```
c===========================================================
c     Computes and reports estimate of machine epsilon.
c
c     Recall: machine epsilon is smallest positive 'eps'
c     such that
c
c             (1.0d0 + eps ) .ne. (1.0d0)
c
c     Program accepts optional argument which specifies
c     division factor: values close to 1.0 will result
c     in more accurate estimate of machine epsilon.
c===========================================================
      program         meps

      implicit        none

c-----------------------------------------------------------
c     Note use of 'r8arg', available in 'libp329f.a' which
c     works exactly like 'i4arg' except that it returns
c     a real*8 value parsed from the specified command-line
c     argument
c-----------------------------------------------------------
      real*8          r8arg

      real*8          default_fac
      parameter     ( default_fac = 2.0d0 )

      real*8          eps,         neweps,        fac

      fac = r8arg(1,default_fac)
      write(0,*) 'meps: using division factor: ', fac
```

```
eps    = 1.0d0
neweps = 1.0d0
do while( .true. )
   if( 1.0d0 .eq. (1.0d0 + neweps) ) then
      write(*,*) eps
      stop
   else
      eps    = neweps
      neweps = neweps / fac
   end if
end do

stop

end
```

```
############################################################
# Output from 'meps' on SGIs (IEEE 64-bit floating point).
############################################################

einstein% make meps
        f77 -g -n32 -c meps.f
        f77 -g -n32 -L/usr/localn32/lib -n32 \
             meps.o -lp329f -o meps

einstein% meps
 meps: using division factor:    2.000000000000000
   2.2204460492503131E-16

einstein% meps 1.01
 meps: using division factor:    1.010000000000000
   1.1104218387155329E-16

einstein% meps 1.0001
 meps: using division factor:    1.000100000000000
   1.1102645224601785E-16
```

```
#############################################################
# Output from 'meps' on Cray J90 (Cray 64-bit floating point)
#############################################################

#############################################################
# First consider source code:  Only difference between this
# version and the SGI version is that here we use 'e0'
# rather than 'd0' for real*8 constants.  A double precision
# variable or constant (d0) on a Cray system uses 16 bytes
# and arithmetic involving such quantities is *not*
# implemented in hardware, and consequently is very slow
# compared to real*8 (e0) arithmetic.
#############################################################

charon 21> cat meps.f
c============================================================
c     Computes and reports estimate of machine epsilon.
c
c     Recall: machine epsilon is smallest positive 'eps'
c     such that
c
c             (1.0e0 + eps ) .ne. (1.0e0)
c
c     Program accepts optional argument which specifies
c     division factor: values close to 1.0 will result
c     in more accurate estimate of machine epsilon.
c============================================================
      program         meps

      implicit        none
```

```
c-------------------------------------------------------------
c     Note use of 'r8arg', available in 'libp329f.a' which
c     works exactly like 'i4arg' except that it returns
c     a real   value parsed from the specified command-line
c     argument
c-------------------------------------------------------------
      real              r8arg

      real              default_fac
      parameter       ( default_fac = 2.0e0 )

      real              eps,         neweps,        fac

      fac = r8arg(1,default_fac)
      write(0,*) 'meps: using division factor: ', fac

      eps    = 1.0e0
      neweps = 1.0e0
      do while( .true. )
         if( 1.0e0 .eq. (1.0e0 + neweps) ) then
            write(*,*) eps
            stop
         else
            eps    = neweps
            neweps = neweps / fac
         end if
      end do

      stop

      end
```

```
charon 22> meps
 meps: using division factor: 2.
 7.105427357601001E-15
 STOP executed at line 38 in Fortran routine 'MEPS'
 CPU: 0.005s,  Wallclock: 0.015s,  4.2% of 8-CPU Machine
 Memory HWM: 200255, Stack HWM: 2048, Stack segment expansions: 0

charon 23> meps 1.01
 meps: using division factor: 1.009999999999998
 7.10922461021392E-15
 STOP executed at line 38 in Fortran routine 'MEPS'
 CPU: 0.009s,  Wallclock: 0.020s,  5.6% of 8-CPU Machine
 Memory HWM: 200257, Stack HWM: 2048, Stack segment expansions: 0

charon 24> meps 1.0001
 meps: using division factor: 1.000100000000003
 7.105756717509093E-15
 STOP executed at line 38 in Fortran routine 'MEPS'
 CPU: 0.338s,  Wallclock: 0.348s,  12.1% of 8-CPU Machine
 Memory HWM: 200257, Stack HWM: 2048, Stack segment expansions: 0
```

```
c===========================================================
c     Illustrates IEEE "exceptional" floating point "values"
c
c     By default, floating point exceptions (underflow,
c     overflow, divide by 0, inexact result) are ignored
c     in program execution.  On the SGIs, this behaviour
c     can be changed so that all FPE's are "trapped" and
c     result in termination of program execution by
c
c     (1) Linking with the 'fpe' library
c
c         % $(F77_LOAD) tfpe.o -lp329f -lfpe -o tfpe
c
c     (2) Setting the environment variable 'TRAP_FPE'
c         prior to program execution
c
c         % setenv TRAP_FPE "ALL=ABORT"
c
c     See 'man handle_sigfpes' (FORTRAN pages) for more
c     detailed information.
c===========================================================
      program         tfpe

      implicit        none

      real*8          r8arg

      real*8          divby0,     overflow,    notnumber
```

```
divby0    = 1.0d0 / r8arg(1,0.0d0)
write(0,*) 'divby0 = ', divby0

overflow  = exp(1.0d10)
write(0,*) 'overflow = ', overflow

notnumber = sqrt(-1.0d0)
write(0,*) 'notnumber = ', notnumber


stop

end
```

```
#############################################################
# Output from 'tfpe' on SGIs (Illustrates IEEE exceptional
# values).  This is the normal output where floating
# point exceptions are ignored and program execution
# continues.
#
# 'nan' stands for 'not a number'
#############################################################

einstein% make tfpe
        f77 -g -n32 -c tfpe.f
        f77 -g -n32 -L/usr/localn32/lib -n32 tfpe.o -lp329f -o tfpe

einstein% tfpe
 divby0 = Infinity
 overflow = Infinity
 notnumber = nan
```

```
################################################################
# Output from 'tfpe' on SGIs illustrating exception
# trapping via 'libfpe.a' and 'TRAP_FPE' environment vbl
################################################################
einstein% make tfpe
        f77 -g -n32 -c tfpe.f
        f77 -g -n32 -L/usr/localn32/lib -n32 tfpe.o \
                    -lfpe -lp329f -o tfpe
ld32: WARNING 85: definition of __checktraps in \
      /usr/lib32/mips3/libfpe.so preempts that definition \
      in /usr/lib32/mips3/libc.so.
ld32: WARNING 85: definition of __readenv_sigfpe in \
      /usr/lib32/mips3/libfpe.so preempts that definition in \
      /usr/lib32/mips3/libc.so.

################################################################
# First invocation produces same results as previously.
################################################################
einstein% tfpe
 divby0 = Infinity
 overflow = Infinity
 notnumber = nan
```

```
##############################################################
# Enable trapping of all floating point exceptions.
# See 'man handle_sigfpes' for more information.
##############################################################
einstein% setenv TRAP_FPE "ALL=ABORT"

##############################################################
# Second invocation aborts after first floating point
# exception.
##############################################################
einstein% tfpe




libfpe: PID 3228 aborting; limit reached for trap type DIVZERO
IOT Trap
Abort (core dumped)


##############################################################
# Remember to remove the 'core' file.
##############################################################
einstein% ls core
core

einstein% RM core
```

```
################################################################
# The debugger (dbx) can help you isolate the problem.
################################################################
einstein% dbx tfpe
dbx version 7.1 Dec  3 1996 17:03:19
Executable /usr2/people/phy329/f77/ex6/tfpe
(dbx) run
Process  3233 (tfpe) started
Process  3233 (tfpe) stopped on signal SIGFPE: \
    Floating point exception (handler __catch) at \
    [tfpe:30 +0x1c,0x10001900]

  30  divby0    = 1.0d0 / r8arg(1,0.0d0)
(dbx) list
>*  30        divby0    = 1.0d0 / r8arg(1,0.0d0)
    31        write(0,*) 'divby0 = ', divby0
    32
    33        overflow  = exp(1.0d10)
    34        write(0,*) 'overflow = ', overflow
    35
    36        notnumber = sqrt(-1.0d0)
    37        write(0,*) 'notnumber = ', notnumber
    38
    39
(dbx) quit
```

```
#############################################################
# Output from 'tfpe' on the Cray J90.  The Cray (sensibly)
# traps all floating point exceptions by default.
#############################################################

charon% tfpe
Floating point exception

 Beginning of Traceback:
  Interrupt at address 424b in routine 'TFPE'.
  Called from line 334 (address 22661d) in routine '$START$'.
 End of Traceback.
Floating exception (core dumped)
```

```
c===========================================================
c     Program illustrating "catastrophic" loss of precision
c     resulting from the subtraction of two nearly equal
c     floating point values.
c===========================================================
      program         catprec

      implicit        none

      real*8          x
      parameter     ( x = 0.2d0 )

      integer         i
      real*8          h,     dsinx

      write(*,*) '      h      d(sin) approx   '//
     &           'd(sin) exact    d(sin) err'
      write(*,*)

      h = 0.5d0
      do i = 1 , 16
c-----------------------------------------------------------
c        Algebraically, in the limit h -> 0, dsinx should
c        approach cos(x), but sin(x+h) -> sin(x) so
c        catastrophic loss of precision occurs.
c-----------------------------------------------------------
         dsinx = (sin(x+h) - sin(x)) / h
         write(*,1000) h,  dsinx,  cos(x),   dsinx - cos(x)
1000     format(1P,E12.3,2E16.8,E12.3)
         h = 0.125d0 * h
      end do

      stop
      end
```

```
##############################################################
# Output from 'catprec' illustrating catastrophic precision
# loss due to subtraction of nearly-equal floating point
# values.
##############################################################

einstein% make catprec
        f77 -g -n32 -c catprec.f
        f77 -g -n32 -L/usr/localn32/lib -n32 catprec.o -o catprec

einstein% catprec

       h      d(sin) approx   d(sin) exact    d(sin) err

   5.000E-01  8.91096713E-01  9.80066578E-01  -8.897E-02
   6.250E-02  9.73222242E-01  9.80066578E-01  -6.844E-03
   7.812E-03  9.79280560E-01  9.80066578E-01  -7.860E-04
   9.766E-04  9.79969416E-01  9.80066578E-01  -9.716E-05
   1.221E-04  9.80054450E-01  9.80066578E-01  -1.213E-05
   1.526E-05  9.80065062E-01  9.80066578E-01  -1.516E-06
   1.907E-06  9.80066388E-01  9.80066578E-01  -1.895E-07
   2.384E-07  9.80066554E-01  9.80066578E-01  -2.369E-08
   2.980E-08  9.80066575E-01  9.80066578E-01  -2.731E-09
   3.725E-09  9.80066583E-01  9.80066578E-01   4.719E-09
   4.657E-10  9.80066597E-01  9.80066578E-01   1.962E-08
   5.821E-11  9.80066776E-01  9.80066578E-01   1.984E-07
   7.276E-12  9.80068207E-01  9.80066578E-01   1.629E-06
   9.095E-13  9.80072021E-01  9.80066578E-01   5.444E-06
   1.137E-13  9.80224609E-01  9.80066578E-01   1.580E-04
   1.421E-14  9.80468750E-01  9.80066578E-01   4.022E-04
```