1/91

CNWRA 91-003

Blue + White

preFOR: A PRE-PROCESSOR FOR FORTRAN FILES

USER'S MANUAL

Prepared for

Nuclear Regulatory Commission
Contract NRC-02-88-005

Prepared by

Center for Nuclear Waste Regulatory Analyses
San Antonio, Texas

April 1991

CNWRA 91-003 2

preFOR: A PRE-PROCESSOR FOR FORTRAN FILES

USER'S MANUAL

Prepared for

Nuclear Regulatory Commission
Contract NRC-02-88-005

Prepared by

R. W. Janetzke
B. Sagar

Center for Nuclear Waste Regulatory Analyses
San Antonio, Texas

April 1991

TABLE OF CONTENTS

4

LIST OF FIGURES

5

LIST OF TABLES

6

preFOR: A PRE-PROCESSOR FOR FORTRAN FILES

USER'S MANUAL

ABSTRACT

preFOR is designed to process code written in FORTRAN in which preFOR commands have been embedded. The output of the processing will be a standard FORTRAN file that can be compiled like an other FORTRAN file. The preFOR commands allow the programmer significant flexibility in coding. preFOR commands include creation of code blocks that can be inserted at appropriate places. These insertions can be made conditional on use of a particular central processing unit (CPU), thus increasing code portability. In addition, the preFOR utility provides other features such as numbering of lines in the code, deletion of comments, and trimming of trailing characters in lines of code. preFOR is written in ANSI standard FORTRAN 77.

ACKNOWLEDGMENT

The authors thank Paul Eslinger for providing the UPDATE program on which the preFOR program is based. They also are thankful to Rawley Johnson and Wes Patrick for their reviews of this document.

1. INTRODUCTION

The preFOR computer program described in this report was developed to automate inclusion of certain desirable features in FORTRAN programs. Thus, preFOR can be regarded as a utility to pre-process FORTRAN files. The automated features include line numbering and addition of specified blocks of code at appropriate locations in the overall code. For example, with the aid of the preFOR utility, common blocks can be inserted at appropriate locations in various subroutines. Other functions will be explained in the following.

The preFOR program was obtained by modifying an earlier program named UPDATE which was developed by Dr. P. W. Eslinger of Battelle Pacific Northwest Laboratory (personal communication between Drs. Sagar and Eslinger). The preFOR utility is expected to provide desired code transportability of the total system performance assessment code that is currently under development jointly by the Center for Nuclear Waste Regulatory Analyses (Center) and the Nuclear Regulatory Commission (NRC).

The preFOR program is written in ANSI standard FORTRAN 77 language. A listing of the source program is provided in the appendix.

2. INSTRUCTIONS FOR USING preFOR

The preFOR program needs to be used only if preFOR commands are used in the development of the source code. These preFOR commands are described in the next Section. A distinctive extension to the source code file name may be used to distinguish it from standard FORTRAN files. While any extension may be used, the extension .PRE is recommended. In the following, for illustration, the source code containing one or more of the preFOR commands is named SOURCE.PRE. When invoked, preFOR will process SOURCE.PRE to create a compilable standard FORTRAN file (see Figure 1). The user can choose any name for this standard FORTRAN file but for illustration purposes, it will be called SOURCE.FOR. The SOURCE.FOR file is then ready for compilation like any other FORTRAN file.

The preFOR is fully interactive. When invoked the program will prompt for an input file name. At this prompt, the user should provide the name of the file that contains the source code with embedded preFOR commands, e.g., SOURCE.PRE. By default, preFOR assumes that the input file name has an extension .PRE. That is, if the user types a file name without an extension, e.g., SOURCE; preFOR automatically appends .PRE to it and looks for a file named SOURCE.PRE. However, the user may provide any file name with any extension including no extension at all. For the latter, the file name must end with a period (.), e.g., (SOURCE.).

preFOR checks for an extension simply by looking for a period (.) anywhere in the file name. If one is found the .PRE extension is not added to the input string. It is recommended that the full name of the source file (including extensions) be typed.

The program will then prompt for an output file name. This output file is the compilable FORTRAN file that will be created by the preFOR program. At this prompt, the full name that the user wishes to assign to the compilable FORTRAN file should be typed. While any extension may be used, the extension .FOR is recommended. Thus, if the input file name was SOURCE.PRE, the output file may be named SOURCE.FOR. If a name without any extension is typed, preFOR will automatically add the .FOR extension to the typed name. If no extension is desired, the file name must end with a period (.). Typing of full output file name including extension is recommended.

If the output file name is identical to the input file name then an error condition is identified and processing is stopped. preFOR is designed to not overwrite over existing files. When an output file name is provided, preFOR checks to see if a file with the same name already exists. If no such file exists, processing proceeds. However, if such a file is found, then a prompt asks the user whether the existing file should be deleted. If the user types YES at this prompt, then the existing file is deleted and the processing continues. On the other hand, if the user types NO, further processing is halted.

A sample printout of the prompts and user responses is shown in Figure 2.

3. USE OF preFOR COMMANDS

The preFOR commands that can be used for developing FORTRAN source programs are described in Table 1. below. All preFOR commands begin with an asterisk (*) in column 1. In the following, capital letters are used for that part of the command that must be used exactly as stated while lower case letters are used to indicate variable names to be provided by the user.

Table 1. Commands of preFOR

Command	Description
*CPU cpu	The 'cpu' parameter specifies the name of the CPU (or machine such as IBMPC, CRAY, VAX etc.) that indicates that certain processing by the preFOR is conditional on matching the CPU

name on the *INSERT/cpu command. This command, if used, should be the first executable preFOR command of the source code.

The cpu name is used only for comparisons and has no other meaning. The *CPU command is superfluous unless there exists an *INSERT/cpu command somewhere following it in the file.

Examples:

```
*CPU XYZ
*CPU CRAY
```

*FRAGMENT name
or
*COMDECK name

This command is used to define the beginning of a code fragment and to associate the fragment with a name. The parameter 'name' is a user selected identifier for the fragment which is used to establish a temporary file for storage of the fragment. A fragment can be any FORTRAN compilable block of code such as FORTRAN lines, common blocks, system commands etc. While this command can be used anywhere in the code, generally all fragments are assembled at the beginning of the code.

There is no default value for 'name', it must be supplied.

Examples:

```
*FRAGMENT ABCD
      Y = X**2 + Z
*FRAGMENT SIZE
      COMMON/ VOLUME / V(100)
      COMMON/ AREA   / A1(50), A2(50)
*FRAGMENT STIME
      CALL TIME(I,J,K)
```

*INSERT name
or
*CALL name

This command can be used to insert the named code fragment anywhere in the source code. The code fragment is given a name by using the commands *FRAGMENT or *COMDECK (see above). above). A code fragment may be anything that a FORTRAN compiler will process, e.g., set of FORTRAN lines, system commands, common blocks etc..

preFOR processes this command by searching for the named

10

fragment and replacing the *INSERT statement with the contents of the fragment. If the named fragment is not found, an error is printed and further processing is stopped.

If line numbering is enabled (see *DECK/NUMBER command) the first 4 characters of the parameter 'name' will be used for the inserted lines.

Examples:

```
*INSERT ABCD
*INSERT SIZE
```

*INSERT/cpu name	Inserts the named fragment only if value of 'cpu' matches the value of the 'cpu' parameter in the *CPU command. See *INSERT for more information.

Examples:

```
*INSERT/XYZ SIZE
*INSERT/VAX STIME
```

*COMMENTS/DELETE	Inhibits the writing of comment lines to the output file. This includes any lines encountered as a result of the *CALL or *INSERT command. This can significantly reduce the size of a heavily commented source file. If line numbering is invoked (see *DECK/NUMBER command), comment lines will be ignored for numbering purposes. That is, the resulting file will be numbered consecutively with no gaps indicating the missing comments. This command can be used any number of times in conjunction with the *COMMENTS/KEEP command to toggle the comment processing on and off. In the absence of *COMMENTS/DELETE, comments will be retained.
*COMMENTS/KEEP	Returns comment processing to the state it was in before the *COMMENTS/DELETE command occurred. It can be used any number of times in conjunction with the *COMMENTS/DELETE command to toggle the comment processing on and off. Because preFOR uses *COMMENTS/KEEP as default, this command need not be used if all comments are to be retained.

11

*COMMENTS/UPPER	Converts comment lines to upper case. This is important especially if the 'c' in column one is lower case, since FORTRAN 77 requires an uppercase 'C'. The effect of this command is reversed with the /NOUPPER switch.
*COMMENTS/NOUPPER	This command negates the *COMMENTS/UPPER command. Under default conditions comments are not shifted to upper case.
*DECK name	This command turns the line numbering off. It can be used in conjunction with the *DECK/NUMBER command to toggle line numbering on and off. This command will also terminate the processing of a *COMDECK or *FRAGMENT command. The value of the 'name' parameter usually refers to the current subroutine name, but may be blank.
*DECK/NUMBER name	Invokes numbering of the deck with only the first 4 characters of 'name' concatenated with a 4 digit number in columns 73-80. *DECK/NUMBER may be abbreviated to *DECK/N. The value of the 'name' parameter is used until another *DECK command is encountered. If line numbers are requested without replacing tabs with spaces, they may be artificially shifted to the right when viewed with compilers which honor tabs in their listings. Under default conditions line numbering is not enabled.

Example:

```
*DECK/NUMBER TPA
```

*INPUT/WIDTH=n	Limits the input to the first n columns, the remaining columns are ignored. The only practical values for the parameter 'n' at this time are 72 and 80. With a value of 72, a program which was received with line numbers in columns 73-80 can be processed to produce a much compressed source with line numbers and trailing blanks removed. The default value of n is 80.
*NOTE	This command does not change the values of any preFOR parameters, but will terminate the processing of *COMDECK and *FRAGMENT commands. It is provided as a convenience for commenting the preFOR process.

12

*OUTPUT/TRIM	This command trims trailing blanks and tabs, thus producing a compressed source code. This is the default mode.
*OUTPUT/NOTRIM	This command negates the *OUTPUT/TRIM command.
*OUTPUT/WIDTH=n	Limits the output width to columns 1-n. The only practical values for the parameter 'n' at this time are 72 and 80. This command can process a file that was received in extended column format, that is valid FORTRAN code in columns 73-80, and generate a file which is compatible with FORTRAN 77. This is done by taking the information in 73-80 and wrapping it around to the next line, with a continuation mark in column 6 as required. preFOR wraps both code and comment lines. The new lines thus generated receive unique line numbers if line numbering is enabled. This command is ignored if line numbering is enabled and wrapping is done automatically. Wrapping is not done if *INPUT/WIDTH=72 is used. The default value for n is 80.
*TAB/KEEP	Keeps tabs as they appear in the input file. Can be used in conjunction with the *TAB/REPLACE command to toggle the tab processing on and off. This is the default mode.
*TAB/REPLACE	Replace tabs with blanks as specified with the fixed values 9,17,25,33,41,49,57,65,73. These are coincident with VT100 compatible editors.

An example of use of these commands is provided in Figure 3.

4. DIMENSION PARAMETER OF preFOR

preFOR puts each fragment into a temporary file. The parameter MXCOM defines the maximum number of temporary files allowed to be generated by preFOR. The default value of MXCOM in the current version of preFOR is 250. This parameter declares the array space for the temporary file names of the code fragments. If the number of code fragments exceeds MXCOM it should be increased to match or exceed the number of fragments.

5. PERFORMANCE OF preFOR

Eighty column/notrim mode is the fastest processing mode for preFOR, but it also creates the largest files. Restricting the input width to 72 columns adds about 5% in time. Generating line numbers adds about 46%. Shifting comments to uppercase adds about 12%. A straight file (no preFOR commands) will run with the default parameters and add about 160%, but the resulting file will be the smallest because trim mode is enabled by default.

6. SUMMARY AND CONCLUSIONS

The preFOR computer program is a pre-processor for FORTRAN files. Its commands can provide flexibility in creating a FORTRAN source code. Specifically, code lines that are specific to different computer systems can be programmed as fragments. Using the *CPU and *INSERT/cpu commands, only the applicable fragments will be inserted into the FORTRAN code prior to creating the compilable file. This provides easy maintenance of the source code since machine dependent versions need not be maintained in separate files.

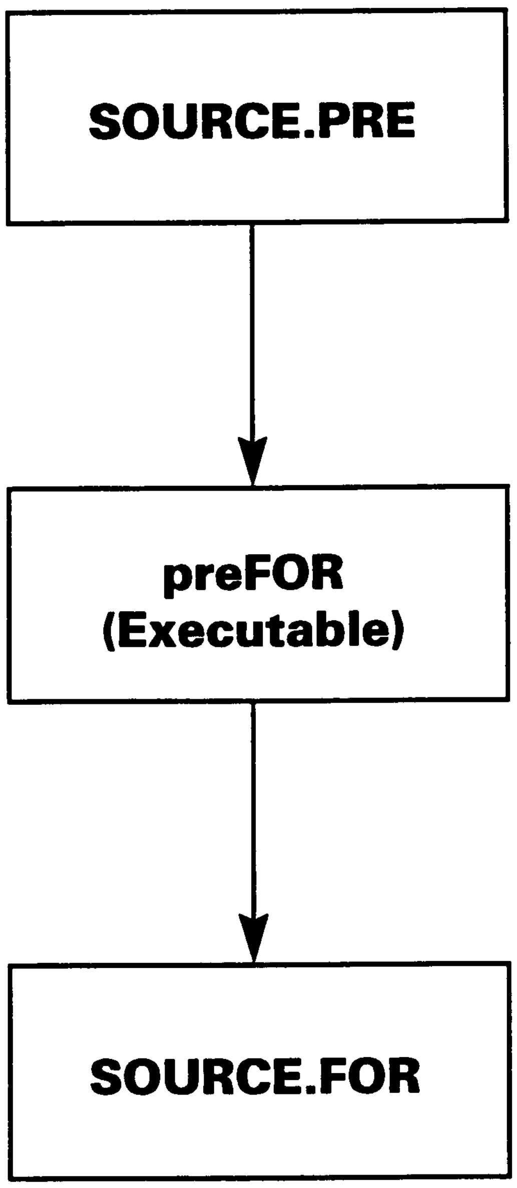
Common blocks which are used in many subroutines can also be coded as fragments and inserted when needed. Any change in a common block, thus needs to be made at only one place. This minimizes chances of errors in defining common blocks.

Other features such as inserting line numbers can help in better maintenance of the source code also.

7. CONTACTS

For any problems related to the use of preFOR or to provide any suggestions for its improvement, contact Dr. Budhi Sagar at (512)522-5252 or Mr. Ron Janetzke at (512)522-3318.

Figure 1. Data Flow Diagram for preFOR.



15

Figure 2. Sample preFOR VAX interactive session.

```
$ RUN PREFOR
preFOR 2.0    26 October 1990     Ron Janetzke
CONVERT A preFOR FILE TO A FORTRAN COMPILE FILE

ENTER THE UPDATE/preFOR SOURCE FILE NAME
PATH NAMES UP TO 64 CHARACTERS ARE ALLOWED
source.pre

ENTER OUTPUT (COMPILE) FILE NAME
PATH NAMES UP TO 64 CHARACTERS ARE ALLOWED
source.for

THE REQUESTED COMPILE FILE ALREADY EXISTS

FILE = SOURCE.FOR


DO YOU WISH TO DELETE IT ?
y



  1 DECKS PROCESSED
  0 INSERT DECKS PROCESSED
   98 EXECUTABLE LINES IN THE COMPILE FILE
   90 COMMENT LINES IN THE COMPILE FILE
  188 TOTAL LINES IN THE COMPILE FILE
FORTRAN STOP
$
```

16

Figure 3. Example showing use of preFOR commands

```
*NOTE This is a sample input file to
*NOTE be processed by preFOR.  The        } A
*NOTE resulting output file is shown
*NOTE below.
*CPU VAX                                  } B
*FRAGMENT 1ST_BLOCK                       } C
c
c     this is code fragment 1 to be
c     inserted in the output.             } D
c
      COMMON /BLOCK/ ONE
*FRAGMENT 2ND_BLOCK                       } E
c
c     this is code fragment 2 to be
c     inserted in the output.             } F
c
      COMMON /BLOCK/ TWO
*FRAGMENT 3RD_BLOCK                       } G
c
c     this is code fragment 3 to be
c     inserted in the output.             } H
c
      COMMON /BLOCK/ THREE
*DECK                                     } I
CCCCCC
C     The main code comments.             } J
CCCCCC
*insert 1ST_BLOCK                         } K
*insert/vax 2ND_BLOCK                     } L
*insert/cray 3RD_BLOCK                    } M
      x=sub(1)
      stop                                } N
      end
*comments/upper                           } O
      function sub(x)                     } P
*insert 1ST_BLOCK                         } Q
*comments/delete                          } R
*insert/vax 2ND_BLOCK                     } S
*insert/cray 3RD_BLOCK                    } T
      sub = x
      return                              } U
      end
```

Figure 4. preFOR output of example shown in previous figure.

```
cccccc                                  }
c     The main code comments.           }  From J
cccccc                                  }
c                                       }
c     this is fragment 1 to be          }
c     inserted in the output.           }  From D
c                                       }
      COMMON /BLOCK/ ONE                }
c                                       }
c     this is fragment 2 to be          }
c     inserted in the output.           }  From F
c                                       }
      COMMON /BLOCK/ TWO                }
      x=sub(1)                          }
      stop                              }  From N
      end                               }
      function sub(x)                   }  From P
C                                       }
C     THIS IS FRAGMENT 1 TO BE          }
C     INSERTED IN THE OUTPUT.           }  From D
C                                       }
      COMMON /BLOCK/ ONE                }
      COMMON /BLOCK/ TWO                }  From F
      sub = x                           }
      return                            }  From U
      end                               }
```

For an explanation of the sections identified with a bold character see Table 2.

Table 2. Explanation of Character Codes in figures 3 & 4.

A -	These lines are for commenting purposes only and are not written to the output file.
B -	This command selects the CPU identifier which will permit the object of an insert command to be written to the output file.
C -	This command defines the beginning of the first code fragment.
D -	These lines are written to a temporary file. Nothing is written to the FORTRAN output file at this time.
E -	This command defines the end of the first fragment and the beginning of the second fragment.
F -	This is the body of the second code fragment.
G -	This command defines the end of the second fragment and the beginning of the third fragment.
H -	This is the body of the third code fragment.
I -	This command defines the end of the third fragment and the beginning of the main FORTRAN code.
J -	Initial comments of the main code.
K -	Retrieve the contents of the first fragment and write them to the FORTRAN output file. This insertion is done regardless of the CPU specified in the CPU command.
L -	Retrieve the contents of the second fragment if the CPU identifier is VAX.
M -	Retrieve the contents of the third fragment if the CPU identifier is CRAY.
N -	Executable section of the main program.
O -	This command converts all comment lines to upper case from this point on.

(continued)

Table 2. Explanation of Character Codes in figures 3 & 4. (continued)

P -	Initial line of a FORTRAN subprogram.
Q -	Retrieve the contents of the first fragment and convert any comments to uppercase as they are written to the FORTRAN output file.
R -	This command inhibits the writing of any comment lines to the FORTRAN output file.
S -	Retrieve the contents of the second fragment and write it to the FORTRAN output file if the current CPU identifier is VAX. However, no comment lines will be written to the FORTRAN output file.
T -	Retrieve the contents of the second fragment and write it to the FORTRAN output file if the current CPU identifier is CRAY. However, no comment lines will be written to the FORTRAN output file.
U -	Body of the subprogram which is written as shown to the FORTRAN output file.

20

APPENDIX: preFOR SOURCE CODE

```
      PROGRAM preFOR
CCCCCC
C                             ACKNOWLEDGMENT
C                             ==============
C
C     Paul Eslinger is acknowledged as the grandfather of preFOR by virtue
C     of his authorship of the precursor program called UPDATE.  Virtually
C     all of his original code remains in this version in various forms.
C
C     ====================================================================
C
C*    NAME:   preFOR
C*
C*    PURPOSE:
C*
C*    THE PURPOSE OF THIS PROGRAM IS TO PROCESS FORTRAN SOURCE
C*    FILES WHICH ARE WRITTEN IN UPDATE FORMAT AND WRITE A SOURCE
C*    FILE SUITABLE FOR SUBMITTING TO A FORTRAN COMPILER.
C     The main feature of preFOR is to allow a code fragment to be
C     placed automatically at various points in the file.  Other
C     features include line numbering, line truncation, CPU specific
C     fragment handling, FORTRAN comment line processing, and tab
C     handling.
C
C     When defining a code fragment or COMDECK no other preFOR commands
C     may be included within the fragment.  This implies, and correctly so,
C     that the preFOR commands are not recursive and cannot be nested.
C
C     USAGE:
C
C     This program is intended for interactive use.  Since most of the
C     default control parameters are set at 'safe' values, a standard
C     program file (i.e. one without preFOR commands), when processed
C     by preFOR and compiled, should give an object file that is identical
C     to the original compilation.  This point is not significant except for
C     users of UPDATE who would expect an error when submitting standard
C     FORTRAN program file as input.  preFOR will, under default
C     conditions, process a valid FORTRAN program without changing it.  No
C     practical use has been found for this feature however.  The only
C     commands that should change the resulting program object file are
C     *COMDECK, *FRAGMENT, *CALL, and *INSERT (and maybe INPUT/WIDTH=72).  A
C     test can be made of the behavior of all the other commands by using
C     them as desired, compile the code, and compare the resulting
C     object file to an original.  The object files should be the same.
C*
C*    CONVENTIONS:
C*
C*                          UPDATE/preFOR DIRECTIVE CARDS
C*     ---------------------------------------------------------------------
C*     *DECK name     DEFINES A DECK (PROGRAM, SUBROUTINE, FUNCTION,
C*                    ETC.). THE CURRENT DECK IDENTIFICATION CONTINUES
C*                    UNTIL A NEW UPDATE/preFOR DIRECTIVE CARD IS ENCOUNTERED.
C*     *COMDECK name  DEFINES A COMDECK (INSERT DECK) UNTIL A NEW
C*                    UPDATE/preFOR DIRECTIVE CARD IS ENCOUNTERED.
C*     *CALL name     SYNTAX TO CAUSE INSERTION OF THE COMDECK WITH
```

22

```
C*                      NAME name AT THE LOCATION OF THE *CALL CARD.
C       *COMMENTS/DELETE    Delete all comment lines from the output file
C                           before the lines are numbered.  Can be used
C                           to cancel the effect of the *COMMENTS/KEEP
C                           command.
C       *COMMENTS/KEEP      Does not delete comment lines from the output
C                           file.  Can be used many times in an input deck.
C                           (Default)
C       *COMMENTS/UPPER     Converts all comment lines to upper case.
C       *COMMENTS/NOUPPER   Negates the *COMMENTS/UPPER command. (Default)
C       *CPU cpu            Specifies the name of the CPU to honored during
C                           the processing of *INSERT/cpu commands.
C       *DECK/NUMBER name   Invokes numbering of the deck with 'name' and 4
C                           digit numbers in columns 73-80.  Only the first
C                           4 characters of 'name' are used.
C       *FRAGMENT name      Specifies the name of the code fragment which is
C                           copied from the input until the next preFOR
C                           command.  Alias for *COMDECK.
C       *INPUT/WIDTH=n      Limits the input to the first n columns, the
C                           remaining columns are ignored.  (n=[72,80])
C                           (Default=80)
C       *INSERT name        Inserts code fragment corresponding to the name
C                           provided.  Alias for *CALL.
C       *INSERT/cpu name    Inserts the named fragment only if 'cpu' matches
C                           the name in the *CPU cpu command.
C       *NOTE               This line is ignored by preFOR and is not output.
C                           A convenience for commenting the preFOR input.
C       *OUTPUT/TRIM        When line numbering is not requested this command
C                           removes trailing blanks and tabs before writing
C                           a line to the output file. (Default)
C       *OUTPUT/NOTRIM      Negates a previous *OUTPUT/TRIM command.
C       *OUTPUT/WIDTH=n     Limits the output width to columns 1-n.  Output
C                           is truncated at column n.  This command is ignored
C                           if *DECK/NUMBER is in effect.  (n=[72,80])
C                           (Default=80)
C       *TAB/KEEP          Keeps tabs as they appear in the input file. (Default)
C       *TAB/REPLACE        Replace tabs with blanks.  Tabs are assumed to be
C                           set at: 9, 17, 25, 33, 41, 49, 57, 65, 73.
C
C*      -----------------------------------------------------------
C       cpu   Denotes an 8 character or less alphanumeric identifier.
C       n     A positive integer, usually 2 digits.
C*      name  DENOTES AN 8 CHARACTER OR LESS ALPHABETIC IDENTIFIER
C*      -----------------------------------------------------------
C
C       Future commands could include blank line processing, comment
C       line custom styling, and string replacement.
C*
C*
C*
C*
C*   RESTRICTIONS:
C*
C*     1. A MAXIMUM OF MXCOM (=250) FRAGMENTS (COMDECKS) CAN BE DEFINED.
C*        THERE MUST BE ENOUGH DIRECTORY ENTRIES AVAILABLE ON THE USER'S DISK TO
C*        ALLOW A SCRATCH FILE FOR EACH FRAGMENT (COMDECK).  THE SCRATCH FILES
```

```
C*       ARE NAMED WITH THE CONVENTION THAT THE FILE FOR *COMDECK name
C*       and *FRAGMENT name IS CALLED name.UUU.
C*
C*    2. THE *COMDECK or *FRAGMENT CARD FOR THE *INSERT DECK name MUST
C        APPEAR IN THE INPUT FILE BEFORE IT IS REFERENCE BY A *CALL or
C        *INSERT CARD.
C*
C     EXTERNAL REFERENCES:
C
C     cname
C     strail
C     upcase
C     upstrg
C     writer
C*
C*  VARIABLE DEFINITIONS:
C*
C*    NAMEDK  : CHARACTER*8 NAME OF THE CURRENT DECK BEING PROCESSED
C*    NAMECM  : CHARACTER*8 NAME OF THE CURRENT COMDECK BEING PROCESSED
C*    LDNAME  : LENGTH OF THE DECK NAME (WITHOUT EXTENSION)
C*    LCNAME  : LENGTH OF THE COMDECK NAME (WITHOUT EXTENSION)
C     LCPUNM  : Length of the CPU name.
C*    MXCOM   : PARAMETER VALUE FOR MAXIMUM NUMBER OF COMDECKS ALLOWED
C*    DECKCD  : LOGICAL FLAG THAT IS FALSE UNTIL THE FIRST *DECK CARD
C*              IS ENCOUNTERED IN THE SOURCE FILE
C*    COMFIL  : CHARACTER*12 CURRENT NAME OF THE SCRATCH COMDECK FILE
C*    CFILES  : CHARACTER*12 ARRAY OF ALL THE NAMES FOR SCRATCH COMDECK
C*              FILES --- ALL THESE FILES WILL BE DELETED ---
C*    INFILE  : CHARACTER*64 NAME FOR SOURCE FILE
C*    OUTFIL  : CHARACTER*64 NAME FOR COMPILE FILE
C*    NUUU    : COUNTER FOR THE NUMBER OF *COMDECK DEFINITIONS
C*    NDDD    : COUNTER FOR THE NUMBER OF *DECK DEFINITIONS
C*    NUMDCK  : COUNTER FOR THE LINE NUMBER IN THE CURRENT DECK
C*    NWRITE  : COUNTER FOR TOTAL NUMBER OF OUTPUT LINES
C*    NCMWRT  : COUNTER FOR NUMBER OF OUTPUT COMMENT LINES
C*    NUMEXE  : COUNTER FOR NUMBER OF OUTPUT EXECUTABLE LINES
C*    NUMCOM  : COUNTER FOR THE LINE NUMBER ON THE CURRENT COMDECK
C*    CARD    : CHARACTER*80 INPUT CARD IMAGE
C*    LINE    : LOGICAL FLAG  .TRUE.  = WRITE LINE NUMBERS
C*                            .FALSE. = DON'T WRITE LINE NUMBERS
C     CPUSWI  : CHARACTER*8 cpu name provided in the *INSERT/cpu command.
C     CPUNAM  : CHARACTER*8 cpu name provided in the *CPU cpu command.  This
C               is to be matched with CPUSWI.
C     LCMKP   : LOGICAL KEEP COMMENTS FLAG.
C     LCMIND  : LOGICAL INDENT COMMENTS FLAG.
C     INDENT  : INTEGER CURRENT INDENT COLUMN.
C     LCMUP   : LOGICAL SHIFT  COMMENTS TO UPPER CASE FLAG.
C     INPWID  : INTEGER INPUT WIDTH IN NUMBER OF COLUMNS
C     OUTWID  : INTEGER OUTPUT WIDTH IN NUMBER OF COLUMNS.
C     LTRIM   : LOGICAL TRIM OUTPUT FLAG.
C     LTABKP  : LOGICAL KEEP TABS FLAG.
C     LTABIN  : LOGICAL INSERT TABS FLAG.
C     TABS    : INTEGER TAB SETTINGS, 9 MAX.
C*
```

```
C*  HISTORY:
C     ---------
C*    As UPDATE
C*    PAUL W. ESLINGER : 24 OCTOBER 1986 : VERSION 1.0
C*    PAUL W. ESLINGER :  1 JUNE    1987 : VERSION 1.1
C*                         FIX A BUG ON MAXIMUM NUMBER OF COMDECK
C*                         FILES THAT ARE ALLOWED.
C*    PAUL W. ESLINGER :  10 JUNE   1987 : VERSION 1.3
C*                         CHANGE MESSAGES TO THE SCREEN
C     ---------
C     As preFOR
C     Ron Janetzke     : 10-26-90 : Version 2.0
C     Ron Janetzke     : 04-22-91 : Version 2.1
C                        Clean up for distribution.
CCCCCC
      PARAMETER ( MXCOM = 250 )
C
      CHARACTER*80 UPSTRG
      CHARACTER*1  UPCASE
      CHARACTER*1  ANS, BLANK, TAB
      CHARACTER*8  cpunam, cpuswi
      CHARACTER*8  NAMEDK, NAMECM
      CHARACTER*12 COMFIL, CFILES(MXCOM)
      CHARACTER*64 INFILE, OUTFIL
      CHARACTER*80 CARD
C
      integer strail
      integer indent
      integer inpwid
      integer outwid
      integer tabs(9)
C
      LOGICAL THERE, DECKCD, LINE
      logical lcmkp
      logical lcmind
      logical lcmup
      logical ltabkp
      logical ltabin
      logical ltrim
C
      DATA BLANK  /' '/
      DATA DECKCD / .FALSE. /
      DATA CFILES / MXCOM * '            ' /
      data tabs   / 9, 17, 25, 33, 42, 49, 57, 65, 73/
CCCCCC
C     Start here.
CCCCCC
      TAB = CHAR(9)
      lcmkp = .true.
      lcmind = .false.
      lcmup  = .false.
      ltabkp = .true.
      ltabin = .false.
      ltrim  = .true.
      indent = 7
      inpwid = 80
      outwid = 80
      line = .false.
      NWRITE = 0
      NCMWRT = 0
      NDDD = 0
C
C *** GET THE INPUT FILE NAME
C
      WRITE(*,100)
  100 FORMAT(
     * ' preFOR 2.0    26 October 1990     Ron Janetzke'/
     * ' CONVERT A preFOR FILE TO A FORTRAN COMPILE FILE'//
     * ' ENTER THE UPDATE/preFOR SOURCE FILE NAME'/
     * ' PATH NAMES UP TO 64 CHARACTERS ARE ALLOWED >> ')
      READ(*,110) INFILE
  110 FORMAT(A64)
C
C     Add the extension .PRE to the input file name if no extension was
C     given on input.
C
      if (index(infile,'.') .eq. 0) then
         infile (len(infile)-strail(infile)+1:) = '.PRE'
      end if
C
C *** CHECK TO SEE IF THE INPUT FILE EXISTS
C *** OPEN THE FILE IF IT EXISTS
C
      INQUIRE(FILE=INFILE,EXIST=THERE)
      IF( THERE ) THEN
        OPEN(7,FILE=INFILE,STATUS='OLD')
      ELSE
        WRITE(*,120) INFILE
  120   FORMAT(' THE REQUESTED UPDATE/preFOR SOURCE FILE DOES NOT',
     &         ' EXIST'/
     *         ' FILE = ',A64/
     *         ' CHECK YOUR DIRECTORY AGAIN')
        STOP
      ENDIF
C
      WRITE(*,130)
  130 FORMAT(' ENTER OUTPUT (COMPILE) FILE NAME'/
     * ' PATH NAMES UP TO 64 CHARACTERS ARE ALLOWED >> ')
      READ(*,110) OUTFIL
CCCCCC
C     Add output file suffix if it was omitted from the input.
CCCCCC
      if (index(outfil,'.') .eq. 0) then
         outfil (len(outfil)-strail(outfil)+1:) = '.FOR'
      end if
C
C *** CHECK FOR THE SAME NAME
C
      infile = upstrg (infile)
      outfil = upstrg (outfil)
      IF( INFILE .EQ. OUTFIL ) THEN
        WRITE(*,140)
  140   FORMAT(/' THE INPUT AND COMPILE FILE NAMES WERE IDENTICAL'/
     *          ' ABORTING preFOR ...')
        STOP
      ENDIF
C
C *** OPEN THE COMPILE FILE IF IT DOES NOT EXIST
C *** IF THE FILE ALREADY EXISTS, MAKE THE USER DELETE IT
C
      INQUIRE(FILE=OUTFIL,EXIST=THERE)
C
      IF( THERE ) THEN
        WRITE(*,150) OUTFIL
  150   FORMAT(' THE REQUESTED COMPILE FILE ALREADY EXISTS'/,/
     *         ' FILE = ',A64/,/
     *         ' DO YOU WISH TO DELETE IT ?')
        READ(*,160) ANS
  160   FORMAT(A1)
        IF( ANS.EQ.'Y' .OR. ANS.EQ.'y' ) THEN
          OPEN(8,FILE=OUTFIL,STATUS='OLD')
          CLOSE(8,STATUS='DELETE')
          OPEN(8,FILE=OUTFIL,STATUS='NEW')
        ELSE
          WRITE(*,165)
  165     FORMAT(/' STOPPING preFOR ...')
          STOP
        ENDIF
      ELSE
        OPEN(8,FILE=OUTFIL,STATUS='NEW')
      ENDIF
C
C *** WRITE ONE BLANK LINE TO THE SCREEN
C
      WRITE(*,166)
  166 FORMAT(/)
C
C *** TOP OF LOOP ON READING CARDS FROM THE SOURCE FILE
C
      NUUU = 0
   10 CONTINUE
C
      READ(7,500,END=60) CARD
  500 FORMAT(A80)
C
   20 CONTINUE
CCCCCC
C     Check for NOTE command.  (No action)
CCCCCC
      IF (CARD(1:5).EQ.'*NOTE' .OR. CARD(1:5).EQ.'*note') go to 10
CCCCCC
C *** CHECK FOR ACTION ON AN UPDATE/preFOR DIRECTIVE
C
C     We can remove the column position restriction for the DECK name imposed
C     by UPDATE if we search for the first non-blank character and pick up
C     the name from that point for the next 8 contiguous non-blank characters.
C
C     NOTE: The last four characters of NAMEDK are not used by preFOR.  The
```

```
C          main purpose of supplying a name via the DECK command is line
C          identification in columns 73-76 with numbers in columns 77-80.
CCCCCC
C     Check for DECK command.
CCCCCC
      IF ( CARD(1:5).EQ.'*DECK' .OR. CARD(1:5).EQ.'*deck') then
       if (CARD(6:6) .NE. '/' ) THEN
        line = .false.
        LDNAME = 0
CCCCCC
C       Build deck name.
CCCCCC
        DO 5 IDN = 7,80
          IF( CARD(IDN:IDN) .NE. BLANK .AND. CARD(IDN:IDN) .NE. TAB
     &        .and. LDNAME .LT. 8) THEN
            LDNAME = LDNAME + 1
            NAMEDK(LDNAME:LDNAME) = CARD(IDN:IDN)
          ELSE IF (LDNAME .GT. 0) THEN
C           Stop building the DECK name at the first blank character.
            GO TO 6
          ENDIF
    5   CONTINUE
CCCCCC
C      Check for DECK/NUMBER command.
CCCCCC
       else if (card(6:7) .eq. '/N' .or. card(6:7) .eq. '/n') then
        line = .true.
C       Skip any trailing characters on the /NUMBER switch. (i.e. 'umber')
        do 7 idn = 8,13
         nstart = idn
           if (card(idn:idn) .eq. blank .or. card(idn:idn) .eq. tab)
     &        then
              go to 8
           end if
    7   continue
CCCCCC
C       Build name for DECK.
CCCCCC
    8   continue
        LDNAME = 0
        DO 9 IDN = nstart,80
          IF( CARD(IDN:IDN) .NE. BLANK .AND. CARD(IDN:IDN) .NE. TAB
     &        .and. LDNAME .LT. 8) THEN
            LDNAME = LDNAME + 1
            NAMEDK(LDNAME:LDNAME) = CARD(IDN:IDN)
          ELSE IF (LDNAME .GT. 0) THEN
C           Stop building the DECK name at the first blank character.
            GO TO 6
          ENDIF
    9   CONTINUE
       END IF

    6   CONTINUE
        DECKCD = .TRUE.
        NUMDCK = 0
        NDDD = NDDD + 1
```

```
cccccc
c       Notify user of current deck name.
cccccc
        WRITE(6,665)
  665   FORMAT('+ PROCESSING DECK:         ')
        WRITE(6,666) (NAMEDK(II:II),II=1,LDNAME)
  666   FORMAT('+ PROCESSING DECK: ',7A1)
        GO TO 10
      ENDIF
cccccc
c     Check for *COMMENT commands.
cccccc
      if (card(1:10) .eq. '*COMMENTS/' .or.
     &    card(1:10) .eq. '*comments/')     then
         card = upstrg (card)
c        Test for KEEP.
         if (card(11:11) .eq. 'K') lcmkp  = .true.
c        Test for DELETE.
         if (card(11:11) .eq. 'D') lcmkp  = .false.
c        Test for INDENT. (Not supported at this time.)
         if (card(11:11) .eq. 'I') lcmind = .true.
c        Test for UPPERcase.
         if (card(11:11) .eq. 'U') lcmup  = .true.
c        Test for NOUPPERcase.
         if (card(11:13) .eq. 'NOU') lcmup  = .false.
      go to 10
      end if
cccccc
c     Check for *INPUT/WIDTH=n command.  (72 or 80 are the only practical
c     values for 'inpwid'.)
cccccc
      if (card(1:12) .eq. '*INPUT/WIDTH' .or.
     &    card(1:12) .eq. '*input/width') then
         read (card,'(13x,i2)') inpwid
         inpwid = max(min(inpwid,80),72)
         go to 10
      end if
cccccc
c     Check for *OUTPUT/WIDTH=n command.  (72 or 80 are the only practical
c     values for 'outwid'.)
cccccc
      if (card(1:13) .eq. '*OUTPUT/WIDTH' .or.
     &    card(1:13) .eq. '*output/width') then
         read (card,'(14x,i2)') outwid
         outwid = max(min(outwid,80),72)
         go to 10
      end if
cccccc
c     Check for *OUTPUT/TRIM command.
cccccc
      if (card(1:12) .eq. '*OUTPUT/TRIM' .or.
     &    card(1:12) .eq. '*output/trim') then
         ltrim = .true.
         go to 10
      end if
cccccc
```

```
C     Check for *OUTPUT/NOTRIM command.
CCCCCC
      if (card(1:12) .eq. '*OUTPUT/NOTR' .or.
     &    card(1:12) .eq. '*output/notr') then
         ltrim = .false.
         go to 10
      end if
CCCCCC
C     Check for *TAB/ commands.
CCCCCC
      if (card(1:5) .eq. '*TAB/' .or.
     &    card(1:5) .eq. '*tab/')     then
         card = upstrg (card)
C        Test for KEEP.
         if (card(6:6) .eq. 'K') ltabkp  = .true.
C        Test for REPLACE. (Not supported at this time.)
         if (card(6:6) .eq. 'R') ltabkp  = .false.
C        Test for INSERT. (Not supported at this time.)
         if (card(6:6) .eq. 'I') ltabin  = .true.
      go to 10
      end if
CCCCCC
C     Check for FRAGMENT command.
CCCCCC
      IF( CARD(1:8).EQ.'*COMDECK' .OR. CARD(1:8).EQ.'*comdeck' .or.
     &    CARD(1:9).EQ.'*FRAGMENT'.OR. CARD(1:9).EQ.'*fragment' ) THEN
        NUUU = NUUU + 1
        IF( NUUU .GT. MXCOM ) THEN
          WRITE(*,170) mxcom
  170     FORMAT(/' ONLY',i4,' *fragment or *COMDECK CARDS ARE',
     &              ' ALLOWED --- STOP'/
     &              ' Check MXCOM parameter in the preFOR program.')
          STOP
        ENDIF
C
C ***   OPEN THE COMFILE
C
        CALL CNAME( CARD, COMFIL, LCNAME )
        CFILES(NUUU) = COMFIL
        OPEN(9,FILE=COMFIL,STATUS='NEW')
        WRITE(*,200) COMFIL
  200   FORMAT('+WRITING FILE: ',A12)
C
C ***   READ THE COMDECK AND WRITE TO THE TEMPORARY FILE
C
   30   CONTINUE
        READ(7,500) CARD
C
C ***   CHECK IF THE INPUT CARD CONTAINS AN UPDATE/preFOR DIRECTIVE
C
        IF( CARD(1:1) .EQ. '*' ) THEN
          CLOSE(9)
          GO TO 20
        ELSE
          WRITE(9,500) CARD
          GO TO 30
        ENDIF
C
      ENDIF
CCCCCC
C     Check for CALL or INSERT command.
CCCCCC
      IF( CARD(1:5).EQ.'*CALL'   .OR. CARD(1:5).EQ.'*call' .or.
     &    CARD(1:7).EQ.'*INSERT' .OR. CARD(1:7).EQ.'*insert' ) THEN
CCCCCC
C     Check for INSERT/cpu command.
CCCCCC
      If (card(8:8) .eq. '/' .and. card(9:9) .ne. blank
     &   .and. card(9:9) .ne. tab) then
         lcpusw = 0
         cpuswi = blank
         do 29 i=9,16
            if (card(I:I) .eq. blank .or. card(I:I) .eq. tab) then
               go to 35
            else
               lcpusw = lcpusw + 1
               cpuswi(lcpusw:lcpusw) = upcase(card(I:I))
            end if
   29    continue
C        End of do.
   35    continue
CCCCCC
C        If cpu name does not match, then skip this insert request.
CCCCCC
         if(cpuswi .ne. cpunam) go to 10
        end if
C
C ***   OPEN THE COMFILE
C
        CALL CNAME( CARD, COMFIL, LCNAME )
        NAMECM = COMFIL(1:LCNAME)
        OPEN(9,FILE=COMFIL,STATUS='OLD')
C
C ***   READ THE COMDECK AND WRITE TO THE COMPILE FILE
C
        NUMCOM = 0
CCCCCC
C       If nddd = 0 then no DECK commands have been found to this point.
C       So set nddd = 1 to keep the counter correct as the counter is only
C       updated when processing an explicit DECK command.
CCCCCC
        if (nddd .eq. 0) nddd = 1
   40   CONTINUE
CCCCCC  *************************
C       READ and WRITE loop.
CCCCCC  *************************
        READ(9,500,END=50) CARD
        NUMCOM = NUMCOM + 1
        CALL WRITER( CARD, NAMECM, LCNAME, NUMCOM, LINE, lcmkp, lcmind,
     &                 indent, lcmup, inpwid, outwid, ltrim,
     &                 ltabkp,ltabin,tabs)
        NWRITE = NWRITE + 1
```

```
C       WRITE(*,190) NAMECM(1:LCNAME), NUMCOM, NWRITE
  190   FORMAT('+',A8,1X,I4,' LINES',2X,I5,' TOTAL LINES')
CCCCCC
C       Only increment the comment line counter if COMMENT/KEEP is enabled.
CCCCCC
        IF ((CARD(1:1).EQ.'C' .OR. CARD(1:1).EQ.'c') .and. lcmkp)
     &      NCMWRT = NCMWRT+1
        GO TO 40
CCCCCC  ***************************
C       End of READ and WRITE loop.
CCCCCC  ***************************
C
C ***   CLOSE THE COMFILE
C
   50   CONTINUE
        CLOSE(9)
        GO TO 10
      ENDIF
CCCCCC
C     End of INSERT or CALL processing.
CCCCCC
C     Check for CPU command.
CCCCCC
      if (card(1:5) .eq. '*CPU ' .or. card(1:5) .eq. '*cpu ') then
C        Find the cpu name.
         lcpunm = 0
         cpunam = blank
         DO 39 I = 5,80
           IF (lcpunm .le. 0) then
             if (card(I:I) .eq. blank .or. card(I:I) .eq. tab) then
                GO TO 39
             else
                lcpunm = lcpunm + 1
                cpunam(lcpunm:lcpunm) = upcase(card(I:I))
                if (lcpunm .ge. 8) go to 55
             end if
           ELSE
             if (card(I:I) .eq. blank .or. card(I:I) .eq. tab) then
                GO TO 55
             else
                lcpunm = lcpunm + 1
                cpunam(lcpunm:lcpunm) = upcase(card(I:I))
                if (lcname .ge. 8) go to 55
             end if
          END IF
   39   CONTINUE
C
   55  CONTINUE
       go to 10
      end if
CCCCCC
C     End of CPU command processing.
CCCCCC
C
C *** WRITE THE CARD IMAGE TO THE COMPILE FILE
C
```

```
cccccc
c       If nddd = 0 then no DECK commands have been found to this point.
c       So set nddd = 1 to keep the counter correct as the counter is only
c       updated when processing an explicit DECK command.
cccccc
        if (nddd .eq. 0) nddd = 1
        NUMDCK = NUMDCK + 1
C        Line removed.
C        CALL WRITER( CARD, NAMEDK, LDNAME, NUMDCK, LINE )
        CALL WRITER( CARD, NAMEDK, LDNAME, NUMDCK, LINE, lcmkp, lcmind,
     &               indent, lcmup, inpwid, outwid, ltrim,
     &               ltabkp,ltabin,tabs)
        NWRITE = NWRITE + 1
C       WRITE(*,190) NAMEDK(1:LDNAME), NUMDCK, NWRITE
        IF ((CARD(1:1).EQ.'C' .OR. CARD(1:1).EQ.'c') .and. lcmkp)
     &       NCMWRT = NCMWRT+1
        GO TO 10
C
C *** END OF FILE
CC
   60 CONTINUE
C
C *** CLEAN UP THE *COMDECK SCRATCH FILES
C
      DO 70 I = 1, MXCOM
        IF( CFILES(I) .NE. '            ' ) THEN
          WRITE(*,210) CFILES(I)
  210     FORMAT('+DELETING FILE: ',A12)
          OPEN(9,FILE=CFILES(I),STATUS='OLD')
          CLOSE(9,STATUS='DELETE')
        ENDIF
   70 CONTINUE
C
C *** FINAL MESSAGES
C
      NUMEXE = NWRITE - NCMWRT
      WRITE(*,220) NDDD, NUUU, NUMEXE, NCMWRT, NWRITE
  220 FORMAT('+                                              '/
     *       ' ',I3,' DECKS PROCESSED'/
     *       ' ',I3,' INSERT DECKS PROCESSED'/
     *       ' ',I5,' EXECUTABLE LINES IN THE COMPILE FILE'/
     *       ' ',I5,' COMMENT LINES IN THE COMPILE FILE'/
     *       ' ',I5,' TOTAL LINES IN THE COMPILE FILE')
C
      STOP
      END
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
      SUBROUTINE CNAME( CARD, COMFIL, LCNAME )
CCCCCC
C     NAME:   CNAME
C*
C*    PURPOSE:
C*
C*    THIS SUBROUTINE WILL GET THE COMDECK FILE NAME FROM THE CARD
C*    IMAGE AND STORE IT IN THE VARIABLE COMFIL
C*
```

```
C*
C*    VARIABLE DEFINITIONS:
C*
C*    CARD     : CHARACTER*80 INPUT CARD IMAGE.  THIS IMAGE SHOULD
C*               START EITHER WITH *COMDECK OR *CALL.  THE LOGIC IN
C*               THE CALLING ROUTINE SHOULD TAKE CARE OF THIS.
C*    COMFIL   : CHARACTER*12 FILE NAME FOR THE COMDECK
C*    LCNAME   : LENGTH OF THE COMDECK NAME (WITHOUT EXTENSION)
C*
C*
C*    CONVENTION:
C*
C*    IF THE CARD IMAGE STARTS WITH *COMDECK name OR *DECK name, THE
C*    VARIABLE COMFIL WILL CONTAIN 'name.UUU'.  IF name IS LONGER
C*    THAN 8 CHARACTERS IT WILL BE TRUNCATED TO 8 CHARACTERS.
C*
CCCCCCC
      CHARACTER*1 blank
      CHARACTER*1 tab
      CHARACTER*12 COMFIL
      CHARACTER*80 CARD
CCCCCC
C     Start here.
CCCCCC
      blank = ' '
      tab   = char(9)
C
C *** FIRST BLANK THE COMDECK FILE NAME
C
      COMFIL = '            '
C
C *** IF THE CARD IMAGE STARTS WITH *COMDECK, *DECK or *FRAGMENT, EXTRACT THE
C *** NAME AND PUT IT IN THE FIRST PART OF COMFIL.  A MAXIMUM OF 8
C *** CHARACTERS ARE ALLOWED, AND TRAILING BLANKS ARE REMOVED.
C *** EMBEDDED BLANKS ARE TREATED AS TERMINATING THE NAME.
C
C     NOTE:  preFOR only uses the first 4 characters for line numbers.
C
C *** IOFF = OFFSET IN NUMBER OF CHARACTERS TO REACH THE START
C *** OF THE COMDECK NAME.
C
      IF( CARD(1:8).EQ.'*COMDECK'  .OR. CARD(1:8).EQ.'*comdeck' ) IOFF=9
      IF( CARD(1:9).EQ.'*FRAGMENT' .OR. CARD(1:9).EQ.'*fragment')IOFF=10
      IF( CARD(1:5).EQ.'*CALL'     .OR. CARD(1:5).EQ.'*call'    ) IOFF=6
      IF( CARD(1:8).EQ.'*INSERT '  .OR. CARD(1:8).EQ.'*insert ' ) IOFF=9
      IF( CARD(1:8).EQ.'*INSERT/'  .OR. CARD(1:8).EQ.'*insert/' ) then
CCCCCC
C     Find the end of the *insert/x... command.
CCCCCC
         do 8 i=9,16
           ioff = i
           if (card(I:I) .eq. blank .or. card(I:I) .eq. tab) go to 9
    8    continue
      end if
    9 continue
C
```

```
C     Allow the file name to start in any column after IOFF.
C
      lcname = 0
      DO 10 I = IOFF, 80
        IF (lcname .le. 0) then
           if (card(I:I) .eq. blank .or. card(I:I) .eq. tab) then
              GO TO 10
           else
              lcname = lcname + 1
              COMFIL(lcname:lcname)  = CARD(I:I)
              if (lcname .ge. 8) go to 20
           end if
        ELSE
           if (card(I:I) .eq. blank .or. card(I:I) .eq. tab) then
              GO TO 20
           else
              lcname = lcname + 1
              COMFIL(lcname:lcname)  = CARD(I:I)
              if (lcname .ge. 8) go to 20
           end if
        END IF
   10 CONTINUE
C
   20 CONTINUE
C
C *** PUT ON THE SCRATCH FILE EXTENSION
C
      I = LCNAME + 1
      J = I + 3
      COMFIL(I:J) = '.UUU'
C
      RETURN
      END
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
      FUNCTION strail (c)
cccccc
c     PURPOSE:  strail finds the number of trailing blanks and tabs
c               in the string c.
c
c     ARGUMENTS:
c
c     c = CHARACTER*(*) input string to be analyzed.
c
c     HISTORY:
c           1.00   11-02-90   Ron Janetzke  Original text.
cccccc
      character*(*) c
      character*1   blank
      character*1   tab
c
      integer stend
      integer i
      integer length
      integer strail
c
      data blank /' '/
```

```
cccccc
c     Start here.
cccccc
      tab    = char(9)
      length = len(c)
      stend  = 0
cccccc
c     If the string length is 0, skip processing and return.
cccccc
      if (length .le. 0) go to 999
c
      do 199 i=length,0,-1
         stend = i
         if (c(i:i) .ne. blank .and. c(i:i) .ne. tab) go to 999
  199 continue
c
  999 continue
      strail = length - stend
      return
      end
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
      SUBROUTINE tabfix (card, tabs)
cccccc
c     PURPOSE:  tabfix removes the tabs from string card and replaces them
c               with spaces as defined by the array of tab stops given in
c               'tabs'.
c
c     ARGUMENTS:
c
c     input:
c          card = CHARACTER*(*) string to be modified.
c          tabs = INTEGER array of tab stop values
c
c     output:
c          card = CHARACTER*(*) string with tabs removed.
c
c     HISTORY:
c            1.00   11-02-90   Ron Janetzke  Original text.
cccccc
      character*(*) card
      integer tabs(9)
c
      integer cend
      integer pos
      integer i
      integer j
      integer nblank
c
      character*1  tab
      character*80 blanks
c
      data blanks /' '/
cccccc
c     Start here.
cccccc
c     Look for a tab in CARD.  If no tabs are found then 'return'.
```

```
cccccc
      tab = char(9)
      if (index(card, tab) .eq. 0) return
c
      cend = len(card)
      pos = 0
cccccc
c     Loop through characters to end of card.
cccccc
      do 299 i=1,cend
c        Do not go past column 80.
         pos = min(80,pos + 1)
c        Look for tabs.
         if (card(pos:pos) .eq. tab) then
            do 199 j=1,9
               if (pos .lt. tabs(j)) then
                  nblank = tabs(J) - pos
                  go to 200
               end if
  199       continue
cccccc      End of do.
  200       continue
cccccc      Insert from 1 to 8 blanks into 'card'.
            if (pos .gt. 1) then
               card = card(1:pos-1)//blanks(1:nblank)//card(pos+1:cend)
            else
               card = blanks(1:nblank) // card(pos+1:cend)
            end if
            pos = pos - 1 + nblank
         end if
  299 continue
      return
      end
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
        FUNCTION upcase (c)
cccccc
c       NAME: upcase
c
c       PURPOSE: Change the case of any lower case alphabetic character to
c                upper case.
c
c       ARGUMENTS:
c               input:
c               c = CHARACTER*1 to be used for possible upshift.
c
c               output:
c               upcase = CHARACTER*1 return value.
c
c       EXAMPLE 1:
c               c = 'a'
c               upcase = 'A'
c
c       REFERENCES:
c               none
c
c       CHANGES:
```

```
C               1.00    08-25-90        Ron Janetzke
C                       Original text.
CCCCCC
        CHARACTER*1 upcase
        CHARACTER*1 c
CCCCCC
C       Start here.
CCCCCC
        if (ichar(c).ge.ichar('a') .and. ichar(c).le.ichar('z')) then
           upcase = char(ichar(c)-32)
        else
           upcase = c
        end if
C
        return
        end
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
      FUNCTION upstrg (c)
CCCCCC
C     PURPOSE:  upstrg converts the input string to uppercase.
C
C     ARGUMENTS:
C
C     c = input, CHARACTER*(*) string to be converted.
C
C     EXTERNAL REFERENCES:
C
C     upcase
C
C     HISTORY:
C           1.00   11-02-90   Ron Janetzke  Original text.
CCCCCC
      character*(*) upstrg
      character*(*) c
      character*1   upcase
C
      integer i
      integer length
C
CCCCCC
C     Start here.
CCCCCC
      length = len(c)
      upstrg = c
CCCCCC
C     Skip processing if string length is 0.
CCCCCC
      if (length .le. 0) go to 999
C
      do 199 i=1,length
         upstrg (i:i) = upcase(c(i:i))
  199 continue
C
  999 continue
      return
      end
```

```
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
      SUBROUTINE WRITER( CARD, NAME, LCNAME, NUM, LINE, lcmkp, lcmind,
     &                   indent, lcmup, inpwid, outwid, ltrim,
     &                   ltabkp, ltabin, tabs)
cccccc
c     NAME:
c
c     writer
c
C*    PURPOSE:
C*
C*    THIS SUBROUTINE PRINTS THE CURRENT CARD IMAGE TO THE OUTPUT
C*    FILE.  THE CARD IMAGE IS PRINTED EITHER WITH OR WITHOUT DECK
C*    IDENTIFIERS AND LINE NUMBERS.
C*
c     NOTE: Indent processing is not implemented, but the variables
c           exist for future enhancement.
C*
C*    Argument DEFINITIONS:
C*
C*    CARD    : CHARACTER*80 CARD IMAGE
C*    NAME    : CHARACTER*8 DECK IDENTIFIER, only the first 4 are used.
C*    NUM     : INTEGER LINE NUMBER COUNTER IN THE DECK OR COMDECK
C*    LCNAME  : LENGTH OF THE COMDECK NAME (WITHOUT EXTENSION)
C*    LINE    : LOGICAL FLAG
C*              .TRUE.  = PRINT DECK IDENTIFIER AND LINE NUMBERS
C*                        WITH THE CARD IMAGE
C*              .FALSE. = PRINT ONLY THE CARD IMAGE
c    lcmkp    : LOGICAL keep comments flag.
c    lcmind   : LOGICAL indent comments flag.
c    indent   : INTEGER column number for indents.
c    lcmup    : LOGICAL shift comments to upper case flag.
c    inpwid   : INTEGER number of columns to accept as input width.
c    outwid   : INTEGER maximum number of columns to output.
c    ltrim    : LOGICAL trim output flag.
c    ltabkp   : LOGICAL keep tab characters flag.
c    ltabin   : LOGICAL insert tabs flag.
c    tabs(9)  : INTEGER tab setting array.
c    blank    : CHARACTER*1 storage area for blank.
c
c     EXTERNAL REFERENCES:
c
c     upcase
c     upstrg
c     strail
c     tabfix
c
c     HISTORY:
c           1.00   11-02-90   Ron Janetzke  Original text.
cccccc
      character*1  blank
      character*1  upcase
      CHARACTER*8  NAME
      CHARACTER*80 CARD
      CHARACTER*80 upstrg
c
```

```
      integer crdlen
      integer strail
      integer indent
      integer inpwid
      integer num
      integer outwid
      integer tabs(9)
c
      logical lcmind
      logical lcmkp
      logical lcmup
      logical lcomnt
      logical ltabrp
      logical ltabkp
      logical ltabin
      logical ltrim
      LOGICAL LINE
c
      data blank /' '/
cccccc
c     Start here.
cccccc
c     Set card length to its trimmed length or else its actual length.
cccccc
      if (ltrim) then
         crdlen = len(card) - strail(card)
      else
         crdlen = len(card)
      end if
cccccc
c     This will cause columns 73-80 to be skipped if inpwid=72.
cccccc
      if (crdlen .gt. inpwid) then
         card = card(1:inpwid)
         crdlen = inpwid
      end if
cccccc
c     Process comments.
cccccc
      if (card(1:1) .eq. 'C' .or. card(1:1) .eq. 'c') then
c        Check flag to keep comments.
         if (lcmkp) then
c           Check for upper case request.
            if (lcmup) then
               card = upstrg(card)
            end if
cccccc
c           Possible code for indenting comments.
c
c            if (lcmind) then
c               call cindnt (card, indent)
c            end if
cccccc
         else
cccccc
c           Skip card and remove comment card count from deck number.
```

```
cccccc
              num = num - 1
              go to 999
           end if
        end if
cccccc
c       Process tabs.
cccccc
        if (.not. ltabkp) then
           call tabfix (card, tabs)
        end if
cccccc
c       Possible code for inserting tabs.
c
c        if (ltabin) then
c           call tabadd (card)
c        end if
cccccc
        IF( LINE ) THEN
C
C ***     WRITE THE CARD IMAGE AND APPEND THE DECK IDENTIFIER AND
C ***     LINE NUMBERS IN COLUMNS 73 THROUGH 82
C
C         Period fill names shorter than 4 characters.
          if (lcname .eq. 1) name(2:4) = '...'
          if (lcname .eq. 2) name(3:4) = '..'
          if (lcname .eq. 3) name(4:4) = '.'
C
CCCCCC    *********************************
C         Write card to output file.
CCCCCC    *********************************
C
          WRITE(8,801) CARD(1:72), NAME(1:4), NUM
  801     FORMAT(A72,A4,I4.4)
C
          if (crdlen .gt. 72) then
C            We will need to wrap this card to the nest one.
             num = num + 1
             if (card(1:1) .ne. 'C' .and. card(1:1) .ne. 'c') then
C               Wrap non-comment.
                write (8,802) (card(L:L),L=73,crdlen),
     &                        (blank,L=crdlen-72,65),
     &                         name(1:4), num
  802           format ('     &',66a1,a4,i4.4)
             else
C               Wrap comment.
                write (8,803) (card(L:L),L=73,crdlen),
     &                        (blank,L=crdlen-72,65),
     &                         name(1:4), num
  803           format ('C     ',66a1,a4,i4.4)
             end if
          end if
          RETURN
CCCCCC
C         Check for card exceeding output width limit.
CCCCCC
```

4/41

```
      else if (crdlen .gt. outwid) then
C          This should only be true if outwid<80, so we will need to
C          wrap these lines also, if it is true.
           write (8,810) (card(L:L),L=1,outwid)
           num = num + 1
C          Wrap comment.
           if (card(1:1) .eq. 'C' .or. card(1:1) .eq. 'c') then
              write (8,808) (card(L:L),L=outwid+1,crdlen)
  808         format ('C     ',66a1)
           else
C             Wrap non-comment.
              write (8,809) (card(L:L),L=outwid+1,crdlen)
  809         format ('     &',66a1)
           end if
      ELSE
C
C ***   PRINT WITHOUT ADDING THE DECK IDENTIFIER AND LINE NUMBERS.
C
        if (ltrim) then
           WRITE(8,810) (CARD(L:L),L=1,crdlen)
        else
           if (outwid .eq. 80) write (8,'(80a)') card(1:80)
           if (outwid .eq. 72) write (8,'(72a)') card(1:72)
        end if
  810   FORMAT(80A1)
C
      ENDIF
C
  999 continue
      RETURN
      END
```