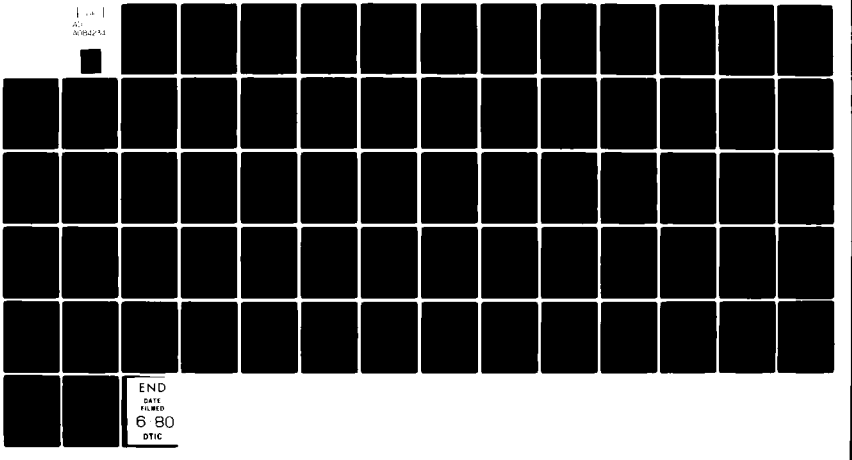
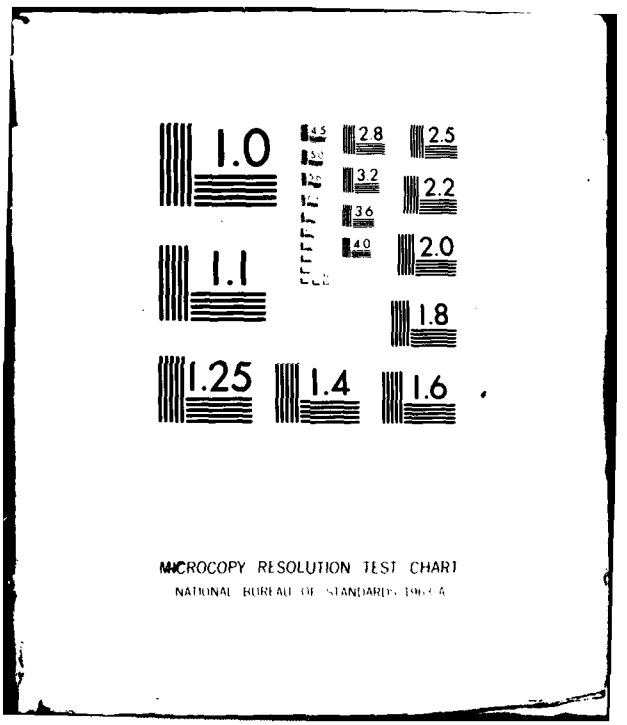
AD-A084 234 VIRGINIA POLYTECHNIC INST AND STATE UNIV WASHINGTON --ETC F/G 9/2
CMS RATFOR SYSTEM MANUAL.(U)
JUL 79 S M CHOQUETTE, R J ORGASS AFOSR-79-0021
UNCLASSIFIED VPI/SU-TM-79-4 AFOSR -TR-80-0277 NL



END
DATE FILMED
6 80
DTIC


1.0
1.1
1.25
1.4
1.6
2.8
2.5
3.2
2.2
3.6
2.0
4.0
1.8
MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS 1963-A

EXTENSION DIVISION

VIRGINIA POLYTECHNIC INSTITUTE AND STATE UNIVERSITY

DEPARTMENT OF COMPUTER SCIENCE
GRADUATE PROGRAM IN NORTHERN VIRGINIA

P. O. Box 17186
Washington, D. C. 20041
(703) 471-4600

CMS RATFOR SYSTEM MANUAL*†

Stephen M. Choquette
and
Richard J. Orgass

Technical Memorandum No. 79-4

July 1, 1979

ABSTRACT

RATFOR is a preprocessor for Fortran that provides modern control structures and a substantial improvement in the syntax of Fortran programs. The output of RATFOR is a Fortran program that is compiled by the Fortran processors and then executed.

The RATFOR preprocessor provides statement grouping, IF-ELSE structures and four loops: DO, FOR, WHILE and REPEAT-UNTIL.

RATFOR source text is free format with multiple statements on a line. Upper and lower case letters are treated as upper case letters except in character constants. There is an include facility so that large programs can be constructed out of a multitude of small files without using the system editor. RATFOR accepts files consisting of fixed length 80 column records with imbedded tabs.

The program described here is an adaptation of the original RATFOR processor written at Bell Laboratories for use in the CMS environment.

This manual provides a detailed description of the processor and is intended for readers who wish to modify the processor. The user's manual is Technical Memorandum No. 79-5.

---

* Research sponsored by the Air Force Office of Scientific Research, Air Force Systems Command, under Grant No. AFOSR-79-0021. The United States Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright notation hereon.

† The information in this document is subject to change without notice. The authors, Virginia Polytechnic Institute and State University, the Commonwealth of Virginia and the United States Government assume no responsibility for errors that may be present in this document or in the program described here.

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE	READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER: AFOSR-TR-80-0277	2. GOVT ACCESSION NO.: AD-A084234
	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle): CMS RATFOR SYSTEM MANUAL	5. TYPE OF REPORT & PERIOD COVERED: Interim rept.
	6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s): Stephen M. Choquette; Richard J. Orgass	8. CONTRACT OR GRANT NUMBER(s): AFOSR-79-0021
9. PERFORMING ORGANIZATION NAME AND ADDRESS: Virginia Polytechnic Inst. & State University, Department of Computer Science, Washington, DC 20041	10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS: 61102F; 2304; A2
11. CONTROLLING OFFICE NAME AND ADDRESS: Air Force Office of Scientific Research/NM, Bolling AFB, Washington, DC 20332	12. REPORT DATE: 1 July 1979
	13. NUMBER OF PAGES: 68
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office)	15. SECURITY CLASS. (of this report): UNCLASSIFIED
	15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited.

14 VPI/SU-TM-79-4

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

RATFOR is a preprocessor for Fortran that provides modern control structures and a substantial improvement in the syntax of Fortran programs. The output of RATFOR is a Fortran program that is compiled by the Fortran processors and then executed.

The RATFOR preprocessor provides statement grouping, IF-ELSE structures and four loops: DO, FOR, WHILE and REPEAT-UNTIL.

(OVER)

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

411731

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

20. Abstract cont.

RATFOR source text is free format with multiple statements on a line. Upper and lower case letters are treated as upper case letters except in character constants. There is an include facility so that large programs can be constructed out of a multitude of small files without using the system editor. RATFOR accepts files consisting of fixed length 80 column records with imbedded tabs.

The program described here is an adaption of theoriginal RATFOR processor written at Bell Laboratories for use in the CMS environment.

This manual provides a detailed description of the processor and is intended for readers who wish to modify the processor.

RATFOR System Manual

* * * TABLE OF CONTENTS * * *

Summary of RATFOR Features

With the growing concern for the cost of program development, the computer industry has seen a shift towards programming languages that emphasize an organized approach to program development. RATFOR is a new, rational approach to programming in FORTRAN; the language offers RATFOR users the universality and efficiency of FORTRAN, while providing decent program flow control structures. RATFOR, implemented as a preprocessor to FORTRAN, offers the busy programmer statement grouping, IF-ELSE segmenting, and DO, FOR, WHILE, and REPEAT-UNTIL loops. Additional RATFOR features and keywords make code maintenance a less painful task.

This paper is the System Manual for the RATFOR preprocessor on the IBM CMS timesharing system. Included in this paper is a language description of RATFOR, an explanation of the preprocessor programs and tables, and a detailed explanation of the RATFOR error messages. Within the language description section is a quick reference guide listing the keyword syntax and RATFOR cosmetic features.

RATFOR is an attempt to hide the worst of FORTRAN's deficiencies (primarily hard to understand code), while retaining the advantages of the language (universality, portability, efficiency). RATFOR offers the user powerful program flow control statements without the FORTRAN necessities of GOTOs and labeled statements. Additionally, the RATFOR language provides the user with many "pretty print" features so that a program in RATFOR would be easier to understand and maintain.

The remainder of the Language Description section will explain both the syntactic and cosmetic features of RATFOR. The program flow control structures are typical of the newer high level programming languages. For a quick reference to RATFOR, flip to the Quick Reference Guide at the end of the Language Description section.

Statement Grouping

Often a programmer will want to group a sequence of statements together for execution on a certain condition. Generally, the programmer needs something to say "if some condition, do these things." In RATFOR, statements can be grouped by enclosing them within brackets. For example,

```
IF (A >= 100)
   { C = C + 1
     SUM = SUM + A
     L = 0
    }
```

The example above is a legal RATFOR program segment. Note that the brackets denote a sequence of statements to execute when A >= 100. The brackets perform the same function as the PL/1 DO/END sequence or the PASCAL BEGIN/END sequence.

To the avid FORTRAN programmer, a few things will stand out in the above example. First, RATFOR allows free form input; statements can occur in any column. When RATFOR encounters a statement starting with an all-numeric field, the processor assumes the field is a FORTRAN label and places it in columns 1-5 of the output. Next, the FORTRAN user will observe that '>=' is not a legal FORTRAN boolean relational operator. RATFOR will translate the more understandable relational operators (>,>=,==,<,<=,¬,¬=,&,|) into their FORTRAN equivalent. The last observation the FORTRAN user will make is the semicolon. Being a free form language, RATFOR allows more than one statement per line. The additional statements must be separated by a semicolon. When only one statement is on a line, the semicolon is optional. Thus, the previous example could be written as:

```
IF(A>=100) {C=C+1;SUM=SUM+A;L=0;}
```

Obviously the first form is easier to understand. One final comment is needed; when a statement is obviously not finished on one line, RATFOR assumes it will be continued on the next line. No character is needed in column 6.

ELSE Clauses

Occassionally the FORTRAN programmer will want to say "if some condition, do these things, otherwise do these." The ELSE clause provides this option. Naturally the ELSE clause may be left off. The full format of the IF statement is

```
IF (legal FORTRAN condition)
   RATFOR statement
 ELSE
   RATFOR statement
```

The RATFOR statement can be in one of three forms: 1) a single FORTRAN statement (no brackets needed), 2) a bracketed segment of RATFOR and FORTRAN statements, or 3) another RATFOR keyword.

Notice that the third case allows us the option of nested IFs. A legal sequence of RATFOR statements is

```
IF (A == B)
   CTR1=CTR1+1
 ELSE IF (A>B)
      CTR2=CTR2+1
   ELSE
      CTR3=CTR3+1
```

Like many languages allowing the IF-ELSE construct, the question arises of which IF does the ELSE match with. This ambiguity is resolved by matching the ELSE with the last unmatched IF. Because RATFOR allows free form input, the user should use indentation to clarify the program listing.

The DO Statement

The RATFOR DO statement is very similar to the FORTRAN DO statement. The major omission is the lack of a statement number. The format of the RATFOR DO statement is

```
DO legal-FORTRAN-DO-test
   RATFOR statement
```

As before, the RATFOR statement can be a single statement or a sequence of RATFOR statements in brackets. Since the RATFOR statement can be another RATFOR statement, the following sequence is legal.

```
IF (A ~= B)
   DO I=1,5
      IF (SWITCH(i) == 1)
         CTR1=CTR1+1
       ELSE
         CTR2=CTR2+1
```

Notice that in each case, a single RATFOR statement follows the IF and DO statements. Occassionally brackets will help clarify the listing, although they are not necessary.

The BREAK Keyword

The BREAK keyword provides a way of exiting a loop without using the FORTRAN GOTO statement. BREAK can be followed by an integer (BREAK N) specifying how many levels of looping to exit from. The following sequence locates the first non-blank character in a string array

```
DO I=1,80
   IF (STRING(I) == BLANK)
      BREAK
```

BREAK jumps to the statement after the end of the specified loop.

The NEXT Keyword

Like BREAK, the NEXT keyword provides a means of loop control without the GOTO statement. NEXT jumps to the iteration step of the specified loop. NEXT can also be followed by an integer (NEXT N) giving the loop level to go to

```
DO I=1,80
   { IF (STR(I)==BLANK)
        NEXT
     STR(I)=STAR
    }
```

This sequence of code sets all non-blank characters in a string array to '*'. STAR is assumed defined above.

The WHILE Statement

WHILE provides a more powerful looping structure than the simple DO loop. The syntax of the WHILE statement is

```
WHILE (legal FORTRAN condition)
     RATFOR statement
```

Notice that the WHILE loop checks the FORTRAN condition at the start of the loop so that the loop may be executed zero times. The sequence

```
I=80
WHILE(STR(I)~=BLANK)
   I=I+1
```

locates the last non-blank character in an input line of 80 characters. Of course the NEXT and BREAK statements can occur within a WHILE loop. The NEXT statement in a WHILE loop goes to the test condition.

The FOR Statement

The FOR statement is yet another powerful RATFOR loop structure. The syntax of the FOR statement is

```
FOR (initial; condition; increment)
    RATFOR statement
```

where initial is any one FORTRAN statement, condition is the stopping condition and increment is the final step (a single statement) in the loop. Any of the fields can be null so long as the semicolon delimiter is present. Our last non-blank character example above is

```
FOR (I=80; STR(I)~=BLANK; I=I-1)
    ;
```

Notice that the actual loop portion is the null statement. This is because everything we will want to do is in the FOR statement. As may be obvious by now, certain loop constructs work better in different situations. Choose the best and simplest for your work. The NEXT statement in a FOR loop goes to the increment step of the loop.

The REPEAT-UNTIL Statement

The REPEAT-UNTIL construct is the last loop control structure in RATFOR. It provides a means of checking the exit condition at the bottom of the loop. Recall that WHILE and FOR check at the top of loops. The syntax of the statement is

```
REPEAT
   RATFOR statement
 UNTIL (legal FORTRAN condition)
```

The UNTIL is optional and, if omitted, provides an infinite loop. Of course the programmer will want to get out of the loop using BREAK, STOP, or RETURN. Caution should be used with the REPEAT-

UNTIL construct as it does not test for the null case.

The RETURN Statement

The standard FORTRAN RETURN mechanism uses the function name to return a value. This is allowable in RATFOR as well as expressions of the form

```
RETURN (value)
```

If there are no parentheses, a normal RETURN is made.

The DEFINE Statement

The DEFINE statement is not an executed RATFOR statement; no FORTRAN code is generated. The statement allows the user to create program definitions to make his program more understandable. The syntax is

```
DEFINE (name, definition)
              or
DEFINE  name, definition
```

Every occurrence of name in the user source code is immediately replaced by the definition. Optimally, DEFINEs should be at the start of the source code to clarify their use. The name portion can be arbitrarily long and must start with a letter. The DEFINE statement can be used to define global constants as follows

```
DEFINE (YES,1)
DEFINE (NO,Ø)
```

With the above statements, we can now say

```
IF (ISMTQ == 1)
   RETURN (YES)
 ELSE
   RETURN (NO)
```

The INCLUDE Statement

The INCLUDE statement inserts files directly in the RATFOR source code input. The statement

```
INCLUDE RATCMN
```

inserts the CMS file RATCMN (possibly containing COMMON blocks) into the user source code in place of the INCLUDE statement. Thus, the programmer would type the program COMMON blocks once into the RATCMN file and then, in each subroutine needing the

COMMON blocks, insert "INCLUDE RATCMN." Of course changes to the RATCMN file would affect every subroutine having the INCLUDE statement. The syntax of the statement is

```
INCLUDE  FN  FT  FM
```

Where FN is the CMS file name (8 characters maximum), FT the CMS file type (optional - Default RATFOR), and FM the CMS file mode (optional - Default A1).

RATFOR Cosmetic Features

As mentioned before, RATFOR provides many cosmetic features to allow the user a sharp looking listing. In addition to a sharper listing, the use of cosmetics makes the source listing more readable and easier to maintain.

First, RATFOR allows free format input. Statements can occur anywhere on a line. If more than one statement is on a line, they must be separated by a semicolon. Blank lines are ignored. The user need not worry about a long statement continuing to a new line; RATFOR can make a fair estimate whether the statement is a continuation. Lines ending with any of the characters

```
= + - * , | & (
```

are assumed to be continued on the next line.

The next cosmetic feature is RATFOR commenting. Comments start with a # and can occur anywhere in a line. Thus, comments can occur next to source statements. Comments are assumed to continue until the end of the line.

RATFOR will perform translation services for the user whenever they are needed, except within single or double quotes:

==	to	.eq.	~=	to	.ne.
>	to	.gt.	>=	to	.ge.
<	to	.lt.	<=	to	.le.
&	to	.and.	|	to	.or.
!	to	.not.	~	to	.not.

Additionally, the statement grouping brackets can be either { and }, [and], or $(and $).

One important cosmetic feature is that RATFOR input can be in upper and lower case. Anything not within single or double quotes is translated to upper case for the CMS FORTRAN compiler.

Lastly, text within matching single or double quotes is converted to its Hollerith equivalent ('string'=6Hstring). Within quoted strings, the backslash '\' serves as an escape character; the next character is taken literally. This way a single quote can be entered as "\'" .

Quick Reference Guide to RATFOR

(Keyword Syntax)

BREAK Keyword

```
BREAK N                (N=1 by default)
```

Exits from N levels of enclosing loops.

DEFINE Statement

```
DEFINE (defined name, defined value)
                    or
DEFINE defined name, defined value
```

Defined name may be arbitrarily long and must start with a letter.

DO Statement

```
DO  legal-FORTRAN-DO-test
    RATFOR statement
```

FOR Statement

```
FOR (initial; condition; increment)
    RATFOR statement
```

Initial - any single FORTRAN statement
Condition - any legal RATFOR condition
Increment - any single FORTRAN statement

IF Statement

```
IF (legal FORTRAN condition)
   RATFOR statement
 ELSE
   RATFOR statement
```

The ELSE is optional and is matched with the last IF.

INCLUDE Statement

```
INCLUDE    FN     FT     FM
```

FN - CMS file name (8 characters maximum)
FT - CMS file type (Optional - Default RATFOR)
FM - CMS file mode (Optional - Default A1)

NEXT

```
NEXT N                        (N=1 by default)
```

Branches to next iteration of Nth loop.

REPEAT-UNTIL Statement

```
    REPEAT
    RATFOR statement
 UNTIL (legal FORTRAN condition)
```

RETURN

```
RETURN (expression)
```

WHILE Statement

```
WHILE (legal FORTRAN condition)
   RATFOR statement
```

*** A RATFOR statement can be any of the following:
1. A single FORTRAN statement
2. A bracketed set of statements
3. Any of the RATFOR statements just described

Quick Reference Guide to RATFOR

(Cosmetics)

* Free form input (ie. spacing is not important).

* Lines ending with = + - * , / | & (are assumed to be continued. No continuation signal is needed.

* Statements beginining with an all-numeric field is assumed to be a FORTRAN label and is placed in columns 1-5 of the output.

* Strings in matching single or double quotes are converted to Hollerith form ('string'=6Hstring).

* Statement grouping using either { and }, [and], or $(and $).

* Comments beginning anywhere in the input, denoted by #.

* Translation services

==	to	.eq.	~=	to	.ne.
>	to	.gt.	>=	to	.ge.
<	to	.lt.	<=	to	.le.
&	to	.and.	|	to	.or.
~	to	.not.			

RATFOR Reference

Kernighan, Brian W. "RATFOR - A Preprocessor for a Rational FORTRAN," Bell Laboratories Technical Report 55, January 1, 1977.

Structure of the RATFOR Preprocessor

The RATFOR preprocessor is organized along lines that will make it easy to maintain and to add new features. The two dominant procedures are LEX and PARSE. LEX translates RATFOR keywords into an internal numeric type. The internal type is expressed in terms of a defined name so that, to modify the type, only one change to the DEFINE statement has to be performed. PARSE takes the internal types assigned by LEX, and calls an appropriate subroutine to parse the RATFOR keyword. For each looping keyword (DO, REPEAT, UNTIL, WHILE, FOR) there are two subroutines; the first generates FORTRAN code for the top of the loop and the second handles code generation for the end of the loop. In addition to the keyword parsing subroutines, there are various internal functions to perform tasks like determining the length of a string.

Since FORTRAN does not handle character strings easily, the preprocessor has been written to make string handling as painless as possible. Character strings are set up as one character per array element. The defined symbol EOS marks the end of a string.

To make life easier, the preprocessor has been written in RATFOR. The original version was bootstrapped up through FORTRAN. Although the FORTRAN representation of the preprocessor exists, it is not the place for changes to the preprocessor. All changes should be made to the RATFOR source code and a new version should be processed through the old code. Having the source code in RATFOR simplifies code maintenance because all of the nice features of the language were used.

Extensive use is made of the RATFOR DEFINE statement in the preprocessor. The DEFINE statement allows users to set up their own definitions. As an example, the user may have "DEFINE (BIGA,32)" in the source code. The result of this is that, when the preprocessor sees BIGA in the user source code, it is immediately replaced with the number 32. In this manner, the code is much more machine independent and is easier to maintain. All user definitions are stored in a large definition table, TABLE, which stores both the user name and the defined value.

Structure of the RATFOR Preprocessor (Continued)

To simplify code maintenance, all COMMON blocks are located in the CMS file, RATCMN RATFOR. When a procedure needs the COMMON blocks, the command "INCLUDE RATCMN" is inserted in the source code. Modification to a COMMON block can now be done once in RATCMN instead of in each procedure using the COMMON block.

The last important structural feature to note is how the preprocessor handles input characters. On CMS, all FORTRAN statements must be in upper case. The RATFOR language allows upper and lower case source code. To convert the FORTRAN statements, the preprocessor maps all characters to upper case, excepting those in character constants. Character constants are left in their original case. The GETCH and INMAP functions handle character constants and mapping.

RATFOR Preprocessor Programs

ALLDIG - Identify whether string is numeric
BALPAR - Determine whether string has balanced parentheses
BRKNXT - Generate code for BREAK and NEXT statements
CONV - Convert string array from 1 character/entry to 4 characters/ent
CTOI - Return numeric representation of specified string
DEFTOK - Retrieve name part of DEFINE statement
DOCODE - Generate code for top of DO loop
DOSTAT - Generate terminating CONTINUE for DO statement
EATUP - Process remainder of input string. Handle continuations
ELSEIF - Generate code for ELSE segment of IF statement
EQUAL - Determine if two strings are equal
ERROR - Print error messages, terminate execution
EXIT - Return control to CMS editor
FORCOD - Generate code for top of FOR loop
FORS - Generate code for bottom of FOR loop
GETCH - Get next input character, map to proper case
GETDEF - Retrieve definition part of DEFINE statement
GETTOK - Process INCLUDE statement
GTOK - Strip comments and re-format input tokens
IFCODE - Generate labels for IF parsing
IFGO - Generate IF..NOT for IF statements
INDEX - Return location of character within a string
INITKW - Initialize the definition table, TABLE
INMAP - Map input characters to internal representation
INSTAL - Place user definitions in definition table, TABLE
INTCV - Convert integer to character string
LABELC - Check on label conflicts
LABGEN - Generate RATFOR labels for parsing loops
LENGTH - Return length of specified string
LEX - Lexically analyze tokens, returning lexical type
LOOKUP - Look up a name in the definition table and return the user definition
NGETCH - Control input from CMS file and input buffer
OTHERC - Handle non-RATFOR statements
OUTCH - Add character to output buffer
OUTCON - Add labeled CONTINUE to output buffer
OUTDON - Fill output buffer through column 80
OUTGO - Add "GOTO N" to output buffer
OUTMAP - Map characters to external representation
OUTNUM - Add label to output buffer
OUTSTR - Place string in output buffer. Convert Holleriths
OUTTAB - Fill columns 1-6 with blanks
PARSE - Controls which code to generate

RATFOR Preprocessor Programs
(Continued)

PBSTR - Add string to the input buffer using PUTBAK
PUTBAK - Add individual character to input buffer
PUTCH - Write characters to CMS output buffer
PUTLIN - Transfer string to output buffer
RELATE - Translate RATFOR relational operators to FORTRAN equivalent
REMARK - Write error message to CMS error file
REPCOD - Generate code for REPEAT statement
RETCOD - Generate code for RETURN statement
SCOPY - Copy one string to another
SYNERR - Print syntax error messages
SYSCAL - Execute CMS commands from executing program
TYPE - Determine whether character is numeric or alpha
UNSTAK - Control loop termination for all loops
UNTILS - Generate code for UNTIL statement
WHILEC - Generate code for top of WHILE loop
WHILES - Generate code for bottom of WHILE loop

Global Tables

```
EXTDIG(10) - External representation of digits 0-9
INTDIG(10) - Internal representation of digits 0-9

EXTLET(26) - External representation of lower case alphabet
INTLET(26) - Internal representation of lower case alphabet

EXTBIG(26) - External representation of upper case alphabet
INTBIG(26) - Internal representation of upper case alphabet

EXTCHR(36) - External representation of special characters
INTCHR(36) - Internal representation of special characters


SDO(3)     - String "DO", followed by EOS
VDO(2)     - Lexical representation of DO

SIF(3)     - String "IF", followed by EOS
VIF(2)     - Lexical representation of IF (LEXIF)

SELSE(5)   - String "ELSE", followed by EOS
VELSE(2)   - Lexical representation of ELSE (LEXELSE)

SWHILE(6)  - String "WHILE", followed by EOS
VWHILE(2)  - Lexical representation of WHILE (LEXWHILE)

SBREAK(6)  - String "BREAK", followed by EOS
VBREAK(2)  - Lexical representation of BREAK (LEXBREAK)

SNEXT(5)   - String "NEXT", followed by EOS
VNEXT(2)   - Lexical representation of NEXT (LEXNEXT)

SREPT(7)   - String "REPEAT", followed by EOS
VREPT(2)   - Lexical representation of REPEAT (LEXREPEAT)

SFOR(4)    - String "FOR", followed by EOS
VFOR(2)    - Lexical representation of FOR (LEXFOR)

SUNTIL(6)  - String "UNTIL", followed by EOS
VUNTIL(2)  - Lexical representation of UNTIL (LEXUNTIL)

SRET(7)    - String "RETURN", followed by EOS
VRET(2)    - Lexical representation of RETURN (LEXRETURN)
```

Global Tables (Cont.)

BUF(300) - Push back buffer - holds strings for later parsing

OUTBUF(81) - Holds output characters, printed when full

FCNAME(30) - Holds function name - will use with RETURN statement

FORSTK(200)- Holds increment clauses of FOR statement - to put at the bottom of the loop.

Global Pointers and Variables

BP - Pointer to the next location in the pushback buffer, OUTBUF.

FORDEP - Current depth (representing # unfinished loops) of the FORSTK table.

LEVEL - Unit number currently being read from. Will vary with INCLUDEs.

LINECT - Line count in the RATFOR input file - used for error message printing.

OUTP - Next location in the output buffer, OUTBUF.

Global Switches

XFER - When xfer = YES, GOTO generation is suppressed. This helps prevent the generation of two successive GOTOs, the second of which is never executed because it is unlabeled. Most of the major routines to parse the IF/ELSE and loop control statements set xfer back to NO.

Organization of Preprocessor Tables

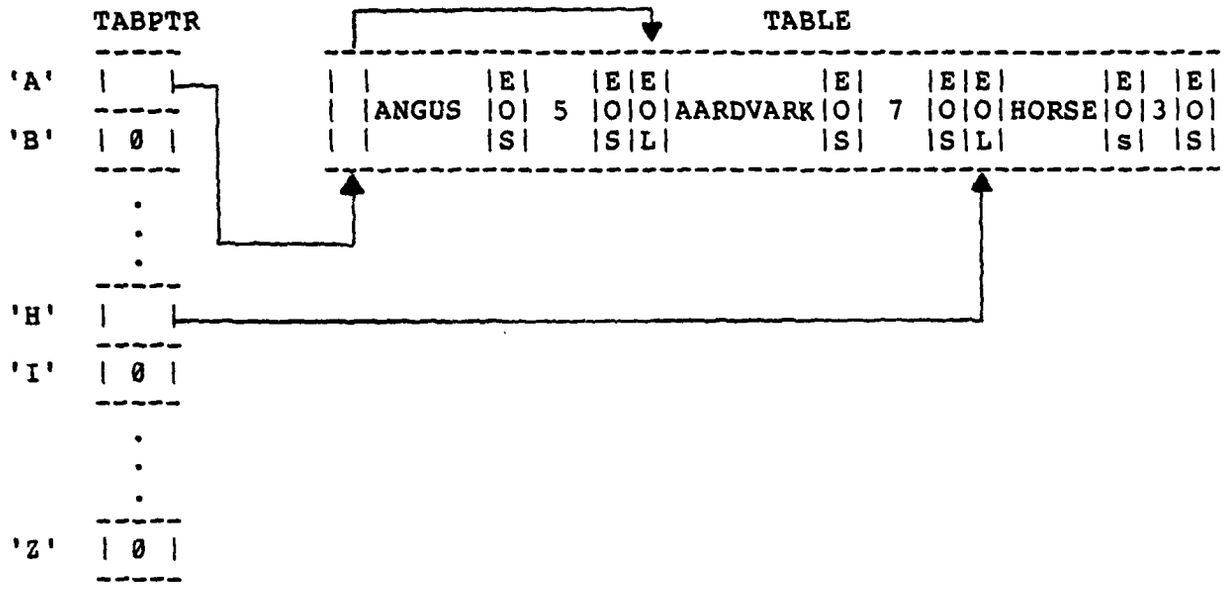
The only table of interest is the definition table, TABLE. This table is organized as a modified linked list. Because an ordinary linked list is too slow, there is a pointer table to speed up processing of user definitions. The pointer table, TABPTR, has one entry for each letter of the alphabet. The entry for a letter will point to a linked list of definitions starting with that letter. If no definitions have been processed for that letter, the TABPTR entry will be zero. AVAIL points to the next available location in TABLE.

The INSTAL routine places a name and user definition in TABLE while LOOKUP retrieves the user definition from the table.

Within each linked list, EOS marks the end of a definition and a name EOL signals the end of a list. If EOL is reached before the name is found (in LOOKUP), the name is not in the definition table.

Important Points

1. The pointer to the next entry in TABLE occurs at the start of the record. Thus, in the example below, AARDVARK is the last entry starting with 'A'.
2. The entries in TABLE are NOT in sorted order. They are placed in the definition table as they are encountered in the source code.
3. There are 41 entries in the TABPTR table because in EBCDIC, the character codes for the letters 'A' - 'Z' are interspersed with unprintable characters. The TABPTR entries for the unprintable characters are never used but cannot be omitted because of how the character is converted into an array subscript. The LOOKUP routine explains the conversion in greater detail.


TABPTR
TABLE
'A'
'B'
'H'
'I'
'Z'
0
0
0
ANGUS
5
AARDVARK
7
HORSE
3

RATFOR Input/Output Files

File Name	Unit No.	Access Mode	Accessed By	Description
XXXX FORTRAN	1	Output	PUTCH	FORTRAN output file
SCRATCH THREE	6	Output	REMARK	Error file
XXXX RATFOR	7	Input	GETCH	User RATFOR source

XXXX is the file name given by the user

Unit numbers 2-5 have been reserved in case later RATFOR modifications require more input or output files. All unit numbers generated by the GETTOK program to handle user INCLUDE statements range from 8 to 99.

RATFOR Error Messages
Explanations and Corrections

"01-Missing Left Paren."

Meaning:	A left parenthesis was missing starting an IF, WHILE, or UNTIL statement.
Correction:	Check to ensure that all parentheses are balanced
Produced By:	BALPAR routine

"02-Missing Parenthesis in Condition."

Meaning:	The parser encountered unbalanced parentheses in an IF, WHILE or UNTIL conditional statement.
Correction:	Check to ensure that all parentheses are balanced within the conditional.
Produced By:	BALPAR routine

"03-Illegal BREAK."

Meaning:	An attempt was made to either generate code for the BREAK statement outside of a loop or to transfer control outside of an illegal number of loops (N too high on "BREAK N").
Correction:	Make sure the BREAK statement occurs within a loop. Check the level of the BREAK with the level of looping.
Produced By:	BRKNXT routine

"04-Illegal NEXT."

Meaning:	An attempt was made to either generate code for the NEXT statement while not in a loop, or to transfer control through too many levels of looping (N too high on "NEXT N").
Correction:	Make sure the NEXT statement occurs within a loop. Check the level of the NEXT with the level of looping.
Produced By:	BRKNXT routine

RATFOR Error Messages
Explanations and Corrections

"05-Unexpected EOF."

Meaning:	The parser was expecting more of the input line when an end of file condition occurred.
Correction:	Check for unfinished continuation lines.
Produced By:	EATUP routine

"06-Unbalanced parentheses."

Meaning:	The parser encountered unbalanced parentheses in an IF, WHILE, or UNTIL statement.
Correction:	Check to ensure that all parentheses are balanced within the statement.
Produced By:	EATUP routine

"07-Missing left paren in FOR statement."

Meaning:	A left parenthesis was expected at the start of the FOR statement.
Correction:	Make sure all necessary parentheses are present.
Produced By:	FORCOD routine

"08-Unbalanced parentheses in FOR clause."

Meaning:	Unbalanced parentheses were encountered while parsing the FOR statement.
Correction:	Check to see that the FOR statement has a balanced number of parentheses.
Produced by:	FORCOD routine

"09-FOR clause too long."

Meaning:	The specified FOR clause was longer than the maximum length (currently 200 characters).
Correction:	Where possible, break the FOR clause into smaller pieces.
Produced by:	FORCOD routine

"10-Non-Alphanumeric name in DEFINE."

Meaning: The name specified in the DEFINE statement had a non-alphanumeric character in it.
Correction: Correct the DEFINE to be composed of numbers and letters only.
Produced By: GETDEF routine

"11-Definition too long."

Meaning: The specified definition was longer than the maximum definition size (currently 200 characters).
Correction: Choose a smaller definition.
Produced By: GETDEF routine

"12-Missing comma in DEFINE."

Meaning: The form of the DEFINE statement having parentheses also has a comma between the name and definition.
Correction: Add the necessary comma.
Produced By: GETDEF routine

"13-Missing right paren in DEFINE."

Meaning: A definition starting with a left parenthesis did not have a matching right parenthesis.
Correction: Add the necessary right parenthesis.
Produced By: GETDEF routine

"14-Unexpected token in DEFINE stmt."

Meaning: An unexpected token was found in the DEFINE statement.
Correction: Correct the DEFINE statement according to the rules specified in the language description.
Produced By: GETDEF routine

"15-INCLUDEs nested too deeply."

Meaning:	More than 92 user-nested INCLUDE files were opened concurrently.
Correction:	99 is the maximum number of concurrent FILEDEFs allowed by CMS for a FORTRAN file (user-defined files start with unit number 7). Check your program for recursion; since this is not checked for, an infinite INCLUDE loop may have occurred.
Produced By:	GETTOK routine

"16-Specified file does not exist."

Meaning:	The file specified in the INCLUDE statement does not exist.
Correction:	Check to make sure the file name, file type (Default RATFOR), and file mode (Default A1) were spelled correctly.
Produced By:	GETTOK routine

"17-Error in defining file."

Meaning:	The CMS FILEDEF statement did not execute correctly. The specified file was not included.
Correction:	Make sure your INCLUDE statement did not include any CMS unprintable characters within the file name.
Produced by:	GETTOK routine

"18-Token too long."

Meaning:	The input token was longer than the maximum token length (currently 200 characters).
Correction:	Shorten the token to something within the length boundary.
Produced By:	GTOK routine

"19-Missing Quote in string."

Meaning:	The string in error was missing its terminating quote mark.
Correction:	Add the required quote mark.
Produced By:	GTOK routine

"20-Too many definitions."

Meaning:	Too many DEFINE statements were encountered.
Correction:	Cut the number of DEFINEs down to the current maximum (6500 characters in all definitions).
Produced By:	INSTAL routine

"21-Warning: Possible label conflict."

Meaning:	A user-defined label may conflict with a RATFOR-generated label.
Correction:	Generally RATFOR-generated labels are of the form 23XXX. If a conflict occurs, choose another user label.
Produced By:	LABELC routine

"22-Illegal ELSE."

Meaning:	An ELSE was encountered that did not have a matching IF statement.
Correction:	Check to ensure that every ELSE has an associated IF statement.
Produced By:	PARSE routine

"23-Stack overflow in parser."

Meaning:	An attempt was made to add too many tokens to the parser stack.
Correction:	Send a copy of your RATFOR file to Dick Orgass - CMS userid ORGASS, along with an explanation of the problem encountered.
Produced By:	PARSE routine

"24-Illegal right brace."

Meaning:	A right brace was encountered that did not have a matching left brace.
Correction:	Ensure that all left braces have a matching number of right braces.
Produced By:	PARSE routine

"25-Unexpected EOF."

Meaning:	The parser was expecting more symbols (possibly loop terminators or brackets) when an end of file condition occurred.
Correction:	Make sure that all loops are in accordance with the language description and that every left bracket has a matching right bracket.
Produced By:	PARSE routine

"26-Too many characters pushed back."

Meaning:	An attempt was made to push more characters on the input buffer than was allowed.
Correction:	Send your RATFOR source to Dick Orgass - CMS userid ORGASS, along with an explanation of the problem encountered.
Produced By:	PBSTR routine

Explanation of DEFINES

```
ALPHA  - Internal code designating alphanumeric character type
AND    - Internal representation for '&'
ARB    - Length of REMARK message buffer
ATSIGN - Internal representation for '@'
BACKSLASH - Internal representation for '\'
BACKSPACE - Internal representation for the back space
BANG   - Internal representation for '!'
BAR    - Internal representation for '|'
BIGA-BIGZ - Upper case letters A-Z
BLANK  - Internal representation for a blank
BUFSIZ - Size of pushback input buffer used by PBSTR
CARET  - Internal representation for '^'
COLON  - Internal representation for ':'
COMMA  - Internal representation for ','
DEFTYPE - Internal code following "DEFINE" in definition table
DIGØ-DIG9 - Digits Ø-9
DIGIT  - Internal code for digits
DOLLAR - Internal representation for '$'
DQUOTE - Internal representation for '"'
EOF    - Internal code signaling End Of File
EOL    - Internal code signifying end of list in definition table
EOS    - Internal code signaling end of string in array
EQUALS - Internal representation for '='
ERROUT - Unit number to write error msgs to
GREATER - Internal representation for '>'
LBRACE - Internal representation for '{'
LBRACK - Internal representation for '['
LESS   - Internal representation for '<'
LETA-LETZ - Lower case letters A-Z
LEXBREAK - Internal lexical code for BREAK keyword
LEXDIGITS - Internal code for lexical type digits
LEXDO  - Internal lexical code for DO keyword
LEXELSE - Internal lexical code for ELSE keyword
LEXFOR - Internal lexical code for FOR keyword
LEXIF  - Internal lexical code for IF keyword
LEXNEXT - Internal lexical code for NEXT keyword
LEXOTHER - Internal lexical code for non-RATFOR token
LEXREPEAT - Internal lexical code for REPEAT keyword
LEXUNTIL - Internal lexical code for UNTIL keyword
LEXWHILE - Internal lexical code for WHILE keyword
LPAREN - Internal representation for '('
MAXCARD - Maximum # characters per input record
MAXCHARS - Maximum # characters for OUTNUM
MAXDEF - Maximum # characters in definition
MAXFOR - Maximum stack space for FOR increment clauses
```

Explanation of DEFINES
(Continued)

```
MAXLINE - Maximum # characters per output line
MAXSTACK - Maximum stack depth for parser
MAXTBL - Maximum # characters in all definitions
MAXTOK - Maximum # characters in a token
MINUS  - Internal representation for '-'
NCHARS - Number of special characters
NEWLINE - Internal code terminating input line
NO     - Negative answer for functions
NOT    - Internal representation for '~'
OR     - Internal representation for OR bar, same as BAR
PERCENT - Internal representation for '%'
PERIOD - Internal representation for '.'
PLUS   - Internal representation for'+'
QMARK  - Internal representation for ?
RBRACE - Internal representation for '}'
RBRACK - Internal representation for ']'
RPAREN - Internal representation for ')'
SEMICOL - Internal representation for ';'
SHARP  - Internal representation for '#'
SLASH  - Internal representation for '/'
SQUOTE - Internal representation for '
STAR   - Internal representation for '*'
STDOUT - Unit number of output FORTRAN file
TAB    - Internal representation for tab character
TILDE  - Internal representation for '~', same as NOT
UNDERLINE - Internal representation for '_'
YES    - Positive answer for functions
CHARACTER - Used to distinguish character variables from
            integers - by appearance only
```

"Expanded Program Descriptions"

Program: *** ALLDIG ***

Type: Integer Function

Parameters: STR - String array of 100 characters maximum

Calls: TYPE - Determine string type (alpha, numeric)

Called from: BRKNXT, LEX

Description: ALLDIG returns "YES" if the specified string is numeric and "NO" otherwise. The code EOS denotes the end of the string.

Error Messages: None

Program: *** BALPAR ***

Type: Subroutine

Parameters: None

Calls: GETTOK - Read next token
OUTSTR - Write token to output file
PBSTR - Place string back in input buffer for later parsing.
SYNERR - Write error messages

Called from: IFGO

Description: BALPAR determines whether the input string has balanced parentheses. Basically, 1 is added to NLPAR for every left parenthesis, while 1 is subtracted for right parentheses. If NLPAR is not zero at the termination of this routine, an error message is printed. The search for parentheses ends when a newline, a left bracket, a semicolon, a right bracket, or an end of file is encountered. OUTSTR handles the tokens within parentheses.

Error Messages: "01-Missing Left Paren."
"02-Missing Parenthesis in Condition."

Program: *** BRKNXT ***

Type: Subroutine

Parameters:
- LABVAL - Array stack keeping track of the labels for loops
- LEXTYP - Array stack giving the loop sequence being parsed
- SP - Stack pointer for LEXTYP and LABVAL arrays
- TOKEN - Token being parsed

Calls:
- ALLDIG - Determine whether numbered BREAK or NEXT
- CTOI - Convert level of BREAK or NEXT from character to numeric
- GETTOK - Get next token (nbr in BREAK or NEXT)
- OUTGO - Generate GOTO for BREAK or NEXT
- PBSTR - Put token back if not number
- SYNERR - Flag syntax errors

Called from: PARSE

Description: This routine generates code for the "BREAK N" and "NEXT N" statements. N may be left off, explaining the business with ALLDIG and PBSTR - if N isn't there, we have another token not dealing with breaks (put it back). For either statement, we have to determine what sequence we are breaking, whether a WHILE, DO, FOR, or a REPEAT sequence. For a NEXT, a GOTO is generated to the specified iteration of the loop. For a BREAK, we go to the statement after the loop. To determine where we are and where to jump to, the LEXTYP and LABVAL arrays are used; these arrays are simulating stacks with SP being the stack pointer.

Error Messages:
- "Ø3-Illegal BREAK."
- "Ø4-Illegal NEXT."

Program: *** CONV ***

Type: Subroutine

Parameters:
EN - Will point to last character converted
OUTSTR - Outgoing string array, 4 characters per array element
STR - Incoming string array, 1 character per array element

Calls: Nothing

Called from: ERROR, SYNERR, GETTOK

Limitations: The number 1077952576 represents 4 blanks. This may not be true in your machine. Also, 64 represents an EBCDIC blank. 256 is the amount the string must be multiplied by to shift the whole string 1 character to the left.

Description: This routine converts a string array stored as 1 character per array location to a left-justified 4 character per location representation. EN points to the last character converted. This routine is needed because CMS requires commands to be in the 4 character "packed" form.

Error Messages: None

Program: *** CTOI ***

Type: Integer Function

Parameters:
I - Starting subscript to IN array
IN - String array to convert to integer

Calls: INDEX - Return a pointer within the digit string to the specified digit.

Called from: BRKNXT

Description: Returns the numeric representation of the specified string. The routine is basically used for user-labeled statements because it skips over tab characters and blanks. The input string ends when the EOS marker is detected.

Error Messages: None

Program: *** DEFTOK ***

Type: Integer Function

Parameters:
- FD - File that token was read from (varies with INCLUDES)
- TOKEN - Stores token to be processed, 200 characters maximum
- TOKSIZ - Size of token to process

Calls:
- GETDEF - Get definition from DEFINE statement
- GTOK - Get new token
- INSTAL - Put definition and name in definition table
- LOOKUP - Look up the token in the definition table to determine if token = "DEFINE"
- PBSTR - Puts DEFN back into the input buffer for later processing

Called from: GETTOK

Description: This routine processes the RATFOR "DEFINE" statement. If the token is "DEFINE", the associated name and definition is read from the input buffer and installed in the big definition table, TABLE. If the token wasn't "DEFINE", the routine places the token back into the input buffer. The routine exits with TOKEN pointing to a new non-DEFINE token to process.

Error Messages: None

Program: *** DOCODE ***

Type: Subroutine

Parameters: LAB - Last label generated

Calls:
EATUP - Add remainder of DO statement to output buffer
LABGEN - Generate label for do statement.
OUTDON - Write output buffer to CMS disk file
OUTNUM - Add label to output buffer
OUTSTR - Output the string "DO"
OUTTAB - Output blanks in first 6 columns.

Called from: PARSE

Description: Generate the code for the "DO" statement. This involves generating a label and inserting it as in a normal FORTRAN "DO" statement. DOSTAT generates the "N CONTINUE" to terminate the DO loop. A label is generated for the statement after the loop - to branch to should a BREAK or NEXT occur within the loop.

Error Messages: None

Program: *** DOSTAT ***

Type: Subroutine

Parameters: LAB - Last label generated

Calls: OUTCON - Write label and CONTINUE to output buffer

Called from: UNSTAK

Description: Generate labeled CONTINUE to terminate DO loop.

Error Messages: None

Program: *** EATUP ***

Type: Subroutine

Parameters: None

Calls: GETTOK - Get next token to identify and parse
OUTSTR - Write token to output buffer
PBSTR - Put back string for later parsing
SYNERR - Announce syntax errors to user

Called from: DOCODE, FORCOD, OTHERC

Description: Process remainder of the statement. Check for balanced parentheses in the rest of the statement.

Error Messages: "Ø5-Unexpected EOF."
"Ø6-Unbalanced Parentheses."

Program: *** ELSEIF ***

Type: Subroutine

Parameters: LAB - FORTRAN label for ELSE portion of an IF statement

Calls: OUTCON - Generate a CONTINUE with a label (LAB) for the ELSE segment
OUTGO - Generates a GOTO to the next statement (over the ELSE segment)

Called from: PARSE

Description: Processes part of an IF/ELSE statement. The IF code has already been generated by IFCODE. This routine generates the GOTO over the ELSE segment, then generates a labeled CONTINUE for the ELSE segment.

Error Messages: None

Program: *** EQUAL ***

Type: Integer Function

Parameters: STR1 and STR2 - Strings to be compared

Calls: Nothing

Called from: GETTOK, LEX

Description: EQUAL returns "YES" if the specified strings are equal, "NO" otherwise. End of string (EOS) is checked in both strings.

Error Messages: None

Program: *** ERROR ***

Type: Subroutine

Parameters: MSG - Message specifying the error detected.

Calls:
- CONV - Convert the 1 character/array element to a 4 character per element representation.
- EXIT - FORTRAN routine to terminate execution
- INTCV - Convert the line number from an integer to its string representation
- REMARK - Write the specified message to an error file

Called from: GETDEF, FORCOD, PARSE, PUTBAK

Description: Announces to the user that a fatal error has occurred. An error message is printed along with the line number (in the generated code) of the error. The FORTRAN EXIT subroutine terminates execution of the RATFOR preprocessor.

Error Messages: None

Program: *** EXIT ***

Type: Subroutine

Parameters: None

Calls: Who knows - CMS routine

Called from: ERROR

Description: EXIT transfers control back to the CMS editor. It is used when a fatal error has been detected while parsing the RATFOR source.

Error Messages: None

Program: *** FORCOD ***

Type: Subroutine

Parameters: LAB - Latest label generated

Calls:
EATUP - Finish parsing rest of line
ERROR - Inform user that FOR clause is too long
GETTOK - Get next token in input
LABGEN - Generate labels for FOR statement
LENGTH - Determine length of FOR arguments
OUTCH - Output parentheses
OUTCON - Generate CONTINUE for top of loop
OUTDON - Generate blanks for columns 72-80
OUTGO - Output GOTO statement
OUTSTR - Output IF-NOT string
OUTTAB - Generate blanks for columns 1-6
PBSTR - Put token back in input buffer
SCOPY - Place token on FOR stack
SYNERR - Flag syntax errors

Called From: PARSE

Description: This routine is a complicated procedure to parse the start of the FOR statement. It must stack the FOR increment condition for insertion at the end of the loop. The initial and termination conditions are parsed and written to the output file. Labels are generated should a BREAK or NEXT statement occur in the input. The parentheses checking function in FORCOD is repetitious with BALPAR.

Error Messages: "07-Missing Left Paren in FOR statement."
"08-Invalid FOR Clause."
"09-FOR clause too long." *
* Fatal error, terminates execution.

Program: *** FORS ***

Type: Subroutine

Parameters: LAB - Latest label generated

Calls:
LENGTH - Determine length of FOR condition
OUTCON - Output CONTINUE for next statement
OUTDON - Place blanks in the rest of the line
OUTGO - Output GOTO for top of FOR loop
OUTNUM - Transfer label to output buffer
OUTSTR - Output the specified string
OUTTAB - Place blanks in columns. 1-6

Called from: PARSE

Description: This routine finishes up the FOR statement which FORCOD started. This involves writing out the increment condition which is obtained from the FORSTK array. The GOTO is written to the top of the FOR loop and a CONTINUE is written for the statement after the loop (in case of BREAKS and NEXTS within the loop). This routine keeps track of our FOR depth level.

Error Messages: None

Program: *** GETCH ***

Type: Integer Function

Parameters: C - Character to be returned
FILE - Input file being read from

Calls: INMAP - Translate the input to internal form

Called from: NGETCH

Description: GETCH reads an input line from the CMS input file (unit number varies depending on which INCLUDE is being processing). All of the input line is mapped to an internal representation. This mapping converts all RATFOR and FORTRAN statements into upper case. All character constants remain in their original case. Character constants are identified by their being either within quotation marks or following the Hollerith "H". Since this is a time consuming process (checking every character to see whether it is a character constant) the checking is only done for three cases. Character constants can only occur 1. within a data statement, 2. within a FORMAT statement, and 3. within a subroutine call. These three cases are recognized by the presence of a comma, a slash, or a left parenthesis. This is rather crude, but, since the user wanted a lower case constant, we must leave it that way. We have no choice regarding RATFOR and FORTRAN statements - CMS FORTRAN will only handle upper case programs.

Error Messages: None

Program: *** GETDEF ***

Type: Subroutine

Parameters:
- DEFN - Definition found by GETDEF
- DEFSIZ - Size of DEFN
- FD - Input file to read from (unit number varies with INCLUDES)
- TOKEN - Token read in by this routine
- TOKSIZ - Length of the token

Calls:
- ERROR - Signal fatal error
- GTOK - Read new token from input file
- NGETCH - Get next character
- PBSTR - Put token back in input buffer
- PUTBAK - Put single character back in input buffer

Called from: DEFTOK

Description: GETDEF processes the RATFOR DEFINE statement. DEFTOK has retrieved the name - it is up to us to get the definition. Recall that there are two forms of DEFINE - with and without parentheses

Error Messages:
"10-Non-Alphanumeric Name in DEFINE." *
"11-Definition too long." *
"12-Missing comma in DEFINE." *
"13-Missing right parenthesis in DEFINE." *
"14-Unexpected token in DEFINE statement." *

* Fatal error, terminates execution

Program: *** GETTOK ***

Type: Integer Function

Parameters:
TOKEN - Token being parsed
TOKSIZ - Size of token being parsed

Calls:
CONV - Convert token to "packed"
DEFTOK - Get new token, possibly DEFINE
EQUAL - Check if token is FUNCTION or INCLUDE
ERROR - Flag syntax errors, terminates execution
INTCV - Convert file number to numeric representation
PBSTR - Put token back in input buffer
SYSCAL - VPI routine to execute CMS commands from executing programs

Called from: BALPAR, BRKXT, EATUP, FORCOD, LEX

Description: This routine processes the RATFOR INCLUDE statement. The INCLUDE allows the inclusion of any source file during execution. The routine also saves the names of functions for use later with the RETURN statement. Up to 92 levels of INCLUDES are allowed. The file to be included, a CMS file, can be identified with or without a file type (default is RATFOR) and with or without a file mode (default is A1). The GETTOK routine first checks whether the file exists. If it does, a unit number is assigned to that file. The CMS FILEDEF statement is executed informing CMS of our new file number. Since INCLUDES can be nested, the input routine (GETCH) reads from the last unit number defined. When a file is finished (EOF), the file is closed and the level (representing the unit #) is decremented. Thus, unit numbers are re-used.

Error Messages: "15-INCLUDEs nested too deeply." *
"16-Specified file does not exist." *
"17-Error in defining file." *

* Fatal error, terminates execution

Program: *** GTOK ***

Type: Integer Function

Parameters:
- FD - Input file to read from (unit # varies with INCLUDEs)
- LEXSTR - Returns string to parse
- TOKSIZ - Size of token being returned

Calls:
- NGETCH - Get single character to form token
- PUTBAK - Put character back in input buffer
- RELATE - Replace RATFOR relational operators with their FORTRAN equivalents
- SYNERR - Flag syntax errors
- TYPE - Identify whether alpha or numeric

Called from: DEFTOK, GETDEF

Description: GTOK returns a token to the calling routine. It strips the input of comments and replaces RATFOR relational operators with their FORTRAN equivalents. If this is a new line, the line count is incremented. Braces are substituted for left and right brackets.

Error Messages: "18-Token too long."
"19-Missing Quote in string."

Program: *** IFCODE ***

Type: Subroutine

Parameters: LAB - Last label generated. Will change on exit from this routine

Calls: IFGO - Parse IF statement
LABGEN - Generate 2 labels (for IF and ELSE)

Called from: PARSE

Description: Generates labels for IF and ELSE (regardless of whether it needs the ELSE). Calls IFGO to parse the IF statement.

Error Messages: None

Program: *** IFGO ***

Type: Subroutine

Parameters: LAB - FORTRAN label to jump to if the IF condition is not true.

Calls: BALPAR - Checks on balanced parentheses. Writes text between parentheses.
OUTCH - Write last ")"
OUTGO - Write "GOTO" with a label
OUTSTR - Writes the string "IF(.NOT."
OUTTAB - write 6 blanks to output buffer

Called from: IFCODE, UNTILS, WHILEC

Description: This routine processes the IF statement. For all IF statements, the parser generates "IF(.NOT.(original condition)) GOTO XXX.

Error Messages: None

Program: *** INDEX ***

Type: Integer Function

Parameters: C - Character to locate in string
STR - String array to perform locate operation upon.

Calls: Nothing

Called from: CTOI

Description: INDEX returns the subscript to the string array where it found the first occurrence of C. If C was not found, 0 is returned. EOS marks the end of the string.

Error Messages: None

Program: *** INITKW ***

Type: Subroutine

Parameters: None

Calls: INSTAL - Put "DEFINE" in definition table

Called from: PARSE

Description: This routine sets the available space pointer to the definition table to 1 and initializes the pointer table, TABPTR, (for definitions) to 0 representing an empty table. The word "DEFINE" is put in the definition table along with a flag, DEFTYPE.

Error Messages: None

Program: *** INMAP ***

Type: Integer Function

Parameters: CASE - Case to put character in
Ø - lower case
1 - upper case
INCHAR - Character to convert

Calls: Nothing

Called from: GETCH

Limitations: Although this routine does the conversions, it is still fairly machined independent. Any changes to the internal/external representations must be done to the COMMON block items.

Description: This routine converts INCHAP from its external representation to an internal representation. CASE specifies whether the character is to be left in lower case or converted to upper case. CASE was necessary because all FORTRAN and RATFOR statements are converted to upper case before parsing; all character constants must stay in their original case. If a conversion cannot be made, the character is left as is.

Error Messages: None

Program: *** INSTAL ***

Type: Subroutine

Parameters: DEFN - Name the preprocessor will replace NAME with - also in the DEFINE statement
NAME - Name the RATFOR programmer will use - specified in a DEFINE statement

Calls: LENGTH - Determine length of NAME string
PUTLIN - Write NAME to error file
ERROR - Write message to error file
SCOPY - Copy NAME and DEFN to definition table

Called from: DEFTOK, INITKW

Limitations: The TABPTR table is 41 long because, although there are only 26 letters, EBCDIC has some non-alpha codes between A and Z. This table size may vary with your machine.

Description: Insert NAME and DEFN into the definition table, TABLE. The format of this table is described later in this documentation. Basically, a linked list is created for each letter in the alphabet pointing to the first definition starting with that letter. AVAIL always points to the next available location in the definition table. EOL signifies the end of a list.

Error Messages: "20-Too many definitions." *

* Fatal error, terminates execution

Program: *** INTCV ***

Type: Subroutine

Parameters: INT - Integer to convert to string representation
STR - 4 character array to hold string repr.

Calls: MOD function

Called from: ERROR, GETTOK, SYNERR

Description: Converts the specified integer to a 4 character representation. The representation is stored 1 character per array element. Leading zeroes are converted to blanks.

Error Messages: None

Program: *** LABELC ***

Type: Subroutine

Parameters: LEXSTR - Lexical string being analyzed (Hopefully FORTRAN label)

Calls: LENGTH - Determine string length
OUTSTR - Output the string
OUTTAB - Output blanks in til column 7
SYNERR - Warn user that his label may conflict with a RATFOR generated label

Called from: PARSE

Description: This routine writes out the RATFOR user's label. The routine checks (partially) for a label conflict by determining if the label, of length 5, starts with 23XXX. If so, we might have a conflict and the user is warned. Regardless, we write out his label and let FORTRAN catch the conflict.

Error Messages: "21-Warning: Possible label conflict."

Program: *** LABGEN ***

Type: Integer Function

Parameters: N - Integer specifying the number of labels to generate.

Calls: Nothing

Called from: DOCODE, FORCOD, IFCODE, REPCOD, WHILEC

Description: This program generates a sequence of N labels, the first of which is returned. A DATA statement provides that label generation starts with label 23000. The generated labels are used parsing FOR, REPEAT, UNTIL, DO and IF statements.

Error Messages: None

Program: *** LENGTH ***

Type: Integer Function

Parameters: STR - String array of 100 characters maximum

Calls: Nothing

Called from: FORCOD, FORS, INSTAL, LABELC, PBSTR, RELATE

Description: This routine determines the length of a string stored as an array. The program assumes one character per array entry with the EOS code denoting the end of the string.

Error Messages: None

Program: *** LEX ***

Type: Integer Function

Parameters: LEXSTR - String to identify the lexical type of

Calls: ALLDIG - Determine whether token is numeric
EQUAL - Identify which token this is
GETTOK - Get new token to check type of

Called from: PARSE, UNTILS

Description: LEX returns the lexical type of the token being parsed. Each RATFOR token (IF, ELSE, WHILE, DO, BREAK, NEXT, FOR, UNTIL, REPEAT, RETURN) has its own lexical type. Digits have their type. Unidentifiable types are classed in the general category type of LEXOTHER.

Error Messages: None

Program: *** LOOKUP ***

Type: Integer Function

Parameters: DEFN - Holds definition found in table
NAME - Name to locate in definition table

Calls: SCOPY - Copy the retrieved definition from the definition table to DEFN parameter.

Called from: DEFTOK

Limitations: The table lookup routine appears to be machine dependent (EBCDIC), but instead only relies on the condition that each letter has its own character code.

Description: LOOKUP searches the definition table, TABLE, for the specified name. If present, the definition is transferred to DEFN and a "YES" is returned. If the definition is absent, "NO" is returned from the function. Refer to the section on table descriptions for an explanation of how the definition table is organized.

Error Messages: None

Program: *** NGETCH ***

Type: Integer Function

Parameters: C - Character to return to calling routine
FD - File to read next character from

Calls: GETCH - Get next character

Called from: GTOK, RELATE

Description: Return a new character to the calling routine. First, check the input buffer where PBSTR has been storing things pushed back. If empty, call GETCH to read from the input file.

Error Messages: None

Program: *** OTHERC ***

Type: Subroutine

Parameters: LEXSTR - String array to put in output buffer

Calls: EATUP - Transfer rest of this line
OUTDON - Place blanks in remaining columns
OUTSTR - Transfer string to buffer
OUTTAB - Write blanks through column 6

Called from: PARSE

Description: OTHERC processes FORTRAN statements. Because RATFOR has free-format input, this routine ensures the proper FORTRAN spacing.

Error Messages: None

Program: *** OUTCH ***

Type: Subroutine

Parameters: C - Character to insert in output buffer

Calls: OUTDON - Fill columns 72-80 with blanks

Called from: FORCOD, IFGO, OUTNUM, OUTSTR, OUTTAB

Description: Add the character, C, to the output buffer. Place blanks in columns 72-80. If we are processing a continuation line, place a star in the next line's column 6.

Error Messages: None

Program: *** OUTCON ***

Type: Subroutine

Parameters: N - Label for "N CONTINUE"

Calls: OUTDON - Write remaining columns of line
OUTNUM - Write N, the label
OUTSTR - Write "CONTINUE"
OUTTAB - Write blanks til column 7

Called from: DOSTAT, ELSEIF, FORCOD, FORS, REPCOD, UNSTAK, UNTILS, WHILEC, WHILES

Description: OUTCON transfers the sequence "N CONTINUE" to the output buffer where N is a label. No sequence is generated for unlabeled continues.

Error Messages: None

Program: *** OUTDON ***

Type: Subroutine

Parameters: None

Calls: PUTLIN - Write entire output line

Called from: DOCODE, FORCOD, FORS, OTHERC, OUTCH, OUTCON, OUTGO

Description: Add end of line (NEWLINE) and end of string (EOS) to output buffer. Write the entire output line to the CMS disk file (via PUTCH).

Error Messages: None

Program: *** OUTGO ***

Type: Subroutine

Parameters: N - Label for "GOTO N"

Calls: OUTDON - Output blanks til end of line
OUTNUM - Outputs label for GOTO
OUTSTR - Output string "GOTO"
OUTTAB - Output blanks in columns 1-6

Called from: BRKNXT, ELSEIF, FORCOD, FORS, IFGO, UNTILS, WHILES

Description: OUTGO transfers the string "GOTO N" to the output buffer. This sequence is used for statements like FOR, REPEAT, etc.

Error Messages: None

Program: *** OUTMAP ***

Type: Integer Function

Parameters: INCHAR - Character to convert

Calls: Nothing

Called from: PUTCH

Limitations: Although this routine does the conversion to a printable representation, it is fairly machine independent. Any changes to the representations must be done to the COMMON block items.

Description: This routine converts the INCHAR from its internal representation (mixed ASCII and EBCDIC) to a representation printable on your machine. If no conversion can be made, the character is left as is.

Error Messages: None

Program: *** OUTNUM ***

Type: Subroutine

Parameters: N - Number to convert to character representation

Calls: MOD - Used in conversion process
OUTCH - Transfer converted number to output buffer

Called from: DOCODE, FORS, OUTCON, OUTGO, OUTSTR, UNTILS, WHILEC

Description: OUTNUM converts a number (generally labels), N, into its character representation. The label is then sent on to the output buffer via the OUTCH subroutine. This routine is the reverse of CTOI, and extremely similar to INTCV (INTCV goes to 4 characters per array element).

Error Messages: None

Program: *** OUTSTR ***

Type: Subroutine

Parameters: STR - String to place in output buffer

Calls: OUTCH - Place single character in output buffer
OUTNUM - Place number in output buffer converting from character to numeric

Called from: BALPAR, DOCODE, EATUP, FORCOD, FORS, IFGO, LABELC, OTHERC, OUTCON, OUTGO

Description: This routine transfers a string to the output buffer. If the string is in single or double quotes, the string is converted to the Hollerith format (eg. 'a' = 1Ha). The transfer to the output buffer continues until the EOS marker is encountered.

Error Messages: None

Program: *** OUTTAB ***

Type: Subroutine

Calls: OUTCH - Write out a blank

Called from: DOCODE, FORCOD, FORS, IFGO, LABELC, OTHERC, OUTcON, OUTGO

Description: OUTTAB writes blanks through column 6 (FORTRAN statements start in column 7).

Error Messages: None

Program: *** PARSE ***

Type: Subroutine

Parameters: None

Calls:
- BRKNXT - Parse BREAK and NEXT statements
- DOCODE - Parse DO statement
- ELSEIF - If present, parse ELSE statement
- ERROR - Flag fatal errors, terminate execution
- FORCOD - Parse FOR statement
- IFCODE - Parse IF statement
- INITKW - Initialize the definition table.
- LABELC - Output user-defined label
- LEX - Lexically analyze input and return parseable token
- OTHERC - Process ordinary FORTRAN statements
- PBSTR - Put string back in input save buffer
- REPCOD - Parse REPEAT statement
- RETCOD - Parse RETURN statement
- SYNERR - Flag syntax errors
- UNSTAK - Finish processing all loops
- WHILEC - Parse WHILE statement

Called from: RATFOR main procedure

Description: PARSE is the RATFOR parser delegating parsing responsibilities to the above-specified subroutines. Most error messages are printed by the subroutines. Bracketed statements are handled here.

Error Messages:
"22-Illegal ELSE."
"23-Stack overflow in parser."
"24-Illegal right brace."
"25-Unexpected EOF." *

* Fatal error, terminates execution

Program: *** PBSTR ***

Type: Subroutine

Parameters: IN - String array to push back in the input buffer 100 characters maximum.

Calls: LENGTH - Determine actual length of IN. EOS marks the end of the string.
PUTBAK - Puts single character back in the input buffer

Called from: BALPAR, BRKNXT, DEFTOK, EATUP, FORCOD, GETDEF, GETTOK
PARSE

Description: Adds the string IN to the input buffer.

Error Messages: "26-Too many characters pushed back."

Program: *** PUTBAK ***

Type: Subroutine

Parameters: C - Character to put back in input buffer

Calls: ERROR - Writes message to error file when trying to put back too many characters

Called from: GETTOK, GTOK, PBSTR, RELATE

Description: Puts character, C, into an input buffer for later parsing. An error message is printed if too many characters are pushed back.

Error Messages: None

Program: *** PUTCH ***

Type: Subroutine

Parameters: C - Character to write to output file
F - File to write to

Calls: OUTMAP - Map the character to printable external representation

Called from: PUTLIN

Description: Place character, C, in the output buffer. When the maximum number per output line has been reached, or at the end of a RATFOR line (C=NEWLINE), write the entire output buffer. Every character is mapped to a printable representation by OUTMAP.

Error Messages: None

Program: *** PUTLIN ***

Type: Subroutine

Parameters: B - String array to write
FILE - Output file to write to

Calls: PUTCH - Write individual string characters

Called from: INSTAL, OUTDON

Description: Write the string array to the output buffer (PUTCH sends it on to the output file). EOS denotes the end of the string array.

Error Messages: None

Program: *** RELATE ***

Type: Subroutine

Parameters: FD - Input file to read from
LAST - Length of new token
TOKEN - Token to parse

Calls: LENGTH - Determine length of new token
NGETCH - Read new token from input file FD
PUTBAK - Place token item back into input buffer
SCOPY - Copy new FORTRAN version of relation operator into TOKEN array

Called from: GTOK

Description: This routine replaces the RATFOR relational operators (see language description) with their FORTRAN equivalent. For example, == is replaced by .EQ. The EOS marker is placed at the end of the new relational operator and the new length is returned. Recall that some tokens require two symbols (">=") while others require just one ("<").

Error Messages: None

Program: *** REMARK ***

Type: Subroutine

Parameters: BUF - Buffer holding message to write to user error file

Calls: Nothing

Called from: ERROR, INSTAL

Limitations: This routine is machine dependent, writing to unit 6 as an error file.

Description: REMARK writes the specified message to a CMS error file.

Error Messages: None

Program: *** REPCOD ***

Type: Subroutine

Parameters: LAB - Last label generated

Calls: LABGEN - Generate labels for REPEAT code
OUTCON - Write labeled "CONTINUE"

Called from: PARSE

Description: This routine begins code generation for the RATFOR REPEAT statement. Recall that most of the work is done at the UNTIL end of the parsing. Labels are generated for later use.

Error Messages: None

Program: *** RETCOD ***

Type: Subroutine

Parameters: None

Calls: EATUP - Finish this line
GETTOK - Get next token (after RETURN)
OUTCH - Output an equal sign
OUTDON - Place blanks in columns 72-80
OUTSTR - Output the function name
OUTTAB - Place blanks in columns 1-6
PBSTR - Place string back in input buffer

Called from: PARSE, SYNERR

Description: Recall that the RETURN statement may be of the form RETURN (YES). This routine retrieves the function name (from FCNAMF) and writes a statement setting it equal to the return value (YES in the example above). The normal RETURN statement (without a value) is also parsed. The routine worrys about brackets and if it finds a right bracket in the input, it keeps it there for later parsing.

Error Messages: None

Program: *** SCOPY ***

Type: Subroutine

Parameters: FROM - Array to copy from
I - Start subscript of FROM
J - Starting subscript of TO
TO - Array to copy to

Calls: Nothing

Called from: FORCOD, INSTAL, LOOKUP, RELATE

Description: Copies the array FROM, starting at I, to the array TO, starting at J. The copy operation continues until an End of String (EOS) is encountered. An EOS marker is then put at the end of TO.

Error Messages: None

Program: *** SYNERR ***

Type: Subroutine

Parameters: MSG - Message specifying the error detected

Calls: CONV - Convert the 1 character per array element string to a 4 character per element representation.
INTCV - Convert the line number from an integer to its string representation.
REMARK - Write the specified message to an error file

Called from: BALPAR, BRKNXT, EATUP, FORCOD, GETTOK, GTOK, LABELC, PARSE

Description: Announces to the user that an error has occurred. An error message is printed along with the line number (in the RATFOR code) of the error.

Error Messages: None

Program: *** SYSCAL ***

Type: Subroutine

Parameters: Field 1 - Array holding CMS command to be executed.
Must have 4 characters / array element
Field 2 - Length of Field 1 parameter
Field 3 - Location to store CMS return code

Calls: Who knows - CMS routine

Called from: GETTOK

Description: SYSCAL is a VPI subroutine that executes CMS commands from within a running program. In this application, it is used to inform CMS of our INCLUDE file. The routine returns an error code reflecting whether the command executed successfully. Please see the CMS User's Guide for a description of the error codes.

Error Messages: None

Program: *** TYPE ***

Type: Integer Function

Parameters: C - Character to determine type of

Calls: Nothing

Called from: ALLDIG, GTOK

Description: Signals if the specified character is numeric or alphabetic. If neither case is true, the character is returned.

Error Messages: None

Program: *** UNSTAK ***

Type : Subroutine

Parameters:
- LABVAL - Array stack keeping track of the labels for our loops
- LEXTYP - Array stack giving the loop sequence we are now processing.
- SP - Stack pointer for LABVAL and LEXTYP arrays
- TOKEN - Token being parsed

Calls:
- DOSTAT - Generate labeled "CONTINUE" for DO statement
- FORS - Process bottom of FOR loop
- OUTCON - Generate labeled "CONTINUE" statement for IF and ELSE statements
- UNTILS - Process bottom of REPEAT/UNTIL loop
- WHILES - Process bottom of WHILE loop

Called from: PARSE

Description: This routine finishes processing for all loops and bracketed statements (IF, ELSE, DO, WHILE, REPEAT/UNTIL, FOR). To do the processing, it calls the appropriate subroutine for the loop we are in. The LABVAL and LEXTYP arrays identify which loop sequence we are in and which labels needs to be generated.

Error Messages: None

Program: *** UNTILS ***

Type: Subroutine

Parameters: LAB - Last label generated
TOKEN - Token just encountered

Calls: IFGO - Generate code for UNTIL condition
LEX - Parse UNTIL condition
OUTCON - Output "CONTINUE" for after REPEAT
OUTGO - Output "GOTO" for REPEAT without UNTIL statement
OUTNUM - Write label for bottom of REPEAT block

Called from: PARSE, UNSTAK

Description: This routine parses the two kinds of REPEAT statements in RATFOR. The first has the keyword UNTIL at the end of the loop. For this case, code is generated for the UNTIL condition, in the form of an IF-NOT jump statement. The other kind of REPEAT has no UNTIL. Theoretically this is an endless loop so only a GOTO statement is generated to the top of the loop. Both cases described above have a post-loop CONTINUE generated.

Error Messages: None

Program: *** WHILEC ***

Type: Subroutine

Parameters: LAB - Latest label generated

Calls: IFGO - Generates IF..NOT for WHILE condition
LABGEN - Generate label for WHILE condition and loop termination condition
OUTCON - Generate labeled CONTINUE
OUTNUM - Output character version of label

Called from: PARSE

Description: Parses RATFOR WHILE statement, generating labels for both the WHILE condition and the loop termination condition. The WHILES routine processes the bottom of the loop.

Error Messages: None

Program: *** WHILES ***

Type: Subroutine

Parameters: LAB - Label for GOTO statement

Calls: OUTCON - Output "CONTINUE" with new label
OUTGO - Output "GOTO" with specified label

Called from: [illegible]STAK

Description: Processes the bottom of a WHILE loop, generating a "GOTO" to the top of the loop and a "CONTINUE" for after the loop.

Error Messages: None


DAT
FILM
6—