1/1

CNWRA INFORMATION PROCESSING STANDARD SOFTWARE SUMMARY

01. Summary Date	02. Summary prepared by (Name and phone)	03. Summary action
Yr. 92 Mo. 01 Day 08		New [x] Replacement [ ] Deletion [ ]
04. Software Date: Yr. Mo. Day	03. Software title: RDFREE; FORTRAN Free Format Read Utility	Previous Internal Software ID

06. Short title: RDFREE; ~~FORTRAN Free Format REad Utility~~ LS 1/9/91

07. Internal Software ID:

08. Software type	09. Processing Mode	10. APPLICATION AREA General		Specific
[ ] Automated Data System	[ ] Interactive	[x] Computer Systems Support/Utility	[ ] Management/Business	I/O Library
[x] Computer Program	[x] Batch	[ ] Scientific/Engineering	[ ] Process Control	
[ ] Subroutine/Module	[ ] Combination	[ ] Bibliographic/Textual	[ ] Other	

11. Submitting organization and address: SwRI

12. Technical contact(s) and phone: Ron Janetzke 512-522-3318

13. Narrative

The RDFREE utility is designed for creating a user friendly data input mechanism for FORTRAN programs. It consists of a suite of subroutines written in ANSI standard FORTRAN 77. These subroutines are designed for reading alphanumeric data without the usual format constraints that are inherent to FORTRAN. That is, the RDFREE utility can be used to read data without specifying any format type (e.g., I, E, F, A, or G formats) and generally (with some exceptions) without constraints of sequence. The primary motivation for developing RDFREE at the Center for Nuclear Waste Regulatory Analyses (CNWRA) was to aid in the development of user friendly computer codes for performance assessment of high-level nuclear waste repository. The basis of RDFREE is to associate alphanumeric data with appropriate keywords. The keywords and the data associated with each keyword are selected by the developer of the FORTRAN application program. The keywords may be partitioned into sets, each set being valid for a different part of the applications program.

14. Keywords: Input, Parsing, Data Conversion

15. Computer manufacturer and model	16. Computer operating system	17. Programming language(s)	18. Number of source program statements
DEC/VAX	VMS	FORTRAN 77	2433
19. Computer memory requirements	**20. Tape drives**	**21. Disk/Drum units**	**22. Terminals**
Virtual	0	0	0

23. Other operational requirements: Requires a MAIN Program driver.

24. Software availability			25. Documentation availability		
Available	Limited	In-house only	Available	Inadequate	In-house only
[x]	[ ]	[ ]	[x]	[ ]	[ ]

26. FOR SUBMITTING ORGANIZATION USE

NOTE: This was used as A subroutine To TPA Phase 2.
GS 12/31/90

CNWRA Form TOP-4 10/90

```
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
        subroutine rdfree (lenmst, master)
CCCCCC
C       NAME: rdfree
C
C       PURPOSE: This module is part of a suite of routines used to analyze
C                a free form input stream in order to generate a consistent
C                array of modifier names and associated data values.
C
C       ARGUMENTS:
C               input:
C               lenmst = INTEGER maximum number of master keywords.
C               master = CHARACTER*6 array of master keywords.
C
C       REFERENCES:
C               exist
C               rdtok
C               type
C               rdfcom
C
C       CHANGES:
C               1.00    08-28-90        Ron Janetzke
C                       Original text.
C               1.01    09-21-90        Ron Janetzke
C                       RDERR added to the argument lists of RDFREE and
C                       RDTOK.
C               1.02    02-12-91        Ron Janetzke
C                       Reflect input on iunit to ounit.
C                       Stop instead of 'go to 999' on all errors.
C               2.00    04-26-91        Ron Janetzke
C                       Pass parameters by common rather than argument list.
CCCCCC
        INTEGER lenmst
        CHARACTER*6  master(lenmst)
CCCCCC
C     RDFREE PARAMETERS
C           Use the following parameters to set up and dimension the
C           work space for the RDFREE routines.
C
C           maxdvl = INTEGER maximum number of numeric values allowed.
C           maxmod = INTEGER maximum number of modifiers allowed.
C           maxsvl = INTEGER maximum number of string values allowed.
C           maxtok = INTEGER maximum number of tokens allowed.  The
C                    dimension for 'token' which is the work array for
C                    subroutine RDTOK.  It should be a minimum of
C                    (maxmod + maxdvl + 2*maxsvl + (the number of noise
C                    words in a line)).  Each noise word is stored as
C                    a token in the work array until it is rejected by
C                    later processing.  Some cases may require a larger
C                    value for 'maxtok' in order to hold some transparent
C                    tokens used in more involved data assignments.
CCCCCC
      INTEGER maxdvl
      INTEGER maxmod
      INTEGER maxsvl
      INTEGER maxtok
C
      PARAMETER (maxdvl=30)
      PARAMETER (maxmod=30)
      PARAMETER (maxsvl=30)
      PARAMETER (maxtok=100)
CCCCCC
C     Declare external reference.
CCCCCC
```

```
      LOGICAL exist
CCCCCC
C     /RDFINT/
C                to RDFREE:
C                eunit = INTEGER FORTRAN error output unit number.  The error
C                        messages sent to this unit should not be ignored.
C                        Any error should be considered fatal for the correct
C                        transfer of data from the input stream into
C                        program variables.
C                iunit = INTEGER FORTRAN input unit number.
C                ounit = INTEGER FORTRAN output unit number.
C
C                from RDFREE:
C                firsti = INTEGER array pointing to the first element of svalue
C                         or dvalue for corresponding modifier (mdfyr).
C                lasti  = INTEGER array pointing to the last element of svalue
C                         of dvalue for corresponding modifier (mdfyr).
C                mcnt   = INTEGER number of returned modifier names.
C                sordi  = INTEGER array indicating svalue, dvalue, or none for
C                         corresponding modifier (mdfyr).
C                         sordi(m) = 0 if keyword name only.
C                        sordi(m) = 1 if values are stored in the svalue array.
C                        sordi(m) = 2 if values are stored in the dvalue array.
C                rderr  = INTEGER read error status from RDTOK.
CCCCCC
      INTEGER eunit
      INTEGER iunit
      INTEGER ounit
C
      INTEGER firsti(maxmod)
      INTEGER lasti(maxmod)
      INTEGER mcnt
      INTEGER rderr
      INTEGER sordi(maxmod)
      COMMON /rdfint/ eunit, iunit, ounit,
     &                mcnt, rderr, sordi, firsti, lasti
CCCCCC
C     /RDFCHA/
C                to RDFREE:
C                token  = CHARACTER*64 work array for token strings.
C
C                from RDFREE:
C                mdfyr  = CHARACTER*6 array of modifier names.
C                svalue = CHARACTER*64 array of string values.
C                title  = CHARACTER*80 array for the title.
CCCCCC
      CHARACTER*6  mdfyr(maxmod)
      CHARACTER*64 svalue(maxsvl)
      CHARACTER*80 title
      CHARACTER*64 token(maxtok)
      COMMON /rdfcha/ mdfyr, svalue, title, token
CCCCCC
C     /RDFDP/
C                from RDFREE:
C                dvalue = double precision array of floating point values.
CCCCCC
      DOUBLE PRECISION dvalue(maxdvl)
      COMMON /rdfdp/ dvalue
CCCCCC
C       dcnt   = data value counter
C       eunit  = error I/O unit number
C       i      = do loop control variable
C       iunit  = input unit number
C       j      = do loop control variable
```

```
C       keyptr = element number of keyword in master keyword array
C       lastin = the last integer which wa stored in the data array
C       mcnt   = modifier counter
C       mod1   = first data element associated with the current modifier
C       mod2   = last  data element associated with the current modifier
C       ounit  = output I/O unit number
C       rderr  = read error number
C       repcnt = repeat counter for the '*' operator
C       repptr = pointer to the data element which is to be repeated. The
C                value to be repeated has already been stored in one of
C                the 2 data arrays when 'repptr' is referenced.
C       scnt   = string value counter
C       tokcnt = token counter
CCCCCC
        INTEGER dcnt
        INTEGER i
        INTEGER j
        INTEGER keyptr
        INTEGER lastin
        INTEGER mod1
        INTEGER mod2
        INTEGER repcnt
        INTEGER repptr
        INTEGER scnt
        INTEGER tokcnt
CCCCCC
C       repval          = temporary storage for the number to be repeated.
CCCCCC
        DOUBLE PRECISION repval
CCCCCC
C       lastc             = previous token
C       c               = current token
C       ctype           = current token type
C       mdtyp           = current modifier type
C       type            = external function name
C       ASTER           = string paramaeter
C       STRING          = string paramaeter
C       UNKNO           = string paramaeter
C       ALPHA           = string paramaeter
C       NUMER           = string paramaeter
C       INTEG           = string paramaeter
C       FLOAT           = string paramaeter
CCCCCC
        CHARACTER*64 lastc
        CHARACTER*64 c
        CHARACTER*6  ctype
        CHARACTER*6  mdtyp
        CHARACTER*6  type
        CHARACTER*6  ASTER
        CHARACTER*6  STRING
        CHARACTER*6  UNKNO
        CHARACTER*6  ALPHA
        CHARACTER*6  NUMER
        CHARACTER*6  INTEG
        CHARACTER*6  FLOAT
        CHARACTER*6  NULL
CCCCCC
C       true   = logical parameter
C       false  = logical parameter
C       repeat = flag to indicate when a repeated data value is to be
C                processed.
CCCCCC
        LOGICAL true
        LOGICAL false
```

```
      LOGICAL repeat
CCCCCC
      parameter (ASTER = '*')
      parameter (STRING= 'string')
      parameter (UNKNO = 'unkno')
      parameter (ALPHA = 'alpha')
      parameter (NUMER = 'numer')
      parameter (INTEG = 'integ')
      parameter (FLOAT = 'float')

      parameter (true = .TRUE.)
      parameter (false = .FALSE.)
CCCCCC
C     start here.
CCCCCC

C     Read the first set of records making up the first data line.

      call rdtok (iunit, ounit, eunit, maxtok, token, tokcnt,title,
     &            rderr)

C     Initialize control variables.

      repeat = false
      repval = 0
      mdtyp = NULL
      lastc = NULL

      do 99 i=1,maxmod
         sordi(i) = 0
         firsti(i) = 0
         lasti(i) = 0
   99 continue

      mcnt   = 0
      dcnt   = 0
      scnt   = 0

C     Analyze each token.

      do 199 i=1,tokcnt
         c = token(i)

C        Find token type.

         ctype = type(c)

         if (ctype .eq. ALPHA) then
            if (mod1 .gt. 0) then

C           The previous modifier was in the process of counting its
C           associated data values.  Complete bookkeeping on the data
C           value pointers of the previous modifier and then proceed.

            firsti(mcnt) = mod1
            lasti(mcnt) = mod2
         end if

C        The token is alphabetic and to be used as a keyword/modifier.
C        But first check for sufficient modifier name array space.

         if (mcnt .eq. maxmod) then
            write (eunit,*) 'RDFREE argument MAXMOD exceeded for ',
     &                      'keyword storage.'
```

```
            stop
         end if

C        Do bookkeeping for adding current token ('c') to modifier
C        names only if it passes the test against the master keyword list.

         if (exist(c,lenmst,master,keyptr)) then

C        token is in the master list, so add it to the
C        modifier array.

         mcnt = mcnt + 1
         mdfyr(mcnt) = c

C        Initialize bookkeeping data associated with this pointer.

         sordi(mcnt) = 0
         mdtyp = NULL
         mod1 = 0
         mod2 = 0
      else

C        Do not update the 'lastc' variable and select the
C        next token.

         go to 199
      end if

    else if (ctype .eq. STRING) then
CCCCCC
C     If the modifier already has a numeric assigned to it
C     then send an error message and stop.
CCCCCC
         if (mdtyp .eq. NUMER) then
            write (eunit,*) 'String type appears in a ',
     &                      'numeric data type field for ',
     &                      'modifier ',mdfyr(mcnt)
            stop
         end if

C        This token is a string value.  But first check
C        for sufficient string array space.

         if (scnt .eq. maxsvl) then
            write (eunit, *) 'RDFREE argument MAXSVL ',
     &                       'exceeded for',
     &                       ' string values.'
            stop
         end if

C        Do bookkeeping for the string value.  Start at string
C        index 2 in order to skip the leading single quote which
C        was saved as part of the token by RDTOK.

         scnt = scnt + 1
         svalue(scnt) = c(2:)
         mdtyp = STRING

         if (mod1 .eq. 0) then

C           Do bookkeeping for the string value array for the first
C           string of the current keyword/modifier.

            mod1 = scnt
```

```
            mod2 = scnt
            sordi(mcnt) = 1
         else

C           Do bookkeeping for the second and following string
C           values of the current keyword/modifier.

            mod2 = scnt
         end if

         if (repeat .and. repcnt .gt. 1) then

C           Process a repeated value.  But first check for sufficient
C           storage in the string array.

            if (scnt+repcnt .gt. MAXSVL) then
               write (eunit,*) 'A repeat integer of ', repcnt,
     &                         'caused RDFREE argument MAXSVL ',
     &                         'for the string value array ',
     &                         'to overflow.'
               stop
            else
               repptr = scnt

C              No need to start at j=1 because 'c' has been stored
C              in 'svalue' above.
               do 149 j=2,repcnt
                 scnt = scnt + 1
                 svalue(scnt) = svalue(repptr)
                 mod2 = scnt

  149          continue

C              repeat is now complete, so clear control variables.

               repeat = false
               repcnt = 0
            end if
         end if

      else if (ctype .eq. INTEG) then
CCCCCC
C         If the current token type is INTEG there are several states
C         possible.  It may be any of the following:
C         - a numeric item for a numeric keyword
C         - the repeat factor for a numeric item
C         - the repeat factor for a string item
C         - a numeric item to be repeated
CCCCCC
          if (mdtyp .eq. NULL .and. .not. repeat) then
             if (i+1              .le. tokcnt .and.
     &           token(i+1)       .eq. ASTER  .and.
     &           i+2              .le. tokcnt .and.
     &          (type(token(i+2)) .eq. INTEG  .or.
     &           type(token(i+2)) .eq. FLOAT  .or.
     &           type(token(i+2)) .eq. STRING)) then
                read (c,701) lastin
  701           format (bn, i24)
             else
                if (dcnt .eq. maxdvl) then
                   write (eunit,*) 'RDFREE argument MAXDVL ',
     &                             'exceeded for data values.'
                   stop
                end if
```

```
            mdtyp = NUMER
            dcnt = dcnt + 1
            read (c,701) lastin
            dvalue(dcnt) = lastin
            mod1 = dcnt
            mod2 = dcnt
            sordi(mcnt) = 2

            end if
         else if (mdtyp .eq. NULL .and. repeat) then

C              Process a repeated value.  But first check for sufficient
C              storage in the data array.

               if (dcnt+repcnt .gt. MAXDVL) then
                  write (eunit, *) 'RDFREE argument MAXDVL ',
     &                             'exceeded for',
     &                             ' data values.'
                  stop
               else
                  mdtyp = NUMER
                  dcnt = dcnt + 1
                  read (c,701) lastin
                  dvalue(dcnt) = lastin
                  mod1 = dcnt
                  mod2 = dcnt
                  sordi(mcnt) = 2
                  repval = dvalue(dcnt)
                  if (repcnt .gt. 1) then
C                    No need to start at j=1 because 'c' has been stored
C                    in 'dvalu' above.
                     do 139 j=2,repcnt
                        dcnt = dcnt + 1
                        dvalue(dcnt) = repval
                        mod2 = dcnt
  139                 continue
                   end if
                   repeat = false
                   repcnt = 0
                   lastin = 0
                end if

             else if (mdtyp .eq. NUMER .and. .not. repeat) then
                 if (i+1 .le. tokcnt .and.
     &              token(i+1) .eq. ASTER) then
                    if (i+2 .le. tokcnt .and.
     &                 type(token(i+2)) .eq. INTEG .or.
     &                 type(token(i+2)) .eq. FLOAT) then
                       read (c,701) lastin
                    else
                       write (eunit,*) 'Mixed mode ',
     &                                 'assignment while processing ',
     &                                 'a field for ',
     &                                  mdfyr(mcnt)
                       stop
                    end if
                 else
                    if (dcnt .eq. maxdvl) then
                      write (eunit,*) 'RDFREE argument MAXDVL ',
     &                                'exceeded for data values.'
                      stop
                    end if
                    dcnt = dcnt + 1
                    read (c,701) lastin
```

```
                 dvalue(dcnt) = lastin
                 mod2 = dcnt
              end if

           else if (mdtyp .eq. NUMER .and. repeat) then

C                Process a repeated value.  But first check for sufficient
C                storage in the data array.

                 if (dcnt+repcnt .gt. MAXDVL) then
                    write (eunit, *) 'RDFREE argument MAXDVL ',
     &                               'exceeded for',
     &                               ' data values.'
                    stop
                 else
                    dcnt = dcnt + 1
                    read (c,701) lastin
                    dvalue(dcnt) = lastin
                    mod2 = dcnt
                    repval = dvalue(dcnt)
                    if (repcnt .gt. 1) then
C                   No need to start at j=1 because 'c' has been stored
C                   in 'dvalu' above.
                       do 141 j=2,repcnt
                         dcnt = dcnt + 1
                         dvalue(dcnt) = repval
                         mod2 = dcnt
  141                  continue
                    end if
                    repeat = false
                    repcnt = 0
                    lastin = 0
                 end if

           else if (mdtyp .eq. STRING) then
              if (i+1 .le. tokcnt .and.
     &            token(i+1) .eq. ASTER) then
                 if (i+2 .le. tokcnt .and.
     &              type(token(i+2)) .eq. STRING) then
                    read (c,701) lastin
                 else
                    write (eunit,*) 'Mixed mode or incomplete ',
     &                              'assignment while processing',
     &                              'a repeated field for ',
     &                               mdfyr(mcnt)
                    stop
                 end if
              else
                    write (eunit,*) 'Mixed mode ',
     &                              'assignment while processing ',
     &                              'a field for ',
     &                               mdfyr(mcnt)
                 stop
              end if
           end if

        else if (ctype .eq. FLOAT) then
           if (mdtyp .ne. STRING) then
C                This token is a floating point numeric.  But first check
C                for sufficient data array space.

                 mdtyp = NUMER
                 if (dcnt .eq. maxdvl) then
                    write (eunit, *) 'RDFREE argument MAXDVL ',
```

```
     &                                  'exceeded for',
     &                                  ' data values.'
                        stop
                     end if

C                    Do bookkeeping for the numeric data value.

                     dcnt = dcnt + 1
                     read (c,702) dvalue(dcnt)
  702                format (bn,d24.16)

                  if (mod1 .eq. 0) then

C                    Do bookkeeping for the data value array for the first
C                    number of the current keyword/modifier.

                     mod1 = dcnt
                     mod2 = dcnt
                     sordi(mcnt) = 2
                  else

C                    Do bookkeeping for the second and following data values
C                    of the current keyword/modifier.

                     mod2 = dcnt
                  end if

                  if (repeat .and. repcnt .gt. 1) then

C                    Process a repeated value.  But first check for sufficient
C                    storage in the data array.

                     if (dcnt+repcnt .gt. MAXDVL) then
                        write (eunit, *) 'RDFREE argument MAXDVL ',
     &                                   'exceeded for',
     &                                   ' data values.'
                        stop
                     else
                        repval = dvalue(dcnt)

C                       No need to start at j=1 because 'c' has been stored
C                       in 'svalue' above.
                        do 159 j=2,repcnt
                          dcnt = dcnt + 1
                          dvalue(dcnt) = repval
                          mod2 = dcnt
  159                   continue
                        repeat = false
                        repcnt = 0
                     end if
                  end if
               else
                  write (eunit,*) 'Attempt to assign a numeric ',
     &                            'value to a string type keyword.'
                  stop
               end if

            else if (c .eq. ASTER) then

C                  This token is an asterisk.  But first check the last token
C                  for type = INTEGER.

                  if (type(lastc) .ne. INTEG) then
                     write (eunit,*) 'Error non-INTEGER repeat factor.'
```

```
                  stop
               else
C                 Set up controls for processing a repeated data value.

                  repcnt = lastin
                  repeat = true
               end if
            end if
            lastc = c
  199 continue

          if (mod1 .gt. 0) then

C            Both pointers have been used for bookkeeping, so return them.

             firsti(mcnt) = mod1
             lasti(mcnt) = mod2
          end if

  999     continue
          return
          end
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
          function type (C)
CCCCCC
C         NAME: type
C
C         PURPOSE: This module is part of a suite of routines used to analyze
C                  a free form input stream in order to generate a consistent
C                  array of modifier names and associated data values.  It is
C                  used by RDFREE to classify a token as alphabetic, numeric,
C                  string, or an * operator.
C
C         ARGUMENTS:
C                  input:
C                  c = CHARACTER*24 token string.
C
C                  output:
C                  type = CHARACTER*6 token classification.
C
C         REFERENCES:
C                  none
C
C         CHANGES:
C                  1.00     08-28-90        Ron Janetzke
C                           Original text.
C                  1.01     09-24-90        Ron Janetzke
C                           Delete equal type and add string type.
CCCCCC

C         temp  = scratch area
C         type  = function name
C         c     = function argument
C         ASTER = string parameter
C         UNKNO = string parameter
C         ALPHA = string parameter
C         NUMER = string parameter
C         INTEG = string parameter
C         FLOAT = string parameter
C         OPER  = string parameter
C         STRING = string parameter

          CHARACTER*6  temp
          CHARACTER*6  type
```

```
      CHARACTER*24 c
      CHARACTER*6  ASTER
      CHARACTER*6  UNKNO
      CHARACTER*6  ALPHA
      CHARACTER*6  NUMER
      CHARACTER*6  INTEG
      CHARACTER*6  FLOAT
      CHARACTER*6  OPER
      CHARACTER*6  STRING


      parameter (ASTER = '*')
      parameter (UNKNO = 'unkno')
      parameter (ALPHA = 'alpha')
      parameter (NUMER = 'numer')
      parameter (INTEG = 'integ')
      parameter (FLOAT = 'float')
      parameter (OPER  = 'oper')
      parameter (STRING = 'string')

CCCCCC
C     Start here.
CCCCCC
      temp = UNKNO

      if (len(c) .gt. 0) then

C        Assign temp the value of OPER, NUMER, STRING, or ALPHA.

         if (c .eq. ASTER) then
              temp = OPER
         else if (c(1:1) .eq. '+' .or.
     &             c(1:1) .eq. '-' .or.
     &             c(1:1) .eq. '.' .or.
     &            (c(1:1) .ge. '0' .and.
     &             c(1:1) .le. '9'      )) then
              temp = NUMER
         else if (c(1:1) .eq. '''') then
              temp = STRING
         else
              temp = ALPHA
         end if

         if (temp .eq. NUMER) then

C        Refine the token classification to be either INTEG or FLOAT.

            if (index(c,'.') .ne. 0 .or.
     &         index(c,'E') .ne. 0 .or.
     &         index(c,'D') .ne. 0) then
               temp = FLOAT
            else
               temp = INTEG
            end if
         end if

         type = temp
      end if

      return
      end
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
      subroutine rdtok (iunit, ounit, eunit, maxtok, token, toknum,
     &                  title, rderr)
```

```
CCCCCC
C       NAME: rdtok
C
C       PURPOSE:
C               RDTOK reads records from the input unit.  Each record is
C               analyzed character by character in order to group the
C               characters into token strings.  Comments beginning with \
C               are ignored.  Reading of the records continues until an
C               end of line terminator is found (|).
C
C       METHOD:
C               A series of logical variables are used to hold the current
C               state of the token generator.  These variables determine
C               which set of characters are permitted for a given token
C               type, which set are separators , and which set may initiate
C               the processing of another token.  A keyword token is
C               different than a modifier token only in the sense that it
C               appears first in the input stream.
C
C       ARGUMENTS:
C               input:
C               iunit = INTEGER FORTRAN input unit number.
C               ounit = INTEGER FORTRAN output unit number.
C               eunit = INTEGER FORTRAN error output unit number.
C               maxtok= INTEGER maximum number of tokens allowed.
C
C               output:
C               rderr = INTEGER error status of read statement.
C               token = CHARACTER*(*) array of token strings.
C               toknum = INTEGER number of tokens returned.
C               title  = CHARACTER*80 title string stored separate from
C                        tokens.
C
C       REFERENCES:
C               lastr
C               lcmnt
C               ldigt
C               lexin
C               lflnm
C               lkwin
C               lkywd
C               lmdfr
C               lmdin
C               lnmbr
C               lnmin
C               lperd
C               lquote
C               lsepr
C               lsign
C               lvbar
C               upcase
C
C       CHANGES:
C               1.00    08-23-90        Ron Janetzke
C                       Original text.
C               1.01    09-20-90        Ron Janetzke
C                       RDERR is set to -1 for end of file and added to
C                       argument list.
C               1.02    09-24-90        Ron Janetzke
C                       Insert '.' in floating point number if missing before
C                       exponent.  Use ' for string delimiter, and allow
C                       multiple string data values.  Move '=' to separator
C                       list.
C               1.03    10-19-90        Ron Janetzke
```

```
C                         Add RDERR=1, and RDERR=2.
C                         Add exponent specific code. (lfexp)
C                 1.04    02-05-91        Ron Janetzke
C                         Function LSEPR changed to add TAB as separator.
CCCCCC

C         RECLEN = the numbers of columns used for input should match this
C                  specification of record length.
C         title  = special variable to hole a title string, if one is found
C                  in the input records.
C         rec    = input record buffer.
C
C         blank  = string parameter
C         upcase = function
C         null   = null string character
C         lastc  = previous character
C         c      = current character
C         token  = output argument
C         temp5  = scratch area for the first 5 characters of 'rec'.
C
C         eunit  = error I/O unit
C         i      = loop indes
C         iunit  = input unit
C         maxtok = maximum number of tokens
C         mxtlen = maximum token length
C         ounit  = output unit
C         pcount = count of periods in the current numeric token (1 max).
C         rderr  = error number as a result of the read.
C         toklen = length of the current token
C         toknum = number of the current token
C         xcount = count of the number of initial exponent characters [E,D]
C                  (1 max).
C
C         lfastr = look for asterisk
C         lfcmnt = look for comment character
C         lfdigt = look for digit
C         lfexin = look for exponent initial character
C         lfexp  = look for exponent
C         lfflnm = look for file name or any quoted string
C         lfkwin = look for initial keyword character
C         lfkywd = look for keyword character
C         lfmdfr = look for modifier character
C         lfmdin = look for initial modifier character
C         lfnmbr = look for number
C         lfnmin = look for initial number
C         lfperd = look for period
C         lfqtin = look for initial quote
C         lfqtfn = look for final quote
C         lfsepr = look for separator character
C         lfsign = look for + or - sign
C         lfvbar = look for vertical bar
C
C         lkywdf = keyword found flag
C
C         lastr  = function
C         lcmnt  = function
C         ldigt  = function
C         lexin  = function
C         lflnm  = function
C         lkwin  = function
C         lkywd  = function
C         lmdfr  = function
C         lmdin  = function
C         lnmbr  = function
```

```
C         lnmin   = function
C         lperd   = function
C         lquote   = function
C         lsepr   = function
C         lsign   = function
C         lvbar   = function
C
C         false   = parameter
C         true    = parameter

          INTEGER RECLEN
          parameter (RECLEN = 80)
          CHARACTER*80 title
          CHARACTER*80 rec

          CHARACTER*1  blank
          CHARACTER*1  upcase
          CHARACTER*1  null
          CHARACTER*1  lastc
          CHARACTER*1  c
          CHARACTER*(*) token(*)
          CHARACTER*5  temp5

          INTEGER eunit
          INTEGER i
          INTEGER iunit
          INTEGER maxtok
          INTEGER mxtlen
          INTEGER ounit
          INTEGER pcount
          INTEGER rderr
          INTEGER toklen
          INTEGER toknum

          LOGICAL lfastr
          LOGICAL lfcmnt
          LOGICAL lfdigt
          LOGICAL lfexin
          LOGICAL lfexp
          LOGICAL lfflnm
          LOGICAL lfkwin
          LOGICAL lfkywd
          LOGICAL lfmdfr
          LOGICAL lfmdin
          LOGICAL lfnmbr
          LOGICAL lfnmin
          LOGICAL lfperd
          LOGICAL lfqtin
          LOGICAL lfqtfn
          LOGICAL lfsepr
          LOGICAL lfsign
          LOGICAL lfvbar

          LOGICAL lkywdf

          LOGICAL lastr
          LOGICAL lcmnt
          LOGICAL ldigt
          LOGICAL lexin
          LOGICAL lflnm
          LOGICAL lkwin
          LOGICAL lkywd
          LOGICAL lmdfr
          LOGICAL lmdin
```

```
      LOGICAL lnmbr
      LOGICAL lnmin
      LOGICAL lperd
      LOGICAL lquote
      LOGICAL lsepr
      LOGICAL lsign
      LOGICAL lvbar

      LOGICAL false
      LOGICAL true

      parameter (false=.false.)
      parameter (true=.true.)
      parameter (blank = ' ')

CCCCCC
C     Start here.
C     Initialize all state variables.
CCCCCC
      mxtlen = len (token(1))
      null   = char(0)
      lfastr = false
      lfcmnt = true
      lfdigt = false
      lfexin = false
      lfexp  = false
      lfflnm = false
      lfkwin = true
      lfkywd = false
      lfmdfr = false
      lfmdin = false
      lfnmbr = false
      lfnmin = false
      lfperd = false
      lfqtin = false
      lfqtfn = false
      lfsepr = true
      lfsign = false
      lfvbar = true

      pcount = 0

      toklen = 0
      toknum = 0
      lastc = blank

      do 99 i = 1,maxtok
         token(i) = ' '
   99 continue

  100 continue
      read  (iunit, '(a)', end=990) rec
      write (ounit, '(x,a)') rec
CCCCCC
C     Look for title.
CCCCCC
      temp5 = rec
      do 140 i=1,5
         temp5(i:i) = upcase(temp5(i:i))
  140 continue
      if (temp5 .eq. 'TITLE') then
CCCCCC
C        Title is stored outside of the token array in a separate variable.
C        'temp5' will be returned as a keyword in the token array.
```

```
CCCCCC
            title = rec(7:80)
            toknum = 1
            toklen = min(5,mxtlen)
            token(toknum) = temp5
            go to 999
         end if
CCCCCC
C        Analyze each character in 'rec'.
CCCCCC
         do 200 i = 1,RECLEN

C          Only convert to uppercase when not analyzing a quoted string.

           if (.not. lfflnm) then
              c = upcase(rec(i:i))
           else
              c = rec(i:i)
           end if
CCCCCC
C          Look for end of line vertical bar (|).
CCCCCC
           if (lfvbar .and. lvbar(c)) then
C                write (*,*) c,'lfvbar'

                go to 999
           end if

CCCCCC
C          Look for comment initial back-slash (\).
CCCCCC
           if (lfcmnt .and. lcmnt(c)) then
C                write (*,*) c,'lfcmnt'
C             Terminate the current token if needed.
              if (.not. lsepr(lastc)) then
                 toknum = toknum + 1
                 if (toknum .gt. maxtok) then
                    write (eunit,*) 'ERROR: number of tokens exceeds '
     &                                ,'MAXTOK'
                    go to 999
                 end if
                 toklen = 0
              end if

                 pcount = 0
C                lfastr =
                 lfcmnt = true
                 lfdigt = true
                 lfexin = false
                 lfexp  = false
                 lfflnm = false
                 lfkwin = false
                 lfkywd = false
                 lfmdfr = false
                 lfmdin = true
                 lfnmbr = false
                 lfnmin = true
                 lfperd = true
                 lfqtin = true
                 lfqtfn = false
                 lfsepr = true
                 lfsign = true
                 lfvbar = true
```

```
            go to 100
         end if

CCCCCC
C        Look for keyword initial letter.
CCCCCC
         if (lfkwin .and. lkwin(c)) then
C              write (*,*) c,'lfkwin'
               if (toklen .eq. 0) then
                  toknum = toknum + 1
                  if (toknum .gt. maxtok) then
                     write (eunit,*) 'ERROR: number of tokens exceeds '
     &                                ,'MAXTOK'
                     go to 999
                  end if
               end if
               toklen = min(mxtlen,toklen +1)
               token (toknum) (toklen:toklen) = c

CCCCCC
C              Logical flag indicates keyword has been found.
CCCCCC
               lkywdf = true

C              Set analyzer state variables.

               lfastr = false
               lfcmnt = true
               lfdigt = true
               lfexin = false
               lfexp  = false
               lfflnm = false
               lfkwin = false
               lfkywd = true
               lfmdfr = false
               lfmdin = false
               lfnmbr = false
               lfnmin = false
               lfperd = false
               lfqtin = false
               lfqtfn = false
               lfsepr = true
               lfsign = false
               lfvbar = true

               go to 199
         end if
CCCCCC
C        Look for keyword secondary characters.
CCCCCC
         if (lfkywd .and. lkywd(c)) then
C              write (*,*) c,'lfkywd'
               toklen = min(mxtlen,toklen +1)
               token (toknum) (toklen:toklen) = c

C              Set analyzer state variables.

               lfastr = false
               lfcmnt = true
               lfdigt = true
               lfexin = false
               lfexp  = false
               lfflnm = false
               lfkwin = false
```

```
                lfkywd = true
                lfmdfr = false
                lfmdin = false
                lfnmbr = false
                lfnmin = false
                lfperd = false
                lfqtin = false
                lfqtfn = false
                lfsepr = true
                lfsign = false
                lfvbar = true

                go to 199
            end if

CCCCCC
C           Look for modifier initial letter.
CCCCCC
            if (lfmdin .and. lmdin(c)) then
C                 write (*,*) c,'lfmdin'
                  if (toklen .eq. 0)  then
                     toknum = toknum + 1
                     if (toknum .gt. maxtok) then
                        write (eunit,*) 'ERROR: number of tokens exceeds '
     &                                  ,'MAXTOK'
                        go to 999
                     end if
                  end if
                  toklen = min(mxtlen,toklen +1)
                  token (toknum) (toklen:toklen) = c

                  lfastr = false
                  lfcmnt = true
                  lfdigt = true
                  lfexin = false
                  lfexp  = false
                  lfflnm = false
                  lfkwin = false
                  lfkywd = false
                  lfmdfr = true
                  lfmdin = false
                  lfnmbr = false
                  lfnmin = false
                  lfperd = false
                  lfqtin = false
                  lfqtfn = false
                  lfsepr = true
                  lfsign = false
                  lfvbar = true

                  go to 199
            end if
CCCCCC
C           Look for modifier secondary characters.
CCCCCC
            if (lfmdfr .and. lmdfr(c)) then
C                 write (*,*) c,'lfmdfr'
                  toklen = min(mxtlen,toklen +1)
                  token (toknum) (toklen:toklen) = c

                  lfastr = false
                  lfcmnt = true
                  lfdigt = true
                  lfexin = false
```

```
                lfexp  = false
                lfflnm = false
                lfkwin = false
                lfkywd = false
                lfmdfr = true
                lfmdin = false
                lfnmbr = false
                lfnmin = false
                lfperd = false
                lfqtin = false
                lfqtfn = false
                lfsepr = true
                lfsign = false
                lfvbar = true

                go to 199
            end if

CCCCCC
C           Look for number initial sign.
CCCCCC
            if (lfnmin .and. lfsign .and. lsign(c)) then
C                 write (*,*) c,'lfnmin'
                  if (toklen .eq. 0)  then
                     toknum = toknum + 1
                     if (toknum .gt. maxtok) then
                        write (eunit,*) 'ERROR: number of tokens exceeds '
     &                                  ,'MAXTOK'
                        go to 999
                     end if
                  end if
                  toklen = min(mxtlen,toklen +1)
                  token (toknum) (toklen:toklen) = c

                  lfastr = false
                  lfcmnt = true
                  lfdigt = true
                  lfexin = false
                  lfexp  = false
                  lfflnm = false
                  lfkwin = false
                  lfkywd = false
                  lfmdfr = false
                  lfmdin = false
                  lfnmbr = true
                  lfnmin = false
                  lfperd = true
                  lfqtin = false
                  lfqtfn = false
                  lfsepr = true
                  lfsign = false
                  lfvbar = true

                  go to 199
            end if

CCCCCC
C           Look for number initial digit.
CCCCCC
            if (lfnmin .and. lfdigt .and. ldigt(c)) then
C                 write (*,*) c,'lfnmin,lfdigt'
                  if (toklen .eq. 0) then
                     toknum = toknum + 1
                     if (toknum .gt. maxtok) then
```

```
                 write (eunit,*) 'ERROR: number of tokens exceeds '
     &                           ,'MAXTOK'
                 go to 999
              end if
           end if
           toklen = min(mxtlen,toklen +1)
           token (toknum) (toklen:toklen) = c

           lfastr = true
           lfcmnt = true
           lfdigt = true
           lfexin = true
           lfexp  = false
           lfflnm = false
           lfkwin = false
           lfkywd = false
           lfmdfr = false
           lfmdin = false
           lfnmbr = true
           lfnmin = false
           lfperd = true
           lfqtin = false
           lfqtfn = false
           lfsepr = true
           lfsign = false
           lfvbar = true

           go to 199
        end if

CCCCCC
C       Look for number initial period.
CCCCCC
        if (lfnmin .and. lfperd .and. lperd(c)) then
C          write (*,*) c,'lfnmin,lfperd'
           if (toklen .eq. 0) then
              toknum = toknum + 1
              if (toknum .gt. maxtok) then
                 write (eunit,*) 'ERROR: number of tokens exceeds '
     &                           ,'MAXTOK'
                 go to 999
              end if
           end if
           toklen = min(mxtlen,toklen +1)
           token (toknum) (toklen:toklen) = c

           lfastr = false
           lfcmnt = true
           lfdigt = true
           lfexin = false
           lfexp  = false
           lfflnm = false
           lfkwin = false
           lfkywd = false
           lfmdfr = false
           lfmdin = false
           lfnmbr = true
           lfnmin = false
           lfperd = false
           lfsepr = false
           lfsign = false
           lfvbar = true

           go to 199
```

```
          end if

CCCCCC
C         Look for number digit.
CCCCCC
          if (lfnmbr .and. lfdigt .and. ldigt(c)) then
C               write (*,*) c,'lfnmbr,lfdigt'
                toklen = min(mxtlen,toklen +1)
                token (toknum) (toklen:toklen) = c

                lfastr = true
                lfcmnt = true
                lfdigt = true
                lfexin = true
                lfexp  = false
                lfflnm = false
                lfkwin = false
                lfkywd = false
                lfmdfr = false
                lfmdin = false
                lfnmbr = true
                lfnmin = false
                lfperd = true
                lfqtin = false
                lfqtfn = false
                lfsepr = true
                lfsign = false
                lfvbar = true

                go to 199
          end if

CCCCCC
C         Look for number internal period.
CCCCCC
          if (lfnmbr .and. lfperd .and. lperd(c)) then
C               write (*,*) c,'lfnmbr,lfperd'
                toklen = min(mxtlen,toklen +1)
                token (toknum) (toklen:toklen) = c
                pcount = pcount + 1

                lfastr = false
                lfcmnt = true
                lfdigt = true
                lfexin = true
                lfexp  = false
                lfflnm = false
                lfkwin = false
                lfkywd = false
                lfmdfr = false
                lfmdin = false
                lfnmbr = true
                lfnmin = false
                lfperd = false
                lfqtin = false
                lfqtfn = false
                lfsepr = true
                lfsign = false
                lfvbar = true

                go to 199
          end if

CCCCCC
```

```
C         Look for exponent initial letter E or D.
CCCCCC
          if (lfnmbr .and. lfexin .and. lexin(c)) then
Cc              write (*,*) c,'lfnmbr,lfexin'
C
C               Add a period if there is none for this token yet.
C
                if (pcount .eq. 0) then
                   toklen = min(mxtlen,toklen + 1)
                   token(toknum) (toklen:toklen) = '.'
                   pcount = pcount + 1
                end if
                toklen = min(mxtlen,toklen +1)
                token (toknum) (toklen:toklen) = c

                lfastr = false
                lfcmnt = true
                lfdigt = true
                lfexin = false
                lfexp  = true
                lfflnm = false
                lfkwin = false
                lfkywd = false
                lfmdfr = false
                lfmdin = false
                lfnmbr = false
                lfnmin = false
                lfperd = false
                lfqtin = false
                lfqtfn = false
                lfsepr = false
                lfsign = true
                lfvbar = true

                go to 199
          end if

CCCCCC
C         Look for exponent initial sign.
CCCCCC
          if (lfexp .and. lfsign .and. lsign(c)) then
Cc              write (*,*) c,'lfexp,lsign'
                toklen = min(mxtlen,toklen +1)
                token (toknum) (toklen:toklen) = c

                lfastr = false
                lfcmnt = false
                lfdigt = true
                lfexin = false
                lfexp  = true
                lfflnm = false
                lfkwin = false
                lfkywd = false
                lfmdfr = false
                lfmdin = false
                lfnmbr = false
                lfnmin = false
                lfperd = false
                lfqtin = false
                lfqtfn = false
                lfsepr = false
                lfsign = false
                lfvbar = false
```

```
            go to 199
        end if

CCCCCC
C       Look for exponent initial digit.
CCCCCC
        if (lfexp .and. lfdigt .and. ldigt(c)) then
Cc          write (*,*) c,'lfexp,lfdigt,ldigt'
            toklen = min(mxtlen,toklen +1)
            token (toknum) (toklen:toklen) = c

            lfastr = false
            lfcmnt = true
            lfdigt = true
            lfexin = false
            lfexp  = true
            lfflnm = false
            lfkwin = false
            lfkywd = false
            lfmdfr = false
            lfmdin = false
            lfnmbr = false
            lfnmin = false
            lfperd = false
            lfqtin = false
            lfqtfn = false
            lfsepr = true
            lfsign = false
            lfvbar = true

            go to 199
        end if

CCCCCC
C       Look for exponent secondary digits.
CCCCCC
C        if (lfnmbr .and. lfdigt .and. ldigt(c)) then
Cc          write (*,*) c,'lfnmbr,lfdigt'
C            toklen = min(mxtlen,toklen +1)
C            token (toknum) (toklen:toklen) = c
C
C            lfastr = false
C            lfcmnt = true
C            lfdigt = true
C            lfexin = false
C            lfexp  = false
C            lfflnm = false
C            lfkwin = false
C            lfkywd = false
C            lfmdfr = false
C            lfmdin = false
C            lfnmbr = true
C            lfnmin = false
C            lfperd = false
C            lfqtin = false
C            lfqtfn = false
C            lfsepr = true
C            lfsign = false
C            lfvbar = true
C
C            go to 199
C        end if

CCCCCC
```

```
C         Look for asterisk (*) operator.
CCCCCC
          if (lfastr .and. lastr(c)) then
Cc              write (*,*) c,'lfastr'
                toknum = toknum + 1
                if (toknum .gt. maxtok) then
                   write (eunit,*) 'ERROR: number of tokens exceeds '
     &                              ,'MAXTOK'
                   go to 999
                end if

                toklen = 1
                token (toknum) (toklen:toklen) = c

C               A token length of 0 will flag other sections of the code
C               that an initial character of a number or keyword is
C               expected.

                toklen = 0

                lfastr = false
                lfcmnt = true
                lfdigt = true
                lfexin = false
                lfexp  = false
                lfflnm = false
                lfkwin = false
                lfkywd = false
                lfmdfr = false
                lfmdin = false
                lfnmbr = false
                lfnmin = true
                lfperd = true
                lfqtin = true
                lfqtfn = false
                lfsepr = true
                lfsign = true
                lfvbar = true

                go to 199
          end if

CCCCCC
C         Look for initial quote (') operator.
CCCCCC
          if (lfqtin .and. lquote(c)) then
C               write (*,*) c,'lfquote'
                toknum = toknum + 1
                if (toknum .gt. maxtok) then
                   write (eunit,*) 'ERROR: number of tokens exceeds '
     &                              ,'MAXTOK'
                   go to 999
                end if

                toklen = 1
                token (toknum) (toklen:toklen) = c

                lfastr = false
                lfcmnt = false
                lfdigt = true
                lfexin = false
                lfexp  = false
                lfflnm = true
                lfkwin = false
```

```
                lfkywd = false
                lfmdfr = false
                lfmdin = false
                lfnmbr = false
                lfnmin = false
                lfperd = false
                lfqtin = false
                lfqtfn = true
                lfsepr = true
                lfsign = false
                lfvbar = true

                go to 199
            end if

CCCCCC
C           Look for final quote (') operator.
CCCCCC
            if (lfqtfn .and. lquote(c)) then
C                write (*,*) c,'final quote'
CCCCCC
C               Disregard final quote as part of a token, but change
C               the anayzer state variables.
CCCCCC
                toklen = 0

                lfastr = false
                lfcmnt = true
                lfdigt = true
                lfexin = false
                lfexp  = false
                lfflnm = false
                lfkwin = false
                lfkywd = false
                lfmdfr = false
                lfmdin = false
                lfnmbr = false
                lfnmin = false
                lfperd = false
                lfqtin = true
                lfqtfn = false
                lfsepr = true
                lfsign = false
                lfvbar = true

                go to 199
            end if

CCCCCC
C           Look for generic string, also used for file names.
CCCCCC
            if (lfflnm .and. lflnm(c)) then
C                write (*,*) c,'lfflnm'
                toklen = min(mxtlen,toklen +1)
                token (toknum) (toklen:toklen) = c

                lfastr = true
                lfcmnt = true
                lfdigt = true
                lfexin = false
                lfexp  = false
                lfflnm = true
                lfkwin = false
                lfkywd = false
```

```
                lfmdfr = false
                lfmdin = false
                lfnmbr = false
                lfnmin = false
                lfperd = false
                lfqtin = false
                lfqtfn = true
                lfsepr = false
                lfsign = false
                lfvbar = true

                go to 199
            end if

CCCCCC
C           Look for separators.
CCCCCC
            if (lfsepr .and. lsepr(c)) then
C                write (*,*) c,'lfsepr'
                 if (.not. lsepr(lastc)) then
                   toklen = 0
                 end if

                 pcount = 0
C                lfastr =
                 lfcmnt = true
                 lfdigt = true
                 lfexin = false
                 lfexp  = false
                 lfflnm = false
                 lfkwin = false
                 lfkywd = false
                 lfmdfr = false
                 lfmdin = true
                 lfnmbr = false
                 lfnmin = true
                 lfperd = true
                 lfqtin = true
                 lfqtfn = false
                 lfsepr = true
                 lfsign = true
                 lfvbar = true

C               write (*,*)
C      &         'lfastr', lfastr ,
C      &         'lfcmnt', lfcmnt ,
C      &         'lfdigt', lfdigt ,
C      &         'lfexin', lfexin ,
C      &         'lfexp ', lfexp  ,
C      &         'lfflnm', lfflnm ,
C      &         'lfkwin', lfkwin ,
C      &         'lfkywd', lfkywd ,
C      &         'lfmdfr', lfmdfr ,
C      &         'lfmdin', lfmdin ,
C      &         'lfnmbr', lfnmbr ,
C      &         'lfnmin', lfnmin ,
C      &         'lfperd', lfperd ,
C      &         'lfqtin', lfqtin ,
C      &         'lfqtfn', lfqtfn ,
C      &         'lfsepr', lfsepr ,
C      &         'lfsign', lfsign ,
C      &         'lfvbar', lfvbar

                 go to 199
```

```
          end if

CCCCCC
C         All others are errors.
CCCCCC
          if (toknum .eq. 0) then
             write (eunit,*) 'Keyword was not found as first token.'
             rderr = 1
          else
             write (eunit,*)
     &           ' ERROR:',c,' in column',i,' toknum=',toknum,
     &           ' toklen=', toklen
             rderr = 2
             go to 999
          end if

  199 continue

C       Save current character into previous character buffer.

        lastc = c
  200 continue
CCCCCC
C       End of record.
CCCCCC
        if (.not. lsepr(lastc)) then
           toklen = 0
        end if

        pcount = 0
C       lfastr =
        lfcmnt = true
        lfdigt = true
        lfexin = false
        lfexp  = false
        lfflnm = false
        lfkwin = false
        lfkywd = false
        lfmdfr = false
        lfmdin = true
        lfnmbr = false
        lfnmin = true
        lfperd = true
        lfqtin = true
        lfqtfn = false
        lfsepr = true
        lfsign = true
        lfvbar = true

        go to 100
CCCCCC
C       Exit.
CCCCCC
  990   CONTINUE
        RDERR = -1

  999   CONTINUE
        return
        end
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
        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
C
        CHARACTER*1 upcase
        CHARACTER*1 c

        if (ichar(c).ge.ichar('a') .and. ichar(c).le.ichar('z')) then
           upcase = char(ichar(c)-32)
        else
           upcase = c
        end if

        return
        end
CCCCCCCCCCC
CCCCCCCCCCC
        function lastr(c)
CCCCCC
C       NAME: lastr
C
C       PURPOSE: Compare c to the replication operator character (*) for RDTOK.
C
C       ARGUMENTS:
C               input:
C               c = CHARACTER*1 to be used for the compare.
C
C               output:
C               lastr = LOGICAL return value.
C
C       EXAMPLE 1:
C               c = '*'
C               lastr = .true.
C
C       REFERENCES:
C               none
C
C       CHANGES:
C               1.00    08-25-90        Ron Janetzke
C                       Original text.
CCCCCC
C
        LOGICAL lastr
        CHARACTER*1 c

        if (c .eq. '*') then
```

```
          lastr = .true.
        else
          lastr = .false.
        end if

        return
        end
CCCCCCCCCCC
CCCCCCCCCCC
        function lcmnt(c)
CCCCCC
C       NAME: lcmnt
C
C       PURPOSE: Compare c to the comment initiator character (\) for RDTOK.
C
C       ARGUMENTS:
C               input:
C               c = CHARACTER*1 to be used for the compare.
C
C               output:
C               lcmnt = LOGICAL return value.
C
C       EXAMPLE 1:
C               c = '\'
C               lcmnt = .true.
C
C       REFERENCES:
C               none
C
C       CHANGES:
C               1.00    08-25-90        Ron Janetzke
C                       Original text.
CCCCCC
C
        LOGICAL lcmnt
        CHARACTER*1 c

        if (c .eq. '\') then
          lcmnt = .true.
        else
          lcmnt = .false.
        end if

        return
        end
CCCCCCCCCCC
CCCCCCCCCCC
        function ldigt(c)
CCCCCC
C       NAME: ldigt
C
C       PURPOSE: Compare c to the digit (0-9) character set for RDTOK.
C
C       ARGUMENTS:
C               input:
C               c = CHARACTER*1 to be used for the compare.
C
C               output:
C               ldigt = LOGICAL return value.
C
C       EXAMPLE 1:
C               c = '0'
C               ldigt = .true.
C
```

```
C       REFERENCES:
C               none
C
C       CHANGES:
C               1.00    08-25-90        Ron Janetzke
C                       Original text.
CCCCCC
C
        LOGICAL ldigt
        CHARACTER*1 c

        if (ichar(c).ge.ichar('0') .and. ichar(c).le.ichar('9')) then
          ldigt = .true.
        else
          ldigt = .false.
        end if

        return
        end
CCCCCCCCCCC
CCCCCCCCCCC
        function lquote(c)
CCCCCC
C       NAME: lquote
C
C       PURPOSE: Compare c to the string assignment operator (=) for RDTOK.
C
C       ARGUMENTS:
C               input:
C               c = CHARACTER*1 to be used for the compare.
C
C               output:
C               lquote = LOGICAL return value.
C
C       EXAMPLE 1:
C               c = ''''
C               lquote = .true.
C
C       REFERENCES:
C               none
C
C       CHANGES:
C               1.00    09-24-90        Ron Janetzke
C                       Redo of function 'lequl' to make paired quotes
C                       required for string data values.
CCCCCC
C
        LOGICAL lquote
        CHARACTER*1 c

        if (c .eq. '''') then
          lquote = .true.
        else
          lquote = .false.
        end if

        return
        end
CCCCCCCCCCC
CCCCCCCCCCC
        function lexin(c)
CCCCCC
C       NAME: lexin
C
```

```
C       PURPOSE: Compare c to the exponent initiator characters (E,D)
C                for RDTOK.
C
C       ARGUMENTS:
C               input:
C               c = CHARACTER*1 to be used for the compare.
C
C               output:
C               lexin = LOGICAL return value.
C
C       EXAMPLE 1:
C               c = 'E'
C               lexin = .true.
C
C       REFERENCES:
C               none
C
C       CHANGES:
C               1.00    08-25-90        Ron Janetzke
C                       Original text.
CCCCCC
C
        LOGICAL lexin
        CHARACTER*1 c

        if (c .eq. 'E' .or. c .eq. 'D') then
          lexin = .true.
        else
          lexin = .false.
        end if

        return
        end
CCCCCCCCCCC
CCCCCCCCCCC
        function lflnm(c)
CCCCCC
C       NAME: lflnm
C
C       PURPOSE: Compare c to the file name character set for RDTOK.
C
C       ARGUMENTS:
C               input:
C               c = CHARACTER*1 to be used for the compare.
C
C               output:
C               lflnm = LOGICAL return value.
C
C       EXAMPLE 1:
C               c = 'A'
C               lflnm = .true.
C
C       REFERENCES:
C               none
C
C       CHANGES:
C               1.00    08-25-90        Ron Janetzke
C                       Original text.
C               1.01    09-24-90        Ron Janetzke
C                       Extend the beginning of the set from $ to !.
C               1.02    10-19-90        Ron Janetzke
C                       Extend the end of the range from ' to z.
C               1.03    10-24-90        Ron Janetzke
C                       Extend the beginning of the set from ! to SPACE.
```

```
CCCCCC
C
        LOGICAL lflnm
        CHARACTER*1 c

        if (ichar(c).ge.ichar(' ') .and. ichar(c).le.ichar('&') .or.
     &      ichar(c).ge.ichar('(') .and. ichar(c).le.ichar('z')) then
          lflnm = .true.
        else
          lflnm = .false.
        end if

        return
        end
CCCCCCCCCCC
CCCCCCCCCCC
        function lkwin(c)
CCCCCC
C       NAME: lkwin
C
C       PURPOSE: Compare c to the keyword initiator character set for RDTOK.
C
C       ARGUMENTS:
C               input:
C               c = CHARACTER*1 to be used for the compare.
C
C               output:
C               lkwin = LOGICAL return value.
C
C       EXAMPLE 1:
C               c = 'A'
C               lkwin = .true.
C
C       REFERENCES:
C               none
C
C       CHANGES:
C               1.00    08-25-90        Ron Janetzke
C                       Original text.
CCCCCC
C
        LOGICAL lkwin
        CHARACTER*1 c

        if (ichar(c).ge.ichar('A') .and. ichar(c).le.ichar('Z')) then
          lkwin = .true.
        else
          lkwin = .false.
        end if

        return
        end
CCCCCCCCCCC
CCCCCCCCCCC
        function lkywd(c)
CCCCCC
C       NAME: lkywd
C
C       PURPOSE: Compare c to the keyword secondary character set for RDTOK.
C
C       ARGUMENTS:
C               input:
C               c = CHARACTER*1 to be used for the compare.
C
```

```
C               output:
C               lkywd = LOGICAL return value.
C
C       EXAMPLE 1:
C               c = 'A'
C               lkywd = .true.
C
C       REFERENCES:
C               none
C
C       CHANGES:
C               1.00    08-25-90        Ron Janetzke
C                       Original text.
C               1.01    09-24-90        Ron Janetzke
C                       Extend the beginning of the set from $ to !.
CCCCCC
C
        LOGICAL lkywd
        CHARACTER*1 c

        if (ichar(c) .ge. ichar('!') .and. ichar(c) .le. ichar('+')
     &  .or.ichar(c) .ge. ichar('-') .and. ichar(c) .le. ichar('9')
     &  .or.ichar(c) .ge. ichar('@') .and. ichar(c) .le. ichar('_'))
     &  then
          lkywd = .true.
        else
          lkywd = .false.
        end if

        return
        end
CCCCCCCCCCC
CCCCCCCCCCC
        function lmdfr(c)
CCCCCC
C       NAME: lmdfr
C
C       PURPOSE: Compare c to the modifier secondary character set for RDTOK.
C
C       ARGUMENTS:
C               input:
C               c = CHARACTER*1 to be used for the compare.
C
C               output:
C               lmdfr = LOGICAL return value.
C
C       EXAMPLE 1:
C               c = 'A'
C               lmdfr = .true.
C
C       REFERENCES:
C               none
C
C       CHANGES:
C               1.00    08-25-90        Ron Janetzke
C                       Original text.
C               1.01    09-24-90        Ron Janetzke
C                       Extend the beginning of the set from $ to !.
CCCCCC
C
        LOGICAL lmdfr
        CHARACTER*1 c

        if (ichar(c) .ge. ichar('!') .and. ichar(c) .le. ichar('+')
```

```
     &  .or.ichar(c) .ge. ichar('-') .and. ichar(c) .le. ichar('9')
     &  .or.ichar(c) .ge. ichar('@') .and. ichar(c) .le. ichar('_'))
     &  then
          lmdfr = .true.
        else
          lmdfr = .false.
        end if

        return
        end
CCCCCCCCCCC
CCCCCCCCCCC
        function lmdin(c)
CCCCCC
C       NAME: lmdin
C
C       PURPOSE: Compare c to the modifier initiator character set for RDTOK.
C
C       ARGUMENTS:
C               input:
C               c = CHARACTER*1 to be used for the compare.
C
C               output:
C               lmdin = LOGICAL return value.
C
C       EXAMPLE 1:
C               c = 'A'
C               lmdin = .true.
C
C       REFERENCES:
C               none
C
C       CHANGES:
C               1.00    08-25-90        Ron Janetzke
C                       Original text.
CCCCCC
C
        LOGICAL lmdin
        CHARACTER*1 c

        if (ichar(c).ge.ichar('A') .and. ichar(c).le.ichar('Z')) then
          lmdin = .true.
        else
          lmdin = .false.
        end if

        return
        end
CCCCCCCCCCC
CCCCCCCCCCC
        function lnmbr(c)
CCCCCC
C       NAME: lnmbr
C
C       PURPOSE: Compare c to the number secondary character set for RDTOK.
C
C       ARGUMENTS:
C               input:
C               c = CHARACTER*1 to be used for the compare.
C
C               output:
C               lnmbr = LOGICAL return value.
C
C       EXAMPLE 1:
```

```
C               c = '0'
C               lnmbr = .true.
C
C       REFERENCES:
C               none
C
C       CHANGES:
C               1.00    08-25-90        Ron Janetzke
C                       Original text.
CCCCCC
C
        LOGICAL lnmbr
        CHARACTER*1 c

        if (ichar(c) .ge. ichar('$') .and. ichar(c) .le. ichar('9')
     &  .or.ichar(c) .ge. ichar('@') .and. ichar(c) .le. ichar('_'))
     &  then
          lnmbr = .true.
        else
          lnmbr = .false.
        end if

        return
        end
CCCCCCCCCCC
CCCCCCCCCCC
        function lperd(c)
CCCCCC
C       NAME: lperd
C
C       PURPOSE: Compare c to the period (.) for RDTOK.
C
C       ARGUMENTS:
C               input:
C               c = CHARACTER*1 to be used for the compare.
C
C               output:
C               lperd = LOGICAL return value.
C
C       EXAMPLE 1:
C               c = '.'
C               lperd = .true.
C
C       REFERENCES:
C               none
C
C       CHANGES:
C               1.00    08-25-90        Ron Janetzke
C                       Original text.
CCCCCC
C
        LOGICAL lperd
        CHARACTER*1 c

        if (c .eq. '.') then
          lperd = .true.
        else
          lperd = .false.
        end if

        return
        end
CCCCCCCCCCC
CCCCCCCCCCC
```

```
      function lsepr(c)
CCCCCC
C     NAME: lsepr
C
C     PURPOSE: Compare c to the separator characters (SPACE, COMMA, COLON,
C               TAB, and EQUAL) for RDTOK.
C
C     ARGUMENTS:
C             input:
C             c = CHARACTER*1 to be used for the compare.
C
C             output:
C             lsepr = LOGICAL return value.
C
C     EXAMPLE 1:
C             c = ','
C             lsepr = .true.
C
C     REFERENCES:
C             none
C
C     CHANGES:
C             1.00    08-25-90        Ron Janetzke
C                     Original text.
C             1.01    09-24-90        Ron Janetzke
C                     Add equal sign to list of separators.
C             1.02    02-05-91        Ron Janetzke
C                     Add TAB to list of separators.
CCCCCC
C
      LOGICAL lsepr
      CHARACTER*1 c

      if (c .eq. ' ' .or. c .eq. ',' .or. c .eq. ':' .or.
     &    c .eq. char(9) .or. c .eq. '=') then
        lsepr = .true.
      else
        lsepr = .false.
      end if

      return
      end
CCCCCCCCCCC
CCCCCCCCCCC
      function lsign(c)
CCCCCC
C     NAME: lsign
C
C     PURPOSE: Compare c to the numeric sign characters (+,-)
C               for RDTOK.
C
C     ARGUMENTS:
C             input:
C             c = CHARACTER*1 to be used for the compare.
C
C             output:
C             lsign = LOGICAL return value.
C
C     EXAMPLE 1:
C             c = '-'
C             lsign = .true.
C
C     REFERENCES:
C             none
```

```
C
C       CHANGES:
C               1.00    08-25-90        Ron Janetzke
C                       Original text.
CCCCCC
C
        LOGICAL lsign
        CHARACTER*1 c

        if (c .eq. '+' .or. c .eq. '-') then
          lsign = .true.
        else
          lsign = .false.
        end if

        return
        end
CCCCCCCCCCC
CCCCCCCCCCC
        function lvbar(c)
CCCCCC
C       NAME: lvbar
C
C       PURPOSE: Compare c to the end of line character (|)
C                for RDTOK.
C
C       ARGUMENTS:
C               input:
C               c = CHARACTER*1 to be used for the compare.
C
C               output:
C               lvbar = LOGICAL return value.
C
C       EXAMPLE 1:
C               c = '|'
C               lvbar = .true.
C
C       REFERENCES:
C               none
C
C       CHANGES:
C               1.00    08-25-90        Ron Janetzke
C                       Original text.
CCCCCC
C
        LOGICAL lvbar
        CHARACTER*1 c

        if (c .eq. '|') then
          lvbar = .true.
        else
          lvbar = .false.
        end if

        return
        end
CCCCCCCCCCC
      function exist (c,lenmst,master,elepos)
CCCCCC
C       Name: exist
C
C       Purpose:
C                This routine is part of a suite of routines used with
C                the data structures required for RDFREE.  It is for two
```

```
C                 different purposes.  The first is as a function to check
C                 a newly found token by the RDFREE routine to see if it
C                 is a member of the master keyword list.  The second is
C                 as a function for the application program to check for
C                 a specific modifier in the keyword/modifier (mdfyr) list
C                 returned from RDFREE.  This routine searches
C                 the array 'master' for an element equal to 'c'.  It then
C                 returns the element number in 'elepos'.  NOTE: 'c' must
C                 be CHARACTER*n, where n is greater than or equal to 6.
C
C       ARGUMENTS:
C                 input:
C                c       = CHARACTER*6 string to be matched to element of master.
C                 lenmst  = INTEGER length of the 'master' keyword array.
C                 master  = CHARACTER*6 array of keywords to be searched.
C
C                 output:
C                 elepos  = INTEGER element number of first matched string,
C                           0 if string is not found to match any in array master.
C                 exist   = LOGICAL if true then string was found, if false
C                           the string was not found.
C
C       REFERENCES:
C                 none
C
C       HISTORY:
C                 1.00    09-04-90        Ron Janetzke
C                         Original text.
C                 1.01    03-20-91        Ron Janetzke
C                         Error message for incorrect length of first argument.
C                 2.00    05-05-91        Ron Janetzke
C                         Change to function for version 2.00 of RDFREE.
CCCCCC
      LOGICAL        exist
      CHARACTER*(*)  c
      INTEGER        lenmst
      CHARACTER*6    master(lenmst)
      INTEGER        elepos
CCCCCC
      CHARACTER*6  temp
      INTEGER i
CCCCCC
C     Start here.
CCCCCC
      if (len(c) .lt. 6) then
         write (*,'(x,a76/x,a)')
     &         'The length of the first argument in call EXIST '//
     &         'must be at least CHARACTER*6.', c
         stop
      end if
      exist = .FALSE.
      temp = c
      elepos = 0
      if (lenmst .gt. 0) then
         do 199 i=1,lenmst
            if (temp .eq. master(i)) then
               elepos = i
               exist = .TRUE.
               go to 999
            end if
  199    continue
      end if
  999 continue
      return
```

end