Source file: `fdemo2.f`

```
c===========================================================
c     fdemo2:  Program which demonstrates basic usage
c     of character variables in Fortran 77.
c===========================================================
      program        fdemo2

      implicit       none

c-----------------------------------------------------------
c     See below for definition of integer function
c     'indlnb'.   Note that this and other useful routines
c     are available in the 'p410f' library.
c-----------------------------------------------------------
      integer        indlnb

c-----------------------------------------------------------
c     Define some character variables of various lengths
c
c     Note that
c
c        character*1     foo
c
c     and
c
c        character       foo
c
c     are synonymous, i.e. if an explicit length
c     specification is not given, the variable will
c     be a single character long.
c-----------------------------------------------------------
      character*1    c1
      character*2    c2
      character*4    c4
      character*26   lcalph
      character      cc1*1,    cc2*2,   cc4*4
      character*60   buffer

c-----------------------------------------------------------
c     Assignment of constant strings to char. variables.
c     If length of character expression being assigned
c     is less than length of character variable, variable
c     is 'right-padded' with blanks.
c-----------------------------------------------------------
      c1     = 'a'
      c2     = 'bc'
      c4     = 'defg'
      lcalph = 'abcdefghijklmnopqrstuvwxyz'

      write(*,*) 'c1 = ', c1
      write(*,*) 'c2 = ', c2
      write(*,*) 'c4 = ', c4
      write(*,*) 'lcalph = ', lcalph
      call prompt('Through constant assignment')

c-----------------------------------------------------------
c     // is the string concatentation operator
c-----------------------------------------------------------
      write(*,*) 'c1 // c2 // c4 = ', c1 // c2 // c4
      call prompt('Through concatenation')

c-----------------------------------------------------------
c     The integer intrinsic (built-in) function 'len'
c     returns the length of its string argument
c-----------------------------------------------------------
      write(*,*) 'len(c1) = ', len(c1)
      write(*,*) 'len(buffer) = ', len(buffer)
      call prompt('Through string length')

c-----------------------------------------------------------
c     Substring extraction
c-----------------------------------------------------------
      write(*,*) 'lcalph(1:13) = ',  lcalph(1:13)
      write(*,*) 'lcalph(18:18) = ', lcalph(18:18)
      call prompt('Through substring extraction')

c-----------------------------------------------------------
c     Substring assignment
c-----------------------------------------------------------
      c4(4:4) = 'Z'
      write(*,*) 'c4 = ', c4
      call prompt('Through substring assignment')

c-----------------------------------------------------------
c     Use of 'indlnb'
c-----------------------------------------------------------
      buffer = 'somefilename'
      write(*,*) '<' // buffer // '>'
      write(*,*) '<' // buffer(1:indlnb(buffer)) // '>'
      buffer = 'Some multi-word message'
      write(*,*) '<' // buffer // '>'
      write(*,*) '<' // buffer(1:indlnb(buffer)) // '>'
      buffer = ' '
      write(*,*) 'indlnb(buffer) = ', indlnb(buffer)
      call prompt('Through indlnb usage')

      call prompt('Through fdemo2')

      stop
      end

c-----------------------------------------------------------
c     Prints a message on stdout and then waits for input
c     from stdin.
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
         return

 900     continue
         stop
      end

c-----------------------------------------------------------
c     Returns index of last non-blank character in 's',
c     or 0 if the string is completely blank.
c-----------------------------------------------------------
      integer function indlnb(s)

         character*(*)    s

         do indlnb = len(s) , 1 , -1
            if( s(indlnb:indlnb) .ne. ' ' ) return
         end do
         indlnb = 0

         return

      end
```

Source file: `fdemo2-output`

```
############################################################
# Blank lines added for readability.
############################################################
lnx1 1> fdemo2
 c1 = a
 c2 = bc
 c4 = defg
 lcalph = abcdefghijklmnopqrstuvwxyz
 Through constant assignment
 Enter any non-blank character & enter to continue
a

 c1 // c2 // c4 = abcdefg
 Through concatenation
 Enter any non-blank character & enter to continue
a

 len(c1) =            1
```

```
 len(buffer) =           60
 Through string length
 Enter any non-blank character & enter to continue
a

 lcalph(1:13) = abcdefghijklm
 lcalph(18:18) = r
 Through substring extraction
 Enter any non-blank character & enter to continue
a

 c4 = defZ
 Through substring assignment
 Enter any non-blank character & enter to continue
a

 <somefilename                                                  >
 <somefilename>
 <Some multi-word message                                       >
 <Some multi-word message>
 indlnb(buffer) =            0
 Through indlnb usage
 Enter any non-blank character & enter to continue
a

 Through fdemo2
 Enter any non-blank character & enter to continue
a
```

Source file: `first100-generate`

```
############################################################
# 'iota' is an APL-inspired script I wrote to generate
# the integers from 1 to n, one per line.  It comes in
# useful in many instances.  In Linux, there is also
# a command 'seq', which can so the same thing.
############################################################
lnx1 1> iota
usage: iota <n> [<origin|1>]

lnx1 2> which iota
/usr/local/bin/iota

############################################################
# 'mw' is another script which attempts to locate
# the source for a script or other executable, and if
# successful, displays the source.
############################################################
lnx1 3> mw iota
</usr/local/bin/iota>
#!/bin/sh

Usage="usage: iota <n> [<origin|1>]"

case $# in
1) n=$1; origin=1;;
2) n=$1; origin=$2;;
*) echo "$Usage"; exit 1;;
esac

if printf "%d" $n > /dev/null 2>&1 && \
   printf "%d" $n > /dev/null 2>&1 $origin; then
   awk 'BEGIN{for(i=0; i<'$n'; i++) printf "%d\n", i+'$origin'}' < /dev/null
else
   echo "$Usage"; exit 1;
fi

############################################################
# Sample 'iota' invocation.
############################################################
lnx1 4> iota 10
1
2
3
4
5
6
7
8
9
10

############################################################
# Create 'first100' file.
############################################################
lnx1 5> iota 100 > first100

############################################################
# Display first 10 lines of 'first100' using Unix 'head'
# command.  Note use of '!$' (last argument to previous
# command).
###########################################################
lnx1 6> head -10 !$
head -10 first100
1
2
3
4
5
6
7
8
9
10

############################################################
# Display last 10 lines of 'first100' using Unix 'tail'
# command.
############################################################
lnx1 7> tail -10 !$
tail -10 first100
91
92
93
94
95
96
97
98
99
100
```

Source file: `mysum.f`

```
c===========================================================
c     mysum:  reads numbers one per line from stdin
c     and writes sum on stdout.  Ignores invalid inputs
c     but counts number encountered and reports on stderr.
c===========================================================
      program      mysum

      implicit     none

c-----------------------------------------------------------
c     vi:    Current number read from stdin
c     sum:   Current sum of numbers read
c     rc:    For storing return status from READ
c     nbad:  Count of number of bad inputs
c-----------------------------------------------------------
      real*8       vi,         sum
      integer      rc,         nbad

c-----------------------------------------------------------
c     Initialize ...
c-----------------------------------------------------------
      nbad = 0
      sum  = 0.0d0

c-----------------------------------------------------------
c     The following construct is roughly equivalent to
c     a while loop, execution keeps returning to the
c     top of the loop until end of file is detected on
c     stdin.
c-----------------------------------------------------------
 100  continue
         read(*,*,iostat=rc,end=200)  vi
         if( rc .eq. 0 ) then
c-----------------------------------------------------------
c           Read a bona fide real*8 value, update sum.
```

```
c------------------------------------------------------------
            sum = sum + vi
         else
c------------------------------------------------------------
c           Input was invalid.
c------------------------------------------------------------
            nbad = nbad + 1
         end if
      go to 100
 200  continue

c------------------------------------------------------------
c     Write sum on standard output.
c------------------------------------------------------------
      write(*,*) sum

c------------------------------------------------------------
c     Report # of invalid inputs only if there were some.
c------------------------------------------------------------
      if( nbad .gt. 0 ) then
c------------------------------------------------------------
c        Unit 0 is stderr (standard error) on most Unix
c        systems: if you redirect stdin using '>' and this
c        message is tripped, it will still appear on the
c        terminal.
c------------------------------------------------------------
         write(0,*) nbad, ' invalid inputs'
      end if

      stop

      end
```

Source file: `mysum-s.f`

```
c============================================================
c     Less-commented (i.e. more reasonable level of
c     comments) version of mysum.
c============================================================
c     mysum_s:  reads numbers one per line from stdin
c     and writes sum on stdout.  Ignores invalid inputs
c     but counts number encountered and reports on stderr.
c============================================================
      program      mysum

      implicit     none

      real*8       vi,         sum
      integer      rc,         nbad

      nbad = 0
      sum  = 0.0d0

 100  continue
         read(*,*,iostat=rc,end=200)  vi
         if( rc .eq. 0 ) then
            sum = sum + vi
         else
            nbad = nbad + 1
         end if
      go to 100
 200  continue

      write(*,*) sum

      if( nbad .gt. 0 ) then
         write(0,*) nbad, ' invalid inputs'
      end if

      stop

      end
```

Source file: `mysum-output`

```
lnx1 1> mysum
1
2
8
10
^D
    21.00000000000000
lnx1 2> mysum < first100
    5050.000000000000

lnx1 3> mysum
12
2
8
a
10
b
^D
    32.00000000000000
            2 invalid inputs

lnx1 4> mysum < first100 > mysum_result

lnx1 5> more !$
more mysum_result
    5050.000000000000
```

Source file: `dvfrom.f`

```
c============================================================
c     Returns a double precision vector (one-dimensional
c     array) read from file 'fname'.  If 'fname' is the
c     string '-', the vector is read from standard input.
c
c     The file should contain one number per line; invalid
c     input is ignored.
c
c     This routine illustrates  a general technique for
c     reading data from a FORMATTED (ASCII) file.  In
c     Fortran, one associates a "logical unit number"
c     (an integer) with a file via the OPEN statement.
c     The unit number can then be used as the first
c     "argument" of the READ and WRITE statements to
c     perform input and output on the file.
c
c     Fortran reserves the following unit numbers:
c
c     5      terminal input (stdin)
c     6      terminal output (stdout)
c     0      error output on Unix systems (stderr)
c============================================================

      subroutine dvfrom(fname,v,n,maxn)
c------------------------------------------------------------
c     Arguments:
c
c        fname:  (I)    File name
c        v:      (O)    Return vector
c        n:      (O)    Length of v (# read)
c        maxn:   (I)    Maximum number to read
c------------------------------------------------------------
         implicit         none
c------------------------------------------------------------
c        The integer functions 'indlnb' and 'getu' are
c        defined in the 'p410f' library.
c------------------------------------------------------------
         integer          indlnb,      getu
c------------------------------------------------------------
c        Declaration of routine arguments: note
c        "adjustable dimensioning" of v; any array which
c        is declared with adjustable dimensions must be
c        a subroutine argument; any adjustable dimensions
c        must also be subroutine arguments.
c------------------------------------------------------------
         character*(*)    fname
         integer          n,           maxn
         real*8           v(maxn)
```

```
c-----------------------------------------------------------
c       Programming style: Use parameter (ustdin) rather
c       than constant value (5) for stdin logical unit #
c-----------------------------------------------------------
        integer         ustdin
        parameter      ( ustdin = 5 )

c-----------------------------------------------------------
c       Local variables:
c
c       vn:     Current number read from input
c       ufrom:  Logical unit number for READ
c       rc:     For storing return status from READ
c-----------------------------------------------------------
        real*8          vn
        integer         ufrom,      rc

c-----------------------------------------------------------
c       Intialize
c-----------------------------------------------------------
        n = 0

c-----------------------------------------------------------
c       Read from stdin?
c-----------------------------------------------------------
        if( fname .eq. '-' ) then
c-----------------------------------------------------------
c          Set unit number to stdin default
c-----------------------------------------------------------
           ufrom = ustdin
        else
c-----------------------------------------------------------
c          Get an available unit number
c-----------------------------------------------------------
           ufrom = getu()
c-----------------------------------------------------------
c          Open the file for formatted I/O
c-----------------------------------------------------------
           open(ufrom,file=fname(1:indlnb(fname)),
     &          form='formatted',status='old',iostat=rc)
           if( rc .ne. 0 ) then
c-----------------------------------------------------------
c             Couldn't open the file, print error message
c             and return.
c-----------------------------------------------------------
              write(0,*) 'dvfrom: Error opening ',
     &                   fname(1:indlnb(fname))
              return
           end if
        end if

c-----------------------------------------------------------
c       Input numbers into vector (one per line) until
c       EOF or maximum allowable number read
c-----------------------------------------------------------
 100    continue
           read(ufrom,*,iostat=rc,end=200)  vn
           if( rc .eq. 0 ) then
              n = n + 1
              if( n .gt. maxn ) then
                 write(0,*)  'dvfrom: Read maximum of ',
     &                       maxn, ' from ',
     &                       fname(1:indlnb(fname))
                    n = maxn
                    go to 200
              end if
              v(n) = vn
           end if
        go to 100
 200    continue

c-----------------------------------------------------------
c       If we are reading from a file, close the file.
c       This releases the unit number for subsequent use.
c-----------------------------------------------------------
        if( ufrom .ne. ustdin ) then
           close(ufrom)
        end if

        return
      end
```

Source file: `tdvfrom.f`

```
c===========================================================
c     Test program for subroutine 'dvfrom'.
c
c     Program expects one argument which is the filename
c     to be passed to 'dvfrom'
c===========================================================
      program        tdvfrom

      implicit       none

c-----------------------------------------------------------
c     The integer function 'iargc' returns the number of
c     arguments supplied to the program.  It is
c     automatically available to all Fortran programs on
c     most Unix systems, as is 'getarg' (see below).
c-----------------------------------------------------------
      integer        iargc,     indlnb

      integer        maxn
      parameter     ( maxn = 100 000 )
      real*8         v(maxn)
      integer        n

      character*256  fname

c-----------------------------------------------------------
c     Unless exactly one argument is supplied, print usage
c     message and exit.
c-----------------------------------------------------------
      if( iargc() .ne. 1 ) then
         write(0,*) 'usage: tdvfrom <file name>'
         write(0,*)
         write(0,*) '      Use ''tdvfrom -'' to read ',
     &              'from standard input'
         stop
      end if

c-----------------------------------------------------------
c     The subroutine 'getarg' (Unix) takes 2 arguments.
c     The first is an integer input argument specifying
c     which argument is to be fetched, the second is
c     a character output argument which, on return,
c     contains the fetched argument.
c
c     Get the filename.
c-----------------------------------------------------------
      call getarg(1,fname)
c-----------------------------------------------------------
c     Call the routine ...
c-----------------------------------------------------------
      call dvfrom(fname,v,n,maxn)
c-----------------------------------------------------------
c     ... and report how many numbers were read.
c-----------------------------------------------------------
      write(0,*) 'tdvfrom: ', n, ' read from '//
     &          fname(1:indlnb(fname))

      stop
      end
```

Source file: `tdvfrom-output`

```
lnx1 1> tdvfrom
 usage: tdvfrom <file name>

       Use 'tdvfrom -' to read from standard input

lnx1 2> tdvfrom -
1
2
3
4
5
^D
 tdvfrom:           5 read from -
```

```
lnx1 3> tdvfrom first100
 tdvfrom:          100 read from first100
```

Source file: `dvto.f`

```
c===========================================================
c     Writes a double precision vector to file 'fname'.
c     If fname is the string '-' then the vector is written
c     to standard output.
c===========================================================

      subroutine dvto(fname,v,n)
c-----------------------------------------------------------
c     Arguments:
c
c        fname:  (I)    File name
c        v:      (I)    Vector to be written
c        n:      (I)    Length of vector
c-----------------------------------------------------------
         implicit         none

         integer          getu,       indlnb

         character*(*)    fname
         integer          n
         real*8           v(n)

         integer          ustdout
         parameter      ( ustdout = 6 )

         integer          i,      uto,       rc

         if( fname .eq. '-' ) then
            uto = ustdout
         else
            uto = getu()
            open(uto,file=fname(1:indlnb(fname)),
     &           form='formatted',iostat=rc)
            if( rc .ne. 0 ) then
               write(0,*) 'dvto: Error opening ',
     &                    fname(1:indlnb(fname))
               return
            end if
         end if

         do i = 1 , n
            write(uto,*) v(i)
         end do

         if( uto .ne. ustdout ) then
            close(uto)
         end if

         return

      end
```

Source file: `tdvto.f`

```
c===========================================================
c     Test program for subroutine 'dvto'.
c
c     Program expects two arguments, the name of a file
c     for output ('-' for stdout) and the length of the
c     test vector to be written.
c===========================================================
      program        tdvto

      implicit       none

c-----------------------------------------------------------
c     The integer function 'i4arg' is defined in the
c     'p410f' library.  It takes two arguments, the first
c     is an integer specifying which program argument is
c     to be parsed as an integer, and the second is a
c     default value which will be returned if the argument
c     was not supplied or could not be converted to an
c     integer.
c-----------------------------------------------------------
      integer        iargc,      i4arg

      integer        maxn
      parameter    ( maxn = 100 000 )
      real*8         v(maxn)
      integer        n

      integer        i
      character*256  fname

c-----------------------------------------------------------
c     Unless exactly two arguments are supplied, print usage
c     message and exit.
c
c     Note the use of the "logical-if" statement (no then)
c-----------------------------------------------------------
      if( iargc() .ne. 2 ) go to 900

      call getarg(1,fname)
      n = i4arg(2,-1)
      if( n .eq. -1 ) go to 900
c-----------------------------------------------------------
c     Limit the value of n
c-----------------------------------------------------------
      n = min(n,maxn)
c-----------------------------------------------------------
c     Define test vector
c-----------------------------------------------------------
      do i = 1 , n
         v(i) = i
      end do

c-----------------------------------------------------------
c     Call the routine ..
c-----------------------------------------------------------
      call dvto(fname,v,n)

c-----------------------------------------------------------
c     Normal exit
c-----------------------------------------------------------
      stop

c-----------------------------------------------------------
c     Usage exit
c-----------------------------------------------------------
 900  continue
         write(0,*) 'usage: tdvto <file name> <n>'
         write(0,*)
         write(0,*) '      Use ''tdvto -'' to write ',
     &              'to standard output'

      stop

      end
```

Source file: `tdvto-output`

```
lnx1 1> tdvto
 usage: tdvto <file name> <n>

       Use 'tdvto -' to write to standard output

lnx1 2> tdvto -
 usage: tdvto <file name> <n>

       Use 'tdvto -' to write to standard output

lnx1 3> tdvto - 10
    1.000000000000000
    2.000000000000000
    3.000000000000000
    4.000000000000000
    5.000000000000000
    6.000000000000000
    7.000000000000000
    8.000000000000000
    9.000000000000000
    10.00000000000000

lnx1 4> tdvto foo 5
```

```
lnx1 5> cat foo
    1.000000000000000
    2.000000000000000
    3.000000000000000
    4.000000000000000
    5.000000000000000
lnx1 6> tdvfrom foo
 tdvfrom:            5 read from foo

lnx1 7> tdvto - 100 | tdvfrom -
 tdvfrom:          100 read from -
```

Source file: `Makefile`

```
.IGNORE:

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

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

EXECUTABLES = fdemo2 mysum tdvfrom tdvto

all: $(EXECUTABLES)

fdemo2: fdemo2.o
   $(F77_LOAD) fdemo2.o -o fdemo2

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

tdvfrom: tdvfrom.o dvfrom.o
   $(F77_LOAD) tdvfrom.o dvfrom.o -lp410f -o tdvfrom

tdvto: tdvto.o dvto.o
   $(F77_LOAD) tdvto.o dvto.o -lp410f -o tdvto

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

Source file: `make-output`

```
############################################################
# Do the default make (all: $(EXECUTABLES))
############################################################
lnx1 1> make
pgf77 -g -c fdemo2.f
pgf77 -g -L/usr/local/PGI/lib fdemo2.o -o fdemo2
Linking:
pgf77 -g -c mysum.f
pgf77 -g -L/usr/local/PGI/lib mysum.o -o mysum
Linking:
pgf77 -g -c tdvfrom.f
pgf77 -g -c dvfrom.f
pgf77 -g -L/usr/local/PGI/lib tdvfrom.o dvfrom.o -lp410f -o tdvfrom
Linking:
pgf77 -g -c tdvto.f
pgf77 -g -c dvto.f
pgf77 -g -L/usr/local/PGI/lib tdvto.o dvto.o -lp410f -o tdvto

############################################################
# Here's an alias which lists all the executables in a
# directory using the fact that the -F flag to ls appends
# a '*' to the name of such files.  I've included it here
# just to keep you thinking about tailoring your Unix
# environment to suit your own needs. 'sed' is the stream-
# editor, which, like 'awk' and 'perl' can be used to
# manipulate and modify text.
############################################################
lnx1 2> alias lsx  '/bin/ls -F | fgrep \* | sed s/\*//g'

lnx1 3> lsx
fdemo2
mysum
tdvfrom
tdvto

############################################################
# Clean up ...
############################################################
lnx1 4> make clean
rm *.o
rm fdemo2 mysum tdvfrom tdvto

lnx1 5> ls
Makefile  dvto.f    first100   mysum.f    tdvto.f
dvfrom.f  fdemo2.f  mysum-s.f  tdvfrom.f
```