Five Dollars

Cromemco RATFOR

Instruction Manual

CROMEMCO

RATFOR

Reference Manual

CROMEMCO, INC.
28Ø Bernardo Avenue
Mountain View, CA 94Ø43

Part No. Ø23-ØØ67 June 1979

This manual was produced in its entirety with the Cromemco Word Processing System and was printed on a Cromemco Model 3355 printer.

PREFACE

This manual gives a description of Cromemco's implementation of Ratfor, a language and preprocessor whose output is Fortran IV. Ratfor, which is an acronym for RATional FORtran, contains all the numerical processing power of Fortran while providing logic control statements such as FOR, REPEAT...UNTIL, and WHILE that bring the advantages of structured program design to the Fortran programmer. Ratfor is described in detail in the book Software Tools by B. W. Kernighan of Bell Laboratories and P. J. Plauger of Yourdon inc. Software Tools is published by Addison-Wesley Publishing Company.

CROMEMCO RATFOR
Reference Manual

Contents

SECTION 1

INTRODUCTION

Cromemco Ratfor is Cromemco's version of the Ratfor preprocessor referred to in the book Software Tools by B. W. Kernighan and P. J. Plauger, published by Addison-Wesley Publishing Company. Ratfor, which is an acronym for RATional FORtran, is a structured Fortran-like language which is translated by the preprocessor into Fortran IV. The Cromemco Ratfor preprocessor supports all the features in the original preprocessor of Kernighan and Plauger, along with some additional features intended to make it more useful.

Ratfor is a language specifically designed to realize the benefits of structured programming. It promotes top-down thinking and program design. Ratfor programs are easy to write and easy to read. They are self-documenting, easy to debug, and easy to change.

Ratfor is a new language. It contains all of the numerical processing power of Fortran, and it also provides logic control statements to make Fortran easier to use. Fortran programmers will be repaid many times for the small effort that it takes to become familiar with the additional Ratfor control statements.

Several steps are performed in translating Ratfor programs into executable machine code. Programs are first written in Ratfor, then translated by the Ratfor preprocessor into Fortran, compiled using the Cromemco Fortran compiler (supplied as part of the Cromemco Ratfor package), and finally loaded into memory with the Linker.

The Cromemco Ratfor preprocessor has itself been written in Ratfor. Some of the primitive subprograms used by Ratfor to communicate with the user have been included in this manual as examples of the use of Ratfor. The generated output for these subprograms has also been included.

This manual is a reference for Cromemco Ratfor. The book Software Tools, supplied as part of the

Cromemco Ratfor package, provides a comprehensive tutorial on the Ratfor language.

Hardware requirements for Ratfor are a Cromemco computer with at least 48K of memory, two disk drives, and a CRT terminal. A printer is optional.

Software requirements are CDOS (Cromemco Disk Operating System), the Cromemco Text Editor, the Cromemco Fortran compiler, and Link, the Cromemco linking loader. These are all supplied with the Ratfor package.

Ratfor programs may be written in lower case or upper case. Throughout this manual, Ratfor statements, such as FOR, have been printed in upper case to set them off from the surrounding text.

SECTION 2

FEATURES

The Ratfor preprocessor receives as input a program written in Ratfor and outputs a program in Fortran which is then compiled with the Fortran compiler. The Fortran compiler is supplied on a separate disk with this package.

Cromemco Ratfor generates Fortran output that is somewhat easier to read than the Fortran output from the Software Tools Ratfor. The output from this original Ratfor contained no spaces, so much of it was difficult to read. In addition, all statements began in column 7 rather than preserving the indented outline structure of the Ratfor source.

However, the Fortran output of Cromemco Ratfor has been made as readable as possible for the user who would like to read and study it. Readable Fortran output is an additional learning aid for Fortran programmers who want to become familiar with Ratfor.

The Fortran output of Ratfor has the following special features:

1. The Fortran output associated with the Ratfor control structures is indented according to structured programming principles.

2. The parent Ratfor control structures BREAK, FOR, IF, ELSE, NEXT, REPEAT, and WHILE are marked in the Fortran output with single line comments, as follows:

   ```
   C *** FOR
   ```

   IF's associated with other structures (FOR, REPEAT, and WHILE) and generated by the preprocessor are not marked as IF statements.

3. Single line comments in a Ratfor program appear in the Fortran output as three comment lines; a blank comment line is added before and after the original comment. A blank comment in a Ratfor program results in a single blank comment in the Fortran output. Multiple line comments in a Ratfor program appear in the Fortran output preceded and followed by a single blank comment. Comments are indented according to the structure in which they are included.

4. Each Ratfor control structure in the translated Fortran output ends in a Fortran CONTINUE statement which is always vertically below the beginning of that structure. That is, the CONTINUE that ends a structure is indented to match the first statement of the structure.

5. Cromemco Ratfor will accept Ratfor programs in lower case, exactly as they are presented in the text Software Tools. This feature allows the Ratfor program to be written in lower case, and the DEFINEd symbols to be entered in upper case, thus setting them apart from the rest of the program.

SECTION 3

RATFOR SYNOPSIS

This section describes the logical symbols and control structures of the Ratfor language.

3.1 LOGICAL SYMBOLS

Ratfor recognizes the following symbols and converts them into the indicated Fortran operators whenever they are found in a Ratfor program.

Ratfor symbol	Fortran operator
==	.EQ.
^=	.NE.
<	.LT.
<=	.LE.
>	.GT.
>=	.GE.
&	.AND.
|	.OR.
^	.NOT.

3.2 CONTROL STRUCTURES

3.2.1 BREAK STATEMENT

```
break
```

The BREAK statement causes execution to cease within a containing loop and to begin at the first statement following the loop.

The containing loop may be that of a DO, FOR, REPEAT, or WHILE statement.

Example:

```
     do i = 1, 100
          {
          .
          .
          if (string(i) == BLANK)
              break
          .
          .
          }
     count = 1
```

The next statement executed after the BREAK statement will be:

```
     count = 1
```

3.2.2 DEFINE STATEMENT

```
     define(symbol, replacement string)
```

The DEFINE statement permits the programmer to specify a replacement string that is inserted into the program in place of each occurrence of the chosen symbol.

The symbol must be a string containing from one to two hundred letters and digits. Blanks and special characters are not allowed.

The replacement string may contain any sequence of from one to two hundred letters, digits, blanks, and special characters.

The comma must immediately follow the symbol in the DEFINE statement.

The maximum number of definitions is 200.

Ratfor searches the define table each time it reads a symbol from the input Ratfor program. When a match occurs, the replacement string is inserted into the input stream and the string is rescanned for DEFINEd symbols. Because of this, a series of DEFINE statements that effect a circular definition must not appear; Ratfor will loop indefinitely attempting to define the symbol.

Example:

```
define(BLANK, 32)
```

Ratfor will replace the symbol BLANK with the integer 32 each time it finds BLANK in the input. (32 is the ASCII representation of the space character.)

3.2.3 DO STATEMENT

```
do index = start, limit, increment
   statement
```

The DO statement is nearly identical in form to the Fortran DO statement. It is identical in function.

The controlling parameters index, start, limit, and increment have the same functions as defined in the Cromemco Fortran IV manual.

The statement within the DO loop may be any Ratfor statement, including a compound statement.

Example:

```
do i = 1, 1ØØ
   {
   ptotal(i) = part1(i) + part2(i)
   call print
   }
```

The two statements within the braces will be executed one hundred times.

3.2.4 FOR STATEMENT

```
for (initialize; condition; reinitialize)
   statement
```

The FOR statement defines an initialize statement that is executed once, a loop whose statements are repetitively executed as long as the controlling condition is true, and a reinitialize statement that is executed at the end of each pass through the loop.

The condition is tested prior to each execution of the loop, so it is possible for the loop not to be executed at all.

Execution proceeds to the first statement following the FOR statement when the condition is found to be false.

The initialize, condition, and reinitialize parts are each optional. The semicolons separating the parts are not optional; they must be used even when one or more of the parts are omitted. Omitting the condition part yields an infinite loop. The initialize and reinitialize statements must be single Fortran statements. The condition may consist of multiple comparisons. The statement within the loop may be any Ratfor statement, including a compound statement.

Example:

```
for (i = 1; i <= limit; i = i + 1)
    {
    ptotal(i) = part1(i) + part2(i)
    call print (ptotal(i))
    }
```

First, i is set to 1, then testing, looping, and incrementing begins. The two statements within braces will be executed and i will be incremented as long as i is less than or equal to limit.

3.2.5 IF STATEMENT

```
if (condition)
    statement-1
else
    statement-2
```

The condition is tested. If true, statement-1 is executed and statement-2 is skipped; otherwise, statement-1 is skipped and statement-2 is executed. The condition may consist of multiple comparisons, and statement-1 and statement-2 may be any Ratfor statements, including compound statements. The ELSE and statement-2 are optional; when the ELSE is omitted, statement-2 must also be omitted.

Example:

```
if (amount < Ø)
   call debit(amount)
else
   call credit(amount)
```

When amount is less than zero, the routine called debit will be executed. When amount is greater than or equal to zero, the routine called credit will be executed.

3.2.6 INCLUDE STATEMENT

```
include filename
```

Ratfor replaces the INCLUDE statement with the contents of the specified (Ratfor source) file, thereby inserting additional statements into the program.

The filename is not enclosed in quotes, and it has the form:

1-8 character name, period, 1-3 character extension

The file name and extension must contain only letters and digits. Special characters are not allowed.

The period and extension are optional. No default value is assigned to the extension.

The file must reside on the same disk as the Ratfor program being processed.

INCLUDE statements may be nested up to three deep.

The INCLUDE statement must be the only statement on the line.

Example:

```
     include maknam.rfr
```

The contents of the file with the name maknam.rfr will be inserted into the program in place of this INCLUDE statement.

3.2.7 NEXT STATEMENT

```
     next
```

The NEXT statement causes the rest of the containing loop to be skipped and execution to proceed with the next iteration of the loop.

For the DO, REPEAT...UNTIL, and WHILE loops, execution proceeds to the condition test.

For the FOR loop, execution proceeds to the reinitialize statement.

For an infinite REPEAT loop, execution proceeds to the top of the loop.

Example:

```
     for (i = 1; i <= limit; i = i + 1)
         {
         if (amount(i) >= Ø)
            next
         debtot = debtot + amount(i)
         }
```

These statements will add all elements of the array amount, skipping those elements with values greater than or equal to zero.

3.2.8 NULL STATEMENT

```
;
```

The semicolon may be used to occupy a position usually occupied by another Ratfor statement.

Example:

```
for (i = 1; string(i) == BLANK & i <= limit; i = i + 1)
    ;
```

These statements will scan the array string until a non-blank character is found. The subscript i will be left pointing to this character.

3.2.9 REPEAT STATEMENT

```
repeat
   statement
   until (condition)
```

The REPEAT statement defines a loop whose statements are repetitively executed until the controlling condition is true, and a condition that is tested after each pass through the loop.

Since the condition is tested after each pass through the loop, the loop will always be executed at least once. Execution proceeds to the first statement following the REPEAT statement when the condition is found to be true.

The condition may consist of multiple comparisons and the statement within the loop may be any RATFOR statement, including a compound statement.

The UNTIL part is optional. When it is omitted, the REPEAT statement defines an infinite loop.

Example:

```
     repeat
        {
        errors = NO
        call getstr (string, length)
        if (string(1) == BLANK)
           {
           errors = YES
           next
           }
        call proces (string, length)
        } until (errors == NO)
```

The statements within the braces will be executed until the variable errors is equal to the DEFINEd symbol NO.

3.2.1Ø WHILE STATEMENT

```
     while (condition)
        statement
```

The WHILE statement defines a loop whose statements are repetitively executed as long as the controlling condition is true, and a condition that is tested before each pass through the loop.

Since the condition is tested before each pass through the loop, it is possible for the loop not to be executed at all. Execution proceeds to the first statement following the WHILE statement when the condition is found to be false.

The condition may consist of multiple comparisons and the statement within the loop may be any Ratfor statement, incuding a compound statement.

Example:

```
     i = 1
     while (i <= 1Ø)
        {
        write (3, 1) i
        1 format (1x, 'Enter entry # ', I2)
        read (3, 2) score(i)
        2 format (F5.2)
        i = i + 1
        }
```

The statements within the braces will be executed 1Ø times, writing the message to the terminal and reading a value into the next element of the array, score, each time.

3.3 OTHER STATEMENTS

Any Fortran statements may be used. A statement label may appear anywhere on the line preceding a Fortran statement, either fully left-justified or indented to the current indentation level. In practice, only FORMAT statements require statement labels. (See the preceding section for an example.)

3.4 MISCELLANEOUS FEATURES

1. Comments

   A sharp sign (#) appearing anywhere on a line causes the rest of the line to be treated as a comment, unless the # is part of a quoted literal.

   Ratfor comments appearing at the end of some control structures will be processed before the ending CONTINUE is generated, and the corresponding Fortran comments may appear before the CONTINUE. If this is a problem, move the Ratfor comments down one statement.

2. Compound statement

   Braces, {}, can be used to enclose single or multiple Ratfor or Fortran statements so that the enclosed block of statements may appear anywhere that a single statement may appear, except as the initialize or reinitialize part of the FOR statement.

   Braces are commonly used to enclose several statements that are to be executed as a block following a DO, FOR, IF, ELSE, REPEAT, or WHILE statement.

   Compound statements may be nested; e.g., one of the statements within braces might be a FOR statement which might itself contain a compound statement.

Example:

```
     if (count == Ø)
        {
        call remark ('Invalid number of entries.')
        return
        }
     else
        {
        for (i = 1; i <= count; i = i + 1)
           {
           total(i) = total(i) + score(i)
           call print (score(i))
           }
        return
        }
```

If count is equal to zero, two statements will be executed, the call and the return. Otherwise, two other statements will be executed, the FOR and the return. Within the FOR statement, two statements, the assignment and the call, will be repetitively executed so long as the variable i is less than or equal to count.

3. Conditions

Conditions may represent comparisons between two operands, or multiple comparisons connected by the logical OR operator, |, and the logical AND operator, &.

Examples:

```
     if (a == b | a == c)
        .
        .
        .
```

In this example, the condition will be true if a is equal to b or if a is equal to c.

```
     while (char ^= COMMA &
            char ^= EQUALS &
            char ^= EOS &
            i <= maxsiz)
            .
            .
            .
```

For the condition in this example to be true, char must not be equal to any one of the DEFINEd symbols COMMA, EQUALS, or EOS, and i must be less than or equal to maxsiz.

4. Indentation

The statements in a Ratfor program should be indented according to the conventions of structured programming to allow for readability. This may easily be accomplished using the variable tab stop capability of the Cromemco Text Editor.

5. Literals

Ratfor does not recognize Hollerith literals. ("1H ," translates to "1H," which is incorrect.) It does recognize literal strings between single or double quotes and translates them to Hollerith literals. For example, "'This'" translates to "4HThis" (the double quotes in each case are not part of the construction).

6. Long lines

Ratfor generates Fortran continuation lines when it finds extended conditions in IF, FOR, REPEAT, or WHILE statements that extend to the next line. Ratfor also generates Fortran continuation lines when a line ends with a comma, as might occur in a long FORMAT statement.

7. Names

Names used in a Ratfor program must be valid Fortran names (six or fewer alphanumeric characters the first of which must be a letter).

8. Nested statements

Ratfor statements may be nested within other Ratfor statements to a maximum level of 1ØØ. A single or compound Ratfor statement is said to be nested within another Ratfor statement when it is used as the statement associated with a DO, FOR, IF, ELSE, REPEAT, or WHILE structure. Further, all single or compound statements within the braces of a compound statement are said to be nested within those braces.

Example:

```
repeat
   {
   flag = Ø
   call getwrd (word, wrdlen)
   if (match (word, 'Overdue') == YES)
      {
      call badlst
      flag = 1
      }
   } until (flag == 1)
```

This example illustrates four levels of nesting. A compound statement is nested within the REPEAT. The assignment, call, and IF statements are nested at the same level within the compound statement. Another compound statement is nested within the IF. Finally the call and the assignment statements are nested at the same level within the compound statement.

SECTION 4

PROCESSING RATFOR PROGRAMS

4.1 Ratfor Command Line

Instructions are given to Ratfor through the console keyboard. Input and output file names and listing options are specified either when Ratfor is executed by entering the file names and options on the line following the word Ratfor before the return key is pressed, or when Ratfor issues a prompt (*) to the terminal. In both cases, the form of the command is the same:

```
output name, listing option = input name
```

The names have the form:

```
drive:name.extension
```

where:

drive — optional
when present it must be a single letter in the range A-Z; when omitted it is assigned a default value;

colon — required when drive is specified, otherwise it must not be used;

name — 1 - 8 characters in length
any characters are valid except spaces, control characters, any of ? * , = / or the DEL character (ASCII 7FH);

period — required when extension is specified, otherwise it must not be used;

extension — optional
when present, it is 1 - 3 characters long, with the same character restrictions as the name; when omitted it is assigned a default value.

Default values are assigned as follows:

input	drive defaults to the current drive; extension defaults to "RFR"
output	drive defaults to the current drive; name defaults to the input name; extension defaults to "FOR"

There are three listing options:

TTY:	directs the listing to the console terminal only;
PRT:	directs the listing to the printer only;
<nothing>	suppresses the listing.

When there is nothing to the left of the equal sign, the output drive, name, and extension default as defined above, and the listing is suppressed.

The output is suppressed when there is no name preceding the comma.

The listing is suppressed when there is no listing option.

Both the output and the listing are suppressed when the equal sign is preceded by only a comma.

Examples:

```
RATFOR    ,TTY: = GETLIN
```

The input file GETLIN.RFR is read from the current disk and the listing is directed to the terminal. No output file is produced.

```
RATFOR
(version message is written to terminal)
(prompting message is written to terminal)

*B:GETLIN,PRT: = GETLIN
```

The input file GETLIN.RFR is read from the current disk, an output file with the name GETLIN.FOR is written to disk drive B, and the listing is directed to the printer.

```
RATFOR =MERGE
```

The input file MERGE.RFR is read from the current disk and an output file with the name MERGE.FOR is written to the current disk. No listing is produced.

4.2 Sample Ratfor Programs

Sample 1:

```
  program echo
#
# This program reads up to 8Ø characters from the
# terminal, converts all lower case letters to upper
# case, and writes them back to the terminal.
#
define(BIGA, 65)
define(BLANK,32)
define(CHARACTER,logical)
define(CRT,3)
define(CRTLINE,8Ø)
define(DIGØ,48)
define(LETA, 97)
define(LETZ, 122)
define(MINUS,45)
define(PERIOD,46)
define(PLUS,43)
#
  CHARACTER outmap
  CHARACTER line (8Ø)
  integer i, bcount
#
```

```
  repeat                                #until nothing
                                        #is entered
     {
     call remark ('Enter line to be echoed:.')
     call remark (' .')
     for (i = 1; i <= 8Ø; i = i + 1)    #blank fill line
        line(i) = BLANK
     read (CRT, 1) (line(i), i = 1, 8Ø)
     bcount = Ø
     for (i = 1; i <= 8Ø; i = i + 1)    #convert all letters
        {                               #to upper case
        if (line(i) == BLANK)
           bcount = bcount + 1
        else
           line(i) = outmap (line(i))
        }
     if (bcount == 8Ø)
        break
     write (CRT, 2) (line(i), i = 1, 8Ø)
     call remark (' .')
     }
  call remark ('End of demonstration.')
1 format (CRTLINE a1)
2 format (1x, CRTLINE a1)
  end
include outmap.rfr
include remark.rfr
```

The program ECHO INCLUDEs two files that contain two primitives used in Ratfor, OUTMAP and REMARK. Listings of the contents of these files follow.

<u>File outmap.rfr</u>

```
   CHARACTER function outmap(inchar)
#  outmap - convert to ascii upper case for fortran output
   CHARACTER inchar
#
   if (inchar >= LETA & inchar <= LETZ)
      outmap = inchar - LETA + BIGA
   else
      outmap = inchar
   return
   end
```

File remark.rfr

```
   subroutine remark(buf)
#  remark - print warning message
   CHARACTER buf(CRTLINE)
#  find position of period in input string
   for (k = 1; buf(k) ^= PERIOD & k < CRTLINE; k = k + 1)
      ;
   if (buf(k) ^= PERIOD)
      write (CRT, 1Ø)
   if (k > 1)
      k = k - 1
   if (buf(1) == MINUS | buf(1) == BLANK | buf(1) == PLUS
      | buf(1) == DIGØ)
      write (CRT, 11) (buf(i), i = 1, k)
   else
      write (CRT, 12) (buf(i), i = 1, k)
   return
#
1Ø format (' ', 'no period in remark')
11 format (CRTLINE a1)
12 format (' ',CRTLINE a1)
   end
```

Note in the above that OUTMAP is a function whereas REMARK is a subroutine.

Sample 2:

```
  program roots
#
# This program calculates and displays the same table
# of square roots displayed by the example program
# ROOTS in the Cromemco Fortran manual.
#
  real a (1Ø)
  integer i, j, k
#
# These lines display the heading for the table
#
  write (3, 1)
1 format (1x, /////, 25x, 'Table of Square Roots', /)
  write (3, 2)
2 format (1X, 8X, 'Ø', 6X, '1', 6X, '2', 6X, '3', 6X,
                  '4', 6X, '5', 6X, '6', 6X, '7', 6X,
                  '8', 6X, '9', /)
```

```
#
# The following lines calculate 10 square roots, load
# them into an array, and display the elements of the
# array.  This repeats 10 times to display the entire
# table.
#
  for (i = 0; i <= 9; i = i + 1)
     {
     for (j = 0; j <= 9; j = j + 1)
        a(j + 1) = sqrt (float(10 * i + j))
     write (3, 3) i, (a(k), k = 1, 10)
     }
  write (3, 4)
3 format (1x, i1, '-', 10f7.3)
4 format (1x, //////)
  end
```

4.3 Sample Programs after Processing

ECHO and ROOTS have been translated using Ratfor. Listings of the generated Fortran programs follow.

Sample 1:

The following command line instructs Ratfor to produce a Fortran file with the name ECHO.FOR and to direct the listing to the terminal. The listing which follows is that directed to the terminal.

```
A.ratfor echo,tty:=echo
```

```
      PROGRAM ECHO
C
C
C      THIS PROGRAM READS UP TO 80 CHARACTERS FROM THE
C      TERMINAL, CONVERTS ALL LOWER CASE LETTERS TO UPPER
C      CASE, AND WRITES THEM BACK TO THE TERMINAL.
C
C
C
      LOGICAL OUTMAP
      LOGICAL LINE(80)
      INTEGER I,BCOUNT
C
C      *** REPEAT
      CONTINUE
23000 CONTINUE
C
C        UNTIL NOTHING
C        IS ENTERED
C
        CALL REMARK(25HEnter line to be echoed:.)
        CALL REMARK(2H .)
```

```
C        *** FOR
        CONTINUE
        I=1
23003   IF(.NOT.(I.LE.80))GOTO 23005
C
C          BLANK FILL LINE
C
          LINE(I)=32
23004     I=I+1
          GOTO 23003
23005   CONTINUE
        READ(3,1)(LINE(I),I=1,80)
        BCOUNT=0
C        *** FOR
        CONTINUE
        I=1
23006   IF(.NOT.(I.LE.80))GOTO 23008
C
C          CONVERT ALL LETTERS
C
C
C          TO UPPER CASE
C
C          *** IF
          IF(.NOT.(LINE(I).EQ.32))GOTO 23009
            BCOUNT=BCOUNT+1
            GOTO 23010
C          *** ELSE
23009     CONTINUE
            LINE(I)=OUTMAP(LINE(I))
23010     CONTINUE
23007     I=I+1
          GOTO 23006
23008   CONTINUE
C        *** IF
        IF(.NOT.(BCOUNT.EQ.80))GOTO 23011
C          *** BREAK
          GOTO 23002
23011   CONTINUE
        WRITE(3,2)(LINE(I),I=1,80)
        CALL REMARK(2H .)
23001   GOTO 23000
23002 CONTINUE
      CALL REMARK(21HEnd of demonstration.)
1     FORMAT(80A1)
2     FORMAT(1X,80A1)
      END
      LOGICAL FUNCTION OUTMAP(INCHAR)
C
C      OUTMAP - CONVERT TO ASCII UPPER CASE FOR FORTRAN OUTPUT
C
      LOGICAL INCHAR
C
C      *** IF
      IF(.NOT.(INCHAR.GE.97.AND.INCHAR.LE.122))GOTO 23013
        OUTMAP=INCHAR-97+65
        GOTO 23014
C      *** ELSE
23013 CONTINUE
        OUTMAP=INCHAR
23014 CONTINUE
      RETURN
      END
      SUBROUTINE REMARK(BUF)
C
C      REMARK - PRINT WARNING MESSAGE
C
      LOGICAL BUF(80)
C
C      FIND POSITION OF PERIOD IN INPUT STRING
C
C      *** FOR
      CONTINUE
      K=1
23015 IF(.NOT.(BUF(K).NE.46.AND.K.LT.80))GOTO 23017
23016   K=K+1
        GOTO 23015
23017 CONTINUE
C      *** IF
      IF(.NOT.(BUF(K).NE.46))GOTO 23018
        WRITE(3,10)
23018 CONTINUE
C      *** IF
      IF(.NOT.(K.GT.1))GOTO 23020
        K=K-1
23020 CONTINUE
C      *** IF
      IF(.NOT.(BUF(1).EQ.45.OR.BUF(1).EQ.32.OR.BUF(1).EQ.43.OR.BUF(1).EQ
     *.48))GOTO 23022
        WRITE(3,11)(BUF(I),I=1,K)
        GOTO 23023
C      *** ELSE
23022 CONTINUE
        WRITE(3,12)(BUF(I),I=1,K)
23023 CONTINUE
      RETURN
C
10    FORMAT(1H ,19Hno period in remark)
11    FORMAT(80A1)
12    FORMAT(1H ,80A1)
      END
There were
0
errors in this ratfor run
```

Sample 2:

The following command line instructs Ratfor to output a Fortran file with the name ROOTS.FOR without producing a listing. The listing which follows displays the contents of ROOTS.FOR.

```
A.ratfor =roots

There were
Ø
errors in this Ratfor run
```

```
      PROGRAM ROOTS
C
C
C      THIS PROGRAM CALCULATES AND DISPLAYS THE SAME TABLE
C      OF SQUARE ROOTS DISPLAYED BY THE EXAMPLE PROGRAM
C      ROOTS IN THE CROMEMCO FORTRAN MANUAL.
C
C
      REAL A(1Ø)
      INTEGER I,J,K
C
C
C      THESE LINES DISPLAY THE HEADING FOR THE TABLE
C
C
      WRITE(3,1)
1     FORMAT(1X,/////,25X,21HTable of Square Roots,/)
      WRITE(3,2)
2     FORMAT(1X,8X,1HØ,6X,1H1,6X,1H2,6X,1H3,6X,1H4,6X,1H5,6X,1H6,6X,1H7,
     *6X,1H8,6X,1H9,/)
C
C
C      THE FOLLOWING LINES CALCULATE 1Ø SQUARE ROOTS, LOAD
C      THEM INTO AN ARRAY, AND DISPLAY THE ELEMENTS OF THE
C      ARRAY.  THIS REPEATS 1Ø TIMES TO DISPLAY THE ENTIRE
C      TABLE.
C
C
C      *** FOR
      CONTINUE
      I=Ø
23ØØØ IF(.NOT.(I.LE.9))GOTO 23ØØ2
C        *** FOR
        CONTINUE
        J=Ø
23ØØ3   IF(.NOT.(J.LE.9))GOTO 23ØØ5
          A(J+1)=SQRT(FLOAT(1Ø*I+J))
23ØØ4     J=J+1
          GOTO 23ØØ3
23ØØ5   CONTINUE
        WRITE(3,3)I,(A(K),K=1,1Ø)
23ØØ1   I=I+1
        GOTO 23ØØØ
23ØØ2 CONTINUE
      WRITE(3,4)
3     FORMAT(1X,I1,1H-,1ØF7.3)
4     FORMAT(1X,//////)
      END
```

4.4 Sample Program Compilation and Linking

The Fortran programs in the preceding section were compiled and linked using the following command lines. The Cromemco Fortran manual contains a complete explanation of the use of the compiler and the Link program, along with an explanation of the numbers displayed by the Link program.

Sample 1:

```
A.for =echo
ECHO
OUTMAP
REMARK


A.link echo,echo/n/e

Data     Ø1Ø3     1BE2

[Ø19D    1BE2     27]
```

Sample 2:

```
A.for =roots
ROOTS


A.link roots,roots/n/e

Data     Ø1Ø3     1F3C


[Ø1CE    1F3C     31]
```

4.5 Sample Program Executions

The sample programs ECHO and ROOTS produce the following results when executed.

Sample 1:

B.ECHO

```
Enter line to be echoed:
ECHO converts lower case letters to upper case.

ECHO CONVERTS LOWER CASE LETTERS TO UPPER CASE.

Enter line to be echoed:
Characters such as !, ", #, ?, :, etc.  are not converted.

CHARACTERS SUCH AS !, ", #, ?, :, ETC.  ARE NOT CONVERTED.

Enter line to be echoed:
A blank line terminates the program.

A BLANK LINE TERMINATES THE PROGRAM.

Enter line to be echoed:
<CR>

End of demonstration
```

Sample 2:

A. ROOTS

```
                           TABLE OF SQUARE ROOTS

        Ø      1      2      3      4      5      6      7      8      9

Ø-  Ø.ØØØ  1.ØØØ  1.414  1.732  2.ØØØ  2.236  2.449  2.646  2.828  3.ØØØ
1-  3.162  3.317  3.464  3.6Ø6  3.742  3.873  4.ØØØ  4.123  4.243  4.359
2-  4.472  4.583  4.69Ø  4.796  4.899  5.ØØØ  5.Ø99  5.196  5.292  5.385
3-  5.477  5.568  5.657  5.745  5.831  5.916  6.ØØØ  6.Ø83  6.164  6.245
4-  6.325  6.4Ø3  6.481  6.557  6.633  6.7Ø8  6.782  6.856  6.928  7.ØØØ
5-  7.Ø71  7.141  7.211  7.28Ø  7.348  7.416  7.483  7.55Ø  7.616  7.681
6-  7.746  7.81Ø  7.874  7.937  8.ØØØ  8.Ø62  8.124  8.185  8.246  8.3Ø7
7-  8.367  8.426  8.485  8.544  8.6Ø2  8.66Ø  8.718  8.775  8.832  8.888
8-  8.944  9.ØØØ  9.Ø55  9.11Ø  9.165  9.22Ø  9.274  9.327  9.381  9.434
9-  9.487  9.539  9.592  9.644  9.695  9.747  9.798  9.849  9.899  9.95Ø
```

SECTION 5

ERROR MESSAGES

The Ratfor preprocessor checks the input Ratfor program for Ratfor syntax errors. It does not check for Fortran syntax errors; that job is left for the Fortran compiler. When the preprocessor finds an error, it displays one of the following messages on the terminal and continues processing. Some errors precipitate a cascade of other errors as the preprocessor attempts to translate input Ratfor statements as continuations of a statement which was ignored because of a syntax error. The preprocessor maintains a count of syntax errors encountered and displays this number at the end of the run. The run is terminated if this count exceeds 100.

Most errors are non-fatal; processing will continue after the error message is displayed. Some, including "Definition too long", "Missing right parenthesis", and "Stack overflow in parser" are fatal; processing terminates at that point.

5.1 Error Messages

Can't open include — there is an error in the filename

Definition too long — the symbol or replacement string in the DEFINE statement is longer than 200 characters

Error at line... — this is the line number of the input line containing an error

Illegal break — BREAK may not appear at this point; it may appear only within a DO, FOR, REPEAT or WHILE loop

Illegal else — there is no preceding IF

Illegal next — NEXT may not appear at this point; it may apper only within a DO, FOR, REPEAT, or WHILE loop

Illegal right brace

there is no preceding left brace

Includes nested too deeply

an attempt has been made to nest INCLUDE files more than 3 deep

Invalid for clause — the FOR clause is not closed properly

Invalid input name — the specified input name is not valid; see Section 4.1

Invalid listing option

The specified listing option is not valid; the three options are TTY:, PRT:, and nothing

Invalid name list — the command line contains multiple commas or equal signs, the comma follows the equal sign, or the command line is too long

Invalid output name

the specified output name is not valid; see Section 4.1

Missing comma in define

the first character following the DEFINEd symbol is not a comma

Missing left parenthesis

the first character following the word DEFINE is not a left parenthesis;

or
there is no parenthesis following the IF statement or the FOR statement

Missing parenthesis in condition

there is no right parenthesis in the IF statement

Missing quote — there is no matching quote at the right end of a quoted literal

Missing right parenthesis

there is no right parenthesis in the DEFINE statement

Non-alphabetic name following include

the file name following the INCLUDE contains a character that is neither a letter nor a digit

Non-alphanumeric name

the DEFINEd symbol contains a character that is neither a letter nor a digit

Stack overflow in parser

an attempt has been made to nest more than 1ØØ numeric labels, left braces, or DO, FOR, IF, ELSE, REPEAT, or WHILE statements

Token too long — an attempt has been made to use a symbol longer than 2ØØ characters

Too many characters pushed back

Ratfor scanned more than 3ØØ characters during its look ahead processing; this most commonly occurs in the DEFINE statement when a long replacement string contains symbols that are themselves DEFINEd by long replacement strings

Too many definitions

there are more than 200 DEFINE statements in the program

Unbalanced parentheses

there is a left parenthesis with no matching right parenthesis

Unexpected brace or eof

there is no preceding left brace;
or
the current control block was not finished when end-of-file was encountered

Unexpected eof

there were control blocks that weren't finished when end-of-file was encountered

Warning: possible label conflict

input labels in the range 23000 -23999 may conflict with Ratfor generated labels

INDEX

Cromemco®
280 BERNARDO AVE., MOUNTAIN VIEW, CA 94043