Notes on `Fortran 77` Arrays

- *Recall array declarations:*

```
real*8        a1d(100)
real*8        a2d(100,200)
real*8        a3d(100,200,300)
```

(Standard `Fortran 77` allows up to 7 dimensions, or *rank-7* arrays)

- `Fortran 77` *array storage*
  - `Fortran 77` arrays are *always* stored in *contiguous* memory locations.
  - For 1-d (rank-1) arrays, memory layout is obvious:

```
real*8        v(5)
```

v(1)	v(2)	v(3)	v(4)	v(5)

  - For multidimensional arrays, storage is "linearized" ("one-dimensionalized") using "column-major" order–1st subscript varies most rapidly, then 2nd, then 3rd, etc.
  - 2-d (rank-2) example:

```
real*8        a(3,2)
```

a(1,1)	a(2,1)	a(3,1)	a(1,2)	a(2,2)	a(3,2)

– 3-d (rank-2) example:

```
real*8        b(2,2,2)
```

b(1,1,1)	b(2,1,1)	b(1,2,1)	b(2,2,1)	b(1,1,2)	b(2,1,2)	b(1,2,2)	b(2,2,2)

- It is relatively easy to write `Fortran 77` programs which can handle "run-time dimensioned" arrays *provided* all array manipulation if performed by `subroutines` or `functions`.

- *Computing "effective 1-d index" of multidimensional array element:*
  - 1-d (rank-1)

    ```
    real*8    a1d(d1)

    a1d(i)      --->   v1d( i )
    ```

  - 2-d (rank-2)

    ```
    real*8    a2d(d1,d2)

    a2d(i,j)    --->   v1d( (j-1)*d1 + i )
    ```

  - 3-d (rank-3)

    ```
    real*8    a3d(d1,d2,d3)

    a3d(i,j,k) --->   v1d( (k-1)*d1*d2 + (j-1)*d1 + i )
    ```

    This "linearization" (index, or offset, computation) is essentially how `Fortran 77` handles *all* array expressions.

- *Consequences of* `Fortran 77` *index computation*
  - Index computation makes it apparent why array bounds *must* be passed to a `subroutine` along with the array.
  - From the point of view of *storage* (memory layout), arrays of *any* dimension are *indistinguishable*, provided that they have the same total number of elements. Example:

```
real*8      c1d(64)
real*8      c2d(8,8)
real*8      c3d(4,4,4)
```