUE
      I = LASTP
23253 IF(.NOT.( I .GT. 0)) GOTO 23255
      J = NAMPTR(I)
      CONTINUE
      K = 1
23256 IF(.NOT.( NAME(K) .EQ. TABLE(J) .AND. NAME(K) .NE. 10002))  GOTO
     *23258
      J = J + 1
23257  K = K + 1
      GOTO 23256
23258 CONTINUE
      IF(.NOT.(NAME(K) .EQ. TABLE(J))) GOTO 23259
      CALL SCOPY(TABLE, J+1, DEFN, 1)
      LOOKUP = 1
      RETURN
23259 CONTINUE
23254  I = I - 1
      GOTO 23253
23255 CONTINUE
      LOOKUP = 0
      RETURN
      END
C
C NGETCH - GET A (POSSIBLY PUSHED BACK) CHARACTER
C
      INTEGER FUNCTION NGETCH(C, FD)
      INTEGER GETCH
      INTEGER C
      INTEGER FD
      COMMON /CCHAR/ EXTDIG(10), INTDIG(10), EXTLET(26), INTLET(26),  E
     *XTBIG(26), INTBIG(26), EXTCHR(33), INTCHR(33),  EXTBLK, INTBLK
      INTEGER EXTDIG
```

```
      INTEGER INTDIG
      INTEGER EXTLET
      INTEGER INTLET
      INTEGER EXTBIG
      INTEGER INTBIG
      INTEGER EXTCHR
      INTEGER INTCHR
      INTEGER EXTBLK
      INTEGER INTBLK
      COMMON /CDEFIO/ BP, BUF(300)
      INTEGER BP
      INTEGER BUF
      COMMON /CFOR/ FORDEP, FORSTK(200)
      INTEGER FORDEP
      INTEGER FORSTK
      COMMON /CKEYWD/ SDO, SIF, SELSE, SWHILE, SBREAK, SNEXT, SFOR, SRE
     *PT, SUNTIL, VDO, VIF, VELSE, VWHILE, VBREAK, VNEXT, VFOR, VREPT, V
     *UNTIL
      INTEGER SDO(3), SIF(3), SELSE(5), SWHILE(6), SBREAK(6), SNEXT(5)
      INTEGER SFOR(4), SREPT(7), SUNTIL(6)
      INTEGER VDO(2), VIF(2), VELSE(2), VWHILE(2), VBREAK(2), VNEXT(2)
      INTEGER VFOR(2), VREPT(2), VUNTIL(2)
      COMMON /CLINE/ LEVEL, LINECT(5), INFILE(5)
      INTEGER LEVEL
      INTEGER LINECT
      INTEGER INFILE
      COMMON /CLOOK/ LASTP, LASTT, NAMPTR(200), TABLE(1500)
      INTEGER LASTP
      INTEGER LASTT
      INTEGER NAMPTR
      INTEGER TABLE
      COMMON /COUTLN/ OUTP, OUTBUF(81)
      INTEGER OUTP
      INTEGER OUTBUF
      IF(.NOT.(BP .GT. 0)) GOTO 23261
      C = BUF(BP)
      GOTO 23262
23261 CONTINUE
```

```
      BP = 1
      BUF(BP) = GETCH(C, FD)
23262 CONTINUE
      BP = BP - 1
      NGETCH = C
      RETURN
      END
C
C OPEN - EXCEEDINGLY TEMPORARY VERSION FOR GETTOK
C
      INTEGER FUNCTION OPEN(NAME, MODE)
      INTEGER NAME(30)
      INTEGER CTOI
      INTEGER I, MODE
      I = 1
      OPEN = CTOI(NAME, I)
      RETURN
      END
C
C OTHERC - OUTPUT ORDINARY FORTRAN STATEMENT
C
      SUBROUTINE OTHERC(LEXSTR)
      INTEGER LEXSTR(100)
      CALL OUTTAB
      CALL OUTSTR(LEXSTR)
      CALL EATUP
      CALL OUTDON
      RETURN
      END
C
C OUTCH - PUT ONE CHARACTER INTO OUTPUT BUFFER
C
      SUBROUTINE OUTCH(C)
      INTEGER C
      INTEGER I
      COMMON /CCHAR/ EXTDIG(10), INTDIG(10), EXTLET(26), INTLET(26),  E
     *XTBIG(26), INTBIG(26), EXTCHR(33), INTCHR(33),  EXTBLK, INTBLK
      INTEGER EXTDIG
```

```
      INTEGER INTDIG
      INTEGER EXTLET
      INTEGER INTLET
      INTEGER EXTBIG
      INTEGER INTBIG
      INTEGER EXTCHR
      INTEGER INTCHR
      INTEGER EXTBLK
      INTEGER INTBLK
      COMMON /CDEFIO/ BP, BUF(300)
      INTEGER BP
      INTEGER BUF
      COMMON /CFOR/ FORDEP, FORSTK(200)
      INTEGER FORDEP
      INTEGER FORSTK
      COMMON /CKEYWD/ SDO, SIF, SELSE, SWHILE, SBREAK, SNEXT, SFOR, SRE
     *PT, SUNTIL, VDO, VIF, VELSE, VWHILE, VBREAK, VNEXT, VFOR, VREPT, V
     *UNTIL
      INTEGER SDO(3), SIF(3), SELSE(5), SWHILE(6), SBREAK(6), SNEXT(5)
      INTEGER SFOR(4), SREPT(7), SUNTIL(6)
      INTEGER VDO(2), VIF(2), VELSE(2), VWHILE(2), VBREAK(2), VNEXT(2)
      INTEGER VFOR(2), VREPT(2), VUNTIL(2)
      COMMON /CLINE/ LEVEL, LINECT(5), INFILE(5)
      INTEGER LEVEL
      INTEGER LINECT
      INTEGER INFILE
      COMMON /CLOOK/ LASTP, LASTT, NAMPTR(200), TABLE(1500)
      INTEGER LASTP
      INTEGER LASTT
      INTEGER NAMPTR
      INTEGER TABLE
      COMMON /COUTLN/ OUTP, OUTBUF(81)
      INTEGER OUTP
      INTEGER OUTBUF
      IF(.NOT.(OUTP .GE. 72)) GOTO 23263
      CALL OUTDON
      CONTINUE
      I = 1
```

```
23265  IF(.NOT.( I .LT. 6)) GOTO 23267
       OUTBUF(I) = 32
23266   I = I + 1
       GOTO 23265
23267  CONTINUE
       OUTBUF(6) = 42
       OUTP = 6
23263  CONTINUE
       OUTP = OUTP + 1
       OUTBUF(OUTP) = C
       RETURN
       END
C
C OUTCON - OUTPUT "N   CONTINUE"
C
       SUBROUTINE OUTCON(N)
       INTEGER N
       INTEGER CONTIN(9)
       DATA CONTIN(1) /99/
       DATA CONTIN(2) /111/
       DATA CONTIN(3) /110/
       DATA CONTIN(4) /116/
       DATA CONTIN(5) /105/
       DATA CONTIN(6) /110/
       DATA CONTIN(7) /117/
       DATA CONTIN(8) /101/
       DATA CONTIN(9) /10002/
       IF(.NOT.(N .GT. 0))  GOTO 23268
       CALL OUTNUM(N)
23268  CONTINUE
       CALL OUTTAB
       CALL OUTSTR(CONTIN)
       CALL OUTDON
       RETURN
       END
C
C OUTDON - FINISH OFF AN OUTPUT LINE
C
```

```
      SUBROUTINE OUTDON
      COMMON /CCHAR/ EXTDIG(10), INTDIG(10), EXTLET(26), INTLET(26),  E
     *XTBIG(26), INTBIG(26), EXTCHR(33), INTCHR(33),  EXTBLK, INTBLK
      INTEGER EXTDIG
      INTEGER INTDIG
      INTEGER EXTLET
      INTEGER INTLET
      INTEGER EXTBIG
      INTEGER INTBIG
      INTEGER EXTCHR
      INTEGER INTCHR
      INTEGER EXTBLK
      INTEGER INTBLK
      COMMON /CDEFIO/ BP, BUF(300)
      INTEGER BP
      INTEGER BUF
      COMMON /CFOR/ FORDEP, FORSTK(200)
      INTEGER FORDEP
      INTEGER FORSTK
      COMMON /CKEYWD/ SDO, SIF, SELSE, SWHILE, SBREAK, SNEXT, SFOR, SRE
     *PT, SUNTIL, VDO, VIF, VELSE, VWHILE, VBREAK, VNEXT, VFOR, VREPT, V
     *UNTIL
      INTEGER SDO(3), SIF(3), SELSE(5), SWHILE(6), SBREAK(6), SNEXT(5)
      INTEGER SFOR(4), SREPT(7), SUNTIL(6)
      INTEGER VDO(2), VIF(2), VELSE(2), VWHILE(2), VBREAK(2), VNEXT(2)
      INTEGER VFOR(2), VREPT(2), VUNTIL(2)
      COMMON /CLINE/ LEVEL, LINECT(5), INFILE(5)
      INTEGER LEVEL
      INTEGER LINECT
      INTEGER INFILE
      COMMON /CLOOK/ LASTP, LASTT, NAMPTR(200), TABLE(1500)
      INTEGER LASTP
      INTEGER LASTT
      INTEGER NAMPTR
      INTEGER TABLE
      COMMON /COUTLN/ OUTP, OUTBUF(81)
      INTEGER OUTP
      INTEGER OUTBUF
```

```
      OUTBUF(OUTP+1) = 10
      OUTBUF(OUTP+2) = 10002
      CALL PUTLIN(OUTBUF, 1)
      OUTP = 0
      RETURN
      END
C
C OUTGO - OUTPUT "GOTO  N"
C
      SUBROUTINE OUTGO(N)
      INTEGER N
      INTEGER GOTO(6)
      DATA GOTO(1) /103/
      DATA GOTO(2) /111/
      DATA GOTO(3) /116/
      DATA GOTO(4) /111/
      DATA GOTO(5) /32/
      DATA GOTO(6) /10002/
      CALL OUTTAB
      CALL OUTSTR(GOTO)
      CALL OUTNUM(N)
      CALL OUTDON
      RETURN
      END
C
C OUTMAP - CONVERT RIGHT ADJ ASCII TO LEFT ADJUSTED EXTERNAL REP
C
      INTEGER FUNCTION OUTMAP(INCHAR)
      INTEGER I, INCHAR
      COMMON /CCHAR/ EXTDIG(10), INTDIG(10), EXTLET(26), INTLET(26),  E
     *XTBIG(26), INTBIG(26), EXTCHR(33), INTCHR(33),  EXTBLK, INTBLK
      INTEGER EXTDIG
      INTEGER INTDIG
      INTEGER EXTLET
      INTEGER INTLET
      INTEGER EXTBIG
      INTEGER INTBIG
      INTEGER EXTCHR
```

```
      INTEGER INTCHR
      INTEGER EXTBLK
      INTEGER INTBLK
      COMMON /CDEFIO/ BP, BUF(300)
      INTEGER BP
      INTEGER BUF
      COMMON /CFOR/ FORDEP, FORSTK(200)
      INTEGER FORDEP
      INTEGER FORSTK
      COMMON /CKEYWD/ SDO, SIF, SELSE, SWHILE, SBREAK, SNEXT, SFOR, SRE
     *PT, SUNTIL, VDO, VIF, VELSE, VWHILE, VBREAK, VNEXT, VFOR, VREPT, V
     *UNTIL
      INTEGER SDO(3), SIF(3), SELSE(5), SWHILE(6), SBREAK(6), SNEXT(5)
      INTEGER SFOR(4), SREPT(7), SUNTIL(6)
      INTEGER VDO(2), VIF(2), VELSE(2), VWHILE(2), VBREAK(2), VNEXT(2)
      INTEGER VFOR(2), VREPT(2), VUNTIL(2)
      COMMON /CLINE/ LEVEL, LINECT(5), INFILE(5)
      INTEGER LEVEL
      INTEGER LINECT
      INTEGER INFILE
      COMMON /CLOOK/ LASTP, LASTT, NAMPTR(200), TABLE(1500)
      INTEGER LASTP
      INTEGER LASTT
      INTEGER NAMPTR
      INTEGER TABLE
      COMMON /COUTLN/ OUTP, OUTBUF(81)
      INTEGER OUTP
      INTEGER OUTBUF
      IF(.NOT.(INCHAR .EQ. INTBLK)) GOTO 23270
      OUTMAP = EXTBLK
      RETURN
23270 CONTINUE
      DO23272I = 1, 10
      IF(.NOT.(INCHAR .EQ. INTDIG(I))) GOTO 23274
      OUTMAP = EXTDIG(I)
      RETURN
23274 CONTINUE
23272 CONTINUE
```

```
23273   CONTINUE
        DO23276I = 1, 26
        IF(.NOT.(INCHAR .EQ. INTLET(I))) GOTO 23278
        OUTMAP = EXTBIG(I)
        RETURN
23278   CONTINUE
23276   CONTINUE
23277   CONTINUE
        DO23280I = 1, 26
        IF(.NOT.(INCHAR .EQ. INTBIG(I))) GOTO 23282
        OUTMAP = EXTBIG(I)
        RETURN
23282   CONTINUE
23280   CONTINUE
23281   CONTINUE
        DO23284I = 1, 33
        IF(.NOT.(INCHAR .EQ. INTCHR(I))) GOTO 23286
        OUTMAP = EXTCHR(I)
        RETURN
23286   CONTINUE
23284   CONTINUE
23285   CONTINUE
        OUTMAP = INCHAR
        RETURN
        END
C
C OUTNUM - OUTPUT DECIMAL NUMBER
C
        SUBROUTINE OUTNUM(N)
        INTEGER CHARS(10)
        INTEGER ITOC
        INTEGER I, LEN, N
        LEN = ITOC(N, CHARS, 10)
        CONTINUE
        I = 1
23288   IF(.NOT.( I .LE. LEN))  GOTO 23290
        CALL OUTCH(CHARS(I))
23289    I = I + 1
```

```
      GOTO 23288
23290 CONTINUE
      RETURN
      END
C
C OUTSTR - OUTPUT STRING
C
      SUBROUTINE OUTSTR(STR)
      INTEGER C, STR(100)
      INTEGER I, J
      CONTINUE
      I = 1
23291 IF(.NOT.( STR(I) .NE. 10002)) GOTO 23293
      C = STR(I)
      IF(.NOT.(C .NE. 39 .AND. C .NE. 34))   GOTO 23294
      CALL OUTCH(C)
      GOTO 23295
23294 CONTINUE
      I = I + 1
      CONTINUE
      J = I
23296 IF(.NOT.( STR(J) .NE. C))  GOTO 23298
23297  J = J + 1
      GOTO 23296
23298 CONTINUE
      CALL OUTNUM(J-I)
      CALL OUTCH(104)
      CONTINUE
23299 IF(.NOT.( I .LT. J)) GOTO 23301
      CALL OUTCH(STR(I))
23300  I = I + 1
      GOTO 23299
23301 CONTINUE
23295 CONTINUE
23292  I = I + 1
      GOTO 23291
23293 CONTINUE
      RETURN
```

```
      END
C
C OUTTAB - GET PAST COLUMN 6
C
      SUBROUTINE OUTTAB
      COMMON /CCHAR/ EXTDIG(10), INTDIG(10), EXTLET(26), INTLET(26),  E
     *XTBIG(26), INTBIG(26), EXTCHR(33), INTCHR(33),  EXTBLK, INTBLK
      INTEGER EXTDIG
      INTEGER INTDIG
      INTEGER EXTLET
      INTEGER INTLET
      INTEGER EXTBIG
      INTEGER INTBIG
      INTEGER EXTCHR
      INTEGER INTCHR
      INTEGER EXTBLK
      INTEGER INTBLK
      COMMON /CDEFIO/ BP, BUF(300)
      INTEGER BP
      INTEGER BUF
      COMMON /CFOR/ FORDEP, FORSTK(200)
      INTEGER FORDEP
      INTEGER FORSTK
      COMMON /CKEYWD/ SDO, SIF, SELSE, SWHILE, SBREAK, SNEXT, SFOR, SRE
     *PT, SUNTIL, VDO, VIF, VELSE, VWHILE, VBREAK, VNEXT, VFOR, VREPT, V
     *UNTIL
      INTEGER SDO(3), SIF(3), SELSE(5), SWHILE(6), SBREAK(6), SNEXT(5)
      INTEGER SFOR(4), SREPT(7), SUNTIL(6)
      INTEGER VDO(2), VIF(2), VELSE(2), VWHILE(2), VBREAK(2), VNEXT(2)
      INTEGER VFOR(2), VREPT(2), VUNTIL(2)
      COMMON /CLINE/ LEVEL, LINECT(5), INFILE(5)
      INTEGER LEVEL
      INTEGER LINECT
      INTEGER INFILE
      COMMON /CLOCK/ LASTP, LASTT, NAMPTR(200), TABLE(1500)
      INTEGER LASTP
      INTEGER LASTT
      INTEGER NAMPTR
```

```
      INTEGER TABLE
      COMMON /COUTLN/ OUTP, OUTBUF(81)
      INTEGER OUTP
      INTEGER OUTBUF
      CONTINUE
23302 IF(.NOT.(OUTP .LT. 6))  GOTO 23303
      CALL OUTCH(32)
      GOTO 23302
23303 CONTINUE
      RETURN
      END
C
C PARSE - PARSE RATFOR SOURCE PROGRAM
C
      SUBROUTINE PARSE
      INTEGER LEXSTR(200)
      INTEGER LEX
      INTEGER LAB, LABVAL(100), LEXTYP(100), SP, TOKEN
      CALL INITKW
      SP = 1
      LEXTYP(1) = 10003
      CONTINUE
      TOKEN = LEX(LEXSTR)
23304 IF(.NOT.( TOKEN .NE. 10003))  GOTO 23306
      IF(.NOT.(TOKEN .EQ. 10261))   GOTO 23307
      CALL IFCODE(LAB)
      GOTO 23308
23307 CONTINUE
      IF(.NOT.(TOKEN .EQ. 10266))   GOTO 23309
      CALL DOCODE(LAB)
      GOTO 23310
23309 CONTINUE
      IF(.NOT.(TOKEN .EQ. 10263))   GOTO 23311
      CALL WHILEC(LAB)
      GOTO 23312
23311 CONTINUE
      IF(.NOT.(TOKEN .EQ. 10268))   GOTO 23313
      CALL FORCOD(LAB)
```

```
      GOTO 23314
23313 CONTINUE
      IF(.NOT.(TOKEN .EQ. 10269))   GOTO 23315
      CALL REPCOD(LAB)
      GOTO 23316
23315 CONTINUE
      IF(.NOT.(TOKEN .EQ. 10260))   GOTO 23317
      CALL LABELC(LEXSTR)
      GOTO 23318
23317 CONTINUE
      IF(.NOT.(TOKEN .EQ. 10262))   GOTO 23319
      IF(.NOT.(LEXTYP(SP) .EQ. 10261)) GOTO 23321
      CALL ELSEIF(LABVAL(SP))
      GOTO 23322
23321 CONTINUE
      CALL SYNERR(13HILLEGAL ELSE.)
23322 CONTINUE
23319 CONTINUE
23318 CONTINUE
23316 CONTINUE
23314 CONTINUE
23312 CONTINUE
23310 CONTINUE
23308 CONTINUE
      IF(.NOT.(TOKEN.EQ.10261 .OR. TOKEN.EQ.10262 .OR. TOKEN.EQ.10263
     *     .OR. TOKEN.EQ.10268 .OR. TOKEN.EQ.10269         .OR. TOKEN.E
     *Q.10266 .OR. TOKEN.EQ.10260 .OR. TOKEN.EQ.123))  GOTO 23323
      SP = SP + 1
      IF(.NOT.(SP .GT. 100))  GOTO 23325
      CALL ERROR(25HSTACK OVERFLOW IN PARSER.)
23325 CONTINUE
      LEXTYP(SP) = TOKEN
      LABVAL(SP) = LAB
      GOTO 23324
23323 CONTINUE
      IF(.NOT.(TOKEN .EQ. 125))  GOTO 23327
      IF(.NOT.(LEXTYP(SP) .EQ. 123))   GOTO 23329
      SP = SP - 1
```

```
      GOTO 23330
23329 CONTINUE
      CALL SYNERR(20HILLEGAL RIGHT BRACE.)
23330 CONTINUE
      GOTO 23328
23327 CONTINUE
      IF(.NOT.(TOKEN .EQ. 10267))   GOTO 23331
      CALL OTHERC(LEXSTR)
      GOTO 23332
23331 CONTINUE
      IF(.NOT.(TOKEN .EQ. 10264 .OR. TOKEN .EQ. 10265))  GOTO 23333
      CALL BRKNXT(SP, LEXTYP, LABVAL, TOKEN)
23333 CONTINUE
23332 CONTINUE
23328 CONTINUE
      TOKEN = LEX(LEXSTR)
      CALL PBSTR(LEXSTR)
      CALL UNSTAK(SP, LEXTYP, LABVAL, TOKEN)
23324 CONTINUE
23305  TOKEN = LEX(LEXSTR)
      GOTO 23304
23306 CONTINUE
      IF(.NOT.(SP .NE. 1)) GOTO 23335
      CALL SYNERR(15HUNEXPECTED EOF.)
23335 CONTINUE
      RETURN
      END
C
C PBSTR - PUSH STRING BACK ONTO INPUT
C
      SUBROUTINE PBSTR(IN)
      INTEGER IN(100)
      INTEGER LENGTH
      INTEGER I
      CONTINUE
      I = LENGTH(IN)
23337 IF(.NOT.( I .GT. 0)) GOTO 23339
      CALL PUTBAK(IN(I))
```

```
23338   I = I - 1
        GOTO 23337
23339   CONTINUE
        RETURN
        END
C
C PUTBAK - PUSH CHARACTER BACK ONTO INPUT
C
        SUBROUTINE PUTBAK(C)
        INTEGER C
        COMMON /CCHAR/ EXTDIG(10), INTDIG(10), EXTLET(26), INTLET(26),  E
     *XTBIG(26), INTBIG(26), EXTCHR(33), INTCHR(33),  EXTBLK, INTBLK
        INTEGER EXTDIG
        INTEGER INTDIG
        INTEGER EXTLET
        INTEGER INTLET
        INTEGER EXTBIG
        INTEGER INTBIG
        INTEGER EXTCHR
        INTEGER INTCHR
        INTEGER EXTBLK
        INTEGER INTBLK
        COMMON /CDEFIO/ BP, BUF(300)
        INTEGER BP
        INTEGER BUF
        COMMON /CFOR/ FORDEP, FORSTK(200)
        INTEGER FORDEP
        INTEGER FORSTK
        COMMON /CKEYWD/ SDO, SIF, SELSE, SWHILE, SBREAK, SNEXT, SFOR, SRE
     *PT, SUNTIL, VDO, VIF, VELSE, VWHILE, VBREAK, VNEXT, VFOR, VREPT, V
     *UNTIL
        INTEGER SDO(3), SIF(3), SELSE(5), SWHILE(6), SBREAK(6), SNEXT(5)
        INTEGER SFOR(4), SREPT(7), SUNTIL(6)
        INTEGER VDO(2), VIF(2), VELSE(2), VWHILE(2), VBREAK(2), VNEXT(2)
        INTEGER VFOR(2), VREPT(2), VUNTIL(2)
        COMMON /CLINE/ LEVEL, LINECT(5), INFILE(5)
        INTEGER LEVEL
        INTEGER LINECT
```

```
      INTEGER INFILE
      COMMON /CLOOK/ LASTP, LASTT, NAMPTR(200), TABLE(1500)
      INTEGER LASTP
      INTEGER LASTT
      INTEGER NAMPTR
      INTEGER TABLE
      COMMON /COUTLN/ OUTP, OUTBUF(81)
      INTEGER OUTP
      INTEGER OUTBUF
      BP = BP + 1
      IF(.NOT.(BP .GT. 300))  GOTO 23340
      CALL ERROR(32HTOO MANY CHARACTERS PUSHED BACK.)
23340 CONTINUE
      BUF(BP) = C
      RETURN
      END
C
C PUTCH (INTERIM VERSION)  PUT CHARACTERS
C
      SUBROUTINE PUTCH(C, F)
      INTEGER BUF(81), C
      INTEGER OUTMAP
      INTEGER F, I, LASTC
      DATA LASTC /0/, IBLNK /1H /
      IF(.NOT.(LASTC .GE. 81 .OR. C .EQ. 10))   GOTO 23342
      IF(.NOT.( LASTC .LE. 0 ))  GOTO 23344
      WRITE(F,2)
2        FORMAT(/)
      GOTO 23345
23344 CONTINUE
      IF(LASTC.GE.80)GO TO 19000
      L=LASTC + 1
      DO 30 K=L,80
      BUF(K)=IBLNK
   30 CONTINUE
      LASTC=80
19000 CONTINUE
      WRITE(F, 1)  (BUF(I), I = 1, LASTC)
```

```
1          FORMAT(80 A1)
23345   CONTINUE
        LASTC = 0
23342   CONTINUE
        IF(.NOT.(C .NE. 10)) GOTO 23346
        LASTC = LASTC + 1
        BUF(LASTC) = OUTMAP(C)
23346   CONTINUE
        RETURN
        END
C
C PUTLIN - PUT OUT LINE BY REPEATED CALLS TO PUTCH
C
        SUBROUTINE PUTLIN(B, F)
        INTEGER B(100)
        INTEGER F, I
        CONTINUE
        I = 1
23348   IF(.NOT.( B(I) .NE. 10002))   GOTO 23350
        CALL PUTCH(B(I), F)
23349    I = I + 1
        GOTO 23348
23350   CONTINUE
        RETURN
        END
C
C RELATE - CONVERT RELATIONAL SHORTHANDS INTO LONG FORM
C
        SUBROUTINE RELATE(TOKEN, LAST, FD)
        INTEGER NGETCH
        INTEGER TOKEN(100)
        INTEGER LENGTH
        INTEGER FD, LAST
        INTEGER DOTGE(5), DOTGT(5), DOTLT(5), DOTLE(5)
        INTEGER DOTNE(5), DOTNOT(6), DOTEQ(5), DOTAND(6), DOTOR(5)
        DATA DOTGE(1), DOTGE(2), DOTGE(3), DOTGE(4), DOTGE(5)/ 46, 103, 1
     *01, 46, 10002/
        DATA DOTGT(1), DOTGT(2), DOTGT(3), DOTGT(4), DOTGT(5)/ 46, 103, 1
```

```
     *16, 46, 10002/
       DATA DOTLE(1), DOTLE(2), DOTLE(3), DOTLE(4), DOTLE(5)/ 46, 108, 1
     *01, 46, 10002/
       DATA DOTLT(1), DOTLT(2), DOTLT(3), DOTLT(4), DOTLT(5)/ 46, 108, 1
     *16, 46, 10002/
       DATA DOTNE(1), DOTNE(2), DOTNE(3), DOTNE(4), DOTNE(5)/ 46, 110, 1
     *01, 46, 10002/
       DATA DOTEQ(1), DOTEQ(2), DOTEQ(3), DOTEQ(4), DOTEQ(5)/ 46, 101, 1
     *13, 46, 10002/
       DATA DOTOR(1), DOTOR(2), DOTOR(3), DOTOR(4), DOTOR(5)/ 46, 111, 1
     *14, 46, 10002/
       DATA DOTAND(1), DOTAND(2), DOTAND(3), DOTAND(4), DOTAND(5), DOTAN
     *D(6) /46, 97, 110, 100, 46, 10002/
       DATA DOTNOT(1), DOTNOT(2), DOTNOT(3), DOTNOT(4), DOTNOT(5), DOTNO
     *T(6) /46, 110, 111, 116, 46, 10002/
       IF(.NOT.(NGETCH(TOKEN(2), FD) .NE. 61))   GOTO 23351
       CALL PUTBAK(TOKEN(2))
23351  CONTINUE
       IF(.NOT.(TOKEN(1) .EQ. 62))   GOTO 23353
       IF(.NOT.(TOKEN(2) .EQ. 61))   GOTO 23355
       CALL SCOPY(DOTGE, 1, TOKEN, 1)
       GOTO 23356
23355  CONTINUE
       CALL SCOPY(DOTGT, 1, TOKEN, 1)
23356  CONTINUE
       GOTO 23354
23353  CONTINUE
       IF(.NOT.(TOKEN(1) .EQ. 60))   GOTO 23357
       IF(.NOT.(TOKEN(2) .EQ. 61))   GOTO 23359
       CALL SCOPY(DOTLE, 1, TOKEN, 1)
       GOTO 23360
23359  CONTINUE
       CALL SCOPY(DOTLT, 1, TOKEN, 1)
23360  CONTINUE
       GOTO 23358
23357  CONTINUE
       IF(.NOT.(TOKEN(1) .EQ. 33))   GOTO 23361
       IF(.NOT.(TOKEN(2) .EQ. 61))   GOTO 23363
```

```
      CALL SCOPY(DOTNE, 1, TOKEN, 1)
      GOTO 23364
23363 CONTINUE
      CALL SCOPY(DOTNOT, 1, TOKEN, 1)
23364 CONTINUE
      GOTO 23362
23361 CONTINUE
      IF(.NOT.(TOKEN(1) .EQ. 61))   GOTO 23365
      IF(.NOT.(TOKEN(2) .EQ. 61))   GOTO 23367
      CALL SCOPY(DOTEQ, 1, TOKEN, 1)
      GOTO 23368
23367 CONTINUE
      TOKEN(2) = 10002
23368 CONTINUE
      GOTO 23366
23365 CONTINUE
      IF(.NOT.(TOKEN(1) .EQ. 38))   GOTO 23369
      CALL SCOPY(DOTAND, 1, TOKEN, 1)
      GOTO 23370
23369 CONTINUE
      IF(.NOT.(TOKEN(1) .EQ. 124))  GOTO 23371
      CALL SCOPY(DOTOR, 1, TOKEN, 1)
      GOTO 23372
23371 CONTINUE
      TOKEN(2) = 10002
23372 CONTINUE
23370 CONTINUE
23366 CONTINUE
23362 CONTINUE
23358 CONTINUE
23354 CONTINUE
      LAST = LENGTH(TOKEN)
      RETURN
      END
C
C REMARK - PRINT WARNING MESSAGE
C
      SUBROUTINE REMARK(BUF)
```

```
      INTEGER BUF(100), I
      WRITE(9, 10) (BUF(I), I = 1, 8)
10       FORMAT(' ',8A4)
      RETURN
      END
C
C REPCOD - GENERATE CODE FOR BEGINNING OF REPEAT
C
      SUBROUTINE REPCOD(LAB)
      INTEGER LABGEN
      INTEGER LAB
      CALL OUTCON(0)
      LAB = LABGEN(3)
      CALL OUTCON(LAB)
      LAB = LAB + 1
      RETURN
      END
C
C SCOPY - COPY STRING AT FROM(I) TO TO(J)
C
      SUBROUTINE SCOPY(FROM, I, TO, J)
      INTEGER FROM(100), TO(100)
      INTEGER I, J, K1, K2
      K2 = J
      CONTINUE
      K1 = I
23373 IF(.NOT.( FROM(K1) .NE. 10002))  GOTO 23375
      TO(K2) = FROM(K1)
      K2 = K2 + 1
23374  K1 = K1 + 1
      GOTO 23373
23375 CONTINUE
      TO(K2) = 10002
      RETURN
      END
C
C SYNERR - REPORT RATFOR SYNTAX ERROR
C
```

```
      SUBROUTINE SYNERR(MSG)
      INTEGER LC(81), MSG(81)
      INTEGER ITOC
      INTEGER I, JUNK
      COMMON /CCHAR/ EXTDIG(10), INTDIG(10), EXTLET(26), INTLET(26),  E
     *XTBIG(26), INTBIG(26), EXTCHR(33), INTCHR(33),  EXTBLK, INTBLK
      INTEGER EXTDIG
      INTEGER INTDIG
      INTEGER EXTLET
      INTEGER INTLET
      INTEGER EXTBIG
      INTEGER INTBIG
      INTEGER EXTCHR
      INTEGER INTCHR
      INTEGER EXTBLK
      INTEGER INTBLK
      COMMON /CDEFIC/ BP, BUF(300)
      INTEGER BP
      INTEGER BUF
      COMMON /CFOR/ FORDEP, FORSTK(200)
      INTEGER FORDEP
      INTEGER FORSTK
      COMMON /CKEYWD/ SDO, SIF, SELSE, SWHILE, SBREAK, SNEXT, SFOR, SRE
     *PT, SUNTIL, VDO, VIF, VELSE, VWHILE, VBREAK, VNEXT, VFOR, VREPT, V
     *UNTIL
      INTEGER SDO(3), SIF(3), SELSE(5), SWHILE(6), SBREAK(6), SNEXT(5)
      INTEGER SFOR(4), SREPT(7), SUNTIL(6)
      INTEGER VDO(2), VIF(2), VELSE(2), VWHILE(2), VBREAK(2), VNEXT(2)
      INTEGER VFOR(2), VREPT(2), VUNTIL(2)
      COMMON /CLINE/ LEVEL, LINECT(5), INFILE(5)
      INTEGER LEVEL
      INTEGER LINECT
      INTEGER INFILE
      COMMON /CLOOK/ LASTP, LASTT, NAMPTR(200), TABLE(1500)
      INTEGER LASTP
      INTEGER LASTT
      INTEGER NAMPTR
      INTEGER TABLE
```

```
      COMMON /COUTLN/ OUTP, OUTBUF(81)
      INTEGER OUTP
      INTEGER OUTBUF
      CALL REMARK(14HERROR AT LINE.)
      CONTINUE
      I = 1
23376 IF(.NOT.( I .LE. LEVEL))   GOTO 23378
      CALL PUTCH(32, 9)
      JUNK = ITOC(LINECT(I), LC, 81)
      CALL PUTLIN(LC, 9)
23377  I = I + 1
      GOTO 23376
23378 CONTINUE
      CALL PUTCH(58, 9)
      CALL PUTCH(10, 9)
      CALL REMARK(MSG)
      RETURN
      END
C
C TYPE - RETURN LETTER, DIGIT OR CHARACTER
C
      INTEGER FUNCTION TYPE(C)
      INTEGER C
      IF(.NOT.( C .GE. 48 .AND. C .LE. 57 )) GOTO 23379
      TYPE = 2
      GOTO 23380
23379 CONTINUE
      IF(.NOT.( C .GE. 97 .AND. C .LE. 122 ))   GOTO 23381
      TYPE = 1
      GOTO 23382
23381 CONTINUE
      IF(.NOT.( C .GE. 65 .AND. C .LE. 90 )) GOTO 23383
      TYPE = 1
      GOTO 23384
23383 CONTINUE
      TYPE = C
23384 CONTINUE
23382 CONTINUE
```

```
23380  CONTINUE
       RETURN
       END
C
C UNSTAK - UNSTACK AT END OF STATEMENT
C
       SUBROUTINE UNSTAK(SP, LEXTYP, LABVAL, TOKEN)
       INTEGER LABVAL(100), LEXTYP(100), SP, TOKEN
       CONTINUE
23385  IF(.NOT.( SP .GT. 1))    GOTO 23387
       IF(.NOT.(LEXTYP(SP) .EQ. 123))    GOTO 23388
       GOTO 23387
23388  CONTINUE
       IF(.NOT.(LEXTYP(SP) .EQ. 10261 .AND. TOKEN .EQ. 10262))  GOTO 233
     *90
       GOTO 23387
23390  CONTINUE
       IF(.NOT.(LEXTYP(SP) .EQ. 10261)) GOTO 23392
       CALL OUTCON(LABVAL(SP))
       GOTO 23393
23392  CONTINUE
       IF(.NOT.(LEXTYP(SP) .EQ. 10262)) GOTO 23394
       IF(.NOT.(SP .GT. 2)) GOTO 23396
       SP = SP - 1
23396  CONTINUE
       CALL OUTCON(LABVAL(SP)+1)
       GOTO 23395
23394  CONTINUE
       IF(.NOT.(LEXTYP(SP) .EQ. 10266)) GOTO 23398
       CALL DOSTAT(LABVAL(SP))
       GOTO 23399
23398  CONTINUE
       IF(.NOT.(LEXTYP(SP) .EQ. 10263)) GOTO 23400
       CALL WHILES(LABVAL(SP))
       GOTO 23401
23400  CONTINUE
       IF(.NOT.(LEXTYP(SP) .EQ. 10268)) GOTO 23402
       CALL FORS(LABVAL(SP))
```

```
      GOTO 23403
23402 CONTINUE
      IF(.NOT.(LEXTYP(SP) .EQ. 10269)) GOTO 23404
      CALL UNTILS(LABVAL(SP), TOKEN)
23404 CONTINUE
23403 CONTINUE
23401 CONTINUE
23399 CONTINUE
23395 CONTINUE
23393 CONTINUE
23386  SP = SP - 1
      GOTO 23385
23387 CONTINUE
      RETURN
      END
C
C UNTILS - GENERATE CODE FOR UNTIL OR END OF REPEAT
C
      SUBROUTINE UNTILS(LAB, TOKEN)
      INTEGER PTOKEN(200)
      INTEGER LEX
      INTEGER JUNK, LAB, TOKEN
      CALL OUTNUM(LAB)
      IF(.NOT.(TOKEN .EQ. 10270))   GOTO 23406
      JUNK = LEX(PTOKEN)
      CALL IFGO(LAB-1)
      GOTO 23407
23406 CONTINUE
      CALL OUTGO(LAB-1)
23407 CONTINUE
      CALL OUTCON(LAB+1)
      RETURN
      END
C
C WHILEC - GENERATE CODE FOR BEGINNING OF WHILE
C
      SUBROUTINE WHILEC(LAB)
      INTEGER LABGEN
```

```
      INTEGER LAB
      CALL OUTCON(0)
      LAB = LABGEN(2)
      CALL OUTNUM(LAB)
      CALL IFGO(LAB+1)
      RETURN
      END
C
C WHILES - GENERATE CODE FOR END OF WHILE
C
      SUBROUTINE WHILES(LAB)
      INTEGER LAB
      CALL OUTGO(LAB)
      CALL OUTCON(LAB+1)
      RETURN
      END
```

APPENDIX B

Ratfor Version of The Text Formatter

```
define(ALPHA,10100)
define(AMPER,38)    % ampersand
define(ARB,100)
define(ATSIGN,64)
define(BACKSLASH,92)
define(BACKSPACE,8)
define(BANG,33)    % exclamation mark
define(BAR,124)
define(BIGA,65)
define(BIGB,66)
define(BIGC,67)
define(BIGD,68)
define(BIGE,69)
define(BIGF,70)
define(BIGG,71)
define(BIGH,72)
define(BIGI,73)
define(BIGJ,74)
define(BIGK,75)
define(BIGL,76)
define(BIGM,77)
define(BIGN,78)
define(BIGO,79)
define(BIGP,80)
define(BIGQ,81)
define(BIGR,82)
define(BIGS,83)
define(BIGT,84)
define(BIGU,85)
define(BIGV,86)
define(BIGW,87)
define(BIGX,88)
define(BIGY,89)
define(BIGZ,90)
define(BLANK,32)
define(BUFSIZE,300)    % pushback buffer for ngetch and putbak
define(COLON,58)
define(COMMA,44)
```

```
define(DEFTYPE,10010)
define(DIG0,48)
define(DIG1,49)
define(DIG2,50)
define(DIG3,51)
define(DIG4,52)
define(DIG5,53)
define(DIG6,54)
define(DIG7,55)
define(DIG8,56)
define(DIG9,57)
define(DIGIT,2)
define(DOLLAR,36)
define(DQUOTE,34)
define(EOF,10003)
define(EOS,10002)
define(EQUALS,61)
define(ERR,10001)
define(ERROUT,6)    % temporarily same as standard output
define(GREATER,62)
define(LBRACE,123)
define(LBRACK,91)
define(LESS,60)
define(LETA,97)
define(LETB,98)
define(LETC,99)
define(LETD,100)
define(LETE,101)
define(LETF,102)
define(LETG,103)
define(LETH,104)
define(LETI,105)
define(LETJ,106)
define(LETK,107)
define(LETL,108)
define(LETM,109)
define(LETN,110)
define(LETO,111)
```

```
define(LETP,112)
define(LETQ,113)
define(LETR,114)
define(LETS,115)
define(LETT,116)
define(LETTER,1)
define(LETU,117)
define(LETV,118)
define(LETW,119)
define(LETX,120)
define(LETY,121)
define(LETZ,122)
define(LEXBREAK,10264)
define(LEXDIGITS,10260)
define(LEXDO,10266)
define(LEXELSE,10262)
define(LEXFOR,10268)
define(LEXIF,10261)
define(LEXNEXT,10265)
define(LEXOTHER,10267)
define(LEXREPEAT,10269)
define(LEXUNTIL,10270)
define(LEXWHILE,10263)
define(LPAREN,40)
define(MAXCARD,80)    % card size
define(MAXCHARS,10)    % characters for outnum
define(MAXDEF,200)    % max chars in a defn
define(MAXFORSTK,200)    % max space for for reinit clauses
define(MAXLINE,80)    % must be 1 more than MAXCARD
define(MAXNAME,30)    % file name size in gettok
define(MAXPTR,200)    % number of defines in lookup
define(MAXSTACK,100)    % max stack depth for parser
define(MAXTBL,1500)    % max chars in all definitions
define(MAXTOK,200)    % max chars in a token
define(MINUS,45)
define(NCHARS,33)    % number of special characters
define(NEWLINE,10)
define(NFILES,5)    % max depth of file inclusion
```

```
define(NO,0)
define(NOT,BANG)    % exclamation mark for now; change for ebcdic
define(PERCENT,37)
define(PERIOD,46)
define(PLUS,43)
define(QMARK,63)
define(RBRACE,125)
define(RBRACK,93)
define(READONLY,0)
define(RPAREN,41)
define(SEMICOL,59)
define(SHARP,35)
define(SLASH,47)
define(SQUOTE,39)
define(STAR,42)
define(STDIN,5)
define(STDOUT,2)
define(TAB,9)
define(UNDERLINE,95)
define(YES,1)
define(character,integer)
define(abs,iabs)
define(INSIZE,300)
define(MAXOUT,300)
define(COMMAND,PERIOD)
define(PAGENUM,SHARP)
define(PAGEWIDTH,60)
define(PAGELEN,66)

define(UNKNOWN,0)
define(FI,1)
define(NF,2)
define(BR,3)
define(LS,4)
define(BP,5)
define(SP,6)
define(IN,7)
define(RM,8)
```

```
define(TI,9)
define(CE,10)
define(UL,11)
define(HE,12)
define(FO,13)
define(PL,14)

define(HUGE,1000)
common /cout/ outp, outw, outwds, outbuf(MAXOUT)
   integer outp       % last char position in outbuf; init = 0
   integer outw       % width of text currently in outbuf; init = 0
   integer outwds       % number of words in outbuf; init = 0
   character outbuf       % lines to be filled collect here
common /cpage/ curpag,newpag,lineno,plval,m1val,m2val,m3val,m4val,
   bottom, header(MAXLINE), footer(MAXLINE)
   integer curpag    % current output page number; init = 0
   integer newpag    % next output page number; init = 1
   integer lineno    % next line to be printed; init = 0
   integer plval    % page length in lines; init = PAGELEN = 66
   integer m1val    % margin before and including header
   integer m2val    % margin after header
   integer m3val    % margin after last text line
   integer m4val    % bottom margin, including footer
   integer bottom    % last live line on page, = plval-m3val-m4val
   character header    % top of page title; init = NEWLINE
   character footer    % bottom of page title; init = NEWLINE
common /cparam/ fill, lsval, inval, rmval, tival, ceval, ulval
   integer fill       % fill if YES; init = YES
   integer lsval    % current line spacing; init = 1
   integer inval    % current indent; >= 0; init = 0
   integer rmval    % current right margin; init = PAGEWIDTH = 60
   integer tival    % current temporary indent; init = 0
   integer ceval    % number of lines to center; init = 0
   integer ulval    % number of lines to underline; init = 0
% format - text formatter main program (final version)
   integer inbuf(INSIZE)
   integer getlin
```

```
   call init
   while (getlin(inbuf, STDIN) ~= EOF)
      if (inbuf(1) == COMMAND)    % it's a command
         call comand(inbuf)
      else                        % it's text
         call text(inbuf)
   if (lineno > 0)
      call space(HUGE)            % flush last output
   stop
   end
% getlin - get a line of input code
      integer function getlin(buf, f)
      integer buf(INSIZE), f
      read(f,1,end=100)(buf(i),i=1,80)
    1 format(80a1)
      do  i=1,80
      buf(i)=inmap(buf(i))
      i=80
   20 if(buf(i)~= 32)go to 30
      i=i-1
      go to 20
   30 getlin=i
      buf(i+1)=NEWLINE
      buf(i+2)=EOS
      return
  100 buf(1)=EOF
      getlin=EOF
      return
      end
% brk - end current filled line
   subroutine brk
common /cout/ outp, outw, outwds, outbuf(MAXOUT)
   integer outp, outw, outwds, outbuf

   if (outp > 0)
      outbuf(outp) = NEWLINE
      outbuf(outp+1) = EOS
      call put(outbuf)
```

```
   outp = 0
   outw = 0
   outwds = 0
   return
   end
% center - center a line by setting tival
   subroutine center(buf)
   integer buf(ARB)
   integer max, width
common /cparam/ fill, lsval, inval, rmval, tival, ceval, ulval
   integer fill, lsval, inval, rmval, tival, ceval, ulval

   tival = max((rmval+tival-width(buf))/2, 0)
   return
   end
% comand - perform formatting command
   subroutine comand(buf)
   integer buf(MAXLINE)
   integer comtyp, getval, max
   integer argtyp, ct, spval, val
ccmmcn /cpage/ curpag,newpag,lineno,plval,m1val,m2val,m3val,m4val,
bottom, header(MAXLINE), footer(MAXLINE)
   integer curpag,newpag,lineno,plval,m1val,m2val,m3val,m4val,
   bottom, header, footer
ccmmcn /cparam/ fill, lsval, inval, rmval, tival, ceval, ulval
   integer fill, lsval, inval, rmval, tival, ceval, ulval

   ct = comtyp(buf)
   if (ct == UNKNOWN)   % ignore unknown commands
      return
   val = getval(buf, argtyp)
   if (ct == FI)
      call brk
      fill = YES

   else if (ct == NF)
      call brk
```

```
      fill = NO

   else if (ct == BR)
      call brk
   else if (ct == LS)
      call set(lsval, val, argtyp, 1, 1, HUGE)
   else if (ct == CE)
      call brk
      call set(ceval, val, argtyp, 1, 0, HUGE)

   else if (ct == UL)
      call set(ulval, val, argtyp, 0, 1, HUGE)
   else if (ct == HE)
      call gettl(buf, header)
   else if (ct == FO)
      call gettl(buf, footer)
   else if (ct == BP)
      if (lineno > 0)
         call space(HUGE)
      call set(curpag, val, argtyp, curpag+1, -HUGE, HUGE)
      newpag = curpag

   else if (ct == SP)
      call set(spval, val, argtyp, 1, 0, HUGE)
      call space(spval)

   else if (ct == IN)
      call set(inval, val, argtyp, 0, 0, rmval-1)
      tival = inval

   else if (ct == RM)
      call set(rmval, val, argtyp, PAGEWIDTH, tival+1, HUGE)
   else if (ct == TI)
      call brk
      call set(tival, val, argtyp, 0, 0, rmval)

   else if (ct == PL)
      call set(plval, val, argtyp, PAGELEN,
```

```
            m1val+m2val+m3val+m4val+1, HUGE)
         bottom = plval - m3val - m4val

      return
      end
% comtyp - decode command type
      integer function comtyp(buf)
      integer buf(MAXLINE)

      if (buf(2) == LETF & buf(3) == LETI)
         comtyp = FI
      else if (buf(2) == LETN & buf(3) == LETF)
         comtyp = NF
      else if (buf(2) == LETB & buf(3) == LETR)
         comtyp = BR
      else if (buf(2) == LETL & buf(3) == LETS)
         comtyp = LS
      else if (buf(2) == LETB & buf(3) == LETP)
         comtyp = BP
      else if (buf(2) == LETS & buf(3) == LETP)
         comtyp = SP
      else if (buf(2) == LETI & buf(3) == LETN)
         comtyp = IN
      else if (buf(2) == LETR & buf(3) == LETM)
         comtyp = RM
      else if (buf(2) == LETT & buf(3) == LETI)
         comtyp = TI
      else if (buf(2) == LETC & buf(3) == LETE)
         comtyp = CE
      else if (buf(2) == LETU & buf(3) == LETL)
         comtyp = UL
      else if (buf(2) == LETH & buf(3) == LETE)
         comtyp = HE
      else if (buf(2) == LETF & buf(3) == LETO)
         comtyp = FO
      else if (buf(2) == LETP & buf(3) == LETL)
         comtyp = PL
      else
```

```
      comtyp = UNKNOWN
   return
   end
% gettl - copy title from buf to ttl
   subroutine gettl(buf, ttl)
   integer buf(MAXLINE), ttl(MAXLINE)
   integer i

   i = 1              % skip command name
   while (buf(i) ~= BLANK & buf(i) ~= TAB & buf(i) ~= NEWLINE)
      i = i + 1
   call skipbl(buf, i)       % find argument
   if (buf(i) == SQUOTE | buf(i) == DQUOTE)    % strip quote if found
      i = i + 1
   call scopy(buf, i, ttl, 1)
   return
   end
% getval - evaluate optional numeric argument
   integer function getval(buf, argtyp)
   integer buf(MAXLINE)
   integer ctoi
   integer argtyp, i

   i = 1              % skip command name
   while (buf(i) ~= BLANK & buf(i) ~= TAB & buf(i) ~= NEWLINE)
      i = i + 1
   call skipbl(buf, i)       % find argument
   argtyp = buf(i)
   if (argtyp == PLUS | argtyp == MINUS)
      i = i + 1
   getval = ctoi(buf, i)
   return
   end
% getwrd - get non-blank word from in(i) into  out, increment i
   integer function getwrd(in, i, out)
   integer in(MAXLINE), out(MAXLINE)
   integer i, j
```

```
   while (in(i) == BLANK | in(i) == TAB)
      i = i + 1
   j = 1
   while (in(i)¬=EOS & in(i)¬=BLANK & in(i)¬=TAB & in(i)¬=NEWLINE)
      out(j) = in(i)
      i = i + 1
      j = j + 1

   out(j) = EOS
   getwrd = j - 1
   return
   end
% index - find character c in string str
   integer function index(str, c)
   integer c, str (ARB)
   for (index = 1; str(index) ¬= EOS; index = index + 1)
      if (str(index) == c)
         return
   index = 0
   return
   end
% init - set parameters to default values
   subroutine init

ccmmon /cout/ outp, outw, outwds, outbuf(MAXOUT)
   integer outp, outw, outwds, outbuf
common /cpage/ curpag,newpag,lineno,plval,m1val,m2val,m3val,m4val,
bottom, header(MAXLINE), footer(MAXLINE)
   integer curpag,newpag,lineno,plval,m1val,m2val,m3val,m4val,
   bottom, header, footer
common /cparam/ fill, lsval, inval, rmval, tival, ceval, ulval
   integer fill, lsval, inval, rmval, tival, ceval, ulval

   inval = 0
   rmval = PAGEWIDTH
   tival = 0
   lsval = 1
   fill = YES
```

```
   ceval = 0
   ulval = 0
   lineno = 0
   curpag = 0
   newpag = 1
   plval = PAGELEN
   m1val = 3; m2val = 2; m3val = 2; m4val = 3
   bottom = plval - m3val - m4val
   header(1) = NEWLINE; header(2) = EOS    % initial titles
   footer(1) = NEWLINE; footer(2) = EOS
   outp = 0
   outw = 0
   outwds = 0

   return
   end
% leadbl - delete leading blanks, set tival
   subroutine leadbl(buf)
   integer buf(MAXLINE)
   integer max
   integer i, j
common /cparam/ fill, lsval, inval, rmval, tival, ceval, ulval
   integer fill, lsval, inval, rmval, tival, ceval, ulval

   call brk
   for (i = 1; buf(i) == BLANK; i = i + 1)    % find 1st non-blank
      ;
   if (buf(i) ~= NEWLINE)
      tival = i - 1
   for (j = 1; buf(i) ~= EOS; j = j + 1)      % move line to left
      buf(j) = buf(i)
      i = i + 1

   buf(j) = EOS
   return
   end
% length - compute length of string
   integer function length(str)
```

```
   integer str(ARB)
   for (length = 0; str(length+ 1) ~= EOS; length = length + 1)
      ;
   return
   end
% pfoot - put out page footer
   subroutine pfoot
common /cpage/ curpag,newpag,lineno,plval,m1val,m2val,m3val,m4val,
bottom, header(MAXLINE), footer(MAXLINE)
   integer curpag,newpag,lineno,plval,m1val,m2val,m3val,m4val,
   bottom, header, footer

   call skip(m3val)
   if (m4val > 0)
      call puttl(footer, curpag)
      call skip(m4val-1)

   return
   end
% phead - put out page header
   subroutine phead
common /cpage/ curpag,newpag,lineno,plval,m1val,m2val,m3val,m4val,
bottom, header(MAXLINE), footer(MAXLINE)
   integer curpag,newpag,lineno,plval,m1val,m2val,m3val,m4val,
   bottom, header, footer

   curpag = newpag
   newpag = newpag + 1
   if (m1val > 0)
      call skip(m1val-1)
      call puttl(header, curpag)

   call skip(m2val)
   lineno = m1val + m2val + 1
   return
   end
% put - put out line with proper spacing and indenting
   subroutine put(buf)
```

```
   integer buf(MAXLINE)
   integer min
   integer i
common /cpage/ curpag,newpag,lineno,plval,m1val,m2val,m3val,m4val,
bottom, header(MAXLINE), footer(MAXLINE)
   integer curpag,newpag,lineno,plval,m1val,m2val,m3val,m4val,
   bottom, header, footer
common /cparam/ fill, lsval, inval, rmval, tival, ceval, ulval
   integer fill, lsval, inval, rmval, tival, ceval, ulval

   if (lineno == 0 | lineno > bottom)
      call phead
   for (i = 1; i <= tival; i = i + 1)       % indenting
      call putc(BLANK)
   tival = inval
   call putlin(buf, STDOUT)
   call skip(min(lsval-1, bottom-lineno))
   lineno = lineno + lsval
   if (lineno > bottom)
      call pfoot
   return
   end
% puttl - put out title line with optional page number
   subroutine puttl(buf, pageno)
   integer buf(MAXLINE)
   integer pageno
   integer i

   for (i = 1; buf(i) ~= EOS; i = i + 1)
      if (buf(i) == PAGENUM)
         call putdec(pageno, 1)
      else
         call putc(buf(i))
   return
   end
% putwrd - put a word in outbuf; includes margin justification
   subroutine putwrd(wrdbuf)
   integer wrdbuf(INSIZE)
```

```
   integer length, width
   integer last, llval, nextra, w
common /cout/ outp, outw, outwds, outbuf(MAXOUT)
   integer outp, outw, outwds, outbuf
common /cparam/ fill, lsval, inval, rmval, tival, ceval, ulval
   integer fill, lsval, inval, rmval, tival, ceval, ulval

   w = width(wrdbuf)
   last = length(wrdbuf) + outp + 1   % new end of outbuf
   llval = rmval - tival
   if (outp > 0 & (outw+w > llval | last >= MAXOUT))     % too big
      last = last - outp      % remember end of wrdbuf
      nextra = llval - outw + 1
      call spread(outbuf, outp, nextra, outwds)
      if (nextra > 0 & outwds > 1)
         outp = outp + nextra
      call brk          % flush previous line

   call scopy(wrdbuf, 1, outbuf, outp+1)
   outp = last
   outbuf(outp) = BLANK      % blank between words
   outw = outw + w + 1      % 1 for blank
   outwds = outwds + 1
   return
   end
% set - set parameter and check range
   subroutine set(param, val, argtyp, defval, minval, maxval)
   integer max, min
   integer argtyp, defval, maxval, minval, param, val

   if (argtyp == NEWLINE)      % defaulted
      param = defval
   else if (argtyp == PLUS)      % relative +
      param = param + val
   else if (argtyp == MINUS)    % relative -
      param = param - val
   else                % absolute
      param = val
```

```
   param = min(param, maxval)
   param = max(param, minval)
   return
   end
% skip - output  n  blank lines
   subroutine skip(n)
   integer i, n

   for (i = 1; i <= n; i = i + 1)
      call putc(PERIOD)
      call putc(NEWLINE)

   return
   end
% skipbl - skip blanks and tabs at lin(i)...
   subroutine skipbl(lin, i)
   integer lin(ARB)
   integer i

   while (lin(i) == BLANK | lin(i) == TAB)
      i = i + 1
   return
   end
% space - space  n  lines or to bottom of page
   subroutine space(n)
   integer min
   integer n
ccmmon /cpage/ curpag,newpag,lineno,plval,m1val,m2val,m3val,m4val,
bottom, header(MAXLINE), footer(MAXLINE)
   integer curpag,newpag,lineno,plval,m1val,m2val,m3val,m4val,
   bottom, header, footer

   call brk
   if (lineno > bottom)
      return
   if (lineno == 0)
      call phead
   call skip(min(n, bottom+1-lineno))
```

```
   lineno = lineno + n
   if (lineno > bottom)
      call pfoot
   return
   end
% spread - spread words to justify right margin
   subroutine spread(buf, outp, nextra, outwds)
   integer buf(MAXOUT)
   integer min
   integer dir, i, j, nb, ne, nextra, nholes, outp, outwds
   data dir /0/

   if (nextra <= 0 | outwds <= 1)
      return
   dir = 1 - dir    % reverse previous direction
   ne = nextra
   nholes = outwds - 1
   i = outp - 1
   j = min(MAXOUT-2, i+ne)    % leave room for NEWLINE, EOS
   while (i < j)
      buf(j) = buf(i)
      if (buf(i) == BLANK)
         if (dir == 0)
            nb = (ne-1) / nholes + 1
         else
            nb = ne / nholes
         ne = ne - nb
         nholes = nholes - 1
         for ( ; nb > 0; nb = nb - 1)
            j = j - 1
            buf(j) = BLANK


      i = i - 1
      j = j - 1

   return
   end
```

```
% text - process text lines (final version)
   subroutine text(inbuf)
   integer inbuf(INSIZE), wrdbuf(INSIZE)
   integer getwrd
   integer i
common /cparam/ fill, lsval, inval, rmval, tival, ceval, ulval
   integer fill, lsval, inval, rmval, tival, ceval, ulval

   if (inbuf(1) == BLANK | inbuf(1) == NEWLINE)
      call leadbl(inbuf)   % move left, set tival
   if (ulval > 0)          % underlining
      call underl(inbuf, wrdbuf, INSIZE)
      ulval = ulval - 1

   if (ceval > 0)          % centering
      call center(inbuf)
      call put(inbuf)
      ceval = ceval - 1

   else if (inbuf(1) == NEWLINE)    % all blank line
      call put(inbuf)
   else if (fill == NO)       % unfilled text
      call put(inbuf)
   else                % filled text
      for (i = 1; getwrd(inbuf, i, wrdbuf) > 0; )
         call putwrd(wrdbuf)
   return
   end
% underl - underline a line
   subroutine underl(buf, tbuf, size)
   integer i, j, size
   integer buf(size), tbuf(size)

   j = 1       % expand into tbuf
   for (i = 1; buf(i) ¬= NEWLINE & j < size-1; i = i + 1)
      tbuf(j) = buf(i)
      j = j + 1
      if (buf(i) ¬= BLANK & buf(i) ¬= TAB & buf(i) ¬= BACKSPACE)
```

```
            tbuf(j) = BACKSPACE
            tbuf(j+1) = UNDERLINE
            j = j + 2


      tbuf(j) = NEWLINE
      tbuf(j+1) = EOS
      call scopy(tbuf, 1, buf, 1)    % copy it back to buf
      return
      end
   % width - compute width of character string
      integer function width(buf)
      integer buf(MAXLINE)
      integer i

      width = 0
      for (i = 1; buf(i) ~= EOS; i = i + 1)
         if (buf(i) == BACKSPACE)
            width = width - 1
         else if (buf(i) ~= NEWLINE)
            width = width + 1
      return
      end
   % ctoi - convert string at in(i) to integer, increment i
      integer function ctoi(in, i)
      integer in(ARB)
      integer index
      integer d, i
   %   string digits "0123456789"
      integer digits(11)
      data digits(1) /DIG0/
      data digits(2) /DIG1/
      data digits(3) /DIG2/
      data digits(4) /DIG3/
      data digits(5) /DIG4/
      data digits(6) /DIG5/
      data digits(7) /DIG6/
      data digits(8) /DIG7/
```

```
   data digits(9) /DIG8/
   data digits(10) /DIG9/
   data digits(11) /EOS/

   while (in(i) == BLANK | in(i) == TAB)
      i = i + 1
   for (ctoi = 0; in(i) ~= EOS; i = i + 1)
      d = index(digits, in(i))
      if (d == 0)          % non-digit
         break
      ctoi = 10 * ctoi + d - 1

   return
   end
% itoc - convert integer  int  to char string in  str
   integer function itoc(int, str, size)
   integer abs, mod
   integer d, i, int, intval, j, k, size
   integer str(size)
%   string digits "0123456789"
   integer digits(11)
   data digits(1) /DIG0/
   data digits(2) /DIG1/
   data digits(3) /DIG2/
   data digits(4) /DIG3/
   data digits(5) /DIG4/
   data digits(6) /DIG5/
   data digits(7) /DIG6/
   data digits(8) /DIG7/
   data digits(9) /DIG8/
   data digits(10) /DIG9/
   data digits(11) /EOS/

   intval = abs(int)
   str(1) = EOS
   i = 1
   repeat                  % generate digits
      i = i + 1
```

```
      d = mod(intval, 10)
      str(i) = digits(d+1)
      intval = intval / 10
        until (intval == 0 | i >= size)
   if (int < 0 & i < size)          % then sign
      i = i + 1
      str(i) = MINUS

   itoc = i - 1
   for (j = 1; j < i; j = j + 1)      % then reverse
      k = str(i)
      str(i) = str(j)
      str(j) = k
      i = i - 1

   return
   end
% putdec - put decimal integer  n  in field width >= w
   subroutine putdec(n, w)
   integer chars(MAXCHARS)
   integer itoc
   integer i, n, nd, w

   nd = itoc(n, chars, MAXCHARS)
   for (i = nd + 1; i <= w; i = i + 1)
      call putc(BLANK)
   for (i = 1; i <= nd; i = i + 1)
      call putc(chars(i))
   return
   end
% scopy - copy string at from(i) to to(j)
   subroutine scopy(from, i, to, j)
   integer from(ARB), to(ARB)
   integer i, j, k1, k2

   k2 = j
   for (k1 = i; from(k1) ~= EOS; k1 = k1 + 1)
      to(k2) = from(k1)
```

```
      k2 = k2 + 1

   to(k2) = EOS
   return
   end
% inmap - convert left adjusted external rep to right adj ascii
   integer function inmap(inchar)
   integer i, inchar
common /cchar/ extdig(10), intdig(10), extlet(26), intlet(26),
   extbig(26), intbig(26), extchr(33), intchr(33),extblk,intblk
   integer extdig, intdig, extlet, intlet, extbig, intbig,
   extchr, intchr, extblk, intblk

   if (inchar == extblk)
      inmap = intblk
      return

   do i = 1, 10
      if (inchar == extdig(i))
         inmap = intdig(i)
         return

   do i = 1, 26
      if (inchar == extlet(i))
         inmap = intlet(i)
         return

   do i = 1, 26
      if (inchar == extbig(i))
         inmap = intbig(i)
         return

   do i = 1, NCHARS
      if (inchar == extchr(i))
         inmap = intchr(i)
         return

   inmap = inchar
```

```
      return
      end
% outmap - convert right adj ascii to left adjusted external rep
      integer function outmap(inchar)
      integer i, inchar
common /cchar/ extdig(10), intdig(10), extlet(26), intlet(26),
     extbig(26), intbig(26), extchr(33), intchr(33),
     extblk, intblk
      integer extdig, intdig, extlet, intlet, extbig, intbig,
     extchr, intchr, extblk, intblk

      if (inchar == intblk)
         outmap = extblk
         return

      do i = 1, 10
         if (inchar == intdig(i))
            outmap = extdig(i)
            return

      do i = 1, 26
         if (inchar == intlet(i))
            outmap = extlet(i)
            return

      do i = 1, 26
         if (inchar == intbig(i))
            outmap = extbig(i)
            return

      do i = 1, NCHARS
         if (inchar == intchr(i))
            outmap = extchr(i)
            return

      outmap = inchar
      return
      end
```

```
% putc (interim version)  put characters
   subroutine putc(c)
   integer buf(MAXLINE), c
   integer outmap
   integer f, i, lastc
   data lastc /0/

   if (lastc >= 300 | c == NEWLINE)
      if ( lastc <= 0 )
         write(f,2)
         2 format(/)
      else

         write(f, 1) (buf(i), i = 1, lastc)
         1 format(120 a1)

      lastc = 0

   if (c ~= NEWLINE)
      lastc = lastc + 1
      buf(lastc) = outmap(c)

   return
   end
% putlin - put out line by repeated calls to putch
   subroutine putlin(b, f)
   integer b(ARB)
   integer f, i

   for (i = 1; b(i) ~= EOS; i = i + 1)
      call putch(b(i), f)
   return
   end
   block data
common /cchar/ extdig(10), intdig(10), extlet(26), intlet(26),
   extbig(26), intbig(26), extchr(NCHARS), intchr(NCHARS),
   extblk, intblk
   integer extdig   % external representation of digits
```

```
integer intdig    % internal rep (ascii)
integer extlet    % external rep of letters (normal case)
integer intlet    % internal rep (ascii lower case)
integer extbig    % external rep of upper case, if used
integer intbig    % internal rep (upper case ascii)
integer extchr    % external rep of special chars
integer intchr    % internal rep (ascii)
integer extblk    % external blank
integer intblk    % internal blank (ascii)
% character set definitions:

data extblk /' '/, intblk /BLANK/

data extdig(1) /'0'/, intdig(1) /DIG0/
data extdig(2) /'1'/, intdig(2) /DIG1/
data extdig(3) /'2'/, intdig(3) /DIG2/
data extdig(4) /'3'/, intdig(4) /DIG3/
data extdig(5) /'4'/, intdig(5) /DIG4/
data extdig(6) /'5'/, intdig(6) /DIG5/
data extdig(7) /'6'/, intdig(7) /DIG6/
data extdig(8) /'7'/, intdig(8) /DIG7/
data extdig(9) /'8'/, intdig(9) /DIG8/
data extdig(10) /'9'/, intdig(10) /DIG9/

% normal case of letters

data extlet(1) /z81404040/, intlet(1) /LETA/
data extlet(2) /z82404040/, intlet(2) /LETB/
data extlet(3) /z83404040/, intlet(3) /LETC/
data extlet(4) /z84404040/, intlet(4) /LETD/
data extlet(5) /z85404040/, intlet(5) /LETE/
data extlet(6) /z86404040/, intlet(6) /LETF/
data extlet(7) /z87404040/, intlet(7) /LETG/
data extlet(8) /z88404040/, intlet(8) /LETH/
data extlet(9) /z89404040/, intlet(9) /LETI/
data extlet(10) /z91404040/, intlet(10) /LETJ/
data extlet(11) /z92404040/, intlet(11) /LETK/
data extlet(12) /z93404040/, intlet(12) /LETL/
```

```
data extlet(13) /z94404040/, intlet(13) /LETM/
data extlet(14) /z95404040/, intlet(14) /LETN/
data extlet(15) /z96404040/, intlet(15) /LETO/
data extlet(16) /z97404040/, intlet(16) /LETP/
data extlet(17) /z98404040/, intlet(17) /LETQ/
data extlet(18) /z99404040/, intlet(18) /LETR/
data extlet(19) /za2404040/, intlet(19) /LETS/
data extlet(20) /za3404040/, intlet(20) /LETT/
data extlet(21) /za4404040/, intlet(21) /LETU/
data extlet(22) /za5404040/, intlet(22) /LETV/
data extlet(23) /za6404040/, intlet(23) /LETW/
data extlet(24) /za7404040/, intlet(24) /LETX/
data extlet(25) /za8404040/, intlet(25) /LETY/
data extlet(26) /za9404040/, intlet(26) /LETZ/

% upper case of letters

data extbig(1) /'A'/, intbig(1) /BIGA/
data extbig(2) /'B'/, intbig(2) /BIGB/
data extbig(3) /'C'/, intbig(3) /BIGC/
data extbig(4) /'D'/, intbig(4) /BIGD/
data extbig(5) /'E'/, intbig(5) /BIGE/
data extbig(6) /'F'/, intbig(6) /BIGF/
data extbig(7) /'G'/, intbig(7) /BIGG/
data extbig(8) /'H'/, intbig(8) /BIGH/
data extbig(9) /'I'/, intbig(9) /BIGI/
data extbig(10) /'J'/, intbig(10) /BIGJ/
data extbig(11) /'K'/, intbig(11) /BIGK/
data extbig(12) /'L'/, intbig(12) /BIGL/
data extbig(13) /'M'/, intbig(13) /BIGM/
data extbig(14) /'N'/, intbig(14) /BIGN/
data extbig(15) /'O'/, intbig(15) /BIGO/
data extbig(16) /'P'/, intbig(16) /BIGP/
data extbig(17) /'Q'/, intbig(17) /BIGQ/
data extbig(18) /'R'/, intbig(18) /BIGR/
data extbig(19) /'S'/, intbig(19) /BIGS/
data extbig(20) /'T'/, intbig(20) /BIGT/
data extbig(21) /'U'/, intbig(21) /BIGU/
```

```
data extbig(22) /'V'/, intbig(22) /BIGV/
data extbig(23) /'W'/, intbig(23) /BIGW/
data extbig(24) /'X'/, intbig(24) /BIGX/
data extbig(25) /'Y'/, intbig(25) /BIGY/
data extbig(26) /'Z'/, intbig(26) /BIGZ/

% special ers. some of these may
% change for your machine

data extchr(1) /'!'/, intchr(1) /NOT/   % use exclam for not-sign
data extchr(2) /'"'/, intchr(2) /DQUOTE/
data extchr(3) /"%"/, intchr(3) /SHARP/
data extchr(4) /'$'/, intchr(4) /DOLLAR/
data extchr(5) /'%'/, intchr(5) /PERCENT/
data extchr(6) /'&'/, intchr(6) /AMPER/
data extchr(7) /"'"/, intchr(7) /SQUOTE/
data extchr(8) /'('/, intchr(8) /LPAREN/
data extchr(9) /')'/, intchr(9) /RPAREN/
data extchr(10) /'*'/, intchr(10) /STAR/
data extchr(11) /'+'/, intchr(11) /PLUS/
data extchr(12) /','/, intchr(12) /COMMA/
data extchr(13) /'-'/, intchr(13) /MINUS/
data extchr(14) /'.'/, intchr(14) /PERIOD/
data extchr(15) /'/'/, intchr(15) /SLASH/
data extchr(16) /':'/, intchr(16) /COLON/
data extchr(17) /';'/, intchr(17) /SEMICOL/
data extchr(18) /'<'/, intchr(18) /LESS/
data extchr(19) /'='/, intchr(19) /EQUALS/
data extchr(20) /'>'/, intchr(20) /GREATER/
data extchr(21) /'?'/, intchr(21) /QMARK/
data extchr(22) /'@'/, intchr(22) /ATSIGN/
data extchr(23) /'['/, intchr(23) /LBRACK/
data extchr(24) /'\'/, intchr(24) /BACKSLASH/
data extchr(25) /']'/, intchr(25) /RBRACK/
data extchr(26) /'_'/, intchr(26) /UNDERLINE/
data extchr(27) /'{'/, intchr(27) /LBRACE/
data extchr(28) /'|'/, intchr(28) /BAR/
data extchr(29) /'}'/, intchr(29) /RBRACE/
```

```
  data extchr(30) /' '/, intchr(30) /BACKSPACE/
data extchr(31) /' '/, intchr(31) /TAB/
  data extchr(32) /'¬'/, intchr(32) /NOT/   % use caret for not-sign
  data extchr(33) /'°'/, intchr(33) /NOT/   % use tilde for not-sign
  % NCHARS is last subscript in this array

  end
```

APPENDIX C

Sample Preprocessed Program Execution

```
vm/370 online      1jh359 qsyosu

logcn vmwr1 300k
ENTER ACCOUNT NUMBER
XXXXXXXX
ENTER SS NUMBER
XXXXXXXX
LOGCN AT 19:13:48 CST TUESDAY 11/08/77
CMS V3 PLC 8 - 8/9/77 05: 16
Y-10/31/77 17:19  * 'CP TERM PROMPT BEL' is now the default.
FORMATTING DISK 'A'
R;
edit getc watfiv
NEWFILE:
EDIT:
case m
input
INPUT:
define(MAXCARD,80)
define(BLANK,32)
define(NEWLINE,10)
define(EOF,1003)
define(EOF,10003)
% copy - copy input characters to output
   integer getc
   integer c
   while(getc(c) -= EOF)
     call putc(c)
   stop
   end
% getc - get character from standard input
   integer function getc(c)
   integer buf(MAXLINE),c
   integer i, lastc
   data lastc/MAXLINE/, buf(MAXLINE)/NEWLINE/
   lastc = lastc + 1
   if(lastc > MAXLINE) $(
     read(10,100,end=10)(buf(i),i=1,MAXCARD)
       100 format(MAXCARD a1)
     lastc = 1
     $)
   c = buf(lastc)
   getc = c
   return
10 c = EOF
   getc = EOF
   return
   end


file
R;
```

```
filedef 05 disk getc watfiv a1 (perm
R;
filedef 01 disk stdout fortran a1 (perm recfm ft block
800 lrecl 80)
R;
runfort ratfor text
EXECUTION BEGINS...


    *** BEGIN RATFOR PREPROCESSOR ***
    *** WAIT ***
    *** END RATFOR PREPROCESSOR ***
R;

type stdout fortran

      INTEGERGETC
      INTEGERC
      CONTINUE
23000 IF(.NOT.(GETC(C).NE.10003))GOTO23001
      CALLPUTC(C)
      GOTO23000
23001 CONTINUE
      STOP
      END
      INTEGERFUNCTIONGETC(C)
      INTEGERBUF(81),C
      DATALASTC/81/,BUF(81)/10/
      LASTC=LASTC+1
      IF(.NOT.(LASTC.GT.81))GOTO23002
      READ(10,100,END=10)(BUF(I),I=1,80)
100   FORMAT(80A1)
      LASTC=1
23002 CONTINUE
      C=BUF(LASTC)
      GETC=C
      RETURN
10    C=10003
      GETC=10003
      RETURN
      END
      SUBROUTINEPUTC(C)
      INTEGERBUF(80),C
      INTEGERI,LASTC
      DATALASTC/0/
      IF(.NOT.(LASTC.GT.80.OR.C.EQ.10))GOTO23004
      CONTINUE
      I=LASTC+1
23006 IF(.NOT.(I.LE.80))GOTO23008
      BUF(I)=32
23007 I=I+1
      GOTO23006
```

```
23008 CONTINUE
      WRITE(9,100)(BUF(I),I=1,80)
100   FORMAT(' ',80A1)
      LASTC=0
23004 CONTINUE
      IF(.NOT.(C.NE.10))GOTO23009
      LASTC=LASTC+1
      BUF(LASTC)=C
23009 CONTINUE
      RETURN
      END

R;
csjob stdout fortran a1 (proc fortgc parm deck punch print
time ,20)

11/08/77 -- YOUR SEQUENCE NUMBER IS VM192148
R;

PRT FILE 7939 FROM CS
PUN FILE 7940 FROM OS

R;
read stdout listing
RECORD LENGTH IS '132' BYTES.
R;
type stdout listing

         HASP  SYSTEM  LOG
19.21.56 JOB 627 -- VM192148 --
19.22.23 JOB 627 END EXECUTION.
/VM192148 JOB (XXXXXX,XXXXXXXX, ,5,
/      5000,1601,1),
/      'FIODD, JOHN',TIME=(,20)
**ROUTE PUNCH VMWR1
**ROUTE PRINT VMWR1
/OSJOB  EXEC  FORTGC,PARM='DECK'
/SYSIN  DD  DATA DLM='/*'

SU0031 REGION SIZE - 256K, MAXIMUM CORE USED - 90K
SU0041 EXCP COUNT - UR 343  114, UR 380  63, DA 251
SU0011 STEP 1 FORT EXECUTION TIME = 1.2 SEC RETURN CODE =0

ht
R;
erase stdout listing
R;
read stdout text
R;

filedef 10 disk  tesfile watfiv a1 (perm
R;
```

Sample Execution

```
runfort stdout text
EXECUTION BEGINS...

THIS IS THE CONTENTS OF THE FILE, TESFILE, BEING
PRINTED.  TESFILE IS BEING READ FROM ONE FILE AND
COPIED TO ANOTHER FILE, 09.  A FILE NUMBER OF 09
CAUSES THE OUTPUT TO BE PRINTED AT THE TERMINAL.
R;
logoff
 CHECKPOINTING...
FILE 'STDOUT WATFIV' NOT SAVED; PROJECT SPACE FULL.
DO YOU WISH TO PROCEED WITH LOGOFF (YES/NO) ?
yes
CONNECT= 00:31:42
```

USER'S GUIDE FOR THE RATIONAL FORTRAN PREPROCESSOR SOFTWARE PACKAGE

by

Benzell Floyd

B.S., University of Southern Mississippi, 1974

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

ABSTRACT

The Rational Fortran (RatFor) Preprocessor is a preprocessor which allows for a simple extension of the Fortran computer language. Alone, Fortran is a poor language for programming or for describing programs. The Rational Fortran extension provides modern control flow statements like those in Pl/I, COBOL, and ALGOL. It avoids the use of Go To statements thus enabling one to write structured programs properly. RatFor is easy to read, write, and understand, and readily translates into fortran.

This project describes the implementation and the operation procedures necessary to execute the RatFor Preprocessor and RatFor programs in the IBM 370's CMS environment. Included are detailed descriptions of changes made to the programs; detailed descriptions of RatFor statements and examples of their usage; actual examples of the preprocessor's output; some module access graphs; a module access matrix; and listings of sample executions of RatFor programs that are found in the book, Software Tools, by Brian W. Kernighan and P.J. Plauger. The sample executions will include the RatFor File Copying program and the RatFor Text Formatter program.