

Goals for This Lecture:

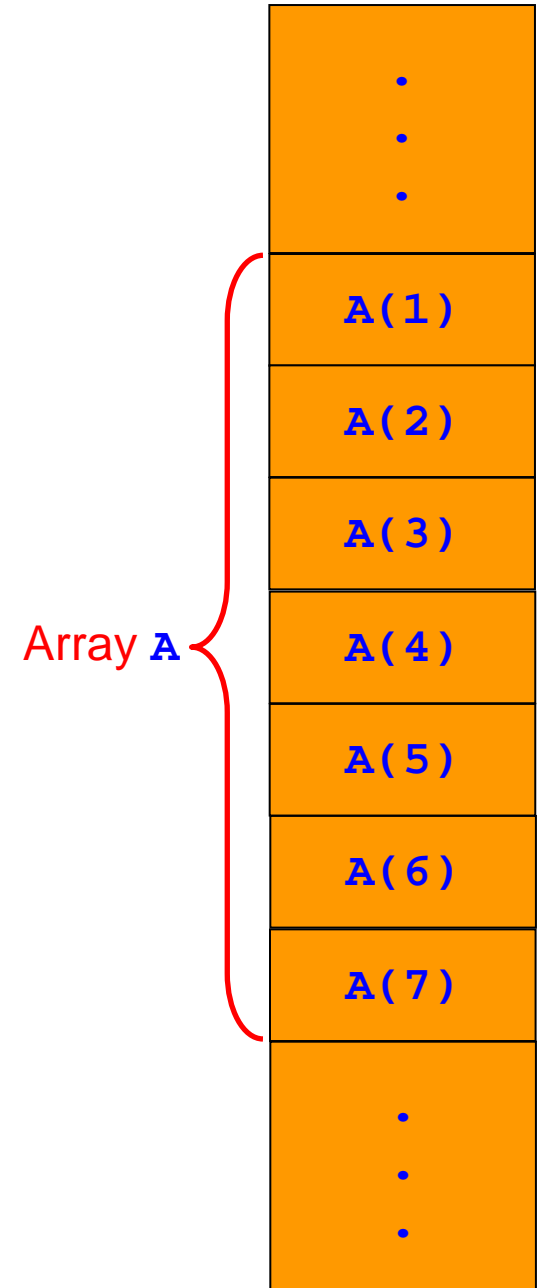
- Introduce Arrays

File Positioning

- Occasionally you may want to backup within a file and write or read a line (record) a second time.
- FORTRAN provides a way to manage positions within a file with the **backup** and **rewind** statements
- The **backup** statement moves backward one record (one line) in the file each time it is executed
 - FORM: **backup**(*lun*)
- The **rewind** statement moves back to the beginning of the file (first record a.k.a. first line) each time it is executed
 - FORM: **rewind**(*lun*)

Arrays

- An array is a group of variables or constants all of the same type referred to by a single name
- Values of the group are stored in consecutive positions in memory
- An individual value within the array is called an element of the array
- Values are identified by an integer valued index or subscript
- Arrays are enumerated lists of values



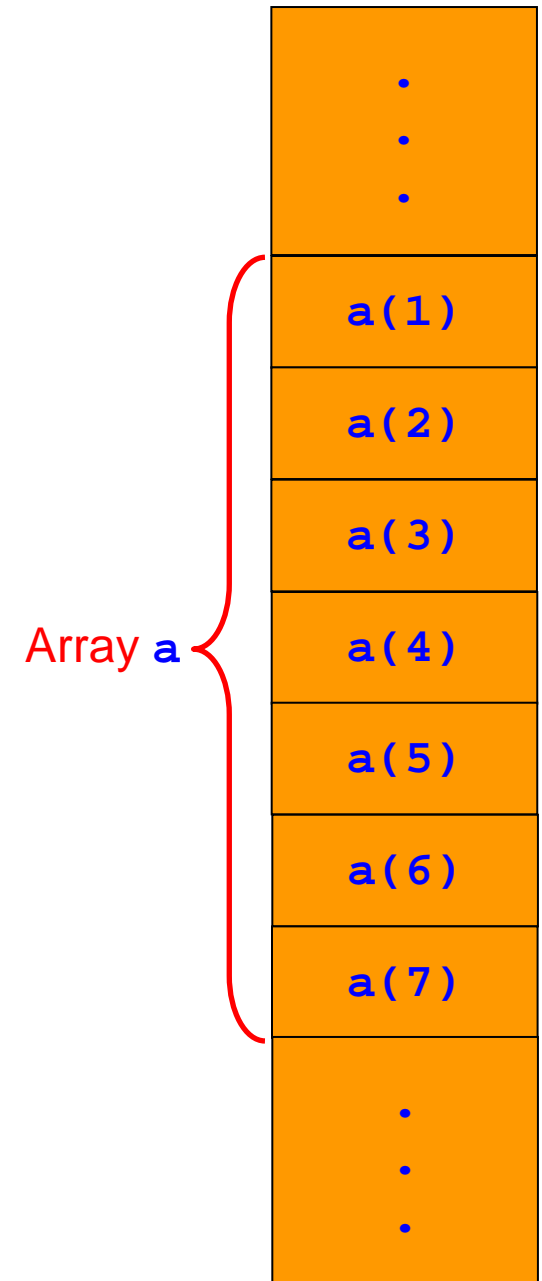
Array Elements

- An array element is referenced by an *integer* index or subscript
- This permits operations on the entire array to be carried out with DO loops
- Example:

```
do i=1,7,1
  a(i) = sin(a(i))
enddo
```

- Much simpler than:

```
a1 = sin(a1)
a2 = sin(a2)
a3 = sin(a3)
a4 = sin(a4)
a5 = sin(a5)
a6 = sin(a6)
a7 = sin(a7)
```



Declaring Arrays

– Arrays can be declared in two ways:

- Method 1 (specifying the extent in the declaration):

```
real :: a(7)
```

- Method 2 (using the dimension attribute):

```
real, dimension(7) :: a
```

– Arrays can be constructed with multiple subscripts so they can be organized into more than one dimension

```
real :: potential(5,7)
```

- The number of subscripts is called the **rank** of the array
- The number of elements in each dimension is called the **extent** of the array in that dimension
- The **shape** of the array is defined as the combination of the array **rank** and its **extent** in each dimension
- The **size** of the array is the total number of elements in the array

Array Constants

– Array Constants can be defined by placing constant values between two special delimiters: (/ and /)

– This is called an **array constructor**

– Example:

```
( / 1.1,2.3,1.5,3.2,-5.1/ )
```

– This is an array constant of rank-1 and extent 5 in dimension 1

– This can be used to initialize an array

– Example:

```
real :: a(5)=(/1.1,2.3,1.5,3.2,-5.1/)
```

Implied DO loop array constructors

– FORM:

```
(/ (arg1,arg2,..., index=start,end,stride) /)
```

- *arg1, arg2, ...* are expressions that are evaluated in order on each pass through the loop.
- *index=start,end,stride* have their usual iterative DO loop meanings
- This form can be used to initialize arrays in type declaration statements

```
real :: a(10)=(/ (2.0*i,0.5*i,i=1,10,2) /)
```

Initializing Arrays

- Like any other variables, arrays must be initialized before they can be used in expressions
- Arrays can be initialized in several ways:

- Assignment statements:

```
do i=1,5,1
  a(i) = 0.0
enddo
```

- Declaration statements with an array constructor:

```
real :: a(5)=(/1.1,2.3,1.5,3.2,-5.1/)
```

- Declaration statements with an implied do loop:

```
real :: a(5)=(/ (1.5*i,i=1,5,1) /)
```

- Read statements:

```
do i=1,5,1
  read(*,*) a(i)
enddo
```

Arrays Subscript Ranges

- An 7-element array declared by

```
integer :: A(7)
```

would normally have it's elements addressed by

```
A(1),A(2),...A(7)
```

- However, it is possible to specify that we would like the subscripts to run over another range such as

```
integer :: A(0:6)
```

or

```
integer :: A(-3:3)
```

- Both examples have an extent of 7 and a size of 7

$extent = upper_bound - lower_bound + 1$

- The general form of declaration is:

```
type, dimension(lower_bound:upper_bound) :: array
```

or

```
type :: array(lower_bound:upper_bound)
```

Example: Reading data into arrays

```
! Purpose: Show how to read data into an array
! Author:  F. Douglas Swesty
! Date:    10/17/2005
program array_read
implicit none      ! Turn off implicit typing
integer, parameter :: lun1=11 ! Define an LUN for the output file
real :: x(10), y(10) ! Define arrays to hold data
integer :: i       ! Loop index
integer :: ierror ! I/O error status variable
open(unit=lun1,file='experiment.dat',status='OLD',iostat=ierror) ! Open the file
if(ierror /= 0) then
  write(*,*) ' Could not read in file!'
  stop
endif
do i=1,10
  read(lun1,*) x(i),y(i) ! Read in (x,y) pairs of data into arrays
enddo

close(unit=lun1)           ! Close the file
stop                       ! Halt execution
end program array_read
```

Using Integer Constants to Declare Arrays

- A good programming practice is to declare arrays using integer constants or parameters which can then appear in the array declaration.

- Example:

```
integer, parameter :: nmax=7  
integer :: A(nmax)
```

- The integer parameter using in an array declaration must be declared in a line appearing prior to the array declaration statement.

- Integer expressions utilizing parameters are permitted in declaration statements.

- Example:

```
integer, paramater :: nmax=5  
integer :: B(1:2*nmax)
```

- This allows for easy program maintenance as DO loops can also utilize the parameters

Example: Reading data into arrays

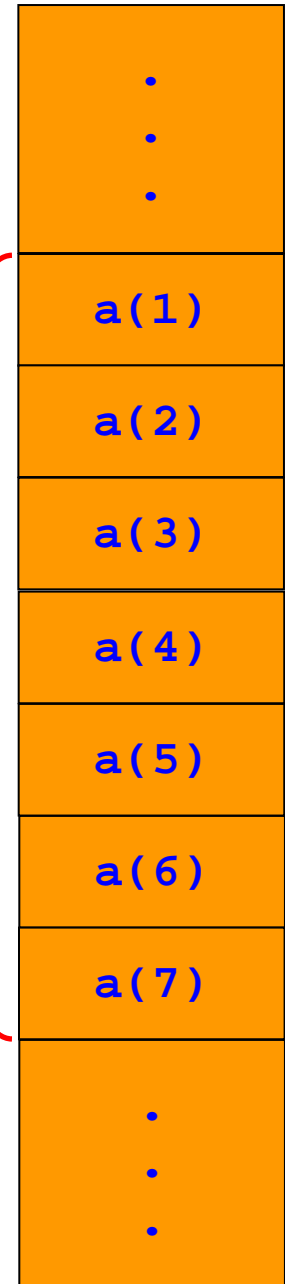
```
! Purpose: Show how to read data into an array with an integer parameter defining the size
! Author:  F. Douglas Swesty
! Date:    10/17/2005
program array_read2
implicit none      ! Turn off implicit typing
integer, parameter :: lun1=11  ! Define an LUN for the output file
integer, parameter :: npts=10  ! Number of data points
real :: x(npts), y(npts)      ! Define arrays to hold data
integer :: i                ! Loop index
integer :: ierror           ! I/O error status variable
open(unit=lun1,file='experiment.dat',status='OLD',iostat=ierror) ! Open the file
if(ierror /= 0) then
  write(*,*) ' Could not read in file!'
  stop
endif
do i=1,npts
  read(lun1,*) x(i),y(i) ! Read in (x,y) pairs of data into arrays
enddo

close(unit=lun1)           ! Close the file
stop                       ! Halt execution
end program array_read2
```

Out-of-bounds Array Subscripts

- What happens if we reference an array element that is below the lower bound of the array or above the upper bound of the array?
- Example: `a(9)`
- These array elements do not exist
- We refer to this as an out-of-bounds array reference and it is an illegal operation
- Most compilers can be told to build in checking for such operations during execution
 - Intel Compiler: add the `-CB` flag to the compilation command
 - Example: `ifort -CB -o array_read2 array_read2.f90`
- This should always be done while debugging code.
- Bounds checking can slow the execution of the code significantly

Array a



Reading Assignment

- Read Sections 6.4-6.7,8.1-8.7