

```

=====
c   arraydemo.f:  Program which demonstrates manipulation
c   of 'run-time' dimensioned arrays in Fortran.
c
c   The program accepts two integer arguments which
c   specify the bounds for the two-dimensional arrays
c   which are to be defined and manipulated.
c
c   The basic guidelines are as follows:
c
c       (1) To deal with run-time defined dimensions,
c           perform all array manipulation (including
c           input and output) in SUBPROGRAMS rather
c           than the main program.
c
c       (2) Always pass ALL bounds of an array, along
c           with the array itself, to subprograms which
c           are to manipulate the array.
c
c       (3) Declare sufficient storage in the main routine
c           to deal with the largest array(s) you
c           anticipate dealing with, but make sure that
c           you always check that the size of the storage
c           is sufficient
c
c       (4) An address of a location in a ONE dimensional
c           array can be passed to a subprogram expecting
c           a multi-dimensional array.
=====

```

```

program          arraydemo

implicit        none

integer         iargc,          i4arg

```

```

c-----
c   Single-dimensional array which can be used to provide
c   storage for the multi-dimensional array manipulation.
c   ("Poor-man's memory allocation")
c-----

      integer          maxq
      parameter      ( maxq = 100 000 )
      real*8         q(maxq)

c-----
c   'Pointer' to next available location in 'q'
c-----

      integer          qnext

c-----
c   'Pointers' for three 2-D arrays ('a1', 'a2', and 'a3')
c-----

      integer          narray
      parameter      ( narray = 3 )
      integer          a1,      a2,      a3

c-----
c   Array bounds which are to be defined at run time
c-----

      integer          n1,      n2

c-----
c   Get the desired array bounds from the command-line
c   and check that there is sufficient 'main-storage'.
c-----

      if( iargc() .ne. 2 ) go to 900
      n1 = i4arg(1,-1)
      n2 = i4arg(2,-1)
      if( n1 .le. 0 .or. n2 .le. 0 ) go to 900
      if( narray * n1 * n2 .gt. maxq ) then
         write(0,*) 'arraydemo: Insufficient main storage'
         stop
      end if

```

```

c-----
c   Initialize the main storage pointer ...
c-----
c       qnext = 1
c-----
c   ... and set up the 'pointers' for the two arrays
c   with bounds (n1,n2).
c-----
c       a1 = qnext
c       qnext = qnext + n1 * n2
c       a2 = qnext
c       qnext = qnext + n1 * n2
c       a3 = qnext
c-----
c   Define and manipulate the 2-d arrays using various
c   subroutines.
c-----
c       call load2d( q(a1), n1, n2,  1.0d0 )
c       call load2d( q(a2), n1, n2, -1.0d0 )
c       call add2d(  q(a1), q(a2), q(a3), n1, n2 )
c-----
c   Dump the 3 arrays to standard error.
c-----
c       call dump2d( q(a1), n1, n2, 'a1' )
c       call dump2d( q(a2), n1, n2, 'a2' )
c       call dump2d( q(a3), n1, n2, 'a1 + a2' )
c
c       stop
c
900  continue
c       write(0,*) 'usage: arraydemo <n1> <n2>'
c       stop
c       end

```

```
c-----  
c   Loads a 2-D array with the values:  
c  
c   a(i,j) = sc * (100 * j + i)  
c-----  
      subroutine load2d(a,d1,d2,sc)  
        implicit      none  
  
        integer      d1,          d2  
        real*8       a(d1,d2)  
        real*8       sc  
  
        integer      i,          j  
  
        do j = 1 , d2  
          do i = 1 , d1  
            a(i,j) = sc * (100.0d0 * j + i)  
          end do  
        end do  
  
        return  
  
end
```

```

c-----
c   Adds 2-D arrays 'a1' and 'a2' element-wise and returns
c   result in 'a3'
c-----
subroutine add2d(a1,a2,a3,d1,d2)
  implicit      none

  integer      d1,      d2
  real*8       a1(d1,d2), a2(d1,d2), a3(d1,d2)

  integer      i,      j

  do j = 1 , d2
    do i = 1 , d1
      a3(i,j) = a1(i,j) + a2(i,j)
    end do
  end do

  return

end

```

```

c-----
c   Dumps 2-d array labelled with 'label' on stderr
c-----
      subroutine dump2d(a,d1,d2,label)
         implicit      none

         integer       d1,       d2
         real*8        a(d1,d2)
         character*(*) label
         integer       i,       j,       st

         if( d1 .gt. 0 .and.  d2 .gt. 0 ) then
            write(0,100) label
100      format( /' <<< ',A,' >>>' /)
            do j = 1 , d2
               st = 1
110      continue
               write(0,120) ( a(i,j) , i = st , min(st+7,d1))
120      format(' ',8F9.3)
               st = st + 8
               if( st .le. d1 ) go to 110
               if( j .lt. d2 ) write(0,*)
            end do
         end if

         return

      end
end

```

Script started on Mon Oct 1 16:36:24 2001

```
#####  
# Sample output from 'arraydemo'  
#####
```

```
lnx1 1> make arraydemo  
pgf77 -g -Msecond_underscore -c arraydemo.f  
pgf77 -g -Msecond_underscore -L/usr/local/PGI/lib arraydemo.o -lp410f -  
Linking:
```

```
lnx1 2> arraydemo  
usage: arraydemo <n1> <n2>  
FORTRAN STOP
```

```
lnx1 3> arraydemo 3 4
```

```
<<< a1 >>>
```

```
101.000 102.000 103.000
```

```
201.000 202.000 203.000
```

```
301.000 302.000 303.000
```

```
401.000 402.000 403.000
```

```
<<< a2 >>>
```

```
-101.000 -102.000 -103.000
```

```
-201.000 -202.000 -203.000
```

```
-301.000 -302.000 -303.000
```

-401.000 -402.000 -403.000

<<< a1 + a2 >>>

0.000 0.000 0.000

0.000 0.000 0.000

0.000 0.000 0.000

0.000 0.000 0.000

FORTRAN STOP



```
#####  
# Illustrates use of 'nth', a script/filter available on the  
# machines for selecting columns from standard input  
#####
```

Script started on Mon Oct 1 16:37:50 2001

lnx1 1> cat powers

|    |     |      |       |
|----|-----|------|-------|
| 1  | 1   | 1    | 1     |
| 2  | 4   | 8    | 16    |
| 3  | 9   | 27   | 81    |
| 4  | 16  | 64   | 256   |
| 5  | 25  | 125  | 625   |
| 6  | 36  | 216  | 1296  |
| 7  | 49  | 343  | 2401  |
| 8  | 64  | 512  | 4096  |
| 9  | 81  | 729  | 6561  |
| 10 | 100 | 1000 | 10000 |

lnx1 2> nth 1 2 < powers

|    |     |
|----|-----|
| 1  | 1   |
| 2  | 4   |
| 3  | 9   |
| 4  | 16  |
| 5  | 25  |
| 6  | 36  |
| 7  | 49  |
| 8  | 64  |
| 9  | 81  |
| 10 | 100 |

lnx1 3> nth 1 3 1 < powers

|   |   |   |
|---|---|---|
| 1 | 1 | 1 |
| 2 | 8 | 2 |

3 27 3  
4 64 4  
5 125 5  
6 216 6  
7 343 7  
8 512 8  
9 729 9  
10 1000 10