

PROGRAM TEST

! Version 1.4.

! This is a sample program using the FMZM modules for doing arithmetic using the FM, IM, and ZM derived types.

! The program's output to the screen is also saved in file SampleFM.out.
! The program checks all the results and the last line of the output file should be
! "All results were ok."

! These examples show various ways to use the FM package, but the methods used are not always
! the most advanced for the sample problem.

USE FMZM

IMPLICIT NONE

! Declare the multiple precision variables. The three types are:
! (FM) for multiple precision real
! (IM) for multiple precision integer
! (ZM) for multiple precision complex

TYPE (FM), SAVE :: X1, X2, X3, X4

TYPE (FM), SAVE, ALLOCATABLE :: A(:,,:), B(:,,:), V1(:), V2(:)

TYPE (IM), SAVE :: I1, I2, I3

TYPE (ZM), SAVE :: Z1, Z2, Z3, Z4

! Declare the function name of a type (FM) function that will be passed as an argument
! to a subroutine called from this program.

TYPE (FM), EXTERNAL :: F

! Declare the other variables (not multiple precision).

CHARACTER(80) :: ST1

CHARACTER(175) :: FMT

INTEGER :: ITER, J, K, KOUT, NERROR

INTEGER :: SEED(7)

DOUBLE PRECISION :: VALUE

! Write output to the screen (unit *), and also to the file SampleFM.out.

KOUT = 18

OPEN (KOUT, FILE='SampleFM.out')

NERROR = 0

! 1. Find a root of the equation $f(x) = x^5 - 3x^4 + x^3 - 4x^2 + x - 6 = 0$.

! Set precision to give at least 60 significant digits.

CALL FM_SET(60)

! Use Newton's method with initial guess $x = 3.12$.
! Horner's rule is used to evaluate the function.

```

!           X1 is the previous iterate.
!           X2 is the current iterate.

!           TO_FM is a function for converting other types of numbers to type FM. Note that
!           TO_FM(3.12) converts the REAL constant to FM, but it is accurate only to single
!           precision, since the number 3.12 cannot be represented exactly in binary and has
!           already been rounded to single precision. Similarly, TO_FM(3.12D0) agrees with
!           3.12 to double precision accuracy, and TO_FM('3.12') or TO_FM(312)/TO_FM(100)
!           agrees to full FM accuracy.
!           Here, TO_FM(3.12) would be ok, since Newton iteration will correct the error
!           coming from single precision, but it is a good habit to use the more accurate
!           form.

X1 = TO_FM('3.12')

!           Print the first iteration.

FMT = "(// ' Sample 1. Real root of f(x) = x**5 - 3x**4 + x**3 - 4x**2 + x - 6 = 0.'//)" &
      "' Iteration      Newton approximation'"
WRITE (* ,FMT)
WRITE (KOUT,FMT)

!           FM_FORM is a formatting subroutine.

CALL FM_FORM('F65.60',X1,ST1)
WRITE (* ,"/I10,4X,A") 0,TRIM(ST1)
WRITE (KOUT, "/I10,4X,A") 0,TRIM(ST1)

DO ITER = 1, 10

!           X3 is f(X1).

X3 = (((X1-3)*X1+1)*X1-4)*X1+1)*X1-6

!           X4 is f'(X1).

X4 = ((5*X1-12)*X1+3)*X1-8)*X1+1

X2 = X1 - X3/X4

!           Print each iteration.

CALL FM_FORM('F65.60',X2,ST1)
WRITE (* ,"/I10,4X,A") ITER,TRIM(ST1)
WRITE (KOUT, "/I10,4X,A") ITER,TRIM(ST1)

!           Stop iterating if X1 and X2 agree to over 60 places.

X4 = ABS(X1-X2)
IF (X4 < 1.0D-61) EXIT

!           Set X1 = X2 for the next iteration.

X1 = X2
ENDDO

!           Check the answer.

```

```
X3 = TO_FM('3.1206562153267265004709560135237974846546239355990660149888284358')
```

```
!           It is slightly safer to do this test with .NOT. instead of  
!           IF (ABS(X3-X2) >= 1.0D-61) THEN  
!           because if the result of ABS(X3-X2) is FM's UNKNOWN value,  
!           the comparison returns false for all comparisons.
```

```
IF (.NOT.(ABS(X3-X2) < 1.0D-61)) THEN  
  NERROR = NERROR + 1  
  WRITE (*, "/' Error in sample case number 1.'/")  
  WRITE (KOUT, "/' Error in sample case number 1.'/")  
ENDIF
```

```
!           2. Higher Precision. Compute the root above to 500 decimal places.
```

```
CALL FM_SET(500)
```

```
!           It is tempting to just say X1 = X3 here to initialize the start of the higher  
!           precision iterations to be the check value defined above. That will not work,  
!           because precision has changed. Most of the digits of X3 may be undefined at  
!           the new precision.  
!           The most flexible way to pad a lower precision value with zeros when raising  
!           precision is to use subroutine FM_EQU, but here it is easier to re-define X1  
!           from scratch at the new precision.
```

```
X1 = TO_FM('3.1206562153267265004709560135237974846546239355990660149888284358')
```

```
DO ITER = 1, 10
```

```
!           X3 is f(X1).
```

```
X3 = (((X1-3)*X1+1)*X1-4)*X1+1)*X1-6
```

```
!           X4 is f'(X1).
```

```
X4 = ((5*X1-12)*X1+3)*X1-8)*X1+1
```

```
X2 = X1 - X3/X4
```

```
!           Stop iterating if X1 and X2 agree to over 500 places.
```

```
X4 = ABS(X1-X2)
```

```
!           Compare this test to the similar one in case 1 above.  
!           For machines with 64-bit double precision, 1.0D-501 would be smaller than the  
!           smallest positive number. So this error tolerance is converted to an FM number  
!           from character form.
```

```
IF (X4 < TO_FM('1.0E-501')) EXIT
```

```
!           Set X1 = X2 for the next iteration.
```

```
X1 = X2
```

```
ENDDO
```

```
!           For very high precision output, it is sometimes more convenient to use FM_PRINT
```

```
!           to format and print the numbers, since the line breaks are handled automatically.
!           The unit number for the output, KW, and the format codes to be used, JFORM1 and
!           JFORM2, are internal FM variables.
!           Subroutine FM_SETVAR is used to re-define these, and the new values will remain in
!           effect for any further calls to FM_PRINT.
```

```
!           Other variables that can be changed and the options they control are listed in the
!           documentation at the top of file FM.f95.
```

```
!           Set the FM_PRINT format to F505.500
```

```
CALL FM_SETVAR(' JFORM1 = 2 ')
CALL FM_SETVAR(' JFORM2 = 500 ')
```

```
!           Set the output screen width to 90 columns.
```

```
CALL FM_SETVAR(' KSWIDE = 90 ')
```

```
FMT = "(///' Sample 2. Find the root above to 500 decimal places.'/)"
WRITE (* ,FMT)
WRITE (KOUT,FMT)
```

```
!           Write to the output file.
```

```
CALL FM_SETVAR(' KW = 18 ')
CALL FM_PRINT(X2)
```

```
!           Write to the screen (unit 6).
```

```
CALL FM_SETVAR(' KW = 6 ')
CALL FM_PRINT(X2)
```

```
!           Check the answer.
```

```
X3 = TO_FM('3.1206562153267265004709560135237974846546239355990660149888284358190264999' // &
           '517954689783257450017151095811923431332682839420040840535954560118152245371' // &
           '792881305271951017118938898212403662058307303983547376913282000110058273504' // &
           '202838670709895619275413484521549282591891156945200789415818387529512010999' // &
           '602155131321076797099026664236992803703462570149559724389392331827597552460' // &
           '610612200485579529156910428115547013787714423708578161025641555097481179969' // &
           '175028390105904786831680128384331143259309155577171683842444352768419176139060')
```

```
IF (.NOT.(ABS(X3-X2) < TO_FM('1.0E-501')))) THEN
  NERROR = NERROR + 1
  WRITE (* ,"/' Error in sample case number 2.'/")
  WRITE (KOUT,"/' Error in sample case number 2.'/")
ENDIF
```

```
!           3. Compute the Riemann zeta function for s=3.
```

```
!           Use Gosper's formula:  $\zeta(3) =$   
!            $(5/4) * \text{Sum} [ (-1)**k * (k!)**2 / ((k+1)**2 * (2k+1)!) ]$   
!           while  $k = 0, 1, \dots$ 
```

```
!           X1 is the current partial sum.  
!           X2 is the current term.  
!           X3 is k!
```

! X4 is $(2k+1)!$

```
CALL FM_SET(60)
X1 = 1
X3 = 1
X4 = 1
DO K = 1, 200
  X3 = K*X3
  J = 2*K*(2*K+1)
  X4 = J*X4
  X2 = X3**2
  J = (K+1)*(K+1)
  X2 = (X2/J)/X4
  IF (MOD(K,2) == 0) THEN
    X1 = X1 + X2
  ELSE
    X1 = X1 - X2
  ENDF
ENDDO
```

! Test for convergence.
! Here the rate of convergence is much slower than in the Newton iterations above.
! Asking for 60 digits in the call to FM_SET will cause the internal precision to
! be set slightly higher than that, giving the user a few guard digits.
! X2 is the difference between the two most recent partial sums, so the test
! below will stop the sum when the last two partial sums agree to at least 65
! significant digits.

```
IF (ABS(X2/X1) < 1.0D-65) THEN
  FMT = "(///' Sample 3.',2X,I5,' terms were added in the zeta sum.'/)"
  WRITE (* ,FMT) K
  WRITE (KOUT,FMT) K
  EXIT
ENDIF
ENDDO
```

! Print the result.

```
X1 = (5*X1)/4
CALL FM_FORM('F63.60',X1,ST1)
WRITE (* , "(' zeta(3) = ',A)") TRIM(ST1)
WRITE (KOUT,"(' zeta(3) = ',A)") TRIM(ST1)
```

! Check the answer.

```
X3 = TO_FM('1.202056903159594285399738161511449990764986292340498881792271555')
IF (.NOT.(ABS(X1-X3) < 1.0D-61)) THEN
  NERROR = NERROR + 1
  WRITE (* ,"/' Error in sample case number 3.'/")
  WRITE (KOUT,"('/ Error in sample case number 3.'/")
ENDIF
```

! 4. Integer multiple precision calculations.

! Fermat's theorem says $x^{(p-1)} \bmod p = 1$ when p is prime and x is not a
! multiple of p .
! If $x^{(p-1)} \bmod p$ gives 1 for some p with several different x 's, then it is
! very likely that p is prime (but it is not certain until further tests are done).

```

!           Find a 70-digit number p that is "probably" prime.

!           Use FM_RANDOM_NUMBER to generate a random 70-digit starting value and search for
!           a prime from that point.

!           Initialize the generator.
!           Note that VALUE is double precision, unlike the similar Fortran intrinsic random
!           number routine, which returns a single-precision result.

SEED = (/ 2718281,8284590,4523536,0287471,3526624,9775724,7093698 /)
CALL FM_RANDOM_SEED_PUT(SEED)

!           I1 is the value p being tested.

I1 = 0
I3 = TO_IM(10)**13
DO J = 1, 6
  CALL FM_RANDOM_NUMBER(VALUE)
  I2 = 1.0D13*VALUE
  I1 = I1*I3 + I2
ENDDO
I3 = TO_IM(10)**70
I1 = MOD(I1,I3)

!           To speed up the search, test only values that are not
!           multiples of 2, 3, 5, 7, 11, 13.

K = 2*3*5*7*11*13
I1 = (I1/K)*K + K + 1
I3 = 3

DO J = 1, 100
  I2 = I1 - 1

!           Compute 3**(p-1) mod p

I3 = POWER_MOD(I3,I2,I1)
IF (I3 == 1) THEN

!           Check that 7**(p-1) mod p is also 1.

I3 = 7
I3 = POWER_MOD(I3,I2,I1)
IF (I3 == 1) THEN
  FMT = "(///' Sample 4.',2X,I5,' values were checked before finding a prime p.'/)"
  WRITE (* ,FMT) J
  WRITE (KOUT,FMT) J
  EXIT
ENDIF
ENDIF

I3 = 3
I1 = I1 + K
ENDDO

!           Print the result.

```

```
CALL IM_FORM('I72',I1,ST1)
WRITE (* , "(' p =',A)") TRIM(ST1)
WRITE (KOUT, "(' p =',A)") TRIM(ST1)
```

! Check the answer.

```
I3 = TO_IM('9552131129056058313103536357738804884840825498503088946760768419490591')
IF (.NOT.(I1 == I3)) THEN
  NERROR = NERROR + 1
  WRITE (* , "(' Error in sample case number 4.'//)")
  WRITE (KOUT, "(' Error in sample case number 4.'//)")
ENDIF
```

! 5. Log Integral function.

! Estimate the number of primes less than $10^{*}30$.

```
FMT = "(///' Sample 5. Log integral. Estimate the number of primes less than 10**30.'// &
      " It should be accurate to about 15 significant digits.'//)"
WRITE (* , FMT)
WRITE (KOUT, FMT)
```

```
I2 = TO_IM(LOG_INTEGRAL(TO_FM('1.0E+30')))
```

! Print the result.

```
CALL IM_FORM('I30',I2,ST1)
WRITE (* , "(' int(li(1.0e+30)) = ',A)") TRIM(ST1)
WRITE (KOUT, "(' int(li(1.0e+30)) = ',A)") TRIM(ST1)
```

! Check the answer.

```
I3 = TO_IM('14692398897720447639079087669')
IF (.NOT.(I2 == I3)) THEN
  NERROR = NERROR + 1
  WRITE (* , "(' Error in sample case number 5.'//)")
  WRITE (KOUT, "(' Error in sample case number 5.'//)")
ENDIF
```

! 6. Gamma function.

! Check that $\text{gamma}(1/2)$ is $\text{sqrt}(\pi)$

```
FMT = "(///' Sample 6. Check that gamma(1/2) = sqrt(pi).'//)"
WRITE (* , FMT)
WRITE (KOUT, FMT)
```

```
X2 = GAMMA(TO_FM('0.5'))
```

! Print the result.

```
CALL FM_FORM('F63.60',X2,ST1)
WRITE (* , "(' gamma(1/2) = ',A)") TRIM(ST1)
WRITE (KOUT, "(' gamma(1/2) = ',A)") TRIM(ST1)
```

! Check the answer.

```

X3 = SQRT(ACOS(TO_FM(-1)))
IF (.NOT.(ABS(X3-X2) < 1.0D-61)) THEN
    NERROR = NERROR + 1
    WRITE (* ,"/' Error in sample case number 6.'/")
    WRITE (KOUT,"/' Error in sample case number 6.'/")
ENDIF

```

! 7. Psi and polygamma functions.

```

! Rational series can often be summed using these functions.
! Sum (n=1 to infinity) 1/(n**2 * (8n+1)**2) =
! 16*(psi(1) - psi(9/8)) + polygamma(1,1) + polygamma(1,9/8)
! Reference: Abramowitz & Stegun, Handbook of Mathematical Functions,
! chapter 6, Example 10.

```

```

FMT = "(///' Sample 7. Psi and polygamma functions.'/)"
WRITE (* ,FMT)
WRITE (KOUT,FMT)

```

```

X2 = 16*(PSI(TO_FM(1)) - PSI(TO_FM(9)/8)) + POLYGAMMA(1,TO_FM(1)) + POLYGAMMA(1,TO_FM(9)/8)

```

! Print the result.

```

CALL FM_FORM('F65.60',X2,ST1)
FMT = "(' Sum (n=1 to infinity) 1/(n**2 * (8n+1)**2) = '/9X,A)"
WRITE (* ,FMT) TRIM(ST1)
WRITE (KOUT,FMT) TRIM(ST1)

```

! Check the answer.

```

X3 = TO_FM('1.3499486145413024755107829105035147950644978635837270816327396M-2')
IF (.NOT.(ABS(X3-X2) < 1.0D-61)) THEN
    NERROR = NERROR + 1
    WRITE (* ,"/' Error in sample case number 7.'/")
    WRITE (KOUT,"/' Error in sample case number 7.'/")
ENDIF

```

! 8. Incomplete gamma and gamma functions.

```

! Find the probability that an observed chi-square for a correct model should be
! less that 2.3 when the number of degrees of freedom is 5.
! Reference: Knuth, Volume 2, 3rd ed., Page 56, and Press, Flannery, Teukolsky,
! Vetterling, Numerical Recipes, 1st ed., Page 165.

```

```

FMT = "(///' Sample 8. Incomplete gamma and gamma functions.'/)"
WRITE (* ,FMT)
WRITE (KOUT,FMT)

```

```

X1 = TO_FM(5)/2
X2 = INCOMPLETE_GAMMA1(X1,TO_FM('2.3')/2) / GAMMA(X1)

```

! Print the result.

```

CALL FM_FORM('F62.60',X2,ST1)
WRITE (* ,"/(' Probability = ',A)") TRIM(ST1)

```



```
WRITE (KOUT, "(' Probability = ',A)") TRIM(ST1)
```

! Check the answer.

```
X3 = TO_FM('0.19373313011487144632751025918250599953472318607121386973066283739')
IF (.NOT.(ABS(X3-X2) < 1.0D-61)) THEN
  NERROR = NERROR + 1
  WRITE (*, "(/' Error in sample case number 8.'/)")
  WRITE (KOUT, "(/' Error in sample case number 8.'/)")
ENDIF
```

! 9. Error function.

! Find the probability that a value drawn from a normal distribution is within
! 1 or 2 or 3 standard deviations from the mean.

```
FMT = "(//' Sample 9. Error function. Probability that a value drawn from a normal'/'/' &
      "' distribution is within k standard deviations from the mean.'/)"
WRITE (*, FMT)
WRITE (KOUT, FMT)
```

```
DO K = 1, 3
  X1 = K / SQRT(TO_FM(2))
  X2 = ERF(X1)
```

! Print the results.

```
CALL FM_FORM('F52.50', X2, ST1)
WRITE (*, "(' k = ',I2,', probability = ',A)") K, TRIM(ST1)
WRITE (KOUT, "(' k = ',I2,', probability = ',A)") K, TRIM(ST1)
```

! Check the answer.

```
IF (K == 1) THEN
  X3 = TO_FM('0.68268949213708589717046509126407584495582593345320878197478890049')
ELSE IF (K == 2) THEN
  X3 = TO_FM('0.95449973610364158559943472566693312505644755259664313203266799974')
ELSE
  X3 = TO_FM('0.99730020393673981094669637046481004524434126368323870127155602929')
ENDIF
IF (.NOT.(ABS(X3-X2) < 1.0D-61)) THEN
  NERROR = NERROR + 1
  WRITE (*, "(/' Error in sample case number 9.'/)")
  WRITE (KOUT, "(/' Error in sample case number 9.'/)")
ENDIF
ENDDO
```

! 10. Array operations.

! Find the dominant eigenvalue and a corresponding eigenvector for this 5x5 matrix:

```
!           3  1  4  1  5
!           9  2  6  5  3
!    A =    5  8  9  7  9
!           3  2  3  8  4
!           6  2  6  4  3
```

```
!           Use the power method. Compute  $B = A^n$ . If  $v_1$  is an initial guess for the
!           largest magnitude eigenvector,  $v_2 = B*v_1$  should be a more accurate approximation.
!           The ratio of the elements of  $v_3 = A*v_2$  to those of  $v_2$  gives an estimate of the
!           corresponding eigenvalue. By repeatedly squaring the matrix, each iteration uses
!           the next higher power of 2 for  $n$ .
```

```
FMT = "(///' Sample 10. Eigenvalue from matrix powers.')"
WRITE (* ,FMT)
WRITE (KOUT,FMT)
```

```
!           These type FM arrays were declared as allocatable. Allocate them now, and initialize.
```

```
ALLOCATE( A(5,5) )
ALLOCATE( B(5,5) )
ALLOCATE( V1(5) )
ALLOCATE( V2(5) )
```

```
!           To initialize the matrix, we can use array sections to set one row at a time, and the
!           FMZM interface will take care of converting from integer to type (FM). If the values
!           were not integers, we could say  $A(1,1:5) = (/ TO\_FM(' 3.7 '), TO\_FM(' 4.2 '), \text{etc.})$ 
```

```
A(1,1:5) = (/ 3, 1, 4, 1, 5 /)
A(2,1:5) = (/ 9, 2, 6, 5, 3 /)
A(3,1:5) = (/ 5, 8, 9, 7, 9 /)
A(4,1:5) = (/ 3, 2, 3, 8, 4 /)
A(5,1:5) = (/ 6, 2, 6, 4, 3 /)
```

```
!           Initialize all elements of the initial guess vector to 1.
```

```
V1 = 1
```

```
B = A
WRITE (* ,"(/' Iteration   eigenvalue approximation ')")
WRITE (KOUT,"(/' Iteration   eigenvalue approximation ')")
DO J = 1, 7
  B = MATMUL(B,B)
  V1 = MATMUL(B,V1)
  V2 = MATMUL(A,V1)
  X1 = V2(1) / V1(1)
  CALL FM_FORM('F64.57',X1,ST1)
  WRITE (* ,"(/I10,A)") J,TRIM(ST1)
  WRITE (KOUT,"(/I10,A)") J,TRIM(ST1)
ENDDO
```

```
!           Normalize the eigenvector (L-2 norm).
```

```
V2 = V2 / NORM2(V2)
WRITE (* ,"(/' The corresponding eigenvector is'/)")
WRITE (KOUT,"(/' The corresponding eigenvector is'/)")
DO J = 1, 5
  CALL FM_FORM('F61.57',V2(J),ST1)
  WRITE (* ,"(A)") TRIM(ST1)
  WRITE (KOUT,"(A)") TRIM(ST1)
ENDDO
```

```
!           Check the answer.
```

```

X3 = TO_FM('23.91276717232132858935703922800330450554912919599927298216827247803204')
IF (.NOT.(ABS(X3-X1) < 1.0D-61)) THEN
  NERROR = NERROR + 1
  WRITE (* , "(/' Error in sample case number 10.'/)")
  WRITE (KOUT, "(/' Error in sample case number 10.'/)")
ENDIF

```

! 11. Function and subroutine example.

! Find the integral from 0 to 1/2 of $2*\exp(-x**2)/\text{sqrt}(\pi)$.

! The exact value of the integral is $\text{erf}(1/2)$.

! Use a simple numerical integration routine to apply an integration rule
! using 100 intervals.

```
CALL FM_SET(40)
```

```
FMT = "(///' Sample 11. Function and subroutine example.'/)"
```

```
WRITE (* , FMT)
```

```
WRITE (KOUT, FMT)
```

```
X1 = 0
```

```
X2 = TO_FM(' 0.5 ')
```

```
CALL PLAN_9(F, X1, X2, 100, X3)
```

! Print the result.

```
CALL FM_FORM('F32.30', X3, ST1)
```

```
WRITE (* , "( ' Integral = ', A)") TRIM(ST1)
```

```
WRITE (KOUT, "( ' Integral = ', A)") TRIM(ST1)
```

! Check the answer.

```
X4 = ERF(TO_FM('0.5'))
```

```
IF (.NOT.(ABS(X3-X4) < 1.0D-31)) THEN
```

```
  NERROR = NERROR + 1
```

```
  WRITE (* , "(/' Error in sample case number 11.'/)")
```

```
  WRITE (KOUT, "(/' Error in sample case number 11.'/)")
```

```
ENDIF
```

! Complex arithmetic.

! Set precision to give at least 30 significant digits.

```
CALL FM_SET(30)
```

! 12. Find a complex root of the equation

! $f(x) = x**5 - 3x**4 + x**3 - 4x**2 + x - 6 = 0$.

! Newton's method with initial guess $x = .56 + 1.06 i$.

! Z1 is the previous iterate.

! Z2 is the current iterate.

```
Z1 = TO_ZM('.56 + 1.06 i')
```

```

!           Print the first iteration.

FMT = "(///' Sample 12. Complex root of f(x) = x**5 - 3x**4 + x**3 - 4x**2 + x - 6 = 0.'," &
      //"///' Iteration      Newton approximation')"
WRITE (*      ,FMT)
WRITE (KOUT,FMT)
CALL ZM_FORM('F32.30', 'F32.30', Z1, ST1)
WRITE (*      , "(/I6,4X,A)") 0, TRIM(ST1)
WRITE (KOUT, "(/I6,4X,A)") 0, TRIM(ST1)

DO ITER = 1, 10

!           Z3 is f(Z1).

Z3 = (((Z1-3)*Z1+1)*Z1-4)*Z1+1)*Z1-6

!           Z4 is f'(Z1).

Z4 = (((5*Z1-12)*Z1+3)*Z1-8)*Z1+1

Z2 = Z1 - Z3/Z4

!           Print each iteration.

CALL ZM_FORM('F32.30', 'F32.30', Z2, ST1)
WRITE (*      , "(/I6,4X,A)") ITER, TRIM(ST1)
WRITE (KOUT, "(/I6,4X,A)") ITER, TRIM(ST1)

!           Stop iterating if Z1 and Z2 agree to over 30 places.

IF (ABS(Z1-Z2) < 1.0D-31) EXIT

!           Set Z1 = Z2 for the next iteration.

Z1 = Z2
ENDDO

!           Check the answer.

Z3 = TO_ZM('0.561958308335403235498111195347453 + 1.061134679604332556983391239058885 i')
IF (.NOT.(ABS(Z3-Z2) < 1.0D-31)) THEN
  NERROR = NERROR + 1
  WRITE (*      , "(/' Error in sample case number 12.'/)")
  WRITE (KOUT, "(/' Error in sample case number 12.'/)")
ENDIF

!           13. Compute exp(1.23-2.34i).

!           Use the direct Taylor series.

!           Z1 is x.
!           Z2 is the current term, x**n/n!.
!           Z3 is the current partial sum.

Z1 = TO_ZM('1.23-2.34i')
Z2 = 1

```

```

Z3 = 1
DO K = 1, 100
  Z2 = Z2*Z1/K
  Z4 = Z3 + Z2

```

! Test for convergence.

! This is a common way to check for series convergence -- wait until the term
! being added is so close to zero that the sum does not change. That is fine
! here, because we are using the default round-to-nearest rounding mode.

! There is a pitfall if we were to re-run the program with a different rounding
! mode. For example, if we change the rounding mode to round toward +infinity,
! then at 30-digit precision the addition $1.2 + 3.4e-100$ rounds up to $1.200\dots001$
! and so the test to see if the sum did not change might never be satisfied.
! This problem can occur with either type FM or ZM sums.

! For cases where other rounding modes might be used, doing the convergence check
! like we did in the zeta sum of example 3 above is better. Here that would be
! IF (ABS(Z2/Z3) < 1.0D-35) THEN

```

IF (Z4 == Z3) THEN
  FMT = "(///' Sample 13.',2X,I5,' terms were added to get exp(1.23-2.34i).'/)"
  WRITE (* ,FMT) K
  WRITE (KOUT,FMT) K
  EXIT
ENDIF
Z3 = Z4
ENDDO

```

! Print the result.

```

CALL ZM_FORM('F33.30', 'F32.30', Z3, ST1)
WRITE (* , "( ' Result= ',A)") TRIM(ST1)
WRITE (KOUT, "( ' Result= ',A)") TRIM(ST1)

```

! Check the answer.

```

Z4 = TO_ZM('-2.379681796854777515745457977696745 - 2.458032970832342652397461908326042 i')
IF (.NOT.(ABS(Z4-Z3) < 1.0D-31)) THEN
  NERROR = NERROR + 1
  WRITE (* , "(/' Error in sample case number 13.'/)")
  WRITE (KOUT, "(/' Error in sample case number 13.'/)")
ENDIF

```

! 14. Exception handling.

! Iterate (real) exp(x) starting at 1.0 until overflow occurs.

! Testing to see if a type FM number is one of the special cases (+-overflow,
! +-underflow or unknown) by direct comparison can be tricky. When X1 is
! +overflow, the statement
! IF (X1 == TO_FM(' +OVERFLOW ')) THEN
! will return false, since the comparison routine cannot be sure that two
! different overflowed results would have been equal if the overflow threshold
! had been higher.

!
! Function IS_OVERFLOW can be used to directly check whether a number is + or -
! overflow, so that is a safer test.

!
! The FM warning message is written on unit KW, so in this test it appears on the
! screen and not in the output file.

```
CALL FM_SET(60)
```

```
X1 = TO_FM(1)
```

```
FMT = "(///' Sample 14. Exception handling.'//12X," // &  
      "' Iterate exp(x) starting at 1.0 until overflow occurs.'//" // &  
      "12X,' An FM warning message will be printed before the last iteration.')" "
```

```
WRITE (*,FMT)
```

```
FMT = "(///' Sample 14. Exception handling.'//" // &  
      "12X,' Iterate exp(x) starting at 1.0 until overflow occurs.')" "
```

```
WRITE (KOUT,FMT)
```

```
DO J = 1, 10
```

```
  X1 = EXP(X1)
```

```
  CALL FM_FORM('E560.40',X1,ST1)
```

```
  WRITE (* , "(/' Iteration',I3,5X,A)") J,TRIM(ST1)
```

```
  WRITE (KOUT,"(/' Iteration',I3,5X,A)") J,TRIM(ST1)
```

```
  IF (IS_OVERFLOW(X1)) EXIT
```

```
ENDDO
```

!
! Check that the last result was +overflow.

```
IF (IS_OVERFLOW(X1)) THEN
```

```
  WRITE (* , "(/' Overflow was correctly detected.')" )
```

```
  WRITE (KOUT,"(/' Overflow was correctly detected.')" )
```

```
ELSE
```

```
  NERROR = NERROR + 1
```

```
  WRITE (* , "(/' Error in sample case number 14.'/)" )
```

```
  WRITE (* , "(/' Overflow was not correctly detected.')" )
```

```
  WRITE (KOUT , "(/' Error in sample case number 14.'/)" )
```

```
  WRITE (KOUT , "(/' Overflow was not correctly detected.')" )
```

```
ENDIF
```

```
IF (NERROR == 0) THEN
```

```
  WRITE (* , "(//A/)" ) ' All results were ok -- no errors were found.'
```

```
  WRITE (KOUT,"(//A/)" ) ' All results were ok -- no errors were found.'
```

```
ELSE
```

```
  WRITE (* , "(//I3,A/)" ) NERROR, ' error(s) found.'
```

```
  WRITE (KOUT,"(//I3,A/)" ) NERROR, ' error(s) found.'
```

```
ENDIF
```

```
STOP
```

```
END PROGRAM TEST
```

```
SUBROUTINE PLAN_9(F,A,B,N,RESULT)
```

! Sample subroutine usage for FM.

! Integrate F(X) from A to B using N subintervals, and return the answer in RESULT.

! This does numerical integration using a 9-point rule.

! It is not a very good way to do high-precision integration, but it is a short routine

! and can often get 20 to 30 digits if f(x) is well-behaved and the interval of integration
! is not too big.

```
USE FMZM
IMPLICIT NONE
TYPE (FM) :: A, B, RESULT
TYPE (FM), SAVE :: H, H8, XJ
TYPE (FM), EXTERNAL :: F
INTEGER :: N, J
INTENT (IN) :: N, A, B
INTENT (INOUT) :: RESULT

H = (B - A)/N
H8 = H/8
RESULT = 0
DO J = 1, N
    XJ = A + (J-1)*H
    RESULT = RESULT + 989*F(XJ) + 5888*F(XJ+ H8) - 928*F(XJ+2*H8) + &
        10496*F(XJ+3*H8) - 4540*F(XJ+4*H8) + 10496*F(XJ+5*H8) - &
        928*F(XJ+6*H8) + 5888*F(XJ+7*H8) + 989*F(XJ+8*H8)
ENDDO
RESULT = H*RESULT/28350

END SUBROUTINE PLAN_9

FUNCTION F(X) RESULT (RETURN_VALUE)
```

! Sample function usage for FM.

! The test function for the integration subroutine is $2*\exp(-x**2)/\text{sqrt}(\pi)$.

```
USE FMZM
IMPLICIT NONE
TYPE (FM) :: RETURN_VALUE, X
TYPE (FM), SAVE :: PI
```

! Compare the usage here with the `SQRT(ACOS(TO_FM(-1)))` usage in the gamma example
! in the main program. There pi was only used once, so `ACOS(TO_FM(-1))` is more like
! what a non-multiple-precision program would do to get pi.

! If we need pi in a function like F that will be called hundreds of times, the acos
! call will be done every time. Here, since the argument is -1, the acos routine will
! recognize it as a special case and return the saved value of pi without needlessly
! making the program slower. But if another formula were used, like $\pi = 6*\text{asin}(1/2)$,
! it would be better to call FM_PI, since pi would be computed only once and later calls
! just use the saved value of pi.

! Another reason to call FM_PI instead of using a formula is that in case the calling
! program changed the trig function mode to degrees, instead of the default radians,
! then `ACOS(TO_FM(-1))` would give 180, not pi.

! For this case the $2/\text{sqrt}(\pi)$ could have been factored out of the integral so pi would
! not be needed every time F is called, but it was left in to illustrate similar but
! more complicated situations.

```
CALL FM_PI(PI)
RETURN_VALUE = 2*EXP(-X**2)/SQRT(PI)
```

END FUNCTION F