**Source file:** `fdemo1.f`

```
c===========================================================
c     fdemo1:  Program which demonstrates many of the
c     essential features of Fortran 77. Some 'safe' language
c     extensions are used; all extensions are valid
c     Fortran 90.
c===========================================================

c===========================================================
c     Source code formatting rules:
c
c     Columns    Use
c
c     1-5        numeric statement label
c     6          continuation character: '&' recommended
c     7-72       statement
c
c     BE EXTREMELY CAREFUL NOT TO TYPE BEYOND COLUMN 72!
c===========================================================
C     COMMENT LINES: Use 'c' 'C' or '*' IN FIRST COLUMN
*===========================================================

c-----------------------------------------------------------
c     The 'program' statement names a Fortran main routine.
c     Optional, but recommended and note that there can
c     only be one 'program' (main routine) per executable.
c-----------------------------------------------------------
      program        fdemo1

c===========================================================
c     BEGINNING OF DECLARATION STATEMENTS
c
c     Declarations (or specification statements) must
c     ALWAYS appear before ANY executable statements.
c===========================================================

c-----------------------------------------------------------
c     The 'implicit none' statement is an extension which
c     forces us to explicitly declare all variables and
c     functions (apart from Fortran built in functions).
c     HIGHLY RECOMMENDED.
c-----------------------------------------------------------
      implicit       none

c-----------------------------------------------------------
c     PARAMETERS
c-----------------------------------------------------------
c     The parameter declaration effectively assigns a
c     CONSTANT value to a name.  Note that each
c     parameter statement must be accompanied by an
c     appropriate declaration of the type of the
c     parameter.  Also note that, except in strings,
c     blanks (spaces) are ignored in Fortran---you can
c     use this fact to make code more readable.
c-----------------------------------------------------------
      integer        zero
      parameter     ( zero = 0 )

c-----------------------------------------------------------
c     Always specify floating point constants using
c     scientific notation.  Use 'd' (instead of 'e') for
c     real*8 constants.
c-----------------------------------------------------------
      real*8         pi
      parameter     ( pi   = 3.141 5926 5358 9793 d0 )

      real*8         tiny
      parameter     ( tiny = 1.0 d-50 )

c-----------------------------------------------------------
c     VARIABLES
c-----------------------------------------------------------
c     The main data types we will be using are
c
c     integer,  real*8,  logical,
c     character*1,  character*2, ... etc., character*(*)
c
c     but note that Fortran has support for complex
c     arithmetic.  Note that complex*16 means real*8
c     values are used for both the real and imaginary
c     parts of the variable.
c-----------------------------------------------------------

c-----------------------------------------------------------
c     (a) SCALARS
c-----------------------------------------------------------
      real*8         a,       b,       c
      real*8         res1,    res2,    res3,    res4
      integer        i,       j,       k,       n
      integer        ires1,   ires2,   ires3,   ires4
      logical        switch
      logical        lres1,   lres2,   lres3
      complex*16     ca,      cb
c-----------------------------------------------------------
c     (b) ARRAYS
c-----------------------------------------------------------
      integer        n1,      n2,      n3
      parameter    ( n1 = 4,  n2 = 3,  n3 = 2)

c-----------------------------------------------------------
c     (b.1) 1-D ARRAYS:  Note, in a main program, all
c     dimension bounds must be integer parameters or
c     integer constants.
c-----------------------------------------------------------
      real*8         r1a(n1),   r1b(n2)
      integer        i1i(n1)
c-----------------------------------------------------------
c     (b.2) 2-D ARRAYS:
c-----------------------------------------------------------
      real*8         r2a(n1,n2)
c-----------------------------------------------------------
c     (b.3) 3-D ARRAYS:
c-----------------------------------------------------------
      real*8         r3a(n1,n2,n3)

c===========================================================
c     END OF DECLARATION STATEMENTS
c===========================================================

c===========================================================
c     BEGINNING OF EXECUTABLE STATEMENTS
c===========================================================

c***********************************************************
c     Assignment statements and simple arithmetic
c     expressions
c***********************************************************

c-----------------------------------------------------------
c     Assignment to scalar variables ... again, note
c     the use of scientific notation (d0) to specify
c     a real*8 constant.
c
c     The only valid logical constants are .true. and
c     .false.  (don't forget to include the .'s)
c-----------------------------------------------------------
      a =  0.025d0
      b = -1.234d-16
      c =  1.0d0
      i =  3000
      switch = .true.

c-----------------------------------------------------------
c     Note the use of the continuation character in
c     column 6 to continue a statement on a second line.
c-----------------------------------------------------------
      write(*,*) 'a = ', a,' b = ', b
      write(*,*) ' c = ', c, ' i = ', i,
     &           ' switch = ', switch
      call prompt('Through scalar assignment')

c-----------------------------------------------------------
c     Arithmetic expressions.  Fortran has standard
c     operator precedences except that the exponentiation
c     operator '**' associates RIGHT to LEFT:  e.g.
c
c     i ** j ** k  is equivalent to  i ** (j ** k)
c
c     Parentheses force evaluation of subexpressions.
```

```
c-----------------------------------------------------------
      a = 2.0d0
      b = 3.0d0
      c = 3.0d0

      res1  = a + b
      res2  = a**2 + b**2
      res3  = (a**2 + b**2)**(0.5d0)
      write(*,*) 'res1 = ', res1,' res2 = ', res2
      write(*,*) ' res3 = ', res3
      call prompt('Through real*8 arithmetic expressions')

c-----------------------------------------------------------
c     Notice the integer truncation which occurs when
c     dividing the integer 2 by the integer 3.
c-----------------------------------------------------------
      i = 2
      j = 3
      k = 2

      ires1 = 2 + 3
      ires2 = 2 / 3
      ires3 = i ** j ** k
      ires4 = (i ** j) ** k
      write(*,*) 'ires1 = ', ires1, ' ires2 = ', ires2
      write(*,*) 'ires3 = ', ires3, ' ires4 = ', ires4
      call prompt('Through integer arithmetic expressions')

c-----------------------------------------------------------
c     "Mixed-mode" computations
c-----------------------------------------------------------

c-----------------------------------------------------------
c     i + j  is  computed using integer arithmetic and
c     the result is converted to a real*8 value before being
c     assigned to res2.
c-----------------------------------------------------------
      res1 = i + j

c-----------------------------------------------------------
c     3 / 4 is evaluated using integer arithmetic (yielding
c     0) and then the value is converted to real*8.
c-----------------------------------------------------------
      res2 = 3 / 4

c-----------------------------------------------------------
c     The appearance of a double precision constant
c     forces the division to be computed using real*8
c     arithmetic
c-----------------------------------------------------------
      res3 = 3 / 4.0d0
      write(*,*) 'res1 = ', res1, ' res2 = ', res2
      write(*,*) ' res3 = ', res3
      call prompt('Through mixed-mode arithmetic')

c***********************************************************
c     CONTROL STATEMENTS
c***********************************************************

c***********************************************************
c     DO LOOPS
c
c     Note that 'end do' is not Fortran 77, but a safe
c     extension (it is legal Fortran 90).
c***********************************************************

      do i = 1 , 3
         write(*,*) 'Loop 1: i = ', i
      end do
      call prompt('Through loop 1')

c-----------------------------------------------------------
c     The same do loop with the optional loop increment
c     specified explicitly
c-----------------------------------------------------------
      do i = 1 , 3 , 1
         write(*,*) 'Loop 2: i = ', i
      end do
      call prompt('Through loop 2')

c-----------------------------------------------------------
c     Another do-loop with a non-default loop increment ...
c-----------------------------------------------------------
      do i = 1 , 7 , 2
         write(*,*) 'Loop 3: i = ', i
      end do
      call prompt('Through loop 3')

c-----------------------------------------------------------
c     ... and one with a negative increment
c-----------------------------------------------------------
      do i = 3 , 1 , -1
         write(*,*) 'Loop 4: i = ', i
      end do
      call prompt('Through loop 4')

c-----------------------------------------------------------
c     Nested do-loops.
c-----------------------------------------------------------
      do i = 1 , 3
         do j = 1 , 2
            write(*,*) 'Loop 5: i, j = ', i, j
         end do
      end do
      call prompt('Through loop 5')

c-----------------------------------------------------------
c     Any of the do-loop parameters can be variables,
c     expressions or parameters: safest to ALWAYS use
c     integer values.
c-----------------------------------------------------------
      n = 6
      do i = 2 , n , n / 3
         write(*,*) 'Loop 6: i = ', i
      end do
      call prompt('Through loop 6')


c***********************************************************
c     LOGICAL EXPRESSIONS
c
c     Note that the Fortran comparison and logical
c     operators all have the form: .operator.
c
c     Comparison:   .eq.    .ne.    .gt.    .lt.
c                   .ge.    .le.
c     Logical:      .not. (unary)
c                   .and.   .or.
c***********************************************************

      a = 25.0d0
      b = 12.0d0

      lres1 = a .gt. b
      lres2 = (a .lt. b)  .or.  (b .ge. 0.0d0)
      lres3 = a .eq. b
      write(*,*) 'lres1 = ', lres1, ' lres2 = ', lres2,
     &           ' lres3 = ', lres3
      call prompt('Through basic conditionals')

c***********************************************************
c     IF-THEN-ELSE STATEMENTS.
c***********************************************************

      if( a .gt. b ) then
         write(*,*) a, ' > ', b
      end if
      call prompt('Through if 1')

      if( b .gt. a ) then
         write(*,*) b, ' > ', a
      else
         write(*,*) a, ' > ', b
      end if
      call prompt('Through if 2')

c-----------------------------------------------------------
c     Nested IF statement.
c-----------------------------------------------------------
      if( a .gt. b ) then
         if( a .gt. 2 * b ) then
            write(*,*) a, ' > ', 2 * b
```

```
         else
            write(*,*) a, ' <= ', 2 * b
         end if
      else
         write(*,*) a, ' <= ', b
      end if
      call prompt('Through nested if')

c-------------------------------------------------------------
c     IF ... ELSE IF .. IF construct can be used in lieu
c     of 'CASE' statement.
c-------------------------------------------------------------
      do i = 1 , 4
         if(      i .eq. 1 ) then
            write(*,*) 'Case 1'
         else if( i .eq. 2 ) then
            write(*,*) 'Case 2'
         else if( i .eq. 3 ) then
            write(*,*) 'Case 3'
         else
            write(*,*) 'Default case'
         end if
      end do
      call prompt('Through case via if')

c*************************************************************
c     WHILE LOOPS
c
c     The do while( ... ) ... end do construct is valid
c     Fortran 90, and a safe Fortran 77 extension.
c*************************************************************
      a = 0.1d0
      b = 0.0d0
      do while ( b .le. 1.0d0 )
         write(*,*) 'Do while loop: b = ', b
         b = b + a
      end do
      call prompt('Through while loop')

c*************************************************************
c     USING BUILT-IN (INTRINSIC) FUNCTIONS
c*************************************************************
      res1 = sin(0.3d0 * Pi)
      res2 = cos(0.3d0 * Pi)
      res3 = res1**2 + res2**2
      res4 = sqrt(res3)
      write(*,*) 'res1 = ', res1, ' res2 = ', res2
      write(*,*) 'res3 = ', res3, ' res4 = ', res4
      call prompt('Through built-in fcn 1')

c-------------------------------------------------------------
c     atan, acos, asin, etc. return arctangent, arcosine,
c     arcsine etc. in RADIANS
c-------------------------------------------------------------
      res1 = atan(1.0d0)
      write(*,*) 'res1 = ', res1
      call prompt('Through built-in fcn 2')

c-------------------------------------------------------------
c     min and max will return the minimum and maximum
c     respectively of an arbitrary number of arguments
c     of any UNIQUE data type.  Do NOT mix types in
c     a single statement as in
c
c     write(*,*) min(1,2.0d0)
c-------------------------------------------------------------
      write(*,*) 'min(3.0d0,2.0d0) = ', min(3.0d0,2.0d0)
      write(*,*) 'min(1,-3,5,0) = ', min(1,-3,5,0)
      call prompt('Through built-in fcn 3')

c-------------------------------------------------------------
c     mod is particularly useful for calculating when one
c     integer divides another evenly
c-------------------------------------------------------------
      do i = 0 , 1000
         if( mod(i,100) .eq. 0 ) then
            write(*,*) 'i = ', i
         end if
      end do
      call prompt('Through built-in fcn 4')

c-------------------------------------------------------------
c     Stop program execution
c-------------------------------------------------------------
      call prompt('Through fdemo1')
      stop

c=============================================================
c     END OF EXECUTABLE STATEMENTS
c=============================================================

c-------------------------------------------------------------
c     End of program unit (fdemo1)
c-------------------------------------------------------------
      end

c=============================================================
c     Prints a message on stdout and then waits for input
c     from stdin.
c=============================================================
c-------------------------------------------------------------
c     A new program unit (prompt).
c-------------------------------------------------------------
      subroutine prompt(pstring)

         implicit      none

         character*(*) pstring
         integer       rc
         character*1   resp

         write(*,*) pstring
         write(*,*) 'Enter any non-blank character & '//
     &              'enter to continue'

         read(*,*,iostat=rc,end=900)  resp
c-------------------------------------------------------------
c        Return to calling program.
c-------------------------------------------------------------
         return

 900     continue
c-------------------------------------------------------------
c        Stop program execution.  This section of code is
c        the "end-of-file" handler for standard input
c        (via the end=900 clause of the read statement).
c        In this case, it is perfectly good style simply
c        to quit.
c-------------------------------------------------------------
         stop
c-------------------------------------------------------------
c     End of program unit (prompt).
c-------------------------------------------------------------
      end
```

**Source file:** `Makefile`

```
##############################################################
#  Note that this 'Makefile' assumes that the following
#  environment variables are set:
#
#     F77         ->  name of f77 compiler
#     F77FLAGS    ->  generic f77 flags
#     F77CFLAGS   ->  f77 flags for compilation phase
#     F77LFLAGS   ->  f77 flags for load phase
##############################################################
.IGNORE:

F77_COMPILE  = $(F77) $(F77FLAGS) $(F77CFLAGS)
F77_LOAD     = $(F77) $(F77FLAGS) $(F77LFLAGS)

.f.o:
   $(F77_COMPILE) $*.f

EXECUTABLES = fdemo1

all: $(EXECUTABLES)

fdemo1: fdemo1.o
   $(F77_LOAD) fdemo1.o -o fdemo1

clean:
   rm *.o
   rm $(EXECUTABLES)
```

**Source file:** `fdemo1-output`

```
##############################################################^
Wed Oct  6 15:17:09 PDT 2004
##############################################################

lnx1 1> cat Makefile
##############################################################
#  Note that this 'Makefile' assumes that the following
#  environment variables are set:
#
#     F77         ->  name of f77 compiler
#     F77FLAGS    ->  generic f77 flags
#     F77CFLAGS   ->  f77 flags for compilation phase
#     F77LFLAGS   ->  f77 flags for load phase
##############################################################
.IGNORE:

F77_COMPILE  = $(F77) $(F77FLAGS) $(F77CFLAGS)
F77_LOAD     = $(F77) $(F77FLAGS) $(F77LFLAGS)

.f.o:
   $(F77_COMPILE) $*.f

EXECUTABLES = fdemo1

all: $(EXECUTABLES)

fdemo1: fdemo1.o
   $(F77_LOAD) fdemo1.o -o fdemo1

clean:
   rm *.o
   rm $(EXECUTABLES)

##############################################################
lnx1 3> env | grep '^F77'
F77=pgf77
F77FLAGS=-g
F77CFLAGS=-c
F77LFLAGS=-L/usr/local/PGI/lib

##############################################################
lnx1 4> make

pgf77 -g -c fdemo1.f
pgf77 -g -L/usr/local/PGI/lib fdemo1.o -o fdemo1

##############################################################
# I encourage you to download 'fdemo1.f', compile it,
# and run it INTERACTIVELY yourself.  You should see
# output essentially identical to that shown below.
# Note, however, that both because I'm lazy, as well
# as to illustrate the use of I/O re-direction, I have
# previously prepared a file called 'INPUT', which
# contains many lines consisting of a single character
# These lines will be read by the 'prompt' subroutine
# which, when run interactively, writes a prompt to
# stdout and then waits for input from stdin.
##############################################################

lnx1 5> head -10 < INPUT
q
q
q
q
q
q
q
q
q
q

##############################################################
lnx1 6> fdemo1 < INPUT
 a =   2.5000000000000000E-002 b =  -1.2339999999999998E-016
  c =    1.000000000000000      i =         3000 switch =  T
 Through scalar assignment
 Enter any non-blank character & enter to continue
```

```
############################################################
# Note: For readability, all other instances of the
# following output from the 'prompting' routine have been
# converted to blank lines with a text editor command.
############################################################


 res1 =    5.000000000000000      res2 =    13.00000000000000
  res3 =    3.605551275463989
 Through real*8 arithmetic expressions

 ires1 =           5 ires2 =           0
 ires3 =         512 ires4 =          64
 Through integer arithmetic expressions

 res1 =    5.000000000000000      res2 =   0.0000000000000000E+000
  res3 =   0.7500000000000000
 Through mixed-mode arithmetic

 Loop 1: i =           1
 Loop 1: i =           2
 Loop 1: i =           3
 Through loop 1

 Loop 2: i =           1
 Loop 2: i =           2
 Loop 2: i =           3
 Through loop 2

 Loop 3: i =           1
 Loop 3: i =           3
 Loop 3: i =           5
 Loop 3: i =           7
 Through loop 3

 Loop 4: i =           3
 Loop 4: i =           2
 Loop 4: i =           1
 Through loop 4

 Loop 5: i, j =           1           1
 Loop 5: i, j =           1           2
 Loop 5: i, j =           2           1
 Loop 5: i, j =           2           2
 Loop 5: i, j =           3           1
 Loop 5: i, j =           3           2
 Through loop 5

 Loop 6: i =           2
 Loop 6: i =           4
 Loop 6: i =           6
 Through loop 6

 lres1 =  T lres2 =  T lres3 =  F
 Through basic conditionals

    25.00000000000000      >    12.00000000000000
 Through if 1

    25.00000000000000      >    12.00000000000000
 Through if 2

    25.00000000000000      >    24.00000000000000
 Through nested if

 Case 1
 Case 2
 Case 3
 Default case
 Through case via if

 Do while loop: b =   0.0000000000000000E+000
 Do while loop: b =   0.1000000000000000
 Do while loop: b =   0.2000000000000000
 Do while loop: b =   0.3000000000000000
 Do while loop: b =   0.4000000000000000
 Do while loop: b =   0.5000000000000000
 Do while loop: b =   0.6000000000000000
 Do while loop: b =   0.7000000000000000
 Do while loop: b =   0.7999999999999999
 Do while loop: b =   0.9000000000000000
 Do while loop: b =   0.9999999999999998
 Through while loop

 res1 =   0.8090169943749475      res2 =   0.5877852522924732
 res3 =    1.000000000000000      res4 =    1.000000000000000
 Through built-in fcn 1

 res1 =   0.7853981633974483
 Through built-in fcn 2

 min(3.0d0,2.0d0) =    2.000000000000000
 min(1,-3,5,0) =          -3
 Through built-in fcn 3

 i =           0
 i =         100
 i =         200
 i =         300
 i =         400
 i =         500
 i =         600
 i =         700
 i =         800
 i =         900
 i =        1000
 Through built-in fcn 4

 Through fdemo1

lnx1 7> exit
exit
```