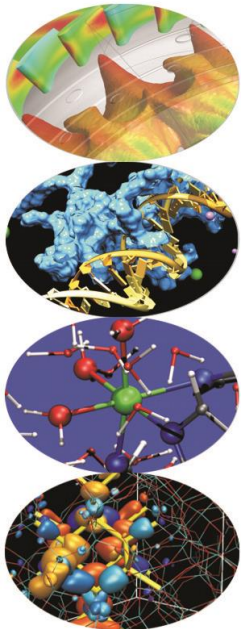




Vettori e Matrici (array)





Array - Introduzione

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

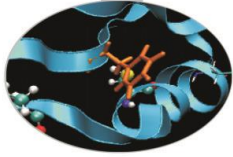
Il Fortran consente di gestire:

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

Gli array in Fortran 90 possono essere di 3 tipi:

- Statici
- Semi-dinamici
- Dinamici

Array - Introduzione

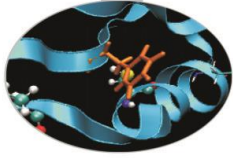


Array statici: hanno dimensioni fissate al momento della dichiarazione e il numero di elementi nell'array non può essere modificato durante l'esecuzione del programma.

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



Rango

numero di dimensioni della matrice

Estensione

numero di elementi in una dimensione

Forma

vettore delle estensioni

Dimensione

prodotto delle estensioni

Conformità

due matrici con la stessa forma

Per dichiarare un array è necessario fornire 3 informazioni:

tipo degli elementi

numero di dimensioni

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
```



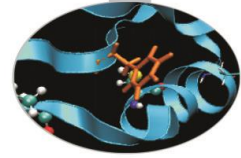
Array - Esempi

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

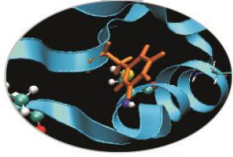
Dichiarazione compatta: per dichiarare le dimensioni dell'array si può anche non utilizzare l'attributo **DIMENSION**, bensì la forma:

```
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 - Inizializzazione

Costruttore: per inizializzare array monodimensionali si può utilizzare una lista di variabili (separate da virgola) racchiusa tra parentesi e slash:

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

Esempi:

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

Con DO implicito:

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



Array - Inizializzazione

Per le **matrici** non esiste una sintassi compatta, ma è necessario combinare il costruttore dei vettori con la funzione intrinseca `RESHAPE (SOURCE, SHAPE [, PAD] [, ORDER])`, che genera una matrice di forma `SHAPE` prendendo il valore degli elementi da `SOURCE`. Se questi non sono sufficienti si usano gli elementi di `PAD`, eventualmente duplicandoli. Il vettore `ORDER` deve avere la stessa dimensione di `SHAPE` e specifica l'ordine col quale devono essere generate le dimensioni.



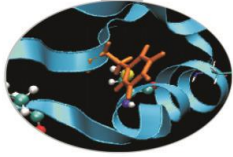
Array - Inizializzazione

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** utilizzato nell'esecuzione dei calcoli, **prima** che ciò che appare alla sinistra venga preso in considerazione:

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

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
```

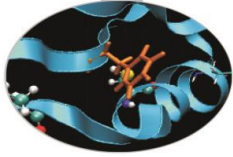
Il seguente ciclo:

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

```
END DO
```

invece equivale a :

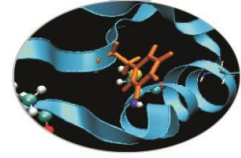
```
a(2:n) = (/ (SUM(a(1:i)), i=2, n) /)
```



Esercizi

1. Generare una matrice A 9×9 i cui elementi siano $A(i,j) = ij$ (es. $a(3,4)=34$) e stamparla a video e su file.
2. Scrivere un programma che costruisca e stampi una matrice 10×10 che rappresenti la Tavola Pitagorica
3. Scrivere un programma che generi una scacchiera 8×8 con "B" e "W" in posizioni alternate, usando la notazione vettoriale.

Soluzioni



Esercizio 1/0

```
PROGRAM matrice

  IMPLICIT NONE

  INTEGER, PARAMETER :: n=9

  REAL, DIMENSION(n,n) :: A, B, C

  INTEGER, DIMENSION(n) :: v

  INTEGER :: i,j

  DO i=1,n
    DO j=1,n
      A(i,j)=REAL(i*10+j)
    END DO
  END DO

  OPEN(11,FILE='matrice.dat',STATUS='replace')
```

Soluzioni



Esercizio 1/1

```
DO i=1,n
    WRITE (*,*) A(i,:)
    WRITE (11,*) A(i,:)

END DO

CLOSE (11)

A=0.0

v=(/(REAL(i),i=1,n)/)

B = SPREAD(v,DIM=1,NCOPIES=n)
C = SPREAD(v,DIM=2,NCOPIES=n)

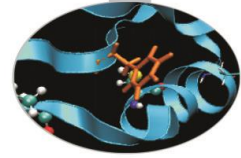
A=B+C*10

DO i=1,n
    WRITE (*,*) A(i,:)

END DO

END PROGRAM matrice
```

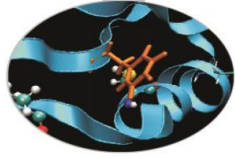
Soluzioni



Esercizio 2/0

```
PROGRAM Pitagora
    IMPLICIT NONE
    INTEGER, DIMENSION(10,10) :: a,b,p
    INTEGER, DIMENSION(10) :: v
    INTEGER :: i
    v=(/(i,i=1,10)/)
    WRITE(*,*) "v = "
    WRITE(*,100) v
    a = SPREAD(v,DIM=1,NCOPIES=10)
    b = SPREAD(v,DIM=2,NCOPIES=10)
    WRITE(*,*) "A = "
    DO i=1,10
        WRITE(*,100) a(i,:)
    END DO
```

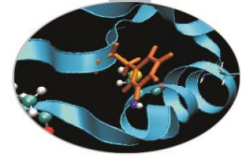
Soluzioni



Esercizio 2/1

```
WRITE (*,*) "B = "  
    DO i=1,10  
        WRITE (*,100) b(i,:)   
    END DO  
p=a*b  
WRITE (*,*) "P = "  
    DO i=1,10  
        WRITE (*,100) p(i,:)   
    END DO  
  
100    FORMAT(10(1x,i3))  
END PROGRAM Pitagora
```


Soluzioni



Esercizio 3/0

PROGRAM chequerboard

```
IMPLICIT NONE
```

```
CHARACTER (LEN = 1) , DIMENSION (8,8) :: chboard
```

```
! Version 1: 4 statements:
```

```
WRITE(*, '(' Version 1 - 4 statements: '//)')
```

```
chboard (1:8:2, ::2) = 'B'
```

```
chboard (2:8:2, ::2) = 'W'
```

```
chboard (1:8:2, 2::2) = 'W'
```

```
chboard (2:8:2, 2::2) = 'B'
```

```
WRITE(*, '(8A4)') chboard
```

```
READ (*, *)
```

Soluzioni



Esercizio 3/1

```
! Version 2: 3 statements:
WRITE(*, '(//'' Version 2 - 3 statements: ''//)')

chboard = 'B'
chboard (2:8:2, ::2) = 'W'
chboard (1:8:2, 2::2) = 'W'

WRITE(*, '(8A4)') chboard
READ (*, *)

STOP
END PROGRAM chequerboard
```