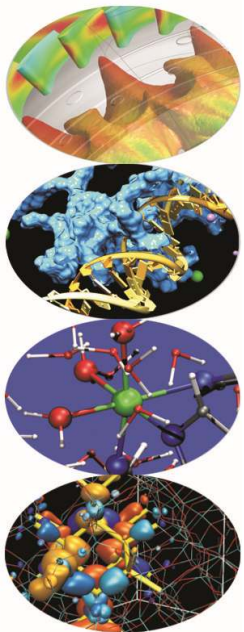


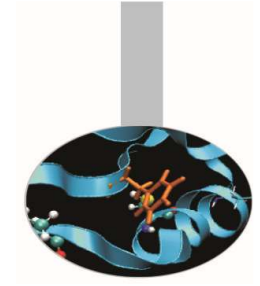
# Vettori e Matrici (array)

*Introduction to modern Fortran*

Paolo Ramieri, *CINECA*

*Maggio 2016*





# Array - Introduzione

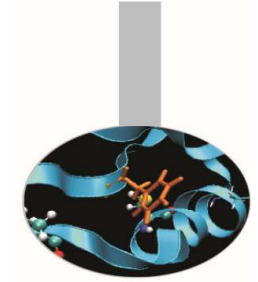
Gli array sono serie di variabili dello **stesso tipo**, ciascuna accessibile mediante indici numerici.

Il Fortran consente di gestire:

- Vettori
- Matrici
- Matrici multidimensionali (fino a 7 dimensioni)

Gli array in Fortran 90 possono essere di 3 tipi:

- Statici
- Semi-dinamici
- Dinamici

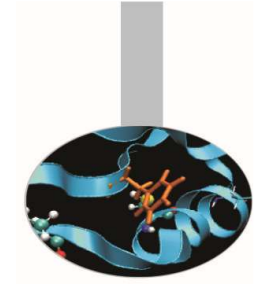


# Array - Introduzione

**Array statici:** hanno dimensioni fissate al momento della dichiarazione e il numero di elementi nell'array non può essere modificato durante l'esecuzione del programma.

**Array semi-dinamici:** all'interno di una procedura non hanno dimensioni fissate, ma esse vengono definite con il passaggio degli argomenti alla procedura. Sono anche detti **array automatici**.

**Array dinamici:** la dimensione di questi array può essere modificata durante l'esecuzione del programma. Sono anche detti array **allocabili**.



# Array - Terminologia

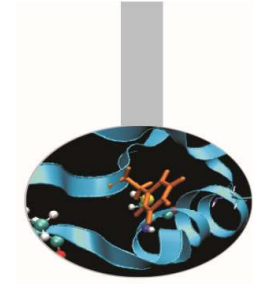
<b>Rango</b>	numero di dimensioni della matrice
<b>Estensione</b>	numero di elementi in una dimensione
<b>Forma</b>	vettore delle estensioni
<b>Dimensione</b>	prodotto delle estensioni
<b>Conformità</b>	due matrici con la stessa forma

Per dichiarare un array è necessario fornire 3 informazioni:

*tipo* degli elementi

*Rango*: numero di dimensioni

*Forma*: numero di elementi per ogni dimensione



# Array - Dichiarazione

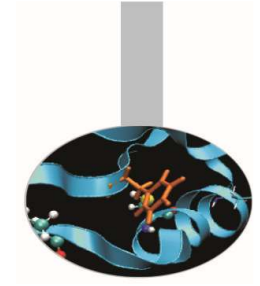
La dichiarazione delle dimensioni è lasciata all'attributo di dichiarazione DIMENSION secondo la sintassi generale:

```
TIPO, DIMENSION ([x1:]x2, [y1:]y2, ...) :: A
```

Ogni dimensione è separata da virgole e i limiti degli indici per ogni dimensione sono separati da ":"

Se come nell'esempio che segue non è specificato l'indice inferiore, si assume che l'array parta dall'indice 1.

```
REAL, DIMENSION (40, 60) :: A
```



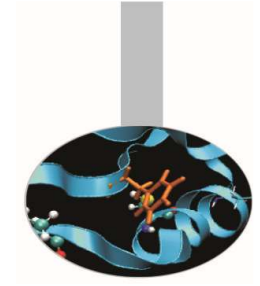
# Array - Esempi

**Uso di PARAMETER:** in generale possono essere utili uno o più PARAMETER per dichiarare le dimensioni degli array statici.

```
INTEGER, PARAMETER :: n=10  
REAL, DIMENSION(n,n+1) :: A,B,C
```

**Nelle subroutine** è possibile utilizzare argomenti di tipo INTEGER per dichiarare le dimensioni di array passati ad una subroutine e array automatici.

```
SUBROUTINE lavora(n,...)  
    INTEGER :: n  
    REAL, DIMENSION(n,2*n) :: A,B,C
```



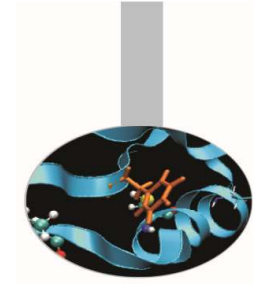
# Array - Esempi

**Dichiarazione compatta:** per dichiarare le dimensioni dell'array si può anche non utilizzare l'attributo **DIMENSION**, bensì la forma:

```
TIPO :: A([x1:]x2, [y1:]y2, ...)
```

Sono pertanto equivalenti le forme:

```
INTEGER, DIMENSION(4,2) :: A,B,C  
INTEGER :: A(4,2), B(4,2), C(4,2)
```



# Array - Esempi

**Dimensioni presunte:** nel caso di array passati in argomento, è possibile evitare di dichiararne le dimensioni, esplicitando **solo il rango**.

Si usa pertanto l'attributo `DIMENSION` (o la forma compatta), ma i valori vengono sostituiti da “:”

```
INTEGER, DIMENSION (:, :) :: c
```





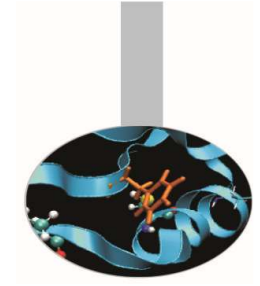
# Array - Inizializzazione

Per leggere o modificare gli elementi di un array è possibile **accedere:**

- ad un singolo elemento
- a sezioni di array (da ... a ...)
- a sottoinsiemi di array utilizzando liste di indici

Per **inizializzare un vettore** possiamo utilizzare:

- assegnazione diretta (componente per componente)
- costrutti DO
- notazione vettoriale
- costruttore (eventualmente con DO implicito)



# Array - Inizializzazione

## Assegnazione diretta:

```
REAL, DIMENSION(3) :: a
    a(1) = 0.0
    a(2) = 1.0
```

## Costrutti DO

```
REAL, DIMENSION(3) :: a
    DO i = 1, 3
        a(i) = REAL(i)
    END DO
```

**Notazione vettoriale:** il Fortran 90 permette di lavorare con vettori e matrici considerandoli nella loro globalità.

```
REAL, DIMENSION(6) :: A = 0.0
REAL, DIMENSION(10,10) :: M = 1.0
```



# Array - Inizializzazione

**Notazione vettoriale** : accesso a sezioni di array

$A(2:5) = 0.1$

$M(2:4, 2:4) = 2.0$

**Sottoinsiemi di indici (solo vettori):**

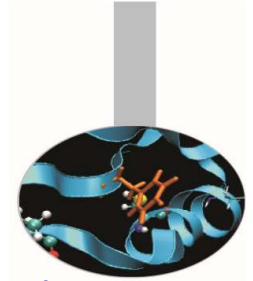
```
REAL, DIMENSION(10) :: a
```

```
INTEGER, DIMENSION(10) :: ip
```

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

```
a(ip) = (/10, 20, 30, 60, 120, 240, 480, 960, 1920, 3840/)
```

Con questo tipo soluzione occorre fare attenzione perché in una assegnazione non si ripetano due indici



**Array Costructor:** per inizializzare array monodimensionali si può utilizzare una lista di variabili (separate da virgola) racchiusa tra (/ ... /) o [...].

Le variabili possono anche essere a loro volta vettori.

```
array = (/ lista /)
```

```
array = [ lista ]
```

**Esempi:**

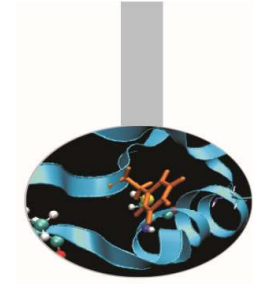
```
INTEGER, DIMENSION(6) :: A = (/1,2,3,4,5,6/)
```

```
REAL VECTOR_X(3), VECTOR_Y(2), RESULT(100)
```

```
. . .
```

```
RESULT(1:8)=[ 1.3, 5.6, VECTOR_X, 2.35, VECTOR_Y]
```

```
INTEGER, DIMENSION(6) :: A = (/ (i, i=1, 6) /)
```



# Array - Inizializzazione

Per array di rango  $> 1$ , l'array constructor non funziona, ma si può utilizzare una appropriata funzione intrinseca:

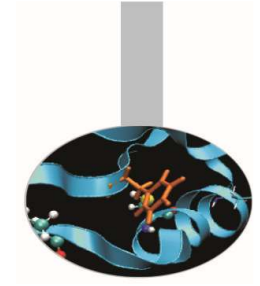
`RESHAPE (SOURCE, SHAPE)`

Che permette di “riformare” un vettore (SOURCE) secondo una qualsiasi possibile “SHAPE”.

`SOURCE = [ 1, 2, 3, 4, 5, 6 ]`

`Y = RESHAPE (SOURCE , [ 3, 2 ])`

1	4
2	5
3	6



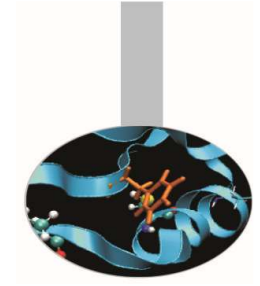
# Array - Inizializzazione

Un esempio:

```
REAL, DIMENSION(3,2) :: ra  
ra = RESHAPE((/(i+j,i=1,3),j=1,2)/), SHAPE = (/3,2/))
```

trasforma il vettore (2,3,4,3,4,5)

```
nella matrice   ra =  2 3  
                  3 4  
                  4 5
```



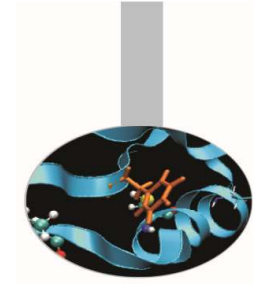
# Notazione Vettoriale

Il Fortran 90 permette di lavorare con vettori e matrici considerandoli nella loro **globalità**, a differenza del fortran 77 che obbliga a lavorare elemento per elemento.

Vettori e matrici devono essere **conformi** per poter essere coinvolti in **operazioni vettoriali**.

E' essenziale ricordare che per calcolare il risultato di un'assegnazione vettori e matrici sono **valutati prima che l'assegnazione abbia luogo**, elemento per elemento. Ovvero si lavora sempre per elementi, con un ciclo DO implicito.

Le **funzioni intrinseche** possono avere **risultato vettoriale**.



# Notazione Vettoriale

## Esempi :

```
REAL, DIMENSION(10,10) :: a, b, c
```

**Queste 3 forme sono equivalenti**

```
a = 0.0
```

```
a(:, :) = 0.0
```

```
a(1:10, 1:10) = 0.0
```

**Queste 3 forme sono equivalenti**

```
c = a * b    ! Prodotto elementale
```

```
c(:, :) = a(:, :) * b(:, :)
```

```
c(1:10, 1:10) = a(1:10, 1:10) * b(1:10, 1:10)
```





# Notazione Vettoriale

## Esempi :

```
REAL, DIMENSION(10,-5:5,3) :: a, b
```

```
REAL, DIMENSION(-4:5,1:5,3) :: c
```

**Si possono specificare solo gli elementi di indice dispari**

```
a(1:10:2,-3:5:2,1:3:2) = b(1:10:2,-3:5:2,1:3:2) *
c(::2,1:5,::2)
```

**Oppure un solo piano**

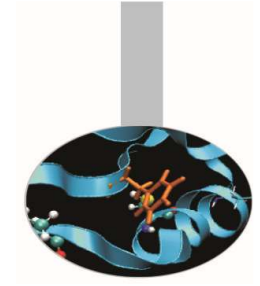
```
a(5,-5:,:) = b(3,-5:,:)
```

**Oppure solo una parte di piano**

```
a(5,1:5,:) = b(5,1:5,:) + c(0,:::)
```

**Ovvero solo alcune righe di un piano**

```
a(1:10:3,0,1:3) = c(1:4,1,:)
```



# Notazione Vettoriale

La notazione vettoriale è una sintassi molto efficace, ma è necessario tener ben presente come vengono eseguite le operazioni.

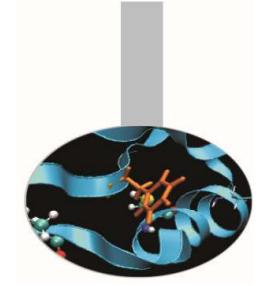
Il principio è che tutto ciò che compare alla destra del segno di assegnazione è **interamente** calcolato, **prima** che l'assegnazione venga fatta:

$$a(2:n) = a(2:n) + b(1:n-1)$$

**equivale a :**

```
DO i = 2, n
    a(i) = a(i) + b(i-1)
END DO
```

# Notazione Vettoriale



$$b(2:n) = a(2:n) + b(1:n-1)$$

**equivale a:**

```
DO i = 2, n
    t(i) = a(i) + b(i-1)
```

```
END DO
```

```
DO i = 2, n
    b(i) = t(i)
```

```
END DO
```

**e non come si potrebbe erroneamente pensare a:**

```
DO i = 2, n
    b(i) = a(i) + b(i-1)
```

```
END DO
```