University Djilali Bounaama, Khemis Miliana, DBKM
Faculty of Material Sciences and Computer Science

COURSE NOTES

ARRAYS

By: Seddik KHERROUBI

Definition:

An array is a collection of elements of the same type, accessed using integer indices. The elements of an array are arranged along one or more axes, which are referred to as the dimensions of the array. In one-dimensional arrays (which can represent mathematical vectors), each element is identified by a single integer. However, Fortran allows arrays with up to 7 dimensions.

A one-dimensional array is sometimes referred to as a vector. It can be represented in the following form:

L(1) L(2) L(3) L(4) ……………………………………. L(n)

- Array dimension: 1
- Array size: n
- The L(i), i=1, 2, …, n (must be of the same type)

Array declaration:

As with simple variables, the type of the array must be specified. All elements of the array are of the same type. If the array name appears in a type declaration (REAL, INTEGER, etc.), the type is resolved. Otherwise, if the array is declared using the DIMENSION specifier, the first letter of the array name can be used to determine its type. If this letter is I, J, K, L, M, or N, the array is of integer type. Otherwise, it is of real type.

Examples:
REAL, DIMENSION(:) :: A(10), B(15, 5), C(3, 7, 9)

These declarations indicate:

- A is a one-dimensional real array with 10 elements (vector).
- B is a two-dimensional real array with 15 rows and 5 columns (matrix).
- C is a three-dimensional real array with dimensions 3, 7, and 9.

These examples demonstrate how to declare arrays in Fortran by specifying their type and dimensions.

Terminology of Arrays:

- Rank of an array: the number of dimensions it has.
- Extent of an array along one of its dimensions: the number of elements in that dimension.
- Bounds of an array along one of its dimensions: the lower and upper limits of the indices in that dimension. The default lower bound is 1.
- Shape of an array: a vector whose components are the extents of the array along its dimensions; its size is equal to the rank of the array.
- Size of an array: the total number of elements in the array, which is the product of the elements of the vector that represents its shape.
  (Two arrays are said to be conformant if they have the same shape.)

Declaration of an Array:

- The declaration of an array is done using the DIMENSION attribute, which specifies the shape of the array, and optionally the bounds separated by the ":" symbol.
- Examples:
  REAL, DIMENSION X(15)
  REAL, DIMENSION Y(1:5, 1:3)
  REAL, DIMENSION Z(-1:3, 0:2)

The array X has a rank of 1, Y and Z have a rank of 2.
The extent of X is 15, while Y and Z have extents of 5 and 3.
The shape of X is the vector (/15/), and the shape of Y and Z is the vector (/5, 3/). The size of arrays X, Y, and Z is 15.
The arrays Y and Z are conformant, meaning they have the same shape.

Array Construction and Display:

To construct a one-dimensional array, you can list its elements and enclose them in (/ ... /). Here are a few examples:

- (/ 1, 2, 3 /) produces the array [1, 2, 3].
- (/ (i, i=1, 100) /) and (/ (i, i=1, 100, 2) /) produce, respectively, the arrays [1, 2, ..., 100] (the array of odd numbers from 1 to 100).

You can combine this approach, as shown in the example (/ (0, (j, j=1, 5), 0, i=1, 6) /), which generates an array of size 42, consisting of 6 repetitions of the sequence 0, 1, 2, 3, 4, 5, 0.

The reshape command (X, (/ m, n /)) allows you to create a rectangular array of size m × n from a one-dimensional array of size mn. The elements are filled successively, row by row. For example, reshape((/ ((i + j, i=1, 100), j=1, 100) /), (/ 100, 100 /)) constructs the addition table of the first 100 integers.

```
Program
Implicit none
Real, dimension(10) :: x
Integer :: i
Do i = 1, 10
x(i) = i**2      Dnd do
Do i = 1, 10
```

```
        Write(*,*) "x(", i, ") = ", x(i)
        End do
        End
```

Function Reshape:

The RESHAPE function in Fortran allows you to modify the shape of a multidimensional array. This function takes two arguments: the original array and the desired new shape. The original array is reorganized to match the new shape by filling the elements in the order of memory storage.
The filling of the array when reshaping depends on the storage order of the elements in memory. In Fortran, the default storage order for multidimensional arrays is column-major convention, also known as Fortran order. This means that the elements are stored column-wise, so the elements of the first column are stored contiguously, followed by the elements of the second column, and so on.
When the RESHAPE function is called with a new number of dimensions or a new shape for the array, the compiler calculates the new size and shape of the array. Then it fills the elements of the array in storage order, taking the elements from the original array in the corresponding order.

For example if the original array has the shape (3, 4) and follows the Fortran storage order, where elements are stored column-wise, the array can be represented as:
A(1,1) A(2,1) A(3,1) A(1,2) A(2,2) A(3,2) A(1,3) A(2,3) A(3,3) A(1,4) A(2,4) A(3,4)
If the RESHAPE function is called with the new shape (6, 2), the compiler will fill the new array by taking the elements from the old array in the following order:
A(1,1) A(2,1) A(3,1) A(4,1) A(5,1) A(6,1) A(1,2) A(2,2) A(3,2) A(4,3) A(5,2) A(6,2)
Note: If the specified new shape for the array does not match the total size of the original array, a compilation or runtime error may occur.
Example:
a = RESHAPE([2.0, 3.0, 1.0, 0.0, -1.0, 4.0, -2.0, 5.0, 2.0], [3, 3])
The statement a = RESHAPE([2.0, 3.0, 1.0, 0.0, -1.0, 4.0, -2.0, 5.0, 2.0], [3, 3]) in Fortran creates an array a with dimensions 3x3 and stores the elements accordingly.

$$\begin{vmatrix} 2 & 0 & 2 \\ 3 & 1 & 5 \\ 1 & 4 & 2 \end{vmatrix}$$

In this case, the elements [2.0, 3.0, 1.0] correspond to the first column of array a, the elements [0.0, -1.0, 4.0] correspond to the second column, and the elements [-2.0, 5.0, 2.0] correspond to the third column. Therefore, the new array a will have the following shape:

Dynamic Allocation:
Definition:

Dynamic allocation (or dynamic memory allocation) is a computer process that allows reserving memory space in the heap of a running program. This technique enables the creation of variables, arrays, and data structures with unknown or variable sizes, unlike static allocation, which requires knowing the size in advance.
In programming languages, an "allocatable" is a type of variable that can be declared to be dynamically allocated. The dynamic allocation operation (allocate) reserves memory space for this variable, while the deallocate operation frees up this space and makes it available for other uses.

```
program allocation_tab
implicit none
integer, allocatable :: tab(:)
integer :: n, i
write(*,*) "Entrez la taille du tableau :"
read(*,*) n
allocate (tab(n))
do i = 1, n
tab(i) = i
end do
write(*,*) "Contenu du tableau :"
write(*,*) tab
deallocate(tab)
End
```

In the example you provided, it declares an integer array tab using the syntax integer, allocatable :: tab(:). Then, it prompts the user to enter the size of the array using read(*,*) n. Next, it uses the dynamic allocation operation allocate(tab(n)) to allocate memory for the array tab with size n. After that, it initializes the array with increasing values from 1 to n, and finally, it displays the content of the array using write(*,*) tab. At the end, the allocated memory for the array is deallocated using deallocate(tab).

Notes:
Real, dimension (100)
It is important to note that the size of the memory allocated for the array or vector V depends on the size of the elements in the array. If you declare V as an array of integers, each element will typically take up 4 bytes of memory (32 bits) on most architectures, so the total size of the memory allocated for V would be 400 bytes. If you declare V as an array of reals (or floats), each element will typically take up 8 bytes of memory (64 bits) on most architectures, so the total size of the memory allocated for V would be 800 bytes.

Examples:

tab(100,100) For a 32-bit operating system, the allocation would be (100x100x4 = 40000 bytes). For a 64-bit operating system, the allocation would be (100x100x8 = 80000 bytes).