# Reading in a 2D array of numbers into Fortran arrays

September 26, 2014

Let's say you have a file `array.txt` that has the contents

```
1.0 1.1 0.0 0.0 0.0
1.2 1.3 1.4 0.0 0.0
0.0 1.5 1.6 1.7 0.0
0.0 0.0 1.8 1.9 1.0
0.0 0.0 0.0 1.1 1.2
```

Just to learn the peculiarities of the `read(*,*)` statement, let's try to read this 5x5 array of numbers into two, 3x3 Fortran arrays.

Method 1

```
  programReadPractice
  implicit none

  real, dimension(3,3) :: array1, array2

  open(12, file="array.txt")

  ! read in values
  read(12,*) array1
  array1 = transpose(array1)
  read(12,*) array2
  array2 = transpose(array2)

  call printMatrix(array1,3,3)
  print*,
  call printMatrix(array2,3,3)

  close(12)

end program ReadPractice

subroutine printMatrix(array, n, m)
```

```
  implicit none
  real, intent(in) :: array(n,m)
  integer, intent(in) :: n,m

  integer :: i

  do i = 1,n
     print*, array(i,:)
  end do
end subroutine printMatrix
```

This results in the output

```
 1.00000000       1.10000002       0.00000000
 0.00000000       0.00000000       1.20000005
 1.29999995       1.39999998       0.00000000

 0.00000000       1.50000000       1.60000002
 1.70000005       0.00000000       0.00000000
 0.00000000       1.79999995       1.89999998
```

Compare this with the file:

```
1.0 1.1 0.0 0.0 0.0
1.2 1.3 1.4 0.0 0.0
0.0 1.5 1.6 1.7 0.0
0.0 0.0 1.8 1.9 1.0
0.0 0.0 0.0 1.1 1.2
```

Some comments on this method:

- No loops necessary!
- Fortran automatically fills the entire arrays with a single `read` statement, but does so by columns. Once we transpose each array, you can see the ordering of the data in the array more closely matches the ordering in the file.
- The last number read into `array1` is the `0.0` to the right of the `1.4` on the second line of `array.txt`. Then, the first number read into `array2` is the `0.0` to the left of the `1.5` on the third line of `array.txt`. That is, the rest of the second line of the file was skipped after the program filled `array1`, and the program jumped to the next line of the file to begin filling `array2`. This is consistent with new `read` statement = new line.

Method 2:

```
program ReadPractice
  implicit none

  real, dimension(3,3) :: array1, array2
  integer :: i, j

  open(12, file="array.txt")

  !  read in values
  read(12,*) ((array1(i,j), j=1,3), i=1,3), &
       ((array2(i,j), j=1,3), i=1,3)

  call printMatrix(array1,3,3)
  print*,
  call printMatrix(array2,3,3)

  close(12)
end program ReadPractice

subroutine printMatrix(array, n, m)
  implicit none
  real, intent(in) :: array(n,m)
  integer, intent(in) :: n,m

  integer :: i

  do i = 1,n
     print*, array(i,:)
  end do
end subroutine printMatrix
```

This results in the output

```
1.00000000        1.10000002        0.00000000
0.00000000        0.00000000        1.20000005
1.29999995        1.39999998        0.00000000

0.00000000        0.00000000        1.50000000
1.60000002        1.70000005        0.00000000
0.00000000        0.00000000        1.79999995
```

Compare this with the file:

```
1.0 1.1 0.0 0.0 0.0
1.2 1.3 1.4 0.0 0.0
0.0 1.5 1.6 1.7 0.0
0.0 0.0 1.8 1.9 1.0
0.0 0.0 0.0 1.1 1.2
```

Some comments on this method:

- This method used a construct known as an "implied do loop", i.e. the `((array1(i,j),j=1,3),i=1,3)`. If you compare the printed `array1` with the file `array.txt` it should be clear what this loop is doing.
- Note that `array1` is filled with the same numbers as in method 1, but because we explicitly stated the order to fill it (fix a row, go across the columns - refer to the implied do loop) there is no need to transpose it.
- Note there is only one `read` statement. Again, `array1` is the same as before, with the last number read in being the `0.0` to the right of the `1.4` in the second line of `array.txt`. However, notice that the program DOES NOT jump to the next line of the file to begin filling `array2`. Instead, it continues where it left off in the file, and the first number read into `array2` is the last `0.0` on the second line of `array.txt`.