```
      PROGRAM TEST
      USE FMZM
      IMPLICIT NONE

!  Examples and advice for using FM_INTEGRATE.

      TYPE (FM), SAVE :: A, B, CHECK, ERR, PI, R1, R2, RESULT, SEVEN, TOL
      TYPE (FM), EXTERNAL :: F
      INTEGER :: K, KPRT, N, N_ERRORS, NW
      CHARACTER(80) :: ST1

      N_ERRORS = 0



!             1.  Start with an integral without singularities on the interval of integration.

!                 integrate  log(t) * cos(t)  from  pi/4 to pi/2.

!                 Set the tolerance to get at least 40 significant digits.
!                 FM_INTEGRATE does a sequence of iterations using the tanh-sinh quadrature formula.
!                 Each iteration uses more points until the last two iterates agree within the
!                 specified tolerance.  Since the next-to-last iterate satisfies the tolerance
!                 and the last iterate (returned as RESULT) is even more accurate, the value
!                 returned from FM_INTEGRATE is usually slightly more accurate than requested.
!                 In this case, the error check below shows that RESULT is actually correct
!                 to about 60 digits.

!                 It is usually best to set FM's precision level to be slightly higher than the
!                 number of digits requested with the tolerance.  Here we set precision to 20
!                 more digits.

      N = 1

!             Call FM_SET to define FM's precision level before any multiple precision variables
!             are defined.  This sets 60-digit precision and TOL is 1.0e-40.

      K = 40
      CALL FM_SET(K+20)
      TOL = TO_FM(10) ** (-K)

      WRITE (*,"(//)")
      CALL FM_PI(PI)

!             A and B are the limits for the integral.

      A = PI / 4
      B = PI / 2

!             KPRT controls trace printing in FM_INTEGRATE.  Setting it to 1 will print a summary
!                  of the call, giving the result, number of function evaluations, and time.
!             NW   is the unit number for this trace output.

      KPRT = 1
      NW = 6

      CALL FM_INTEGRATE(F,N,A,B,TOL,RESULT,KPRT,NW)
```

```
!             For these sample problems the integrals have known closed-form results, so
!             we can check the accuracy of FM_INTEGRATE.

      CHECK = LOG(PI/2) - LOG(PI/4)/SQRT(TO_FM(2)) + SIN_INTEGRAL(PI/4) - SIN_INTEGRAL(PI/2)

      ERR = ABS( RESULT - CHECK )
      CALL FM_FORM('ES12.4',ERR,ST1)
      WRITE (*,"(/10X,A,I2,A,A)") ' Error for case ',N,' = ',TRIM(ST1)
      IF (ERR > TOL) N_ERRORS = N_ERRORS + 1



!             2.  Next do an integral with a singularity at zero
!                 (from "Integrals of Powers of LogGamma" by T. Amdeberhan, M. Coffey, O. Espinosa,
!                 C. Koutschan, D. Manna, and V. Moll, in Proc. Amer. Math. Soc., #139, 2011)

!                 integrate  log( gamma(t) )  from  0 to 1.

!                 Leave precision and tolerance the same as above.

!                 The tanh-sinh algorithm is good at handling singularities at the endpoints,
!                 so this case takes about the same number of function evaluations as case 1.

      N = 2
      WRITE (*,"(//)")
      A = 0
      B = 1
      TOL = TO_FM(10) ** (-K)
      KPRT = 1
      NW = 6

      CALL FM_INTEGRATE(F,N,A,B,TOL,RESULT,KPRT,NW)

      CHECK = LOG( SQRT( 2*PI ) )
      ERR = ABS( RESULT - CHECK )
      CALL FM_FORM('ES12.4',ERR,ST1)
      WRITE (*,"(/10X,A,I2,A,A)") ' Error for case ',N,' = ',TRIM(ST1)
      IF (ERR > TOL) N_ERRORS = N_ERRORS + 1



!             3.  This integral has a pole at pi/2 and a sqrt singularity at zero.
!                 (from "A Comparison of Three High-Precision Quadrature Schemes" by D. H. Bailey,
!                 K. Jeyabalan, and X. S. Li, in Experimental Mathematics, Vol 14 (2005), No. 3)

!                 integrate  sqrt( tan(t) )  from  0 to pi/2.

!                 Set tolerance to give 100 digits.

!                 Here the fact that the pi/2 endpoint is not exactly representable in floating
!                 point form causes a problem.  FM_INTEGRATE will increase precision above the
!                 user's level while computing the integral.  But the endpoints A and B are input
!                 values that were defined at the user's precision, and their extra digits will be
!                 zeros when precision is raised in FM_INTEGRATE.

!                 That is fine for A = 0, but B = pi/2 will still be accurate only to the user's
!                 precision, not to the higher intermediate precision.  The fact that B is a
```

```
!                singularity for the function means that FM_INTEGRATE will need to know the
!                position of B to higher precision to evaluate the integral accurately.

!                The fix is to make a change of variables to get an equivalent integral where
!                both endpoints are exact in floating point.  Leaving a singular endpoint inexact
!                will usually cause FM_INTEGRATE to run much slower, and sometimes fail.

!                Let  u = t * 2 / pi.  Then  t = u * pi / 2  and  dt = pi/2 du.
!                The new form of this integral becomes:

!                integrate  sqrt( abs( tan( u * pi / 2 ) ) ) * pi / 2  from  0 to 1.

!                Now pi will be computed inside function F, so it will be done at whatever higher
!                precision FM_INTEGRATE uses.

!                When changing variables in this case, we also need to defend against rounding
!                errors when computing  u * pi / 2.  When u is very close to 1, rounding could
!                cause u * pi / 2 to round up, giving a value slightly greater than pi/2.
!                Then tan would return a negative value and then sqrt would return unknown,
!                causing the integration to fail.  The fix is to take the absolute value before
!                doing the square root.

      N = 3
      K = 100
      CALL FM_SET(K+20)

!             Precision has increased, so we must get pi at the new precision.

      CALL FM_PI(PI)
      WRITE (*,"(//)")
      A = 0
      B = 1
      TOL = TO_FM(10) ** (-K)
      KPRT = 1
      NW = 6

      CALL FM_INTEGRATE(F,N,A,B,TOL,RESULT,KPRT,NW)

      CHECK = PI * SQRT( TO_FM(2) ) / 2
      ERR = ABS( RESULT - CHECK )
      CALL FM_FORM('ES12.4',ERR,ST1)
      WRITE (*,"(/10X,A,I2,A,A)") ' Error for case ',N,' = ',TRIM(ST1)
      IF (ERR > TOL) N_ERRORS = N_ERRORS + 1



!             4.  integrate  exp( -t**2 / 2 )  from  0 to infinity.
!                 (from "A Comparison of Three High-Precision Quadrature Schemes" by D. H. Bailey,
!                 K. Jeyabalan, and X. S. Li, in Experimental Mathematics, Vol 14 (2005), No. 3)

!                Set tolerance to give 100 digits.

!                Infinite regions must be converted to finite ones.  Let u = 1/(t+1) to get:

!                integrate  exp( -(1/u - 1)**2 / 2 ) / u**2  from  0 to 1.

!                Exponential functions pose another problem for FM_INTEGRATE.
!                When u is very close to zero the exponential can underflow.  FM does not flush
```

```
!                underflows to zero like most floating point systems, so when that value is
!                then divided by the small u**2 FM detects the possibility that this result
!                could be above the underflow threshold.  Since FM can't be sure whether the
!                true function value is below the underflow threshold, unknown is returned.

!                The fix in this case is to see that whenever underflow occurs in this integration
!                the final function value is too small to change the integral.  That is the usual
!                situation whenever F underflows and the final value of the integral is greater
!                than 10**(-10**6) in magnitude, because FM's underflow is less than 10**(-10**8).
!                So we check for underflow after doing the exponential in function F and replace
!                underflowed function values by zero.

      N = 4
      K = 100
      CALL FM_SET(K+20)
      CALL FM_PI(PI)
      WRITE (*,"(//)")
      A = 0
      B = 1
      TOL = TO_FM(10) ** (-K)
      KPRT = 1
      NW = 6

      CALL FM_INTEGRATE(F,N,A,B,TOL,RESULT,KPRT,NW)

      CHECK = SQRT( PI / 2 )
      ERR = ABS( RESULT - CHECK )
      CALL FM_FORM('ES12.4',ERR,ST1)
      WRITE (*,"(/10X,A,I2,A,A)") ' Error for case ',N,' = ',TRIM(ST1)
      IF (ERR > TOL) N_ERRORS = N_ERRORS + 1



!             5.  integrate  log( abs( ( tan(t) + sqrt(7) ) / ( tan(t) - sqrt(7) ) ) )
!                            from  pi/3 to pi/2.
!                 (from "High-Precision Numerical Integration: Progress and Challenges"
!                 by D. H. Bailey and J. M. Borwein (2009))

!                 Set tolerance to give 150 digits.

!                 There is only one singularity, but it is atan(sqrt(7)), which is not an endpoint.
!                 FM_INTEGRATE will initially have very slow convergence and then will try to
!                 isolate the singularity and split into two integrals with the singularity at
!                 endpoints.

!                 This strategy works here, but it is slower and doesn't always succeed.
!                 Case 6 shows a better way to handle interior singularities.

      N = 5
      K = 150
      CALL FM_SET(K+20)
      CALL FM_PI(PI)
      WRITE (*,"(//)")
      A = PI/3
      B = PI/2
      TOL = TO_FM(10) ** (-K)
      KPRT = 1
      NW = 6
```

```
      CALL FM_INTEGRATE(F,N,A,B,TOL,RESULT,KPRT,NW)

      SEVEN = 7
      CHECK = ( SQRT(SEVEN) / 168 ) * ( POLYGAMMA(1,1/SEVEN) + POLYGAMMA(1,2/SEVEN) -  &
                                        POLYGAMMA(1,3/SEVEN) + POLYGAMMA(1,4/SEVEN) -  &
                                        POLYGAMMA(1,5/SEVEN) - POLYGAMMA(1,6/SEVEN) )
      ERR = ABS( RESULT - CHECK )
      CALL FM_FORM('ES12.4',ERR,ST1)
      WRITE (*,"(/10X,A,I2,A,A)") ' Error for case ',N,' = ',TRIM(ST1)
      IF (ERR > TOL) N_ERRORS = N_ERRORS + 1



!             6.  Same integral as case 5.

!                 integrate  log( abs( ( tan(t) + sqrt(7) ) / ( tan(t) - sqrt(7) ) ) )
!                            from  pi/3 to pi/2.

!                 Set tolerance to give 150 digits.

!                 Split into two integrals and change variables to make the endpoints exact.
!                 This will be faster than making FM_INTEGRATE search for the interior singularity
!                 as in case 5.
!                 Call the two function numbers 61 and 62.

!                 1.  from  pi/3  to  atan(sqrt(7)).
!                     Let  u = ( t - pi/3 ) / ( atan( sqrt(7) ) - pi/3 )

!                 2.  from  atan(sqrt(7)) to pi/2.
!                     Let  v = ( t - atan(sqrt(7)) ) / ( pi/2 - atan( sqrt(7) ) )

!                 This gives two integrals from 0 to 1, then we add the two results.

      N = 6
      K = 150
      CALL FM_SET(K+20)
      CALL FM_PI(PI)
      WRITE (*,"(//)")
      A = 0
      B = 1
      TOL = TO_FM(10) ** (-K)
      KPRT = 1
      NW = 6

      CALL FM_INTEGRATE(F,61,A,B,TOL,R1,KPRT,NW)
      CALL FM_INTEGRATE(F,62,A,B,TOL,R2,KPRT,NW)
      RESULT = R1 + R2

      WRITE (*,*) ' '
      WRITE (*,*) ' Adding these last two integrals gives the case 6 result:'
      WRITE (*,*) ' '
      CALL FM_PRINT(RESULT)

      SEVEN = 7
      CHECK = ( SQRT(SEVEN) / 168 ) * ( POLYGAMMA(1,1/SEVEN) + POLYGAMMA(1,2/SEVEN) -  &
                                        POLYGAMMA(1,3/SEVEN) + POLYGAMMA(1,4/SEVEN) -  &
                                        POLYGAMMA(1,5/SEVEN) - POLYGAMMA(1,6/SEVEN) )
```

```
      ERR = ABS( RESULT - CHECK )
      CALL FM_FORM('ES12.4',ERR,ST1)
      WRITE (*,"(/10X,A,I2,A,A)") ' Error for case ',N,' = ',TRIM(ST1)
      IF (ERR > TOL) N_ERRORS = N_ERRORS + 1



!             7.  Same integral as cases 5 and 6.
!                 Combine these two integrals into one, so only one call to FM_INTEGRATE is needed.
!                 This will be faster than doing two calls as in case 6.

!                 integrate  log( abs( ( tan(t) + sqrt(7) ) / ( tan(t) - sqrt(7) ) ) )
!                            from  pi/3 to pi/2.

!                 Set tolerance to give 150 digits.

!                 Split into two integrals and change variables to make the endpoints exact.
!                 Both new integrals are from 0 to 1.

!                 1.  from  pi/3  to  atan(sqrt(7)).
!                     Let  u = ( t - pi/3 ) / ( atan( sqrt(7) ) - pi/3 )

!                 2.  from  atan(sqrt(7)) to pi/2.
!                     Let  v = ( t - atan(sqrt(7)) ) / ( pi/2 - atan( sqrt(7) ) )

      N = 7
      K = 150
      CALL FM_SET(K+20)
      CALL FM_PI(PI)
      WRITE (*,"(//)")
      A = 0
      B = 1
      TOL = TO_FM(10) ** (-K)
      KPRT = 1
      NW = 6

      CALL FM_INTEGRATE(F,7,A,B,TOL,RESULT,KPRT,NW)

      SEVEN = 7
      CHECK = ( SQRT(SEVEN) / 168 ) * ( POLYGAMMA(1,1/SEVEN) + POLYGAMMA(1,2/SEVEN) -  &
                                        POLYGAMMA(1,3/SEVEN) + POLYGAMMA(1,4/SEVEN) -  &
                                        POLYGAMMA(1,5/SEVEN) - POLYGAMMA(1,6/SEVEN) )
      ERR = ABS( RESULT - CHECK )
      CALL FM_FORM('ES12.4',ERR,ST1)
      WRITE (*,"(/10X,A,I2,A,A)") ' Error for case ',N,' = ',TRIM(ST1)
      IF (ERR > TOL) N_ERRORS = N_ERRORS + 1



      WRITE (*,*) ' '
      WRITE (*,*) ' '
      IF (N_ERRORS == 0) THEN
          WRITE (*,*) ' All results were ok -- no errors were found.'
      ELSE
          WRITE (*,*) N_ERRORS,' error(s) were found.'
      ENDIF
      WRITE (*,*) ' '
```

```
      STOP
      END PROGRAM TEST

      FUNCTION F(X,N)     RESULT (RETURN_VALUE)
      USE FMZM
      IMPLICIT NONE

      TYPE (FM) :: RETURN_VALUE, X
      TYPE (FM), SAVE :: C1, C2, PI, SQRT7, TANX
      INTEGER :: N

      IF (N == 1) THEN
          RETURN_VALUE = LOG(X) * COS(X)
      ELSE IF (N == 2) THEN
          RETURN_VALUE = LOG( GAMMA( X ) )
      ELSE IF (N == 3) THEN

!             The original limits from 0 to pi/2 have been changed to 0 to 1.

          CALL FM_PI(PI)
          RETURN_VALUE = PI * SQRT( ABS( TAN( PI * X / 2 ) ) ) / 2
      ELSE IF (N == 4) THEN

!             Exp could underflow and then make F unknown.
!             Check for the underflow and set F = 0 in that case.

          RETURN_VALUE = EXP( -(1 - 1/X)**2 / 2 )
          IF ( IS_UNDERFLOW(RETURN_VALUE) ) THEN
              RETURN_VALUE = 0
          ELSE
              RETURN_VALUE = RETURN_VALUE / X**2
          ENDIF
      ELSE IF (N == 5) THEN
          SQRT7 = SQRT(TO_FM(7))
          TANX = TAN(X)
          RETURN_VALUE = LOG( ABS( ( TANX + SQRT7 ) / ( TANX - SQRT7 ) ) )
      ELSE IF (N == 61) THEN
          CALL FM_PI(PI)

!             It is tempting to compute constants like C1, C2, SQRT7 once and then save them for
!             use in subsequent calls to F.  That can be done, but it is trickier than it seems,
!             since FM_INTEGRATE may call F with different precision levels during one integration,
!             so it is easy to not have the right precision in a saved variable.
!             Here we just compute them each time, making the logic straightforward while the
!             function evaluations are somewhat slower.

          SQRT7 = SQRT(TO_FM(7))
          C1 = ATAN(SQRT7) - PI/3
          TANX = TAN( C1*X + PI/3 )
          RETURN_VALUE = C1 * LOG( ABS( ( TANX + SQRT7 ) / ( TANX - SQRT7 ) ) )
      ELSE IF (N == 62) THEN
          CALL FM_PI(PI)
          SQRT7 = SQRT(TO_FM(7))
          C2 = ATAN(SQRT7)
          C1 = PI/2 - C2
          TANX = TAN( C1*X + C2 )
          RETURN_VALUE = C1 * LOG( ABS( ( TANX + SQRT7 ) / ( TANX - SQRT7 ) ) )
      ELSE IF (N == 7) THEN
```

```
!             Combine the two integrals into one.

          CALL FM_PI(PI)
          SQRT7 = SQRT(TO_FM(7))
          C2 = ATAN(SQRT7)
          C1 = C2 - PI/3
          TANX = TAN( C1*X + PI/3 )
          RETURN_VALUE = C1 * LOG( ABS( ( TANX + SQRT7 ) / ( TANX - SQRT7 ) ) )

          C1 = PI/2 - C2
          TANX = TAN( C1*X + C2 )
          RETURN_VALUE = RETURN_VALUE + C1 * LOG( ABS( ( TANX + SQRT7 ) / ( TANX - SQRT7 ) ) )
      ENDIF

      END FUNCTION F
```