Allocatable Arrays

Fortran 90 allows arrays to be created on-the-fly; these are known as **deferred-shape** arrays.

* Declaration: (note allocatable attribute, fixed rank)

```
integer, dimension(:), allocatable :: ages
real, dimension(:,:), allocatable :: speed
```

* Allocation:

```
read*, isize
allocate(ages(isize), stat=ierr)
if (ierr /= 0) print*, “ages: allocation failed”

allocate(speed(0:isize-1,10), stat=ierr)
if (ierr /= 0) print*, “speed: allocation failed”
```

Deallocating Arrays

Heap storage can be reclaimed using the **DEALLOCATE** statement:

```
if (allocated(ages)) deallocate(ages, stat=ierr)
```

- You'll get an error if you try to deallocate an array without the allocate attribute or an array that has not previously been allocated space.
- If a procedure containing an allocatable array which does not have the save attribute is exited without being deallocated, then this storage becomes inaccessible.

The WHERE statement and construct

Used to assign values to only those elements of an array where is logical condition is true.

* Single statement form:

```
where (a < 0) b = 0          ! a and b must be arrays of the same shape
```

* Block form:

```
where (c /= 0)       ! c /= 0 is a logical
   a = b / c         ! a and b must conform to c
elsewhere
   a = 0             ! the elements of a are set to 0 where they have not
                     ! been set to b/c.
   c = 1             ! the 0 elements of c are set to 1
end where
```

- All statements within a **WHERE** construct must be array assignments.
- The assignments are executed in the order they are written: first those in the **WHERE** block, then those in the **ELSEWHERE** block.
- **WHERE** constructs may not be nested.

Element Renumbering in Expressions

The elements in an expression no longer have the same subscripts as the elements in the arrays that make up the expression. They are renumbered with 1 as the lower bound in each dimension.

```
y(0:7) + z(-7:0)          ! result is an array with subscripts 1, 2, 3, ..., 8


integer, dimension(0:6), parameter :: v = (/ 3, 7, 0, -2, 2, 6, -1 /)
maxloc(v)             ! result is 2
maxloc(v(2:6))        ! result is 4 because the largest entry (6) is in the
                      ! 4th position of the section v(2:6)
```