```fortran
      program test
      use fmzm

!   fm_find_min is a multiple precision function minimization routine that uses Brent's method.

!   The function to be minimized or maximized is  f(x, nf).
!   x  is the argument to the function.
!   nf is the function number in case extrema to several functions are needed.

      implicit none
      character(80)  :: st1, st2

!  Declare the multiple precision variables.

      type (fm), save     :: a, b, tol, xval, fval
      type (fm), external :: f

!             Set the FM precision to 50 significant digits (plus a few more "guard digits")

      call fm_set(50)

!             Find a minimum of the first function,  x**3 - 9*x + 17.
!             a, b are two endpoints of an interval in which the search takes place.

      a = 1
      b = 2

!             tol is the error tolerance.

      tol = epsilon(a)

      write (*,*) ' '
      write (*,*) ' '
      write (*,*) ' Case 1.  Call fm_find_min to find a relative minimum between 1 and 2'
      write (*,*) '          for f(x) = x**3 - 9*x + 17.'
      write (*,*) '          Use kprt = 0, so no output will be done in the routine, then'
      write (*,*) '          write the results from the main program.'

!             For this call no trace output will be done (kprt = 0).

      call fm_find_min(1, a, b, tol, xval, fval, f, 1, 0, 6)

!             Write the result, using f52.50 format, and compare xval to the true minimum, sqrt(3).

      call fm_form('f52.50', xval, st1)
      call fm_form('f52.50', fval, st2)
      write (*, "(/'  A minimum for function 1 is'/'     x  = ', a/'   f(x) = ', a)")  &
              trim(st1), trim(st2)
      call fm_form('es17.10', abs(xval-sqrt(to_fm(3))), st2)
      write (*, "(/'  Error for x = ',a)") trim(st2)

!             Find a maximum of the first function,  x**3 - 9*x + 17.

!             This time we use fm_find_min's built-in trace (kprt = 1) to print the final
!             approximation to the root.  The output will appear on more than one line, to
!             allow for the possibility that precision could be hundreds or thousands of digits,
!             so the number might not fit on one line.
```

```fortran
      write (*,*) ' '
      write (*,*) ' '
      write (*,*) ' Case 2.  Find a relative maximum between -5 and 1.'
      write (*,*) '          Use kprt = 1, so fm_find_min will print the results.'

      call fm_find_min(2, -to_fm('5.0d0'), to_fm('1.0d0'), tol, xval, fval, f, 1, 1, 6)

!              Find a maximum of the first function,  x**3 - 9*x + 17.

!              See what happens when the maximum value is at an endpoint of the search interval.
!              The algorithm still finds a relative maximum in the interior of the interval,
!              not the absolute maximum at x=5.

      write (*,*) ' '
      write (*,*) ' '
      write (*,*) ' Case 3.  Find a relative maximum between -5 and 5.'
      write (*,*) '          Use kprt = 2, so fm_find_min will print all iterations,'
      write (*,*) '          as well as the final results.'

      call fm_find_min(2, -to_fm('5.0d0'), to_fm('5.0d0'), tol, xval, fval, f, 1, 2, 6)

!              Find a minimum of the second function,  gamma(x).

      write (*,*) ' '
      write (*,*) ' '
      write (*,*) ' Case 4.  The gamma function has one minimum for positive x.'
      write (*,*) '          Find it, printing all iterations.'

      call fm_find_min(1, to_fm('0.1d0'), to_fm('3.0d0'), tol, xval, fval, f, 2, 2, 6)


      write (*,*) ' '

      end program test


      function f(x, nf)     result (return_value)
      use fmzm

!   x  is the argument to the function.
!   nf is the function number.

      implicit none
      integer :: nf
      type (fm) :: return_value, x
      intent (in) :: x, nf

      if      (nf == 1) then
          return_value = x**3 - 9*x + 17
      else if (nf == 2) then
          return_value = gamma(x)
      else
          return_value = 3*x**2 + x - 2
      endif

      end function f
```