

Goals for This Lecture:

- Learn about multi-dimensional (rank > 1) arrays
- Learn about multi-dimensional array storage
- Learn about the RESHAPE function
- Learn about allocatable arrays & the ALLOCATE and DEALLOCATE statements
- See an example of allocatable arrays in use

Multi-dimensional Arrays

- Type and shape of a multi-dimensional (rank > 1) array can be declared in two forms:

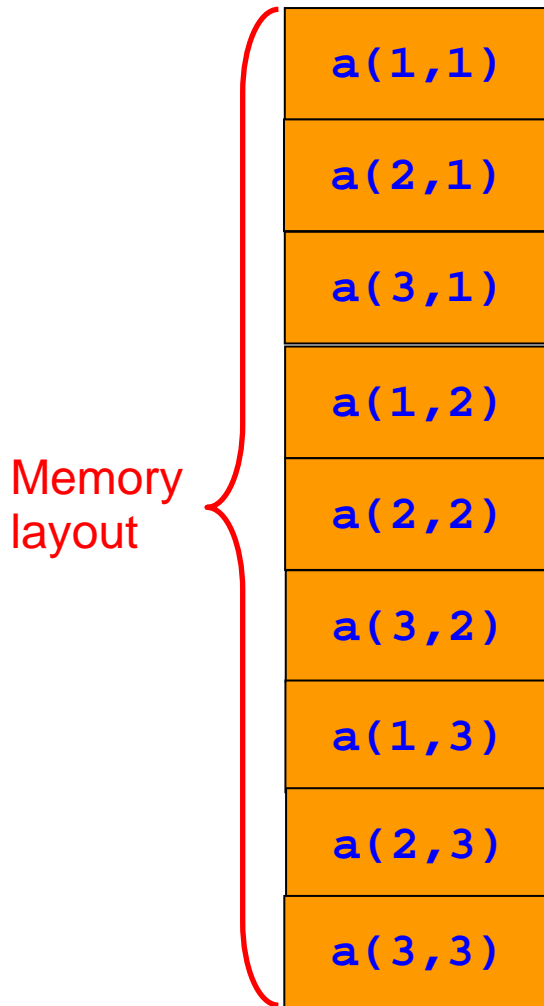
```
real :: data(9,1:2,0:257)
real, dimension(9,1:2,0:257) :: data
```

- Arrays can have up to seven dimensions
- You may not be able to fit a multidimensional array on a given machine!
 - Example:

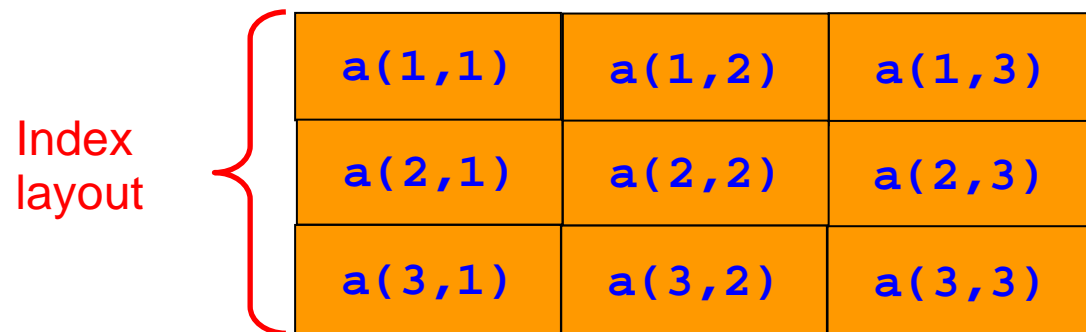
```
real :: data(1000,1000,1000,1000)
```

- This array requires at least 4×10^{12} bytes of storage.
- There are not many platforms with > 4 Terabytes of memory around!

Multi-dimensional Array Storage



- In FORTRAN, multi-dimensional arrays are ordered in column-major order
- Elements are stored in memory in a sequence where the left-most index varies most rapidly
- This is the reverse of other languages like C & C++ where elements are stored in row-major order
- The order in which array elements are accessed controls how fast the code executes



Initializing Multi-dimensional Arrays

- It would be nice if we could initialize multi-dimensional arrays in declaration statements with an array constructor such as:

```
real :: identity(3,3)=(/1.,0.,0.,0.,1.,0.,0.,0.,1./)
```

- However this is illegal as the array constructor on the right-hand-side of the assignment operator is not conformable with the rank-2 array on the left-hand-side
- This can be dealt with with the RESHAPE function:

```
real :: identity(3,3)=reshape(/1.,0.,0.,0.,1.,0.,0.,0.,1./,(/3,3/))
```

- The RESHAPE function converts the 9-element rank-1 array (the first argument) into a rank-2 array (specified by the array in the second argument) of the same size.

The RESHAPE function

- The reshape function accepts two arguments
- The first argument is the array to be reshaped
 - It can be of any type
 - Arrays can be variable or constants (built with an array constructor)
- The second argument is an integer rank-1 array that specifies the shape the resulting array will have
 - The number of elements in the integer array is the dimensionality of the reshaped matrix
 - The value of each element is the extent of the corresponding dimension

Allocatable Arrays

- Thus far we have been working with arrays that are statically allocated
- The size of the array is determined at the time of compilation
- Must know the needed size prior to compiling the program.
- A better solution is to dynamically allocate the array once the size is known (at the time of program execution)
- This can be done with allocatable, a.k.a. deferred shape arrays
- Example:

```
real, allocatable :: data(:)
integer :: size
size = 15
allocate(data(size))
deallocate(data)
```
- Allocatable arrays should always be deallocated after the program ceases to use them

ALLOCATED Statement

- Allocatable arrays can not be used in any way until memory for it has been allocated!
- You can test to see if memory for an array has been allocated by using the ALLOCATED statement
- The ALLOCATED statement returns a value of `.TRUE.` if memory has already been allocated and `.FALSE.` if it has not been allocated.
- Example:

```
real, allocatable :: data(:)
integer :: size
size = 15
if(.not.allocated(data)) allocate(data(size))
    data(1:size) = 1.0
deallocate(data)
```

```

! Purpose: Find max & min values of an arbitrary data set
! Author:  F. Douglas Swesty
! Date:    10/21/2005
program maxmin
implicit none      ! Turn off implicit typing
integer, parameter :: lun1=11  ! Define an LUN for the output file
integer :: npts=0          ! Number of data points
real, allocatable :: x(:)      ! Define array to hold data
integer :: i              ! Loop index
integer :: ierror=0        ! I/O error status variable
real :: max_val, min_val    ! Maximum & minimum values
integer :: max_loc, min_loc  ! Maximum & minimum location

                                ! Open the file
open(unit=lun1,file='experiment.dat',status='OLD',iostat=ierror)
if(ierror /= 0) then
  write(*,*) ' Could not read in file!'
  stop
endif

                                ! Find out how many points there are
do while(ierror == 0)          ! Do while no read error
  read(lun1,*) max_val        ! Read in data into a temp. variable
  npts = npts+1              ! Increment counter
enddo

allocate(x(npts))            ! Allocate space in array
rewind(lun1)                 ! Rewind the file
do i=1,npts,1               ! Rewind the data points
  read(lun1,*) x(i)
enddo
close(unit=lun1)            ! Close the file

```

```

max_val = x(1)              ! Initialize maximum
min_val = x(1)              ! Initialize minimum
max_loc = 1                 ! Initialize maximum location
min_loc = 1                 ! Initialize minimum location

do i=2,npts                 ! Loop over rest of data set

  if(x(i) > max_val) then   ! If x(i) is a new maximum
    max_val = x(i)         ! Reset maximum
    max_loc = i            ! Reset maximum location
  endif

  if(x(i) < min_val) then   ! If x(i) is a new minimum
    min_val = x(i)         ! Reset minimum
    min_loc = i            ! Reset minimum location
  endif

enddo

                                ! Write out results
write(*,*) ' Maximum ',max_val,' at ',max_loc
write(*,*) ' Minimum ',min_val,' at ',min_loc

deallocate(x)               ! deallocate space for array

stop                        ! Halt execution

end program maxmin

```

Reading Assignment

- Read Sections 7.1