

```

! FM_rational 1.4                                David M. Smith                                Rational Arithmetic

! This module extends the definition of basic Fortran arithmetic and function operations so
! they also apply to multiple precision rationals, using version 1.4 of FM.
! The multiple precision rational data type is called
!   TYPE (FM_RATIONAL)

! Each FM rational number A/B consists of two values, with A and B being TYPE(IM) integer multiple
! precision numbers.  Negative rationals are represented with A being negative.

! This module supports assignment, arithmetic, comparison, and functions involving FM_RATIONAL
! numbers.

! Mixed-mode operations, such as adding FM_RATIONAL to IM or machine integer types, are supported.
! In general, operations where both the arguments and results are mathematically rational (machine
! precision integers, TYPE(IM), or TYPE (FM_RATIONAL)) are supported, such as A = 1, A = B - 3, or
! A = B / X_IM, where A and B are FM_RATIONAL, and X_IM is type IM.

! Array operations are also supported, so A, B, and X_IM could be 1- or 2-dimensional arrays in the
! examples above.

! Mixed-mode comparisons are also supported, as with IF (A == 1), IF (A <= B - 3), or
! IF (A > B / X_IM).

! TO_FM_RATIONAL is a function for creating a number of type FM_RATIONAL.
! This function can have one argument, for the common case of creating a rational number with an
! integer value.  For TO_FM_RATIONAL(A), A can be a machine integer or array of integers, a type
! IM value or array, or a character string.

! There is also a two argument form, TO_FM_RATIONAL(A,B), that can be used to create the fraction
! A/B when A and B are both machine precision integers, TYPE(IM) multiple precision integers, or
! character strings representing integers.

! The one argument character form can be used with a single input string having both parts of the
! fraction present and separated by '/', as in TO_FM_RATIONAL(' 41 / 314 ').  This might be more
! readable than the equivalent forms TO_FM_RATIONAL( 41, 314 ) or TO_FM_RATIONAL( '41', '314' ).

! The TO_FM function from the basic FM package has been extended to convert a type FM_RATIONAL to
! a type FM number.  The result is an approximation accurate to the current FM precision.

! RATIONAL_APPROX(A, DIGITS) is a function that converts an FM number A to a rational approximation
! of type FM_RATIONAL that has no more than DIGITS decimal digits in the top and bottom.
! Ex:  A = pi, DIGITS = 2   returns the FM_RATIONAL result    22 /    7
!      A = pi, DIGITS = 3   returns the FM_RATIONAL result   355 /   113
!      A = pi, DIGITS = 6   returns the FM_RATIONAL result  833719 / 265381
! The rational result usually approximates A to about 2*DIGITS significant digits, so
! DIGITS should not be more than about half the precision carried for A.
! Ex:  833719 / 265381 = 3.141592653581077771204419306..., and agrees with pi to about 11 s.d.

! The standard Fortran functions that are available for FM_RATIONAL arguments are the ones that
! give exact rational results.  So if A and B are type FM_RATIONAL variables, A**B, EXP(A), etc.,
! are not provided since the results are not generally exact rational numbers.

```

! But INT(A), FLOOR(A), MAX(A,B), MOD(A,B), etc., do give rational result and are provided.

! AVAILABLE OPERATONS:

!
! =
! +
! -
! *
! /
! ** A ** J is provided for FM_RATIONAL A and machine integer J.
! ==
! /=
! <
! <=
! >
! >=
! ABS(A)
! CEILING(A)
! DIM(A,B) Positive difference. Returns A - B if A > B, zero if not.
! FLOOR(A)
! INT(A)
! IS_UNKNOWN(A) Returns true if A is unknown.
! MAX(A,B,...) Can have from 2 to 10 arguments.
! MIN(A,B,...) Can have from 2 to 10 arguments.
! MOD(A,B) Result is A - int(A/B) * B
! MODULO(A,B) Result is A - floor(A/B) * B
! NINT(A)

! Array operations for functions.

! ABS, CEILING, FLOOR, INT, and NINT can each have a 1- or 2-dimensional array argument.
! They will return a vector or matrix with the function applied to each element of the
! argument.

! DIM, MOD, MODULO work the same way, but the two array arguments for these functions must
! have the same size and shape.

! IS_UNKNOWN is a logical function that can be used with an array argument but does not return
! an array result. It returns "true" if any element of the input array is FM's special UNKNOWN
! value, which comes from undefined operations such as dividing by zero.

! Functions that operate only on arrays.

! DOT_PRODUCT(X,Y) Dot product of rank 1 vectors.
! MATMUL(X,Y) Matrix multiplication of arrays
! Cases for valid argument shapes:
! (1) (n,m) * (m,k) --> (n,k)
! (2) (m) * (m,k) --> (k)
! (3) (n,m) * (m) --> (n)
! MAXVAL(X) Maximum value in the array
! MINVAL(X) Minimum value in the array
! PRODUCT(X) Product of all values in the array
! SUM(X) Sum of all values in the array
! TRANSPOSE(X) Matrix transposition. If X is a rank 2 array with shape (n,m), then
! Y = TRANSPOSE(X) has shape (m,n) with Y(i,j) = X(j,i).

```
TYPE FM_RATIONAL
  TYPE(MULTI) :: NUMERATOR
  TYPE(MULTI) :: DENOMINATOR
END TYPE
```

! Work variables for derived type operations.

```
TYPE (IM), SAVE :: R_1, R_2, R_3, R_4, R_5, R_6
TYPE (FM), SAVE :: F_1, F_2
TYPE (FM_RATIONAL), SAVE :: MT_RM, MU_RM
INTEGER, SAVE :: RATIONAL_EXP_MAX = 0, RATIONAL_SKIP_MAX = 100
LOGICAL, SAVE :: SKIP_GCD = .FALSE.
```

```
INTERFACE TO_FM_RATIONAL
  MODULE PROCEDURE FM_RATIONAL_I
  MODULE PROCEDURE FM_RATIONAL_II
  MODULE PROCEDURE FM_RATIONAL_I1
  MODULE PROCEDURE FM_RATIONAL_I2
  MODULE PROCEDURE FM_RATIONAL_IM
  MODULE PROCEDURE FM_RATIONAL_IMIM
  MODULE PROCEDURE FM_RATIONAL_IM1
  MODULE PROCEDURE FM_RATIONAL_IM2
  MODULE PROCEDURE FM_RATIONAL_ST
  MODULE PROCEDURE FM_RATIONAL_STST
END INTERFACE
```

```
INTERFACE FM_UNDEF_INP
  MODULE PROCEDURE FM_UNDEF_INP_RATIONAL_RM0
  MODULE PROCEDURE FM_UNDEF_INP_RATIONAL_RM1
  MODULE PROCEDURE FM_UNDEF_INP_RATIONAL_RM2
END INTERFACE
```

```
INTERFACE RATIONAL_NUMERATOR
  MODULE PROCEDURE RATIONAL_NUMERATOR_IM
END INTERFACE
```

```
INTERFACE RATIONAL_DENOMINATOR
  MODULE PROCEDURE RATIONAL_DENOMINATOR_IM
END INTERFACE
```

```
INTERFACE ASSIGNMENT (=)
  MODULE PROCEDURE FMEQ_RATIONAL_RMRM
  MODULE PROCEDURE FMEQ_RATIONAL_RMI
  MODULE PROCEDURE FMEQ_RATIONAL_RMIM

  MODULE PROCEDURE FMEQ_RATIONAL_RM1RM
  MODULE PROCEDURE FMEQ_RATIONAL_RM1RM1
  MODULE PROCEDURE FMEQ_RATIONAL_RM1I
  MODULE PROCEDURE FMEQ_RATIONAL_RM1I1
  MODULE PROCEDURE FMEQ_RATIONAL_RM1IM
  MODULE PROCEDURE FMEQ_RATIONAL_RM1IM1

  MODULE PROCEDURE FMEQ_RATIONAL_RM2RM
  MODULE PROCEDURE FMEQ_RATIONAL_RM2RM2
  MODULE PROCEDURE FMEQ_RATIONAL_RM2I
  MODULE PROCEDURE FMEQ_RATIONAL_RM2I2
  MODULE PROCEDURE FMEQ_RATIONAL_RM2IM
  MODULE PROCEDURE FMEQ_RATIONAL_RM2IM2
```

END INTERFACE

CONTAINS

!

TO_FM_RATIONAL

FUNCTION FM_RATIONAL_I(TOP) RESULT (RETURN_VALUE)

USE FMVALS

IMPLICIT NONE

TYPE (FM_RATIONAL) :: RETURN_VALUE

INTEGER :: TOP,N1,N2

INTENT (IN) :: TOP

IF (TOP == 0) THEN

CALL IMST2M('0',RETURN_VALUE%NUMERATOR)

CALL IMST2M('1',RETURN_VALUE%DENOMINATOR)

RETURN

ENDIF

N1 = ABS(TOP)

N2 = 1

IF (TOP > 0) THEN

CALL IMI2M(N1,RETURN_VALUE%NUMERATOR)

ELSE

CALL IMI2M(-N1,RETURN_VALUE%NUMERATOR)

ENDIF

CALL IMI2M(N2,RETURN_VALUE%DENOMINATOR)

CALL FM_MAX_EXP_RM(RETURN_VALUE)

END FUNCTION FM_RATIONAL_I

FUNCTION FM_RATIONAL_II(TOP,BOT) RESULT (RETURN_VALUE)

USE FMVALS

IMPLICIT NONE

TYPE (FM_RATIONAL) :: RETURN_VALUE

INTEGER :: TOP,BOT,N1,N2

INTENT (IN) :: TOP,BOT

IF (BOT == 0) THEN

CALL IMST2M('UNKNOWN',RETURN_VALUE%NUMERATOR)

CALL IMST2M('UNKNOWN',RETURN_VALUE%DENOMINATOR)

RETURN

ENDIF

IF (TOP == 0) THEN

CALL IMST2M('0',RETURN_VALUE%NUMERATOR)

CALL IMST2M('1',RETURN_VALUE%DENOMINATOR)

RETURN

ENDIF

N1 = ABS(TOP)

N2 = ABS(BOT)

CALL FMGCDI(N1,N2)

IF ((TOP > 0 .AND. BOT > 0) .OR. (TOP < 0 .AND. BOT < 0)) THEN

CALL IMI2M(N1,RETURN_VALUE%NUMERATOR)

ELSE

CALL IMI2M(-N1,RETURN_VALUE%NUMERATOR)

ENDIF

CALL IMI2M(N2,RETURN_VALUE%DENOMINATOR)

CALL FM_MAX_EXP_RM(RETURN_VALUE)

END FUNCTION FM_RATIONAL_II

FUNCTION FM_RATIONAL_I1(IVAL) RESULT (RETURN_VALUE)

```

USE FMVALS
IMPLICIT NONE
INTEGER, DIMENSION(:) :: IVAL
TYPE (FM_RATIONAL), DIMENSION(SIZE(IVAL)) :: RETURN_VALUE
INTEGER :: J
DO J = 1, SIZE(IVAL)
    CALL IMI2M(IVAL(J),RETURN_VALUE(J)%NUMERATOR)
    CALL IMI2M(1,RETURN_VALUE(J)%DENOMINATOR)
    CALL FM_MAX_EXP_RM(RETURN_VALUE(J))
ENDDO
END FUNCTION FM_RATIONAL_I1

```

```

FUNCTION FM_RATIONAL_I2(IVAL)      RESULT (RETURN_VALUE)
USE FMVALS
IMPLICIT NONE
INTEGER, DIMENSION(:,:) :: IVAL
TYPE (FM_RATIONAL), DIMENSION(SIZE(IVAL,DIM=1),SIZE(IVAL,DIM=2)) :: RETURN_VALUE
INTEGER :: J,K
DO J = 1, SIZE(IVAL,DIM=1)
    DO K = 1, SIZE(IVAL,DIM=2)
        CALL IMI2M(IVAL(J,K),RETURN_VALUE(J,K)%NUMERATOR)
        CALL IMI2M(1,RETURN_VALUE(J,K)%DENOMINATOR)
        CALL FM_MAX_EXP_RM(RETURN_VALUE(J,K))
    ENDDO
ENDDO
END FUNCTION FM_RATIONAL_I2

```

```

FUNCTION FM_RATIONAL_IM(TOP)      RESULT (RETURN_VALUE)
USE FMVALS
IMPLICIT NONE
TYPE (FM_RATIONAL) :: RETURN_VALUE
INTEGER :: R_SIGN
TYPE (IM) :: TOP
INTENT (IN) :: TOP
CALL FM_UNDEF_INP(TOP)
CALL IMEQ(TOP%MIM,R_1%MIM)
CALL IMI2M(1,R_2%MIM)
IF (IS_UNKNOWN(R_1) .OR. IS_OVERFLOW(R_1)) THEN
    CALL IMST2M('UNKNOWN',RETURN_VALUE%NUMERATOR)
    CALL IMST2M('UNKNOWN',RETURN_VALUE%DENOMINATOR)
    RETURN
ENDIF
IF (R_1 == 0) THEN
    CALL IMST2M('0',RETURN_VALUE%NUMERATOR)
    CALL IMST2M('1',RETURN_VALUE%DENOMINATOR)
    RETURN
ENDIF
R_SIGN = 1
IF (R_1 < 0) THEN
    R_SIGN = -1
ENDIF
CALL IM_ABS(R_1,R_4)
CALL IM_ABS(R_2,R_5)
CALL FM_MAX_EXP_IM(R_4,R_5)
IF (SKIP_GCD .AND. MAX(R_4%MIM%MP(2),R_5%MIM%MP(2)) < RATIONAL_SKIP_MAX) THEN
    IF (R_SIGN == -1) R_4%MIM%MP(1) = -1
    CALL IMEQ(R_4%MIM,RETURN_VALUE%NUMERATOR)
    CALL IMEQ(R_5%MIM,RETURN_VALUE%DENOMINATOR)

```

```

ELSE
  CALL IM_GCD(R_4,R_5,R_3)
  CALL IM_DIV(R_4,R_3,R_1)
  CALL IM_DIV(R_5,R_3,R_2)
  IF (R_SIGN == -1) R_1%MIM%MP(1) = -1
  CALL IMEQ(R_1%MIM,RETURN_VALUE%NUMERATOR)
  CALL IMEQ(R_2%MIM,RETURN_VALUE%DENOMINATOR)
ENDIF
END FUNCTION FM_RATIONAL_IM

FUNCTION FM_RATIONAL_IMIM(TOP,BOT)      RESULT (RETURN_VALUE)
  USE FMVALS
  IMPLICIT NONE
  TYPE (FM_RATIONAL) :: RETURN_VALUE
  INTEGER :: R_SIGN
  TYPE (IM) :: TOP,BOT
  INTENT (IN) :: TOP,BOT
  CALL FM_UNDEF_INP(TOP)
  CALL FM_UNDEF_INP(BOT)
  CALL IMEQ(TOP%MIM,R_1%MIM)
  CALL IMEQ(BOT%MIM,R_2%MIM)
  IF (R_2 == 0 .OR. IS_UNKNOWN(R_1) .OR. IS_OVERFLOW(R_1) .OR. &
      IS_UNKNOWN(R_2) .OR. IS_OVERFLOW(R_2) ) THEN
    CALL IMST2M('UNKNOWN',RETURN_VALUE%NUMERATOR)
    CALL IMST2M('UNKNOWN',RETURN_VALUE%DENOMINATOR)
    RETURN
  ENDIF
  IF (R_1 == 0) THEN
    CALL IMST2M('0',RETURN_VALUE%NUMERATOR)
    CALL IMST2M('1',RETURN_VALUE%DENOMINATOR)
    RETURN
  ENDIF
  R_SIGN = 1
  IF ((R_1 > 0 .AND. R_2 < 0) .OR. (R_1 < 0 .AND. R_2 > 0)) THEN
    R_SIGN = -1
  ENDIF
  CALL IM_ABS(R_1,R_4)
  CALL IM_ABS(R_2,R_5)
  CALL FM_MAX_EXP_IM(R_4,R_5)
  IF (SKIP_GCD .AND. MAX(R_4%MIM%MP(2),R_5%MIM%MP(2)) < RATIONAL_SKIP_MAX) THEN
    IF (R_SIGN == -1) R_4%MIM%MP(1) = -1
    CALL IMEQ(R_4%MIM,RETURN_VALUE%NUMERATOR)
    CALL IMEQ(R_5%MIM,RETURN_VALUE%DENOMINATOR)
  ELSE
    CALL IM_GCD(R_4,R_5,R_3)
    CALL IM_DIV(R_4,R_3,R_1)
    CALL IM_DIV(R_5,R_3,R_2)
    IF (R_SIGN == -1) R_1%MIM%MP(1) = -1
    CALL IMEQ(R_1%MIM,RETURN_VALUE%NUMERATOR)
    CALL IMEQ(R_2%MIM,RETURN_VALUE%DENOMINATOR)
  ENDIF
END FUNCTION FM_RATIONAL_IMIM

FUNCTION FM_RATIONAL_IM1(TOP)      RESULT (RETURN_VALUE)
  USE FMVALS
  IMPLICIT NONE
  TYPE (IM), DIMENSION(:) :: TOP
  TYPE (FM_RATIONAL), DIMENSION(SIZE(TOP)) :: RETURN_VALUE

```

```

INTEGER :: J
DO J = 1, SIZE(TOP)
  CALL IMEQ(TOP(J)%MIM, RETURN_VALUE(J)%NUMERATOR)
  CALL IMI2M(1, RETURN_VALUE(J)%DENOMINATOR)
  CALL FM_MAX_EXP_RM(RETURN_VALUE(J))
ENDDO
END FUNCTION FM_RATIONAL_IM1

FUNCTION FM_RATIONAL_IM2(TOP)      RESULT (RETURN_VALUE)
  USE FMVALS
  IMPLICIT NONE
  TYPE (IM), DIMENSION(:, :) :: TOP
  TYPE (FM_RATIONAL), DIMENSION(SIZE(TOP, DIM=1), SIZE(TOP, DIM=2)) :: RETURN_VALUE
  INTEGER :: J, K
  DO J = 1, SIZE(TOP, DIM=1)
    DO K = 1, SIZE(TOP, DIM=2)
      CALL IMEQ(TOP(J, K)%MIM, RETURN_VALUE(J, K)%NUMERATOR)
      CALL IMI2M(1, RETURN_VALUE(J, K)%DENOMINATOR)
      CALL FM_MAX_EXP_RM(RETURN_VALUE(J, K))
    ENDDO
  ENDDO
END FUNCTION FM_RATIONAL_IM2

FUNCTION FM_RATIONAL_ST(TOP)      RESULT (RETURN_VALUE)
  USE FMVALS
  IMPLICIT NONE
  TYPE (FM_RATIONAL) :: RETURN_VALUE
  INTEGER :: K, R_SIGN
  CHARACTER(*) :: TOP
  INTENT (IN) :: TOP
  K = INDEX(TOP, '/')
  IF (K > 0) THEN
    CALL IMST2M(TOP(1:K-1), R_1%MIM)
    CALL IMST2M(TOP(K+1:LEN(TOP)), R_2%MIM)
  ELSE
    CALL IMST2M(TOP, R_1%MIM)
    CALL IMI2M(1, R_2%MIM)
  ENDF
  IF (R_2 == 0 .OR. IS_UNKNOWN(R_1) .OR. IS_OVERFLOW(R_1) .OR. &
      IS_UNKNOWN(R_2) .OR. IS_OVERFLOW(R_2)) THEN
    CALL IMST2M('UNKNOWN', RETURN_VALUE%NUMERATOR)
    CALL IMST2M('UNKNOWN', RETURN_VALUE%DENOMINATOR)
    RETURN
  ENDF
  IF (R_1 == 0) THEN
    CALL IMST2M('0', RETURN_VALUE%NUMERATOR)
    CALL IMST2M('1', RETURN_VALUE%DENOMINATOR)
    RETURN
  ENDF
  R_SIGN = 1
  IF ((R_1 > 0 .AND. R_2 < 0) .OR. (R_1 < 0 .AND. R_2 > 0)) THEN
    R_SIGN = -1
  ENDF
  CALL IM_ABS(R_1, R_4)
  CALL IM_ABS(R_2, R_5)
  CALL FM_MAX_EXP_IM(R_4, R_5)
  IF (SKIP_GCD .AND. MAX(R_4%MIM%MP(2), R_5%MIM%MP(2)) < RATIONAL_SKIP_MAX) THEN
    IF (R_SIGN == -1) R_4%MIM%MP(1) = -1

```

```

CALL IMEQ(R_4%MIM,RETURN_VALUE%NUMERATOR)
CALL IMEQ(R_5%MIM,RETURN_VALUE%DENOMINATOR)
ELSE
CALL IM_GCD(R_4,R_5,R_3)
CALL IM_DIV(R_4,R_3,R_1)
CALL IM_DIV(R_5,R_3,R_2)
IF (R_SIGN == -1) R_1%MIM%MP(1) = -1
CALL IMEQ(R_1%MIM,RETURN_VALUE%NUMERATOR)
CALL IMEQ(R_2%MIM,RETURN_VALUE%DENOMINATOR)
ENDIF
END FUNCTION FM_RATIONAL_ST

FUNCTION FM_RATIONAL_STST(TOP,BOT)      RESULT (RETURN_VALUE)
USE FMVALS
IMPLICIT NONE
TYPE (FM_RATIONAL) :: RETURN_VALUE
INTEGER :: R_SIGN
CHARACTER(*) :: TOP,BOT
INTENT (IN) :: TOP,BOT
CALL IMST2M(TOP,R_1%MIM)
CALL IMST2M(BOT,R_2%MIM)
IF (R_2 == 0 .OR. IS_UNKNOWN(R_1) .OR. IS_OVERFLOW(R_1) .OR. &
    IS_UNKNOWN(R_2) .OR. IS_OVERFLOW(R_2) ) THEN
CALL IMST2M('UNKNOWN',RETURN_VALUE%NUMERATOR)
CALL IMST2M('UNKNOWN',RETURN_VALUE%DENOMINATOR)
RETURN
ENDIF
IF (R_1 == 0) THEN
CALL IMST2M('0',RETURN_VALUE%NUMERATOR)
CALL IMST2M('1',RETURN_VALUE%DENOMINATOR)
RETURN
ENDIF
R_SIGN = 1
IF ((R_1 > 0 .AND. R_2 < 0) .OR. (R_1 < 0 .AND. R_2 > 0)) THEN
R_SIGN = -1
ENDIF
CALL IM_ABS(R_1,R_4)
CALL IM_ABS(R_2,R_5)
CALL FM_MAX_EXP_IM(R_4,R_5)
IF (SKIP_GCD .AND. MAX(R_4%MIM%MP(2),R_5%MIM%MP(2)) < RATIONAL_SKIP_MAX) THEN
IF (R_SIGN == -1) R_4%MIM%MP(1) = -1
CALL IMEQ(R_4%MIM,RETURN_VALUE%NUMERATOR)
CALL IMEQ(R_5%MIM,RETURN_VALUE%DENOMINATOR)
ELSE
CALL IM_GCD(R_4,R_5,R_3)
CALL IM_DIV(R_4,R_3,R_1)
CALL IM_DIV(R_5,R_3,R_2)
IF (R_SIGN == -1) R_1%MIM%MP(1) = -1
CALL IMEQ(R_1%MIM,RETURN_VALUE%NUMERATOR)
CALL IMEQ(R_2%MIM,RETURN_VALUE%DENOMINATOR)
ENDIF
END FUNCTION FM_RATIONAL_STST

```

!

RATIONAL_NUMERATOR

```

FUNCTION RATIONAL_NUMERATOR_IM(MA)      RESULT (RETURN_VALUE)
USE FMVALS
IMPLICIT NONE

```



```

TYPE (FM_RATIONAL) :: MA
TYPE (IM) :: RETURN_VALUE
INTENT (IN) :: MA
CALL IMEQ(MA%NUMERATOR,RETURN_VALUE%MIM)
END FUNCTION RATIONAL_NUMERATOR_IM

```

!

RATIONAL_DENOMINATOR

```

FUNCTION RATIONAL_DENOMINATOR_IM(MA)      RESULT (RETURN_VALUE)
  USE FMVALS
  IMPLICIT NONE
  TYPE (FM_RATIONAL) :: MA
  TYPE (IM) :: RETURN_VALUE
  INTENT (IN) :: MA
  CALL IMEQ(MA%DENOMINATOR,RETURN_VALUE%MIM)
END FUNCTION RATIONAL_DENOMINATOR_IM

```

```

SUBROUTINE FM_MAX_EXP_RM(MA)
  USE FMVALS
  IMPLICIT NONE
  TYPE (FM_RATIONAL) :: MA
  INTEGER :: NT, NB
  NT = MA%NUMERATOR%MP(2)
  NB = MA%DENOMINATOR%MP(2)
  IF (NT < MEXPOV .AND. NT > RATIONAL_EXP_MAX) RATIONAL_EXP_MAX = NT
  IF (NB < MEXPOV .AND. NB > RATIONAL_EXP_MAX) RATIONAL_EXP_MAX = NB
END SUBROUTINE FM_MAX_EXP_RM

```

```

SUBROUTINE FM_MAX_EXP_IM(MA,MB)
  USE FMVALS
  IMPLICIT NONE
  TYPE (IM) :: MA,MB
  INTEGER :: NT
  NT = MA%MIM%MP(2)
  IF (NT < MEXPOV .AND. NT > RATIONAL_EXP_MAX) RATIONAL_EXP_MAX = NT
  NT = MB%MIM%MP(2)
  IF (NT < MEXPOV .AND. NT > RATIONAL_EXP_MAX) RATIONAL_EXP_MAX = NT
END SUBROUTINE FM_MAX_EXP_IM

```

!

FM_PRINT_RATIONAL

```

SUBROUTINE FM_PRINT_RATIONAL(MA)
  USE FMVALS
  IMPLICIT NONE
  TYPE (FM_RATIONAL) :: MA
  CHARACTER(100) :: ST1, ST2
  CHARACTER(203) :: STR
  INTENT (IN) :: MA
  INTEGER :: J,KPT

```

!

If the top and bottom integers can be printed on one line, as 12 / 7
in fewer than KSWIDE characters, do it. Otherwise call IMPRINT twice.

!

```

CALL IMABS(MA%NUMERATOR,R_1%MIM)
CALL IMABS(MA%DENOMINATOR,R_2%MIM)
CALL IMMPLY(R_1%MIM,R_2%MIM,R_3%MIM)

```

```

IF (TO_IM(10)**(KSWIDE-11) > R_3 .AND. R_1 < TO_IM('1E+99') .AND. R_2 < TO_IM('1E+99')) THEN

```

```

CALL IMFORM('I100',MA%NUMERATOR,ST1)
CALL IMFORM('I100',MA%DENOMINATOR,ST2)
STR = ' '
KPT = 0
DO J = 1, 100
  IF (ST1(J:J) /= ' ') THEN
    KPT = KPT + 1
    STR(KPT:KPT) = ST1(J:J)
  ENDIF
ENDDO
STR(KPT+1:KPT+3) = ' / '
KPT = KPT + 3
DO J = 1, 100
  IF (ST2(J:J) /= ' ') THEN
    KPT = KPT + 1
    STR(KPT:KPT) = ST2(J:J)
  ENDIF
ENDDO
IF (MA%NUMERATOR%MP(1) < 0) THEN
  WRITE (KW, "(6X,A)") STR(1:KPT)
ELSE
  WRITE (KW, "(7X,A)") STR(1:KPT)
ENDIF
ELSE
CALL IMPRINT(MA%NUMERATOR)
WRITE (KW, "(A)" ' ' /'
CALL IMPRINT(MA%DENOMINATOR)
ENDIF
END SUBROUTINE FM_PRINT_RATIONAL

```

```

SUBROUTINE FM_UNDEF_INP_RATIONAL_RM0(MA)
  USE FMVALS
  IMPLICIT NONE
  TYPE (FM_RATIONAL) :: MA
  INTENT (IN) :: MA
  IF (.NOT. ALLOCATED(MA%NUMERATOR%MP)) CALL FM_INPUT_ERROR
  IF (.NOT. ALLOCATED(MA%DENOMINATOR%MP)) CALL FM_INPUT_ERROR
END SUBROUTINE FM_UNDEF_INP_RATIONAL_RM0

```

```

SUBROUTINE FM_UNDEF_INP_RATIONAL_RM1(MA)
  USE FMVALS
  IMPLICIT NONE
  TYPE (FM_RATIONAL), DIMENSION(:) :: MA
  INTEGER :: J
  INTENT (IN) :: MA
  DO J = 1, SIZE(MA)
    IF (.NOT. ALLOCATED(MA(J)%NUMERATOR%MP)) CALL FM_INPUT_ERROR1(J)
    IF (.NOT. ALLOCATED(MA(J)%DENOMINATOR%MP)) CALL FM_INPUT_ERROR1(J)
  ENDDO
END SUBROUTINE FM_UNDEF_INP_RATIONAL_RM1

```

```

SUBROUTINE FM_UNDEF_INP_RATIONAL_RM2(MA)
  USE FMVALS
  IMPLICIT NONE
  TYPE (FM_RATIONAL), DIMENSION(:, :) :: MA
  INTEGER :: J, K
  INTENT (IN) :: MA

```

```

DO J = 1, SIZE(MA,DIM=1)
  DO K = 1, SIZE(MA,DIM=2)
    IF (.NOT. ALLOCATED(MA(J,K)%NUMERATOR%MP)) CALL FM_INPUT_ERROR2(J,K)
    IF (.NOT. ALLOCATED(MA(J,K)%DENOMINATOR%MP)) CALL FM_INPUT_ERROR2(J,K)
  ENDDO
ENDDO
END SUBROUTINE FM_UNDEF_INP_RATIONAL_RM2

```

```

SUBROUTINE FMEQ_RATIONAL(MA,MB)
  USE FMVALS
  IMPLICIT NONE
  TYPE (FM_RATIONAL) :: MA,MB
  INTENT (IN) :: MA
  INTENT (INOUT) :: MB
  CALL IMEQ(MA%NUMERATOR,MB%NUMERATOR)
  CALL IMEQ(MA%DENOMINATOR,MB%DENOMINATOR)
END SUBROUTINE FMEQ_RATIONAL

```

!

=

```

SUBROUTINE FMEQ_RATIONAL_RMRM(MA,MB)
  USE FMVALS
  IMPLICIT NONE
  TYPE (FM_RATIONAL) :: MA,MB
  INTENT (INOUT) :: MA
  INTENT (IN) :: MB
  CALL FM_UNDEF_INP(MB)
  CALL FMEQ_RATIONAL(MB,MA)
  CALL FM_MAX_EXP_RM(MA)
END SUBROUTINE FMEQ_RATIONAL_RMRM

```

```

SUBROUTINE FMEQ_RATIONAL_RMI(MA,IVAL)
  USE FMVALS
  IMPLICIT NONE
  TYPE (FM_RATIONAL) :: MA
  INTEGER :: IVAL
  INTENT (INOUT) :: MA
  INTENT (IN) :: IVAL
  CALL IMI2M(IVAL,MA%NUMERATOR)
  CALL IMI2M(1,MA%DENOMINATOR)
  CALL FM_MAX_EXP_RM(MA)
END SUBROUTINE FMEQ_RATIONAL_RMI

```

```

SUBROUTINE FMEQ_RATIONAL_RMIM(MA,MB)
  USE FMVALS
  IMPLICIT NONE
  TYPE (FM_RATIONAL) :: MA
  TYPE (IM) :: MB
  INTENT (INOUT) :: MA
  INTENT (IN) :: MB
  CALL FM_UNDEF_INP(MB)
  CALL IMEQ(MB%MIM,MA%NUMERATOR)
  CALL IMI2M(1,MA%DENOMINATOR)
  CALL FM_MAX_EXP_RM(MA)
END SUBROUTINE FMEQ_RATIONAL_RMIM

```

!

Array equal assignments for RM.

! (1) rank 1 = rank 0

```
SUBROUTINE FMEQ_RATIONAL_RM1I(MA,IVAL)
  USE FMVALS
  IMPLICIT NONE
  TYPE (FM_RATIONAL), DIMENSION(:) :: MA
  INTEGER :: IVAL,J
  INTENT (INOUT) :: MA
  INTENT (IN) :: IVAL
  DO J = 1, SIZE(MA)
    CALL FMEQ_RATIONAL_RMI(MA(J),IVAL)
  ENDDO
END SUBROUTINE FMEQ_RATIONAL_RM1I
```

```
SUBROUTINE FMEQ_RATIONAL_RM1RM(MA,MB)
  USE FMVALS
  IMPLICIT NONE
  TYPE (FM_RATIONAL), DIMENSION(:) :: MA
  TYPE (FM_RATIONAL) :: MB
  INTEGER :: J
  INTENT (INOUT) :: MA
  INTENT (IN) :: MB
  CALL FMEQ_RATIONAL_RMRM(MT_RM,MB)
  DO J = 1, SIZE(MA)
    CALL FMEQ_RATIONAL_RMRM(MA(J),MT_RM)
  ENDDO
END SUBROUTINE FMEQ_RATIONAL_RM1RM
```

```
SUBROUTINE FMEQ_RATIONAL_RM1IM(MA,MB)
  USE FMVALS
  IMPLICIT NONE
  TYPE (FM_RATIONAL), DIMENSION(:) :: MA
  TYPE (IM) :: MB
  INTEGER :: J
  INTENT (INOUT) :: MA
  INTENT (IN) :: MB
  CALL IMEQ(MB%MIM,R_1%MIM)
  DO J = 1, SIZE(MA)
    CALL FMEQ_RATIONAL_RMIM(MA(J),R_1)
  ENDDO
END SUBROUTINE FMEQ_RATIONAL_RM1IM
```

! (2) rank 1 = rank 1

```
SUBROUTINE FMEQ_RATIONAL_RM1I1(MA,IVAL)
  USE FMVALS
  IMPLICIT NONE
  TYPE (FM_RATIONAL), DIMENSION(:) :: MA
  INTEGER, DIMENSION(:) :: IVAL
  INTEGER :: J
  INTENT (INOUT) :: MA
  INTENT (IN) :: IVAL
  IF (SIZE(MA) /= SIZE(IVAL)) THEN
    CALL IMST2M('UNKNOWN',MT_RM%NUMERATOR)
    CALL IMST2M('UNKNOWN',MT_RM%DENOMINATOR)
    DO J = 1, SIZE(MA)
      CALL IMEQ(MT_RM%NUMERATOR,MA(J)%NUMERATOR)
      CALL IMEQ(MT_RM%DENOMINATOR,MA(J)%DENOMINATOR)
    ENDDO
  ENDIF
END SUBROUTINE FMEQ_RATIONAL_RM1I1
```

```

        ENDDO
        RETURN
    ENDIF
    DO J = 1, SIZE(MA)
        CALL FMEQ_RATIONAL_RMI(MA(J),IVAL(J))
    ENDDO
END SUBROUTINE FMEQ_RATIONAL_RM1I1

```

```

SUBROUTINE FMEQ_RATIONAL_RM1RM1(MA,MB)
    USE FMVALS
    IMPLICIT NONE
    TYPE (FM_RATIONAL), DIMENSION(:) :: MA
    TYPE (FM_RATIONAL), DIMENSION(:) :: MB
    TYPE (FM_RATIONAL), ALLOCATABLE, DIMENSION(:) :: TEMP
    INTEGER :: J,N
    INTENT (INOUT) :: MA
    INTENT (IN) :: MB
    IF (SIZE(MA) /= SIZE(MB)) THEN
        CALL IMST2M('UNKNOWN',MT_RM%NUMERATOR)
        CALL IMST2M('UNKNOWN',MT_RM%DENOMINATOR)
        DO J = 1, SIZE(MA)
            CALL IMEQ(MT_RM%NUMERATOR,MA(J)%NUMERATOR)
            CALL IMEQ(MT_RM%DENOMINATOR,MA(J)%DENOMINATOR)
        ENDDO
        RETURN
    ENDIF
    N = SIZE(MA)

```

! To avoid problems when lhs and rhs are overlapping parts of the same array, move MB
! to a temporary array before re-defining any of MA.

```

    ALLOCATE(TEMP(N))
    DO J = 1, N
        CALL IMEQ(MB(J)%NUMERATOR,TEMP(J)%NUMERATOR)
        CALL IMEQ(MB(J)%DENOMINATOR,TEMP(J)%DENOMINATOR)
    ENDDO
    DO J = 1, N
        CALL FMEQ_RATIONAL_RMRM(MA(J),TEMP(J))
    ENDDO
    DEALLOCATE(TEMP)
END SUBROUTINE FMEQ_RATIONAL_RM1RM1

```

```

SUBROUTINE FMEQ_RATIONAL_RM1IM1(MA,MB)
    USE FMVALS
    IMPLICIT NONE
    TYPE (FM_RATIONAL), DIMENSION(:) :: MA
    TYPE (IM), DIMENSION(:) :: MB
    TYPE (IM), ALLOCATABLE, DIMENSION(:) :: TEMP
    INTEGER :: J,N
    INTENT (INOUT) :: MA
    INTENT (IN) :: MB
    IF (SIZE(MA) /= SIZE(MB)) THEN
        CALL IMST2M('UNKNOWN',MT_RM%NUMERATOR)
        CALL IMST2M('UNKNOWN',MT_RM%DENOMINATOR)
        DO J = 1, SIZE(MA)
            CALL IMEQ(MT_RM%NUMERATOR,MA(J)%NUMERATOR)
            CALL IMEQ(MT_RM%DENOMINATOR,MA(J)%DENOMINATOR)
        ENDDO

```

```

        RETURN
    ENDIF
    N = SIZE(MA)
    ALLOCATE(TEMP(N))
    DO J = 1, SIZE(MA)
        CALL IMEQ(MB(J)%MIM,TEMP(J)%MIM)
    ENDDO
    DO J = 1, SIZE(MA)
        CALL FMEQ_RATIONAL_RMIM(MA(J),TEMP(J))
    ENDDO
    DEALLOCATE(TEMP)
END SUBROUTINE FMEQ_RATIONAL_RM1IM1

```

! (3) rank 2 = rank 0

```

SUBROUTINE FMEQ_RATIONAL_RM2I(MA,IVAL)
    USE FMVALS
    IMPLICIT NONE
    TYPE (FM_RATIONAL), DIMENSION(:,:) :: MA
    INTEGER :: IVAL,J,K
    INTENT (INOUT) :: MA
    INTENT (IN) :: IVAL
    DO J = 1, SIZE(MA,DIM=1)
        DO K = 1, SIZE(MA,DIM=2)
            CALL FMEQ_RATIONAL_RMI(MA(J,K),IVAL)
        ENDDO
    ENDDO
END SUBROUTINE FMEQ_RATIONAL_RM2I

```

```

SUBROUTINE FMEQ_RATIONAL_RM2RM(MA,MB)
    USE FMVALS
    IMPLICIT NONE
    TYPE (FM_RATIONAL), DIMENSION(:,:) :: MA
    TYPE (FM_RATIONAL) :: MB
    INTEGER :: J,K
    INTENT (INOUT) :: MA
    INTENT (IN) :: MB
    CALL FMEQ_RATIONAL_RMRM(MT_RM,MB)
    DO J = 1, SIZE(MA,DIM=1)
        DO K = 1, SIZE(MA,DIM=2)
            CALL FMEQ_RATIONAL_RMRM(MA(J,K),MT_RM)
        ENDDO
    ENDDO
END SUBROUTINE FMEQ_RATIONAL_RM2RM

```

```

SUBROUTINE FMEQ_RATIONAL_RM2IM(MA,MB)
    USE FMVALS
    IMPLICIT NONE
    TYPE (FM_RATIONAL), DIMENSION(:,:) :: MA
    TYPE (IM) :: MB
    INTEGER :: J,K
    INTENT (INOUT) :: MA
    INTENT (IN) :: MB
    CALL IMEQ(MB%MIM,R_1%MIM)
    DO J = 1, SIZE(MA,DIM=1)
        DO K = 1, SIZE(MA,DIM=2)
            CALL FMEQ_RATIONAL_RMIM(MA(J,K),R_1)
        ENDDO
    ENDDO

```

```
ENDDO
END SUBROUTINE FMEQ_RATIONAL_RM2IM
```

! (4) rank 2 = rank 2

```
SUBROUTINE FMEQ_RATIONAL_RM2I2(MA,IVAL)
  USE FMVALS
  IMPLICIT NONE
  TYPE (FM_RATIONAL), DIMENSION(:,:) :: MA
  INTEGER, DIMENSION(:,:) :: IVAL
  INTEGER :: J,K
  INTENT (INOUT) :: MA
  INTENT (IN) :: IVAL
  IF (SIZE(MA,DIM=1) /= SIZE(IVAL,DIM=1) .OR. SIZE(MA,DIM=2) /= SIZE(IVAL,DIM=2)) THEN
    CALL IMST2M('UNKNOWN',MT_RM%NUMERATOR)
    CALL IMST2M('UNKNOWN',MT_RM%DENOMINATOR)
    DO J = 1, SIZE(MA,DIM=1)
      DO K = 1, SIZE(MA,DIM=2)
        CALL IMEQ(MT_RM%NUMERATOR,MA(J,K)%NUMERATOR)
        CALL IMEQ(MT_RM%DENOMINATOR,MA(J,K)%DENOMINATOR)
      ENDDO
    ENDDO
    RETURN
  ENDF
  DO J = 1, SIZE(MA,DIM=1)
    DO K = 1, SIZE(MA,DIM=2)
      CALL FMEQ_RATIONAL_RMI(MA(J,K),IVAL(J,K))
    ENDDO
  ENDDO
END SUBROUTINE FMEQ_RATIONAL_RM2I2
```

```
SUBROUTINE FMEQ_RATIONAL_RM2RM2(MA,MB)
  USE FMVALS
  IMPLICIT NONE
  TYPE (FM_RATIONAL), DIMENSION(:,:) :: MA
  TYPE (FM_RATIONAL), DIMENSION(:,:) :: MB
  TYPE (FM_RATIONAL), ALLOCATABLE, DIMENSION(:,:) :: TEMP
  INTEGER :: J,K
  INTENT (INOUT) :: MA
  INTENT (IN) :: MB
  CALL FM_UNDEF_INP(MB)
  IF (SIZE(MA,DIM=1) /= SIZE(MB,DIM=1) .OR. SIZE(MA,DIM=2) /= SIZE(MB,DIM=2)) THEN
    CALL IMST2M('UNKNOWN',MT_RM%NUMERATOR)
    CALL IMST2M('UNKNOWN',MT_RM%DENOMINATOR)
    DO J = 1, SIZE(MA,DIM=1)
      DO K = 1, SIZE(MA,DIM=2)
        CALL IMEQ(MT_RM%NUMERATOR,MA(J,K)%NUMERATOR)
        CALL IMEQ(MT_RM%DENOMINATOR,MA(J,K)%DENOMINATOR)
      ENDDO
    ENDDO
    RETURN
  ENDF
```

! To avoid problems when lhs and rhs are overlapping parts of the same array, move MB
! to a temporary array before re-defining any of MA.

```
ALLOCATE(TEMP(SIZE(MA,DIM=1),SIZE(MA,DIM=2)))
DO J = 1, SIZE(MA,DIM=1)
```

```

DO K = 1, SIZE(MA,DIM=2)
  CALL IMEQ(MB(J,K)%NUMERATOR,TEMP(J,K)%NUMERATOR)
  CALL IMEQ(MB(J,K)%DENOMINATOR,TEMP(J,K)%DENOMINATOR)
ENDDO
ENDDO
DO J = 1, SIZE(MA,DIM=1)
  DO K = 1, SIZE(MA,DIM=2)
    CALL FMEQ_RATIONAL_RMRM(MA(J,K),TEMP(J,K))
  ENDDO
ENDDO
DEALLOCATE(TEMP)
END SUBROUTINE FMEQ_RATIONAL_RM2RM2

```

```

SUBROUTINE FMEQ_RATIONAL_RM2IM2(MA,MB)
  USE FMVALS
  IMPLICIT NONE
  TYPE (FM_RATIONAL), DIMENSION(:,:) :: MA
  TYPE (IM), DIMENSION(:,:) :: MB
  TYPE (IM), ALLOCATABLE, DIMENSION(:,:) :: TEMP
  INTEGER :: J,K
  INTENT (INOUT) :: MA
  INTENT (IN) :: MB
  IF (SIZE(MA,DIM=1) /= SIZE(MB,DIM=1) .OR. SIZE(MA,DIM=2) /= SIZE(MB,DIM=2)) THEN
    CALL IMST2M('UNKNOWN',MT_RM%NUMERATOR)
    CALL IMST2M('UNKNOWN',MT_RM%DENOMINATOR)
    DO J = 1, SIZE(MA,DIM=1)
      DO K = 1, SIZE(MA,DIM=2)
        CALL IMEQ(MT_RM%NUMERATOR,MA(J,K)%NUMERATOR)
        CALL IMEQ(MT_RM%DENOMINATOR,MA(J,K)%DENOMINATOR)
      ENDDO
    ENDDO
    RETURN
  ENDIF
  ALLOCATE(TEMP(SIZE(MA,DIM=1),SIZE(MA,DIM=2)))
  DO J = 1, SIZE(MA,DIM=1)
    DO K = 1, SIZE(MA,DIM=2)
      CALL IMEQ(MB(J,K)%MIM,TEMP(J,K)%MIM)
    ENDDO
  ENDDO
  DO J = 1, SIZE(MA,DIM=1)
    DO K = 1, SIZE(MA,DIM=2)
      CALL FMEQ_RATIONAL_RMIM(MA(J,K),TEMP(J,K))
    ENDDO
  ENDDO
  DEALLOCATE(TEMP)
END SUBROUTINE FMEQ_RATIONAL_RM2IM2

```

```

SUBROUTINE FMADD_RATIONAL_RMRM_0(MA,MB,MC)
  USE FMVALS
  IMPLICIT NONE
  TYPE (FM_RATIONAL) :: MA,MB,MC
  INTENT (IN) :: MA,MB
  INTENT (INOUT) :: MC
  CALL FM_UNDEF_INP(MA)
  CALL FM_UNDEF_INP(MB)
  CALL IMEQ(MA%NUMERATOR, R_1%MIM)
  CALL IMEQ(MA%DENOMINATOR,R_2%MIM)
  CALL IMEQ(MB%NUMERATOR, R_3%MIM)

```



```

CALL IMEQ(MB%DENOMINATOR,R_4%MIM)
CALL IM_MPY(R_1,R_4,R_5)
CALL IM_MPY(R_2,R_3,R_1)
CALL IM_ADD(R_1,R_5,R_3)
CALL IM_MPY(R_2,R_4,R_5)
CALL FM_MAX_EXP_IM(R_3,R_5)
IF (SKIP_GCD .AND. MAX(R_3%MIM%MP(2),R_5%MIM%MP(2)) < RATIONAL_SKIP_MAX) THEN
    CALL IMEQ(R_3%MIM,MC%NUMERATOR)
    CALL IMEQ(R_5%MIM,MC%DENOMINATOR)
ELSE
    CALL IM_GCD(R_3,R_5,R_1)
    CALL IMDIV(R_3%MIM,R_1%MIM,MC%NUMERATOR)
    CALL IMDIV(R_5%MIM,R_1%MIM,MC%DENOMINATOR)
ENDIF
IF (MC%DENOMINATOR%MP(1) < 0) THEN
    MC%DENOMINATOR%MP(1) = 1
    MC%NUMERATOR%MP(1) = -MC%NUMERATOR%MP(1)
ENDIF
END SUBROUTINE FMADD_RATIONAL_RMRM_0

```

```

SUBROUTINE FMSUB_RATIONAL_RMRM_0(MA,MB,MC)
USE FMVALS
IMPLICIT NONE
TYPE (FM_RATIONAL) :: MA,MB,MC
INTENT (IN) :: MA,MB
INTENT (INOUT) :: MC
CALL FM_UNDEF_INP(MA)
CALL FM_UNDEF_INP(MB)
CALL IMEQ(MA%NUMERATOR, R_1%MIM)
CALL IMEQ(MA%DENOMINATOR,R_2%MIM)
CALL IMEQ(MB%NUMERATOR, R_3%MIM)
CALL IMEQ(MB%DENOMINATOR,R_4%MIM)
CALL IM_MPY(R_1,R_4,R_5)
CALL IM_MPY(R_2,R_3,R_1)
CALL IM_SUB(R_5,R_1,R_3)
CALL IM_MPY(R_2,R_4,R_5)
CALL FM_MAX_EXP_IM(R_3,R_5)
IF (SKIP_GCD .AND. MAX(R_3%MIM%MP(2),R_5%MIM%MP(2)) < RATIONAL_SKIP_MAX) THEN
    CALL IMEQ(R_3%MIM,MC%NUMERATOR)
    CALL IMEQ(R_5%MIM,MC%DENOMINATOR)
ELSE
    CALL IM_GCD(R_3,R_5,R_1)
    CALL IMDIV(R_3%MIM,R_1%MIM,MC%NUMERATOR)
    CALL IMDIV(R_5%MIM,R_1%MIM,MC%DENOMINATOR)
ENDIF
IF (MC%DENOMINATOR%MP(1) < 0) THEN
    MC%DENOMINATOR%MP(1) = 1
    MC%NUMERATOR%MP(1) = -MC%NUMERATOR%MP(1)
ENDIF
END SUBROUTINE FMSUB_RATIONAL_RMRM_0

```

```

SUBROUTINE FMMPY_RATIONAL_RMRM_0(MA,MB,MC)
USE FMVALS
IMPLICIT NONE
TYPE (FM_RATIONAL) :: MA,MB,MC
INTENT (IN) :: MA,MB
INTENT (INOUT) :: MC
INTEGER :: IDIV

```

```

CALL FM_UNDEF_INP(MA)
CALL FM_UNDEF_INP(MB)
CALL IMEQ(MA%NUMERATOR, R_1%MIM)
CALL IMEQ(MA%DENOMINATOR, R_2%MIM)
CALL IMEQ(MB%NUMERATOR, R_3%MIM)
CALL IMEQ(MB%DENOMINATOR, R_4%MIM)
CALL IM_MPY(R_1, R_3, R_5)
CALL IM_MPY(R_2, R_4, R_3)
CALL FM_MAX_EXP_IM(R_3, R_5)
IF (SKIP_GCD .AND. MAX(R_3%MIM%MP(2), R_5%MIM%MP(2)) < RATIONAL_SKIP_MAX) THEN
    CALL IMEQ(R_5%MIM, MC%NUMERATOR)
    CALL IMEQ(R_3%MIM, MC%DENOMINATOR)
ELSE
    CALL IM_GCD(R_3, R_5, R_1)
    IF (R_1%MIM%MP(2) == 1 .AND. R_1%MIM%MP(3) == 1) THEN
        CALL IMEQ(R_5%MIM, MC%NUMERATOR)
        CALL IMEQ(R_3%MIM, MC%DENOMINATOR)
    ELSE IF (R_1%MIM%MP(2) == 1 .AND. R_1%MIM%MP(3) < MXBASE) THEN
        IDIV = R_1%MIM%MP(3)
        CALL IMDIVI(R_5%MIM, IDIV, MC%NUMERATOR)
        CALL IMDIVI(R_3%MIM, IDIV, MC%DENOMINATOR)
    ELSE
        CALL IMDIV(R_5%MIM, R_1%MIM, MC%NUMERATOR)
        CALL IMDIV(R_3%MIM, R_1%MIM, MC%DENOMINATOR)
    ENDF
ENDIF
ENDIF
IF (MC%DENOMINATOR%MP(1) < 0) THEN
    MC%DENOMINATOR%MP(1) = 1
    MC%NUMERATOR%MP(1) = -MC%NUMERATOR%MP(1)
ENDIF
END SUBROUTINE FMMPY_RATIONAL_RMRM_0

SUBROUTINE FMDIV_RATIONAL_RMRM_0(MA, MB, MC)
USE FMVALS
IMPLICIT NONE
TYPE (FM_RATIONAL) :: MA, MB, MC
INTENT (IN) :: MA, MB
INTENT (INOUT) :: MC
CALL FM_UNDEF_INP(MA)
CALL FM_UNDEF_INP(MB)
IF (MA%NUMERATOR%MP(2) == MUNKNO .OR. MA%DENOMINATOR%MP(2) == MUNKNO .OR. &
    MB%NUMERATOR%MP(2) == MUNKNO .OR. MB%DENOMINATOR%MP(2) == MUNKNO .OR. &
    MB%NUMERATOR%MP(3) == 0 ) THEN
    CALL IMST2M('UNKNOWN', MC%NUMERATOR)
    CALL IMST2M('UNKNOWN', MC%DENOMINATOR)
    RETURN
ENDIF
CALL IMEQ(MA%NUMERATOR, R_1%MIM)
CALL IMEQ(MA%DENOMINATOR, R_2%MIM)
CALL IMEQ(MB%NUMERATOR, R_3%MIM)
CALL IMEQ(MB%DENOMINATOR, R_4%MIM)
CALL IM_MPY(R_1, R_4, R_5)
CALL IM_MPY(R_2, R_3, R_4)
CALL FM_MAX_EXP_IM(R_4, R_5)
IF (SKIP_GCD .AND. MAX(R_4%MIM%MP(2), R_5%MIM%MP(2)) < RATIONAL_SKIP_MAX) THEN
    CALL IMEQ(R_5%MIM, MC%NUMERATOR)
    CALL IMEQ(R_4%MIM, MC%DENOMINATOR)
ELSE

```

```

CALL IM_GCD(R_4,R_5,R_1)
CALL IMDIV(R_5%MIM,R_1%MIM,MC%NUMERATOR)
CALL IMDIV(R_4%MIM,R_1%MIM,MC%DENOMINATOR)
ENDIF
IF (MC%DENOMINATOR%MP(1) < 0) THEN
  MC%DENOMINATOR%MP(1) = 1
  MC%NUMERATOR%MP(1) = -MC%NUMERATOR%MP(1)
ENDIF
END SUBROUTINE FMDIV_RATIONAL_RMRM_0

END MODULE FM_RATIONAL_ARITHMETIC_1

```

```

MODULE FM_RATIONAL_ARITHMETIC_2
  USE FM_RATIONAL_ARITHMETIC_1

```

```

INTERFACE OPERATOR (+)

```

```

  MODULE PROCEDURE FMADD_RATIONAL_RM
  MODULE PROCEDURE FMADD_RATIONAL_RM1
  MODULE PROCEDURE FMADD_RATIONAL_RM2

```

```

  MODULE PROCEDURE FMADD_RATIONAL_RMRM
  MODULE PROCEDURE FMADD_RATIONAL_IRM
  MODULE PROCEDURE FMADD_RATIONAL_RMI
  MODULE PROCEDURE FMADD_RATIONAL_IMRM
  MODULE PROCEDURE FMADD_RATIONAL_RMIM

```

```

  MODULE PROCEDURE FMADD_RATIONAL_RM1RM1
  MODULE PROCEDURE FMADD_RATIONAL_RM1RM
  MODULE PROCEDURE FMADD_RATIONAL_RMRM1
  MODULE PROCEDURE FMADD_RATIONAL_I1RM1
  MODULE PROCEDURE FMADD_RATIONAL_RM1I1
  MODULE PROCEDURE FMADD_RATIONAL_I1RM
  MODULE PROCEDURE FMADD_RATIONAL_RMI1
  MODULE PROCEDURE FMADD_RATIONAL_IRM1
  MODULE PROCEDURE FMADD_RATIONAL_RM1I
  MODULE PROCEDURE FMADD_RATIONAL_IM1RM1
  MODULE PROCEDURE FMADD_RATIONAL_IM1RM
  MODULE PROCEDURE FMADD_RATIONAL_IMRM1
  MODULE PROCEDURE FMADD_RATIONAL_RM1IM1
  MODULE PROCEDURE FMADD_RATIONAL_RM1IM
  MODULE PROCEDURE FMADD_RATIONAL_RMIM1

```

```

  MODULE PROCEDURE FMADD_RATIONAL_RM2RM2
  MODULE PROCEDURE FMADD_RATIONAL_RM2RM
  MODULE PROCEDURE FMADD_RATIONAL_RMRM2
  MODULE PROCEDURE FMADD_RATIONAL_I2RM2
  MODULE PROCEDURE FMADD_RATIONAL_I2RM
  MODULE PROCEDURE FMADD_RATIONAL_IRM2
  MODULE PROCEDURE FMADD_RATIONAL_RM2I2
  MODULE PROCEDURE FMADD_RATIONAL_RM2I
  MODULE PROCEDURE FMADD_RATIONAL_RMI2
  MODULE PROCEDURE FMADD_RATIONAL_IM2RM2
  MODULE PROCEDURE FMADD_RATIONAL_IM2RM
  MODULE PROCEDURE FMADD_RATIONAL_IMRM2
  MODULE PROCEDURE FMADD_RATIONAL_RM2IM2
  MODULE PROCEDURE FMADD_RATIONAL_RM2IM
  MODULE PROCEDURE FMADD_RATIONAL_RMIM2

```

END INTERFACE

CONTAINS

!

+

```
FUNCTION FMADD_RATIONAL_RM(MA)      RESULT (RETURN_VALUE)
  USE FMVALS
  IMPLICIT NONE
  TYPE (FM_RATIONAL) :: MA, RETURN_VALUE
  INTENT (IN) :: MA
  CALL FM_UNDEF_INP(MA)
  CALL FMEQ_RATIONAL(MA, RETURN_VALUE)
  IF (RETURN_VALUE%DENOMINATOR%MP(1) < 0) THEN
    RETURN_VALUE%DENOMINATOR%MP(1) = 1
    RETURN_VALUE%NUMERATOR%MP(1) = -RETURN_VALUE%NUMERATOR%MP(1)
  ENDIF
END FUNCTION FMADD_RATIONAL_RM
```

```
FUNCTION FMADD_RATIONAL_RM1(MA)      RESULT (RETURN_VALUE)
  USE FMVALS
  IMPLICIT NONE
  TYPE (FM_RATIONAL), DIMENSION(:) :: MA
  TYPE (FM_RATIONAL), DIMENSION(SIZE(MA)) :: RETURN_VALUE
  INTEGER :: J
  INTENT (IN) :: MA
  CALL FM_UNDEF_INP(MA)
  DO J = 1, SIZE(MA)
    CALL FMEQ_RATIONAL(MA(J), RETURN_VALUE(J))
    IF (RETURN_VALUE(J)%DENOMINATOR%MP(1) < 0) THEN
      RETURN_VALUE(J)%DENOMINATOR%MP(1) = 1
      RETURN_VALUE(J)%NUMERATOR%MP(1) = &
        -RETURN_VALUE(J)%NUMERATOR%MP(1)
    ENDIF
  ENDDO
END FUNCTION FMADD_RATIONAL_RM1
```

```
FUNCTION FMADD_RATIONAL_RM2(MA)      RESULT (RETURN_VALUE)
  USE FMVALS
  IMPLICIT NONE
  TYPE (FM_RATIONAL), DIMENSION(:, :) :: MA
  TYPE (FM_RATIONAL), DIMENSION(SIZE(MA, DIM=1), SIZE(MA, DIM=2)) :: RETURN_VALUE
  INTEGER :: J, K
  INTENT (IN) :: MA
  CALL FM_UNDEF_INP(MA)
  DO J = 1, SIZE(MA, DIM=1)
    DO K = 1, SIZE(MA, DIM=2)
      CALL FMEQ_RATIONAL(MA(J, K), RETURN_VALUE(J, K))
      IF (RETURN_VALUE(J, K)%DENOMINATOR%MP(1) < 0) THEN
        RETURN_VALUE(J, K)%DENOMINATOR%MP(1) = 1
        RETURN_VALUE(J, K)%NUMERATOR%MP(1) = &
          -RETURN_VALUE(J, K)%NUMERATOR%MP(1)
      ENDIF
    ENDDO
  ENDDO
END FUNCTION FMADD_RATIONAL_RM2
```

```
FUNCTION FMADD_RATIONAL_RMRM(MA, MB)  RESULT (RETURN_VALUE)
```