7. Arrays

One-dimensional Arrays

- The dimension of an array may be specified by a type specification statement of the form:

```
REAL, DIMENSION(10) :: A, B
INTEGER, DIMENSION(0:9) :: C
```

  Here, the three arrays A, B, and C have each been dimensioned with 10 storage slots. The index of the real arrays A and B start at 1 while the index for the integer array C starts at 0.

- The value of the individual array elements of the array A may be initialized to the values 1, 2, 3, ..., 10 by either of the two methods:

```
A = (/ 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 /)
```

  or,

```
A = (/ (I, I = 1, 10) /)
```

- The assignment of the values of one array to another is allowed provided that both arrays in question have the same physical dimension. For example,

```
B = A
```

  assigns the previously determined values of the elements of the A array to the array B.

- Operators and functions normally applied to simple expressions may also be applied to arrays having the same number of elements. Such operations are carried out on an element by element basis. For example,

```
A = A + B
C = 2*C
```

  assigns the ith element of A the value of the sum of the ith elements of arrays A and B. Similarly, the ith element of C is assigned the value equal to the ith element of itself multiplied by 2.

- A WHERE construct may be used to assign values to the individual elements of an array with

```
WHERE (logical argument)
   sequence of array assignments
ELSEWHERE
   sequence of array assignments
END WHERE
```

  For example, if A is assigned the values

```
A = (/ (I, I = 1,10) /)
```

  then, we may consider assigning the elements of the array B as

```
WHERE (A > 5)
   B = 1.
ELSEWHERE
```

```
   B = 0.
END WHERE
```

This assigns to B the values 0, 0, 0, 0, 0, 1, 1, 1, 1, 1.

- Several intrinsic array-type functions are available for processing arrays. Some of these include

```
DOT_PRODUCT(A, B):   returns the dot product of A and B
MAXVAL(A):           returns the maximum value in array A
MAXLOC(A):           returns a one-element 1D array whose value is
                     the location of the first occurrence of the
                     maximum value in A
PRODUCT(A):          returns the product of the elements of A
SUM(A):              returns the sum of the elements of A
```

- An array may be *allocatable*, i.e., it may be assigned memory storage during execution. To declare a real allocatable array A, do

```
REAL, DIMENSION(:), ALLOCATABLE :: A
```

At run time, the actual bounds for the array A may be determined by the statement

```
ALLOCATE ( A(N) )
```

where N is an integer variable that has been previously assigned. To ensure that enough memory is available to allocate space for your array, make use of the STAT option of the ALLOCATE command:

```
ALLOCATE ( A(N), STAT = AllocateStatus)
IF (AllocateStatus /= 0) STOP "*** Not enough memory ***"
```

Here, AllocateStatus is an integer variable. AllocateStatus takes the value 0 if allocation is successful or some other machine dependent value of there is insufficient memory.

An array can be released from memory by using the DEALLOCATE command

```
DEALLOCATE (A, STAT = DeAllocateStatus)
```

Again, DeAllocateStatus is an integer variable whose value is 0 if the deallocation was successful.

Multi-dimensional arrays

- Multi-dimensional arrays can be dimensioned with statements like

```
REAL, DIMENSION(2,3) :: A
REAL, DIMENSION(0:1,0:2) :: B
INTEGER, DIMENSION(10,20,3) :: I
```

The maximum limit on the rank (the number of dimensions) of an array is 7.

- The values of the elements of a multi-dimensional array may be assigned in a manner similar to that for the one-dimensional variety. For instance, the values 1, 2, 3, 4, 5, 6 may be assigned to the two-dimensional array A by

```
A = (/ 1, 2, 3, 4, 5, 6 /)
```

This assigns the values of the A array in column order similar to the rules of Fortran 77.

- The assignment of the values of one array to another is allowed provided that both arrays in question have the same physical dimension. For example,

```
B = A
```

  assigns the previously determined values of the elements of the A array to the array B.

- Just like with one-dimensional arrays, operators and functions normally applied to simple expressions may also be applied to multi-dimensional arrays having the same number of elements. Such operations are carried out on an element by element basis.
- A WHERE construct may be used to assign values to the individual elements of an array with

```
WHERE (logical argument)
   sequence of array assignments
ELSEWHERE
   sequence of array assignments
END WHERE
```

- Several intrinsic array-type functions are available for processing multi-dimensional arrays. Some of the more useful ones are

```
MAXVAL(A, D):        returns an array of one less dimension than A
                     containing the maximum values of A along
                     dimension D (if D is omitted, returns the
                     maximum value in the entire array)
MAXLOC(A):           returns a one-element 1D array whose value is
                     the location of the first occurrence of the
                     maximum value in A
SUM(A, D):           returns an array of one less dimension than A
                     containing the sums of the elements of A along
                     dimension D (if D is omitted, returns sum of the
                     elements in the entire array)
MATMUL(A, B):        returns the matrix product of A and B
TRANSPOSE(A):        returns the transpose of the 2D array A
```

- An array may be *allocatable*, i.e., it may be assigned memory storage during execution. To declare a real allocatable array A, do

```
REAL, DIMENSION(:, :), ALLOCATABLE :: A
```

  At run time, the actual bounds for the array A may be determined by the statements

```
ALLOCATE ( A(N, N), STAT = AllocateStatus)
IF (AllocateStatus /= 0) STOP "*** Not enough memory ***"
```

  Here, N and AllocateStatus are integer variables. AllocateStatus takes the value 0 if allocation is successful or some other machine dependent value of there is insufficient memory.

  An array can be released from memory by using the DEALLOCATE command

```
DEALLOCATE (A, STAT = DeAllocateStatus)
```

  Again, DeAllocateStatus is an integer variable whose value is 0 if the deallocation was successful.