```
!  This program tries to automate most of the conversion of a program to use the FM package
!  for multiple precision computation.

!  To convert everything would require almost as much syntax analysis as a full compiler.
!  Instead, the most commonly used Fortran features are handled, by using some shortcuts
!  and simplifying assumptions about the syntax.

!  C2FM.INP is the input file  -- the non multiple precision program.
!  C2FM.OUT is the output file -- the FM version.
!  Several other files are used for temporary information, these can be ignored.

!  Because not all the variables in the original program may need to be multiple precision,
!  this conversion program uses the convention that the constant in the KIND part of the
!  type declaration statement is a code telling when to convert a variable to multiple precision.
!  Make a copy of the original program and call it C2FM.INP, then change each declaration there
!  for which the variables will be multiple precision in the new version to one of the following
!  three types.  Use exactly these types, with all upper case letters and one blank.

!  REAL (KIND(3.1D1))     will be converted to  TYPE (FM),
!  COMPLEX (KIND(3.1D1))  will be converted to  TYPE (ZM),
!  INTEGER (KIND(31))     will be converted to  TYPE (IM).

!  For an existing program, decide which variables need to become multiple precision, and
!  change each declaration to one of these.  In the common case of a program that will not
!  need multiple precision complex or integers, and all reals are double precision before
!  the conversion, this can usually be done with one global change in an editor.

!  Calls to functions DBLE, FLOAT, REAL, SNGL that are in lines that reference multiple precision
!  variables or routines will be converted to calls to TO_FM.  This could give error messages if
!  a REAL call in the original program uses the optional KIND argument, and those will have to
!  be fixed by hand.

!  Any multiple precision functions defined in your program should be declared EXTERNAL along
!  with the proper kind code.  Omitting the external attribute might result in a compiler error
!  message because of trying to initialize it.

!  If your program uses variables that will become multiple precision in a complicated way, then
!  Convert2FM is likely to miss some changes.  These will have to be found and changed by hand,
!  but most of the time they will cause compiler error messages that show where the missed
!  changes were.  For example, if your program has multiple precision components of derived types
!  or pointer aliases to multiple precision variables, Convert2FM might miss some changes.

!  The most time-consuming parts of conversion are usually changing constants,
!     X = Y/3.7   should become   X = Y/TO_FM('3.7'),
!  changing type declarations,
!     REAL (KIND(1.0D0)) :: X, Y   should become   TYPE (FM), SAVE :: X, Y
!  and inserting FM interface instructions like
!     USE FMZM
!     CALL FM_ENTER_USER_ROUTINE
!     CALL FM_EXIT_USER_ROUTINE.
!  Convert2FM should be able to do these automatically.

!  Converting read/write statements is done by converting between machine precision and multiple
!  precision, so that any associated formats do not have to be changed.  These will often need
```

```
!  to be changed by hand to get the desired multiple precision formats.  This is especially true
!  for TYPE (IM) numbers, since the machine precision integer overflow threshold is very low,
!  so writing TO_INT(K) is seldom correct.

!  For example,
!      WRITE (KW,120) N,H,TOL
!  becomes
!      WRITE (KW,120) N,TO_DP(H),TO_DP(TOL)
!  after Convert2FM, which writes the multiple precision values H and TOL only to double
!  precision accuracy.

!  Reads are similar but messier:
!      READ (KR,*) (X(L),L=1,N)
!  becomes
!      ALLOCATE( C2FM_TEMP(1:N) )
!      READ (KR,*) (C2FM_TEMP(L),L=1,N)
!      DO L = 1, N
!         X(L) = C2FM_TEMP(L)
!      ENDDO
!      DEALLOCATE(C2FM_TEMP)
!  where X is type (fm) in the new version.  C2FM_TEMP is a double precision variable in module
!  C2FM_READS that Convert2FM puts at the start of the new program.


      MODULE C2FM_VARS


!  These are the global variables for C2FM.

!  LINE contains one subprogram at a time while the input file is analyzed and converted to FM.

!  ROUTINE_NAME     is the name of the current routine.

!  ROUTINE_TYPE = 1 for the main program
!               = 2 for subroutines
!               = 3 for functions
!               = 4 for modules

!  N_ARGS is the number of arguments in the current routine.

!  ROUTINE_ARGUMENT(K) is the variable name of the Kth argument in the current routine.

!  ARGUMENT_TYPE(K) = 0 if that variable will not become multiple precision
!                   = 1 for TYPE (FM)
!                   = 2 for TYPE (ZM)
!                   = 3 for TYPE (IM)

!  N_VARS is the number of variables in the current routine.

!  ROUTINE_VARIABLE(K) is the variable name of the Kth variable in the current routine that
!                      will become multiple precision.

!  VARIABLE_INITIALIZATION(K) gives any initialization expression in the declaration of
!                             the original program.  These must be made into executable
!                             statements and put at the top of the routine.
```

```
!  VARIABLE_LOCATION(K) records whether that variable is an input argument, a module variable,
!                       or a local variable.

!  ARRAY_SIZE(K) gives the size of a multiple precision variable array.

!  VARIABLE_TYPE(K) = 0, 1, 2, 3 gives these local variable types as with ARGUMENT_TYPE.

!  The next five arrays form a database showing which routine input arguments are multiple
!  precision types.

!  N_ROUTINES is the number of routines in the program.

!  R_NAME(J) is the routine name of the Jth subprogram in the sorted list of names.

!  R_TYPE(J) is the routine type of the Jth subprogram.

!  R_N_ARGS(J) is the number of arguments for the Jth routine.

!  R_ARG_START(J) gives the starting location of the list of argument types for routine J.

!  R_ARGS( R_ARG_START(J) : R_ARG_START(J) + R_N_ARGS(J) - 1 ) gives the argument type of each
!          input argument to the Jth routine, using the same codes as ARGUMENT_TYPE.

!  The next five arrays form a database showing which modules are used.

!  R_N_USED(J) is the number of USE statements in the Jth routine.

!  R_USED_START(J) gives the starting location of the list of used modules for routine J.

!  R_USED( R_USED_START(J) : R_USED_START(J) + R_N_USED(J) - 1 ) gives the module routine names
!          used in the Jth routine.

!  R_N_ALL_USED(J) is the number of modules used in the Jth routine, including indirect usage
!          where a directly used module uses another module, etc.

!  R_ALL_USED( J , 1 : R_N_ALL_USED(J) ) gives the module routine numbers directly and indirectly
!          used in the Jth routine.

!  AFTER_READ holds new conversion statements used after the new read statement gets double
!          precision inputs and then these statements convert those values to multiple precision.


      CHARACTER(132),  SAVE :: LINE(10000)
      CHARACTER(32),   SAVE :: ROUTINE_NAME,ROUTINE_ARGUMENT(100),ROUTINE_VARIABLE(1000)
      CHARACTER(32),   SAVE :: R_NAME(1000),R_USED(10000)
      CHARACTER(132),  SAVE :: VARIABLE_INITIALIZATION(1000)
      CHARACTER(9999), SAVE :: TEMP_STATEMENT,TEMP_ST
      CHARACTER(8),    SAVE :: VARIABLE_LOCATION(1000)
      CHARACTER(132),  SAVE :: AFTER_READ(100)
      INTEGER, SAVE :: ARGUMENT_TYPE(100),VARIABLE_TYPE(1000),ARRAY_SIZE(1000),ROUTINE_TYPE,     &
                       N_ARGS,N_VARS,QUOTE_LEVEL(9999)
      INTEGER, SAVE :: N_ROUTINES,R_TYPE(1000),R_N_ARGS(1000),R_ARG_START(1000),R_ARGS(100000),  &
                       R_N_USED(1000),R_USED_START(1000),R_N_ALL_USED(1000),N_AFTER_READ,        &
                       N_AFTER_ALLOC
      INTEGER, SAVE, DIMENSION(:,:), ALLOCATABLE :: R_ALL_USED
```

```
      END MODULE C2FM_VARS



      PROGRAM C2FM

      USE C2FM_VARS
      IMPLICIT NONE
      INTEGER :: FIRST,J,LAST
      LOGICAL :: FILE_ENDED

      OPEN(22,FILE='C2FM.INP')
      OPEN(23,FILE='C2FM.OUT')
      OPEN(31,FILE='C2FM.ARGS')
      OPEN(32,FILE='C2FM.VARS')
      OPEN(33,FILE='C2FM.MODS')
      OPEN(34,FILE='C2FM.PASS1')
      OPEN(35,FILE='C2FM.VARS2')
      OPEN(36,FILE='C2FM.PASS2')

!             Not all programs use modules.  Write a blank MODS file to erase any information
!             left over from previous conversions.

      WRITE (33,*) ' '
      CLOSE(33)
      OPEN(33,FILE='C2FM.MODS')

!             Make a first pass over the program to find out which arguments to each subprogram
!             will be multiple precision.

      CALL GET_ARGUMENTS

!             Sort the R_NAME database by routine names to speed up later searches.

      CALL SORT_NAMES

!             Expand the R_NAME database by adding any modules used by other modules to
!             the list of used modules in each routine.

      CALL USE_CHECK

!             Make a second variable list for each routine, including any module variables
!             that are used in each routine.

      CALL VAR2_LIST

!             The second pass adds code to initialize the multiple precision variables and
!             record entry and exit to routines so that multiple precision temporary variables
!             do not get discarded too soon.

      REWIND(31)
      REWIND(32)
      REWIND(33)
      REWIND(34)
      REWIND(35)
```

```
      DO J = 1, 10000

!             Read the next routine.

         CALL READ_ROUTINE(34,FIRST,LAST,FILE_ENDED)
         IF (FILE_ENDED .AND. LAST <= 0) EXIT

!             Get the information about local multiple precision variables.

         CALL LOCAL_VARS(32)

!             Write the next routine to the output file.

         CALL WRITE_PASS2(LAST)

         IF (FILE_ENDED) EXIT

      ENDDO

!             The third pass converts floating point constants that appear in expressions
!             involving multiple precision variables, and also converts read, write, print,
!             and allocate statements.

      REWIND(31)
      REWIND(32)
      REWIND(33)
      REWIND(34)
      REWIND(35)
      REWIND(36)

!             Put a module with two allocatable arrays at the top of the FM version of the program.
!             These are used by the new code that handles read statements.

      WRITE (23,"(A)") ' '
      WRITE (23,"(A)") '      MODULE C2FM_READS'
      WRITE (23,"(A)") '         REAL (KIND(1.0D0)), DIMENSION(:),   ALLOCATABLE :: C2FM_TEMP'
      WRITE (23,"(A)") '         REAL (KIND(1.0D0)), DIMENSION(:,:), ALLOCATABLE :: C2FM_TEMP2'
      WRITE (23,"(A)") '      END MODULE C2FM_READS'
      WRITE (23,"(A)") ' '

      DO J = 1, 10000

!             Read the next routine.

         CALL READ_ROUTINE(36,FIRST,LAST,FILE_ENDED)
         IF (FILE_ENDED .AND. LAST <= 0) STOP

!             Get the information about local multiple precision variables.

         CALL LOCAL_VARS(35)

!             Write the next routine to the output file.

         CALL WRITE_PASS3(LAST)
```

```
         IF (FILE_ENDED) STOP

      ENDDO

      END PROGRAM C2FM

      SUBROUTINE ALL_CAPS(STRING,N,RESULT)

!  Convert STRING(1:N) to all upper case and return it an RESULT(1:N).

      IMPLICIT NONE
      INTEGER :: N
      CHARACTER(N) :: STRING,RESULT
      CHARACTER(26), PARAMETER :: LOWER_CASE = 'abcdefghijklmnopqrstuvwxyz',  &
                                  UPPER_CASE = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ'
      INTEGER :: K,KA

      RESULT(1:N) = STRING(1:N)
      DO K = 1, N
         KA = INDEX(LOWER_CASE,STRING(K:K))
         IF (KA > 0) RESULT(K:K) = UPPER_CASE(KA:KA)
      ENDDO

      END SUBROUTINE ALL_CAPS

      SUBROUTINE CONVERT_CONSTANTS(START_OF_STATEMENT,LAST_NONBLANK,END_OF_STATEMENT,J_START,J_END)

!  TEMP_STATEMENT contains the statement.
!  Look for an executable statement where constants should be converted to
!  multiple precision, and insert calls to TO_FM, etc.
!
!  START_OF_STATEMENT is the location of the first nonblank character in the statement.
!  LAST_NONBLANK is the location of the last nonblank character in the statement,
!                including any trailing comments after !.
!  END_OF_STATEMENT is the location of the last nonblank character in the non-comment
!                   part of the statement.
!  J_START and J_END sometimes limit the columns where constants will be changed.

      USE C2FM_VARS
      IMPLICIT NONE
      CHARACTER(63), PARAMETER :: NAME_CHARS =  &
        'ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz_0123456789'
      CHARACTER(32), SAVE :: VAR_NAME,NAME_OF_ROUTINE
      INTEGER :: END_OF_STATEMENT,EXP_CHAR,START_OF_STATEMENT,J,J1,JA,JB,J_START,J_END,K,KA,KB,  &
                 K_END,L,L2,NARG,LAST_NONBLANK,SKIP_UNTIL,LEVEL_P,LEVEL_Q1,LEVEL_Q2,INDEX_ULC
      LOGICAL :: IN_NAME,IN_STRING,IN_CONSTANT,IN_CALL,INTEGER_CONSTANT,  &
                 DIGIT_IN_CONSTANT,ROUTINE_CALL,FM_VAR_FOUND,PREVIOUS_CHAR_NAME

!             See if any multiple precision variable names appear in this statement.
!             Also look for calls to functions or subroutines.

      ROUTINE_CALL = .FALSE.
      FM_VAR_FOUND = .FALSE.
      IN_NAME = .FALSE.
      K_END = J_END
      DO J = START_OF_STATEMENT, END_OF_STATEMENT+1
```

```
        IF (.NOT. IN_NAME) THEN
            J1 = INDEX(NAME_CHARS,TEMP_STATEMENT(J:J))
            IF (J1 > 0 .AND. J1 < 53) THEN
                IN_NAME = .TRUE.
                JA = J
            ENDIF
        ELSE
            J1 = INDEX(NAME_CHARS,TEMP_STATEMENT(J:J))
            IF (J1 == 0) THEN
                IN_NAME = .FALSE.
                JB = J - 1
                VAR_NAME = ' '
                CALL ALL_CAPS(TEMP_STATEMENT(JA:JB),JB-JA+1,VAR_NAME)
                DO K = 1, N_VARS
                   IF (VAR_NAME == ROUTINE_VARIABLE(K)) THEN
                       FM_VAR_FOUND = .TRUE.
                       EXIT
                   ENDIF
                ENDDO
                DO K = 1, N_ROUTINES
                   IF (VAR_NAME == R_NAME(K)) THEN
                       ROUTINE_CALL = .TRUE.
                       NAME_OF_ROUTINE = VAR_NAME
                       EXIT
                   ENDIF
                ENDDO
            ENDIF
        ENDIF
      ENDDO

      IF (.NOT.(FM_VAR_FOUND .OR. ROUTINE_CALL)) RETURN

!             Look for any calls to DBLE, FLOAT, REAL, SNGL.
!             Convert them to TO_FM.

      IN_NAME = .FALSE.
      J = START_OF_STATEMENT
  110    IF (.NOT. IN_NAME) THEN
            J1 = INDEX(NAME_CHARS,TEMP_STATEMENT(J:J))
            IF (J1 > 0 .AND. J1 < 53) THEN
                IN_NAME = .TRUE.
                JA = J
            ENDIF
        ELSE
            J1 = INDEX(NAME_CHARS,TEMP_STATEMENT(J:J))
            IF (J1 == 0) THEN
                IN_NAME = .FALSE.
                JB = J - 1
                VAR_NAME = ' '
                CALL ALL_CAPS(TEMP_STATEMENT(JA:JB),JB-JA+1,VAR_NAME)
                IF (VAR_NAME == 'DBLE' .OR. VAR_NAME == 'REAL' .OR. VAR_NAME == 'SNGL') THEN
                    DO K = LAST_NONBLANK, JB+1, -1
                       TEMP_STATEMENT(K+1:K+1) = TEMP_STATEMENT(K:K)
                    ENDDO
                    TEMP_STATEMENT(JA:JB+1) = 'TO_FM'
                    LAST_NONBLANK = LAST_NONBLANK + 1
```

```
                    END_OF_STATEMENT = END_OF_STATEMENT + 1
                ENDIF
                IF (VAR_NAME == 'FLOAT') THEN
                    TEMP_STATEMENT(JA:JB) = 'TO_FM'
                ENDIF
            ENDIF
        ENDIF
      J = J + 1
      IF (J <= END_OF_STATEMENT+1) GO TO 110

!             Scan for constants to convert.
!             All non-integer constants are converted, except in format statements.

      IN_NAME = .FALSE.
      IN_STRING = .FALSE.
      IN_CONSTANT = .FALSE.
      IN_CALL = .FALSE.
      SKIP_UNTIL = 0
      J = START_OF_STATEMENT
  120    IF (SKIP_UNTIL > 0 .AND. J < SKIP_UNTIL) GO TO 140
         IF (J < J_START .OR. J > K_END) GO TO 140
         IF (.NOT. (IN_NAME .OR. IN_STRING .OR. IN_CONSTANT)) THEN
             J1 = INDEX(NAME_CHARS,TEMP_STATEMENT(J:J))
             IF (J1 > 0 .AND. J1 < 53) THEN
                 IN_NAME = .TRUE.
                 JA = J
             ELSE IF (J1 > 53 .OR. TEMP_STATEMENT(J:J) == '.') THEN
                 IN_CONSTANT = .TRUE.
                 JA = J
                 EXP_CHAR = 0
                 IF (J1 > 53) THEN
                     INTEGER_CONSTANT = .TRUE.
                     DIGIT_IN_CONSTANT = .TRUE.
                 ELSE
                     INTEGER_CONSTANT = .FALSE.
                     DIGIT_IN_CONSTANT = .FALSE.
                 ENDIF
             ELSE IF (TEMP_STATEMENT(J:J) == "'") THEN
                 IN_STRING = .TRUE.
                 JA = J
             ELSE IF (TEMP_STATEMENT(J:J) == '"') THEN
                 IN_STRING = .TRUE.
                 JA = J
             ENDIF
         ELSE IF (IN_NAME) THEN
             J1 = INDEX(NAME_CHARS,TEMP_STATEMENT(J:J))
             IF (J1 == 0) THEN
                 IN_NAME = .FALSE.
                 JB = J - 1
                 VAR_NAME = TEMP_STATEMENT(JA:JB)
                 IF (INDEX_ULC(VAR_NAME,'FORMAT') == 1 .OR. INDEX_ULC(VAR_NAME,'READ')  == 1 .OR.  &
                     INDEX_ULC(VAR_NAME,'WRITE')  == 1 .OR. INDEX_ULC(VAR_NAME,'PRINT') == 1) THEN
                     GO TO 150
                 ENDIF
                 DO K = 1, N_ROUTINES
                    IF (VAR_NAME == R_NAME(K)) THEN
```

```
                    ROUTINE_CALL = .TRUE.
                    NAME_OF_ROUTINE = VAR_NAME
                    IN_CALL = .TRUE.
                    EXIT
                ENDIF
            ENDDO
        ENDIF
    ELSE IF (IN_STRING) THEN
        IF (TEMP_STATEMENT(J:J) == TEMP_STATEMENT(JA:JA)) THEN
            IN_STRING = .FALSE.
        ENDIF
    ELSE IF (IN_CONSTANT) THEN
        J1 = INDEX(NAME_CHARS,TEMP_STATEMENT(J:J))
        IF (J1 > 53) THEN
            DIGIT_IN_CONSTANT = .TRUE.
            IF (J >= END_OF_STATEMENT) THEN
                J = J + 1
                GO TO 130
            ENDIF
            GO TO 140
        ENDIF
        IF (TEMP_STATEMENT(J:J) == '.') THEN
            INTEGER_CONSTANT = .FALSE.
            IF (J >= END_OF_STATEMENT) THEN
                J = J + 1
                GO TO 130
            ENDIF
            GO TO 140
        ENDIF
        IF (EXP_CHAR == 0) THEN
            IF (TEMP_STATEMENT(J:J) == 'E' .OR. TEMP_STATEMENT(J:J) == 'e' .OR.  &
                TEMP_STATEMENT(J:J) == 'D' .OR. TEMP_STATEMENT(J:J) == 'd') THEN
                EXP_CHAR = J
                INTEGER_CONSTANT = .FALSE.
                IF (J >= END_OF_STATEMENT) THEN
                    J = J + 1
                    GO TO 130
                ENDIF
                GO TO 140
            ENDIF
        ENDIF
        IF ((TEMP_STATEMENT(J:J) == '+' .OR. TEMP_STATEMENT(J:J) == '-') .AND.  &
            EXP_CHAR > 0 .AND. J == EXP_CHAR+1) THEN
            IF (J >= END_OF_STATEMENT) THEN
                J = J + 1
                GO TO 130
            ENDIF
            GO TO 140
        ENDIF

130     JB = J - 1
        IN_CONSTANT = .FALSE.
        IF (.NOT. DIGIT_IN_CONSTANT) GO TO 140

!        End of constant found, expand the line and convert it.
```

```
            IF (.NOT. INTEGER_CONSTANT) THEN
                DO K = LAST_NONBLANK, JB+1, -1
                   TEMP_STATEMENT(K+9:K+9) = TEMP_STATEMENT(K:K)
                ENDDO
                TEMP_STATEMENT(JB+8:JB+9) = "')"
                DO K = JB, JA, -1
                   TEMP_STATEMENT(K+7:K+7) = TEMP_STATEMENT(K:K)
                ENDDO
                TEMP_STATEMENT(JA:JA+6) = "TO_FM('"
                SKIP_UNTIL = JB + 10
                K_END = MIN(9999,K_END+9)
                LAST_NONBLANK = LAST_NONBLANK + 9
                END_OF_STATEMENT = END_OF_STATEMENT + 9
            ENDIF
        ENDIF
  140 J = J + 1
      IF (J <= END_OF_STATEMENT+1) GO TO 120

!             Convert complex constants.

!             At this point, a complex constant like  (1.23,4.56)  will have been converted
!             to  (TO_FM('1.23'),TO_FM('4.56'))  if any multiple precision variables also
!             appear in this statement.  Now this needs to become
!             CMPLX(TO_FM('1.23'),TO_FM('4.56')) so it will be converted to type ZM.

  150 PREVIOUS_CHAR_NAME = .FALSE.
      SKIP_UNTIL = 0
      J1 = 0
      J = 1

  160    IF (SKIP_UNTIL > 0 .AND. J < SKIP_UNTIL) GO TO 170
         IF (J < J_START .OR. J > K_END) GO TO 170
         IF (TEMP_STATEMENT(J:J) == ' ') GO TO 170
         IF (.NOT. PREVIOUS_CHAR_NAME .AND. J1 == 0) THEN
             IF (TEMP_STATEMENT(J:J) == '(') THEN
                 J1 = 1
                 JA = J
                 GO TO 170
             ENDIF
         ENDIF
         IF (J1 == 1 .OR. J1 == 3) THEN
             IF (INDEX_ULC(TEMP_STATEMENT(J:J+5),'TO_FM(') == 1) THEN
                 J1 = J1 + 1
                 J = J + 6
                 GO TO 160
             ELSE
                 IF (TEMP_STATEMENT(J:J) == '+' .OR. TEMP_STATEMENT(J:J) == '-') THEN
                     GO TO 170
                 ELSE
                     J1 = 0
                     PREVIOUS_CHAR_NAME = .FALSE.
                     GO TO 160
                 ENDIF
             ENDIF
         ENDIF
         IF (J1 == 2) THEN
```

```
            IF (TEMP_STATEMENT(J:J) == ',') THEN
                J1 = 3
                GO TO 170
            ELSE
                GO TO 170
            ENDIF
        ENDIF
        IF (J1 == 4 .OR. J1 == 5) THEN
            IF (TEMP_STATEMENT(J:J) == ')') THEN
                J1 = J1 + 1
                GO TO 170
            ELSE
                GO TO 170
            ENDIF
        ENDIF

        IF (INDEX(NAME_CHARS,TEMP_STATEMENT(J:J)) > 0) THEN
            PREVIOUS_CHAR_NAME = .TRUE.
        ELSE
            PREVIOUS_CHAR_NAME = .FALSE.
        ENDIF

  170   IF (J1 == 6) THEN
            JB = J

!            End of complex constant found, expand the line and convert it.

            DO K = LAST_NONBLANK, JA, -1
               TEMP_STATEMENT(K+5:K+5) = TEMP_STATEMENT(K:K)
            ENDDO
            TEMP_STATEMENT(JA:JA+4) = "CMPLX"
            SKIP_UNTIL = JB + 6
            K_END = MIN(9999,K_END+5)
            LAST_NONBLANK = LAST_NONBLANK + 5
            END_OF_STATEMENT = END_OF_STATEMENT + 5

            J1 = 0
            PREVIOUS_CHAR_NAME = .FALSE.
        ENDIF

      J = J + 1
      IF (J <= END_OF_STATEMENT+1) GO TO 160

!            Now that constants in this statement have been converted to multiple precision,
!            look for any that are arguments to a function or subroutine and are not multiple
!            precision arguments.

      IF (INDEX_ULC(TEMP_STATEMENT(1:END_OF_STATEMENT),'TO_FM(') <= 0 .AND.  &
          INDEX_ULC(TEMP_STATEMENT(1:END_OF_STATEMENT),'TO_IM(') <= 0) RETURN

      DO J = 1, N_ROUTINES
         KA = INDEX_ULC(TEMP_STATEMENT(1:END_OF_STATEMENT),TRIM(R_NAME(J)))
  180    IF (KA <= 0) CYCLE
         KB = KA + LEN_TRIM(R_NAME(J)) - 1

!            Make sure the match is the full name and not a substring.
```

```
      IF (KA > 1) THEN
          IF (INDEX(NAME_CHARS,TEMP_STATEMENT(KA-1:KA-1)) > 0) CYCLE
      ENDIF
      IF (KB < END_OF_STATEMENT) THEN
          IF (INDEX(NAME_CHARS,TEMP_STATEMENT(KB+1:KB+1)) > 0) CYCLE
      ELSE
          RETURN
      ENDIF

!             Scan the argument list.

      LEVEL_P = 0
      LEVEL_Q1 = 0
      LEVEL_Q2 = 0
      NARG = 0
      JA = 0
      JB = -1
      DO K = KB+1, END_OF_STATEMENT
         IF (TEMP_STATEMENT(K:K) == '(' .AND. LEVEL_Q1 == 0 .AND. LEVEL_Q2 == 0) THEN
             LEVEL_P = LEVEL_P + 1
             IF (LEVEL_P == 1) THEN
                 JA = K + 1
                 NARG = 1
             ENDIF
         ENDIF
         IF (TEMP_STATEMENT(K:K) == ')' .AND. LEVEL_Q1 == 0 .AND. LEVEL_Q2 == 0) THEN
             LEVEL_P = LEVEL_P - 1
             IF (LEVEL_P == 0) THEN
                 JB = K - 1
             ENDIF
         ENDIF
         IF (TEMP_STATEMENT(K:K) == '"' .AND. LEVEL_Q1 == 0) LEVEL_Q2 = 1 - LEVEL_Q2
         IF (TEMP_STATEMENT(K:K) == "'" .AND. LEVEL_Q2 == 0) LEVEL_Q1 = 1 - LEVEL_Q1
         IF (TEMP_STATEMENT(K:K) == ',' .AND. LEVEL_Q1 == 0 .AND. LEVEL_Q2 == 0) THEN
             IF (LEVEL_P == 1) THEN
                 JB = K - 1
             ENDIF
         ENDIF
         IF (JB >= JA) THEN
             IF (R_ARGS(R_ARG_START(J)+NARG-1) <= 0) THEN
                 DO L = JA, JB-9
                    IF (TEMP_STATEMENT(L:L+6) == "TO_FM('") THEN
                        TEMP_STATEMENT(L:L+6) =  &
                        CHAR(0)//CHAR(0)//CHAR(0)//CHAR(0)//CHAR(0)//CHAR(0)//CHAR(0)
                        DO L2 = L, JB
                           IF (TEMP_STATEMENT(L2:L2+1) == "')") THEN
                               TEMP_STATEMENT(L2:L2+1) = CHAR(0)//CHAR(0)
                               EXIT
                           ENDIF
                        ENDDO
                    ENDIF
                    IF (TEMP_STATEMENT(L:L+6) == "TO_IM('") THEN
                        TEMP_STATEMENT(L:L+6) =  &
                        CHAR(0)//CHAR(0)//CHAR(0)//CHAR(0)//CHAR(0)//CHAR(0)//CHAR(0)
                        DO L2 = L, JB
```

```
                              IF (TEMP_STATEMENT(L2:L2+1) == "')") THEN
                                  TEMP_STATEMENT(L2:L2+1) = CHAR(0)//CHAR(0)
                                  EXIT
                              ENDIF
                           ENDDO
                        ENDIF
                     ENDDO
                  ENDIF
                  JA = K + 1
                  NARG = NARG + 1
               ENDIF
               IF (LEVEL_P == 0) EXIT
            ENDDO

!              See if there are more references to this function in this statement.

            IF (JB > 1 .AND. JB < END_OF_STATEMENT) THEN
                KA = INDEX_ULC(TEMP_STATEMENT(JB:END_OF_STATEMENT),TRIM(R_NAME(J)))
                IF (KA > 0) THEN
                    KA = KA + JB - 1
                    GO TO 180
                ENDIF
            ENDIF
         ENDDO

!              Re-pack the statement if any TO_FM or TO_IM calls were removed.

         JA = 0
         DO K = 1, LAST_NONBLANK
            IF (TEMP_STATEMENT(K:K) /= CHAR(0)) THEN
                JA = JA + 1
                IF (K /= JA) TEMP_STATEMENT(JA:JA) = TEMP_STATEMENT(K:K)
            ENDIF
         ENDDO

         END_OF_STATEMENT = END_OF_STATEMENT - (LAST_NONBLANK - JA)
         LAST_NONBLANK = JA

         END SUBROUTINE CONVERT_CONSTANTS

         SUBROUTINE CONVERT_READS(LAST_NONBLANK,END_OF_STATEMENT)

!  TEMP_STATEMENT contains the statement.
!  Look for an executable statement where a read statement reads multiple precision variables,
!  and convert to the new form.
!
!  LAST_NONBLANK is the location of the last nonblank character in the statement,
!                including any trailing comments after !.
!  END_OF_STATEMENT is the location of the last nonblank character in the non-comment
!                   part of the statement.
!
!  The new read statement uses the same format as the original, and reads machine precision values.
!  Then new code converts to multiple precision.
!      READ (KR,*) (X(L),L=1,N)
!  becomes
!      ALLOCATE( C2FM_TEMP(1:N) )
```

```
!      READ (KR,*) (C2FM_TEMP(L),L=1,N)
!      DO L = 1, N
!         X(L) = C2FM_TEMP(L)
!      ENDDO
!      DEALLOCATE(C2FM_TEMP)
!  where X is type (fm) in the new version.

      USE C2FM_VARS
      IMPLICIT NONE
      CHARACTER(63), PARAMETER :: NAME_CHARS =  &
        'ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz_0123456789'
      CHARACTER(32),  SAVE :: VAR_NAME
      CHARACTER(64),  SAVE :: READ_VAR_FM(100),READ_DP_VAR(100),READ_VAR
      CHARACTER(132), SAVE :: READ_SUB_FM1,READ_SUB_FM2
      INTEGER :: END_OF_STATEMENT,LAST_NONBLANK,ID1(6),ID2(6),J,J1,JA,JB,JC,JD,K,KA,KB,  &
                 K_VARIABLE,K_SECTION1,K_SECTION2,L,LAST_NB,P_LEVEL,Q1_LEVEL,Q2_LEVEL,INDEX_ULC,  &
                 N_VARS_FOUND,N_C2FM,BEGIN_READ_VAR_FM(100),END_READ_VAR_FM(100)
      LOGICAL :: IN_NAME,FM_VAR_FOUND

      N_AFTER_READ = 0
      TEMP_ST = TEMP_STATEMENT(1:END_OF_STATEMENT)
      KA = INDEX_ULC(TEMP_ST(1:END_OF_STATEMENT),'READ')
      IF (KA <= 0) RETURN

!             Scan backward from the keyword.  If there is a non-blank character before it,
!             make sure it is a digit (statement label) or right parentheses (end of logical if).

      DO J = KA-1, 1, -1
         IF (TEMP_ST(J:J) == ' ') CYCLE
         IF (INDEX('0123456789',TEMP_ST(J:J)) > 0) EXIT
         IF (TEMP_ST(J:J) == ')') EXIT
         RETURN
      ENDDO

!             Scan forward from the keyword.  The next field should be a format string or statement
!             label, followed by a comma ( '(e15.3,i5)', or 120, ), or a list with unit number,
!             format, ..., enclosed in parentheses ( (KW,120,end=555), or (6,"(e15.3,i5)") ).

      DO J = KA+4, END_OF_STATEMENT
         IF (TEMP_ST(J:J) == ' ') CYCLE
         IF (INDEX('0123456789',TEMP_ST(J:J)) > 0) THEN
             DO K = J+1, END_OF_STATEMENT
                IF (TEMP_ST(K:K) == ' ') CYCLE
                IF (INDEX('0123456789',TEMP_ST(K:K)) > 0) CYCLE
                IF (TEMP_ST(K:K) == ',') THEN
                    KB = K + 1
                    GO TO 110
                ENDIF
             ENDDO
             RETURN
         ENDIF
         IF (TEMP_ST(J:J) == '"') THEN
             DO K = J+1, END_OF_STATEMENT
                IF (TEMP_ST(K:K) == '"') THEN
                    DO L = K+1, END_OF_STATEMENT
                       IF (TEMP_ST(L:L) == ' ') CYCLE
```

```
                    IF (TEMP_ST(L:L) == ',') THEN
                        KB = L + 1
                        GO TO 110
                    ENDIF
                ENDDO
            ENDIF
        ENDDO
        RETURN
    ENDIF
    IF (TEMP_ST(J:J) == "'") THEN
        DO K = J+1, END_OF_STATEMENT
            IF (TEMP_ST(K:K) == "'") THEN
                DO L = K+1, END_OF_STATEMENT
                    IF (TEMP_ST(L:L) == ' ') CYCLE
                    IF (TEMP_ST(L:L) == ',') THEN
                        KB = L + 1
                        GO TO 110
                    ENDIF
                ENDDO
            ENDIF
        ENDDO
        RETURN
    ENDIF
    IF (TEMP_ST(J:J) == "(") THEN
        Q1_LEVEL = 0
        Q2_LEVEL = 0
        DO K = J+1, END_OF_STATEMENT
            IF (TEMP_ST(K:K) == "'" .AND. Q2_LEVEL == 0) Q1_LEVEL = 1 - Q1_LEVEL
            IF (TEMP_ST(K:K) == '"' .AND. Q1_LEVEL == 0) Q2_LEVEL = 1 - Q2_LEVEL
            IF (TEMP_ST(K:K) == ')' .AND. Q1_LEVEL == 0 .AND. Q2_LEVEL == 0) THEN
                KB = K + 1
                GO TO 110
            ENDIF
        ENDDO
        RETURN
    ENDIF
ENDDO
RETURN

!             Now scan the i/o list starting at position KB and look for names of
!             multiple precision variables.

  110 JA = 0
    JB = 0
    N_AFTER_READ = 0
    N_VARS_FOUND = 0
    N_C2FM = 0
    READ_SUB_FM1 = ' '
    READ_SUB_FM2 = ' '
    LAST_NB = LAST_NONBLANK
    IN_NAME = .FALSE.
    FM_VAR_FOUND = .FALSE.
    Q1_LEVEL = 0
    Q2_LEVEL = 0
    DO J = KB, END_OF_STATEMENT
        IF (TEMP_ST(J:J) == "'" .AND. Q2_LEVEL == 0) Q1_LEVEL = 1 - Q1_LEVEL
```

```
        IF (TEMP_ST(J:J) == '"' .AND. Q1_LEVEL == 0) Q2_LEVEL = 1 - Q2_LEVEL
        IF (Q1_LEVEL == 1 .OR. Q2_LEVEL == 1) CYCLE
        IF (.NOT. IN_NAME) THEN
            J1 = INDEX(NAME_CHARS,TEMP_ST(J:J))
            IF (J1 > 0) THEN
                IN_NAME = .TRUE.
                JA = J
                IF (J == END_OF_STATEMENT) THEN
                    JB = J
                    VAR_NAME = ' '
                    CALL ALL_CAPS(TEMP_ST(JA:JB),JB-JA+1,VAR_NAME)
                    DO K = 1, N_VARS
                       IF (VAR_NAME == ROUTINE_VARIABLE(K)) THEN
                           FM_VAR_FOUND = .TRUE.
                           K_VARIABLE = K
                           EXIT
                       ENDIF
                    ENDDO
                ENDIF
            ENDIF
        ELSE
            J1 = INDEX(NAME_CHARS,TEMP_ST(J:J))
            IF (J1 == 0 .OR. J == END_OF_STATEMENT) THEN
                IN_NAME = .FALSE.
                IF (J1 == 0) THEN
                    JB = J - 1
                ELSE
                    JB = J
                ENDIF
                VAR_NAME = ' '
                CALL ALL_CAPS(TEMP_ST(JA:JB),JB-JA+1,VAR_NAME)
                DO K = 1, N_VARS
                   IF (VAR_NAME == ROUTINE_VARIABLE(K)) THEN
                       FM_VAR_FOUND = .TRUE.
                       K_VARIABLE = K
                       EXIT
                   ENDIF
                ENDDO
            ENDIF
        ENDIF

        IF (FM_VAR_FOUND) THEN
            JC = JB

!             Check for a subscript after the name.

!             K_SECTION1 is positive if the "subscript" is really an array section.
!             Only 1 or 2 dimensional sections and only the simplest forms are handled.
!             For example, a( 2:n:2 ) or a( j , : ) are not recognized.

            P_LEVEL = 0
            K_SECTION1 = 0
            K_SECTION2 = 0
            ID1 = 0
            ID2 = 0
            DO K = JB+1, END_OF_STATEMENT
```

```
            IF (TEMP_ST(K:K) == ' ') CYCLE
            IF (TEMP_ST(K:K) == "'" .AND. Q2_LEVEL == 0) Q1_LEVEL = 1 - Q1_LEVEL
            IF (TEMP_ST(K:K) == '"' .AND. Q1_LEVEL == 0) Q2_LEVEL = 1 - Q2_LEVEL
            IF (Q1_LEVEL == 1 .OR. Q2_LEVEL == 1) CYCLE
            IF (TEMP_ST(K:K) == ':' .AND. Q1_LEVEL == 0 .AND. Q2_LEVEL == 0 .AND.  &
                P_LEVEL == 1) THEN
                IF (K_SECTION1 == 0) THEN
                    K_SECTION1 = K
                    ID1(4) = K - 1
                    ID1(5) = K + 1
                ELSE IF (K_SECTION2 == 0) THEN
                    K_SECTION2 = K
                    ID1(4) = K - 1
                    ID1(5) = K + 1
                ELSE
                    K_SECTION1 = 0
                ENDIF
                CYCLE
            ENDIF
            IF (TEMP_ST(K:K) == '(' .AND. Q1_LEVEL == 0 .AND. Q2_LEVEL == 0) THEN
                P_LEVEL = P_LEVEL + 1
                IF (P_LEVEL == 1) ID1(3) = K + 1
                CYCLE
            ENDIF
            IF (P_LEVEL == 0) EXIT
            IF (TEMP_ST(K:K) == ')' .AND. Q1_LEVEL == 0 .AND. Q2_LEVEL == 0) THEN
                P_LEVEL = P_LEVEL - 1
                IF (P_LEVEL == 0) THEN
                    IF (K_SECTION2 > 0) THEN
                        ID2(6) = K - 1
                    ELSE IF (K_SECTION1 > 0) THEN
                        ID1(6) = K - 1
                    ENDIF
                    JC = K
                    EXIT
                ENDIF
            ENDIF
        ENDDO

!             Look for an implied do after the subscript.
!             This will also handle two implied do loops after a 2-dimensional array.

        JD = JC
        IF (JC > JB .AND. K_SECTION1 == 0) THEN
            DO K = JC+1, END_OF_STATEMENT
                IF (TEMP_ST(K:K) == ' ') CYCLE
                IF (TEMP_ST(K:K) == ',') THEN
                    CALL IMPLIED_DO(TEMP_ST,END_OF_STATEMENT,K,ID1,JD)
                    IF (JD > JC) THEN
                        DO L = JD+1, END_OF_STATEMENT
                            IF (TEMP_ST(L:L) == ' ') CYCLE
                            IF (TEMP_ST(L:L) == ',') THEN
                                CALL IMPLIED_DO(TEMP_ST,END_OF_STATEMENT,L,ID2,JD)
                            ENDIF
                            EXIT
                        ENDDO
```

```
                ENDIF
              ENDIF
              EXIT
            ENDDO
          ENDIF

!           Replace the multiple precision variable name in the read list by a temporary double
!           precision variable, and generate new statements to convert the d.p. results to
!           multiple precision.

          IF (VARIABLE_TYPE(K_VARIABLE) == 1) THEN
            N_VARS_FOUND = N_VARS_FOUND + 1
            N_C2FM = N_C2FM + 1
            IF (JC == JB) THEN
              IF (READ_SUB_FM1(1:1) == ' ') THEN
                READ_VAR_FM(N_C2FM) = TEMP_ST(JA:JC)
                BEGIN_READ_VAR_FM(N_C2FM) = JA
                END_READ_VAR_FM(N_C2FM) = JC
                READ_SUB_FM1(1:1) = '1'
                READ_DP_VAR(N_C2FM) = 'C2FM_TEMP('//TRIM(READ_SUB_FM1)//')'
              ELSE
                READ_VAR_FM(N_C2FM) = TEMP_ST(JA:JC)
                BEGIN_READ_VAR_FM(N_C2FM) = JA
                END_READ_VAR_FM(N_C2FM) = JC
                READ_SUB_FM1 = TRIM(READ_SUB_FM1)//'+1'
                READ_DP_VAR(N_C2FM) = 'C2FM_TEMP('//TRIM(READ_SUB_FM1)//')'
              ENDIF
            ELSE IF (K_SECTION1 <= 0) THEN
              IF (ID1(1) <= 0) THEN
                IF (READ_SUB_FM1(1:1) == ' ') THEN
                  READ_VAR_FM(N_C2FM) = TEMP_ST(JA:JC)
                  BEGIN_READ_VAR_FM(N_C2FM) = JA
                  END_READ_VAR_FM(N_C2FM) = JC
                  READ_SUB_FM1(1:1) = '1'
                  READ_DP_VAR(N_C2FM) = 'C2FM_TEMP('//TRIM(READ_SUB_FM1)//')'
                ELSE
                  READ_VAR_FM(N_C2FM) = TEMP_ST(JA:JC)
                  BEGIN_READ_VAR_FM(N_C2FM) = JA
                  END_READ_VAR_FM(N_C2FM) = JC
                  READ_SUB_FM1 = TRIM(READ_SUB_FM1)//'+1'
                  READ_DP_VAR(N_C2FM) = 'C2FM_TEMP('//TRIM(READ_SUB_FM1)//')'
                ENDIF
              ELSE IF (ID2(1) <= 0) THEN
                IF (READ_SUB_FM1(1:1) == ' ') THEN
                  READ_VAR_FM(N_C2FM) = TEMP_ST(JA:JC)
                  BEGIN_READ_VAR_FM(N_C2FM) = JA
                  END_READ_VAR_FM(N_C2FM) = JC
                  IF (TEMP_ST(ID1(3):ID1(4)) == '1') THEN
                    READ_SUB_FM1 = TEMP_ST(ID1(5):ID1(6))
                    READ_DP_VAR(N_C2FM) = 'C2FM_TEMP'//TEMP_ST(JB+1:JC)
                  ELSE
                    READ_SUB_FM1 = TEMP_ST(ID1(5):ID1(6))//'-'//  &
                                   TEMP_ST(ID1(3):ID1(4))//'+1'
                    READ_DP_VAR(N_C2FM) = 'C2FM_TEMP'//TEMP_ST(JB+1:JC)
                  ENDIF
                ELSE
```

```
                READ_VAR_FM(N_C2FM) = TEMP_ST(JA:JC)
                BEGIN_READ_VAR_FM(N_C2FM) = JA
                END_READ_VAR_FM(N_C2FM) = JC
                IF (TEMP_ST(ID1(3):ID1(4)) == '1') THEN
                    READ_DP_VAR(N_C2FM) = 'C2FM_TEMP'//TEMP_ST(JB+1:JC-1)//'+'//  &
                                          TRIM(READ_SUB_FM1)//')'
                    READ_SUB_FM1 = TRIM(READ_SUB_FM1)//'+'//TEMP_ST(ID1(5):ID1(6))
                ELSE
                    READ_DP_VAR(N_C2FM) = 'C2FM_TEMP'//TEMP_ST(JB+1:JC-1)//'+'//  &
                                          TRIM(READ_SUB_FM1)//')'
                    READ_SUB_FM1 = TRIM(READ_SUB_FM1)//'+'//TEMP_ST(ID1(5):ID1(6))//  &
                                   '-'//TEMP_ST(ID1(3):ID1(4))//'+1'
                ENDIF
            ENDIF
        ELSE
            IF (READ_SUB_FM2(1:1) == ' ') THEN
                READ_VAR_FM(N_C2FM) = TEMP_ST(JA:JC)
                BEGIN_READ_VAR_FM(N_C2FM) = JA
                END_READ_VAR_FM(N_C2FM) = JC
                IF (TEMP_ST(ID1(3):ID1(4)) == '1' .AND.  &
                    TEMP_ST(ID2(3):ID2(4)) == '1') THEN
                    READ_SUB_FM2 = TEMP_ST(ID1(5):ID1(6))//','//TEMP_ST(ID2(5):ID2(6))
                    READ_DP_VAR(N_C2FM) = 'C2FM_TEMP2'//TEMP_ST(JB+1:JC)
                ELSE
                    READ_SUB_FM2 = TEMP_ST(ID1(5):ID1(6))//'-'//        &
                                   TEMP_ST(ID1(3):ID1(4))//'+1'//','//  &
                                   TEMP_ST(ID2(5):ID2(6))//'-'//        &
                                   TEMP_ST(ID2(3):ID2(4))//'+1'
                    READ_DP_VAR(N_C2FM) = 'C2FM_TEMP2'//TEMP_ST(JB+1:JC)
                ENDIF
            ELSE
                WRITE (*,*) ' '
                WRITE (*,*) ' Convert2FM warning!    Only one 2-dimensional'//  &
                            ' implied do can be handled in one read statement.'
                WRITE (*,*) ' This statement seems to have more than one:'
                WRITE (*,*) TRIM(TEMP_ST)
                WRITE (*,*) ' '
            ENDIF
        ENDIF
    ELSE IF (K_SECTION2 <= 0) THEN
        IF (READ_SUB_FM1(1:1) == ' ') THEN
            READ_VAR_FM(N_C2FM) = TEMP_ST(JA:JC)
            BEGIN_READ_VAR_FM(N_C2FM) = JA
            END_READ_VAR_FM(N_C2FM) = JC
            IF (TEMP_ST(ID1(3):ID1(4)) == '1') THEN
                READ_SUB_FM1 = TEMP_ST(ID1(5):ID1(6))
                READ_DP_VAR(N_C2FM) = 'C2FM_TEMP'//TEMP_ST(JB+1:JC)
            ELSE
                READ_DP_VAR(N_C2FM) = 'C2FM_TEMP'//TEMP_ST(JB+1:JB+1)//'1:'//  &
                                      TEMP_ST(ID1(5):ID1(6))//'-'//  &
                                      TEMP_ST(ID1(3):ID1(4))//'+1'//')'
                READ_SUB_FM1 = TEMP_ST(ID1(5):ID1(6))//'-'//  &
                               TEMP_ST(ID1(3):ID1(4))//'+1'
            ENDIF
        ELSE
            READ_VAR_FM(N_C2FM) = TEMP_ST(JA:JC)
```

```
                    BEGIN_READ_VAR_FM(N_C2FM) = JA
                    END_READ_VAR_FM(N_C2FM) = JC
                    IF (TEMP_ST(ID1(3):ID1(4)) == '1') THEN
                        READ_DP_VAR(N_C2FM) = 'C2FM_TEMP'//TEMP_ST(JB+1:JB+1)//  &
                                              TRIM(READ_SUB_FM1)//'+'//TEMP_ST(JB+2:JC-1)//  &
                                              '+'//TRIM(READ_SUB_FM1)//')'
                        READ_SUB_FM1 = TRIM(READ_SUB_FM1)//'+'//TEMP_ST(ID1(5):ID1(6))
                    ELSE
                        READ_DP_VAR(N_C2FM) = 'C2FM_TEMP'//TEMP_ST(JB+1:JB+1)//  &
                                              TRIM(READ_SUB_FM1)//'+'//'1:'//  &
                                              TEMP_ST(ID1(5):ID1(6))//'-'//  &
                                              TEMP_ST(ID1(3):ID1(4))//'+1'//  &
                                              '+'//TRIM(READ_SUB_FM1)//')'
                        READ_SUB_FM1 = TRIM(READ_SUB_FM1)//'+'//TEMP_ST(ID1(5):ID1(6))//  &
                                       '-'//TEMP_ST(ID1(3):ID1(4))//'+1'
                    ENDIF
                ENDIF
            ENDIF
            CALL READ_VAR_SUBSCRIPT(READ_DP_VAR(N_C2FM))

!           Record the new conversion statements that will be written after this
!           read statement.

            N_AFTER_READ = N_AFTER_READ + 1
            IF (ID1(1) == 0) THEN
                AFTER_READ(N_AFTER_READ) = '      '//TRIM(READ_VAR_FM(N_C2FM))//' = '//     &
                                           TRIM(READ_DP_VAR(N_C2FM))
            ELSE IF (ID2(1) == 0) THEN
                AFTER_READ(N_AFTER_READ) = '      '//'DO '//TEMP_ST(ID1(1):ID1(2))//        &
                                           ' = '//TEMP_ST(ID1(3):ID1(4))//', '//            &
                                           TEMP_ST(ID1(5):ID1(6))
                N_AFTER_READ = N_AFTER_READ + 1
                AFTER_READ(N_AFTER_READ) = '         '//TRIM(READ_VAR_FM(N_C2FM))//' = '//  &
                                           TRIM(READ_DP_VAR(N_C2FM))
                N_AFTER_READ = N_AFTER_READ + 1
                AFTER_READ(N_AFTER_READ) = '      ENDDO'
            ELSE
                AFTER_READ(N_AFTER_READ) = '      '//'DO '//TEMP_ST(ID1(1):ID1(2))//        &
                                           ' = '//TEMP_ST(ID1(3):ID1(4))//', '//            &
                                           TEMP_ST(ID1(5):ID1(6))
                N_AFTER_READ = N_AFTER_READ + 1
                AFTER_READ(N_AFTER_READ) = '         '//'DO '//TEMP_ST(ID2(1):ID2(2))//     &
                                           ' = '//TEMP_ST(ID2(3):ID2(4))//', '//            &
                                           TEMP_ST(ID2(5):ID2(6))
                N_AFTER_READ = N_AFTER_READ + 1
                AFTER_READ(N_AFTER_READ) = '            '//TRIM(READ_VAR_FM(N_C2FM))//      &
                                           ' = '//TRIM(READ_DP_VAR(N_C2FM))
                N_AFTER_READ = N_AFTER_READ + 1
                AFTER_READ(N_AFTER_READ) = '         ENDDO'
                N_AFTER_READ = N_AFTER_READ + 1
                AFTER_READ(N_AFTER_READ) = '      ENDDO'
            ENDIF
        ENDIF

        P_LEVEL = 0
        Q1_LEVEL = 0
```