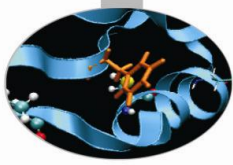
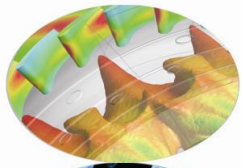
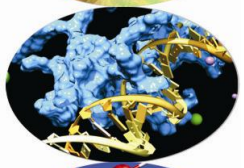
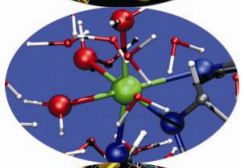
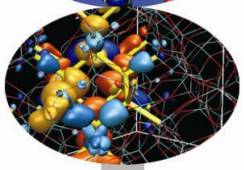
CINECA SCAI
SuperComputing Applications and Innovation



Gestione della Memoria

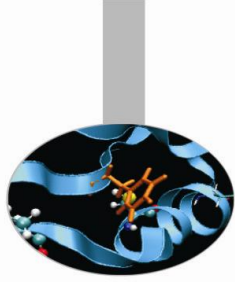





Introduction to Fortran 90

Paolo Ramieri, *CINECA*

Aprile 2013

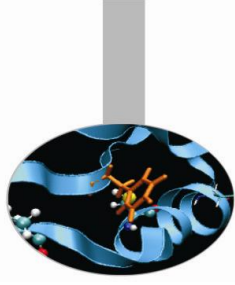
CINECA



Gestione della memoria

Il Fortran 90 permette di gestire dinamicamente la memoria, in almeno quattro modi diversi:

- array automatici
- array a dimensioni presunte
- array allocabili
- sinonimi (puntatori)



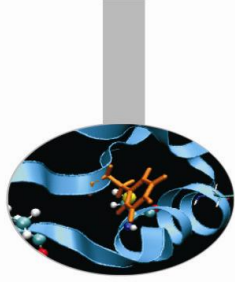
Array automatici

Vettori e matrici sono detti "**allocati automaticamente**" se sono **dichiarati** in una **procedura**, ma **non sono passati in argomento**.

Le loro **dimensioni** sono **definite da variabili passate in argomento** o tramite `USE` di un modulo o ereditate dalla procedura ospite.

In tal modo le **dimensioni** di un vettore automatico possono essere **diverse a seconda delle chiamate di procedura**.

La **memoria** per vettori e matrici automatici è **riservata** quando la procedura è attivata ed è sempre **rilasciata** all'uscita.

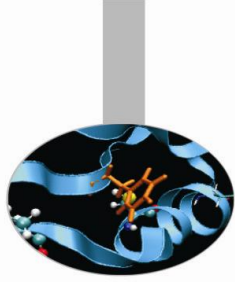


Array automatici - Esempi

Esempio 1:

```
SUBROUTINE auto(n,a)
    IMPLICIT NONE
    INTEGER :: n
    REAL, DIMENSION (n,n), INTENT(INOUT) :: a
    REAL, DIMENSION (n,n) :: work1
    REAL, DIMENSION (SIZE(a,1)) :: work2
END SUBROUTINE auto
```

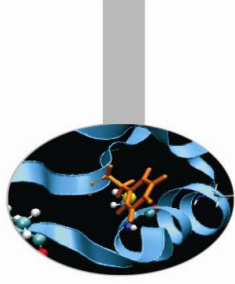
`work1` e `work2` sono array automatici. Essi prendono le dimensioni dall'intero `n` e dall'array `a` che sono argomenti della procedura.



Array automatici - Esempi

Esempio 2: Uso dei moduli

```
MODULE auto_mod
    INTEGER :: n
    CONTAINS
       SUBROUTINE auto_sub
          REAL, DIMENSION(n) :: w
          WRITE(*,*) 'Bounds and size of w: ', LBOUND(w),
  UBOUND(w), SIZE(w)
       END SUBROUTINE auto_sub
END MODULE auto_mod
PROGRAM auto_array
    USE auto_mod
    INTEGER :: i
    DO i=,10
       n=i
      CALL auto_sub
    ENDDO
END PROGRAM auto_array
```



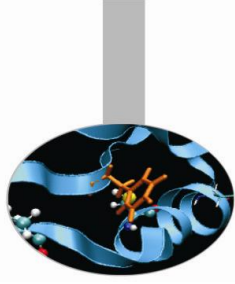
Array a dimensione presunte

Si dicono "**array a dimensioni presunte**", array (dichiarati in procedura) le cui **dimensioni non** sono **note**, ma vengono di volta in volta imposte dall'**argomento** corrispondente passato alla procedura.

Quando si dichiara un array a dimensioni presunte, le sue dimensioni sono indicate con : (o in alternativa con [**lower_bound**]:).

Pertanto questi array rendono non necessario passare ad una procedura le dimensioni di vettori o matrici passati in argomento.

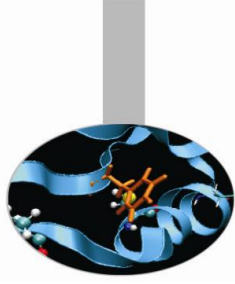
Tuttavia è necessario che l'**interfaccia** della procedura sia **esplicita**; pertanto questa deve essere fornita dal programmatore se la procedura è esterna.



Array a dimensioni presunte

Esempio:

```
PROGRAM DimmAss
      IMPLICIT NONE
      INTERFACE
         SUBROUTINE sub(a,b,c)
            REAL, DIMENSION(:,:), INTENT(IN) :: a, b
            REAL, DIMENSION(0:,2:), INTENT(INOUT) :: c
         END SUBROUTINE sub
      END INTERFACE
      REAL, DIMENSION(0:9,10) :: r
      CALL sub(r,r(0:4,2:6),r(0:4,2:6))
END PROGRAM DimmAss
SUBROUTINE sub(a,b,c)
      REAL, DIMENSION(:,:), INTENT(IN) :: a, b
      REAL, DIMENSION(0:,2:), INTENT(INOUT) :: c
END SUBROUTINE sub
```



Array allocabili

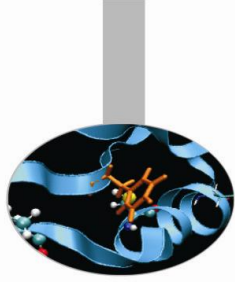
Il Fortran 90 permette di definire array **senza specificarne a priori le dimensioni** che verranno poi **assegnate in corso di esecuzione**, pertanto la memoria riservata può essere rilasciata e riassegnata dinamicamente utilizzandola solo quando effettivamente necessario.

Gli array allocabili vengono dichiarati mediante l'attributo di dichiarazione `ALLOCATABLE`.

E' inoltre necessario specificare il rango dell'array mediante l'attributo `DIMENSION`, ma non le dimensioni.

Esempi di dichiarazione:

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



ALLOCATE e DEALLOCATE

In seguito alla dichiazione di un array **allocabile** il programma non riserva memoria per l'array in questione. Esistono apposite istruzioni per gestire la memoria degli array allocabili.

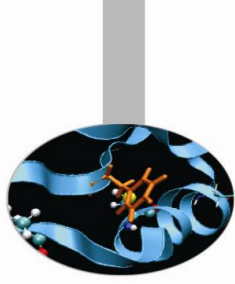
Per **riservare memoria** si usa l'istruzione:

```
ALLOCATE(nome_array([x1:]x2,[y1:]y2,...),STAT=st)
```

Per **rilasciare** la **memoria** si usa l'istruzione:

```
DEALLOCATE(nome_array)
```

La memoria dei vettori allocabili dev'essere assegnata e rilasciata nel programma principale o nella procedura dove sono dichiarati.



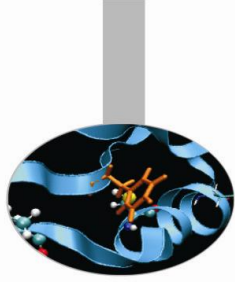
ALLOCATE e DEALLOCATE

Esempio: Array allocati in seguito alla lettura delle dimensioni

```
PROGRAM alloca
  INTEGER :: n1, n2,st
  REAL, DIMENSION(:,:), ALLOCATABLE :: B
  READ(*,*) n1, n2
  ALLOCATE(B(n1,n2),STAT=st)
  IF ( st /= 0 ) STOP " Errore allocazione B(:,:)"
  DEALLOCATE(B)
  STOP
END PROGRAM alloca
```

L'attributo `SAVE` permette al vettore di rimanere allocato in uscita dalla procedura, e ne salva i valori delle componenti:

```
REAL, DIMENSION(:), ALLOCATABLE, SAVE :: a
```



La funzione ALLOCATED

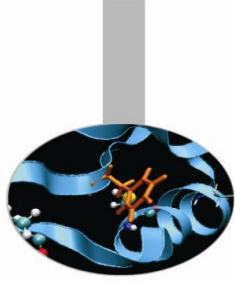
Un array allocabile può essere in 2 possibili stati:

- allocato, ossia con memoria associata
- al momento non allocato, ossia non ha memoria riservata

Lo stato di un array può essere verificato mediante la funzione logica `ALLOCATED(nome_array)`.

Questa funzione ritorna:

`.TRUE.`	se l'array ha memoria associata
`.FALSE.`	altrimenti



La funzione ALLOCATED

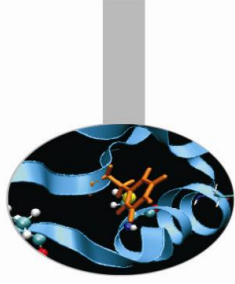
Esempio 1 :

```
IF ( ALLOCATED(a) ) DEALLOCATE(a,STAT=st)
```

Esempio 2:

```
IF ( .NOT. ALLOCATED(a) ) ALLOCATE(a(1:10),STAT=st)
```

Nota: immediatamente dopo la sua dichiarazione un array si trova nello stato "non allocato“.

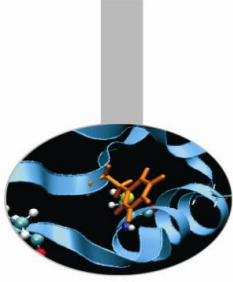


Array allocabili - restrizioni

Ci sono tre **restrizioni** nell'uso dei vettori allocabili:

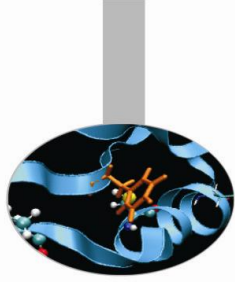
- non possono essere argomento di procedure
- non possono essere risultato di una funzione
- non possono essere usati nella definizione di tipi derivati.

Array – Esempio riassuntivo



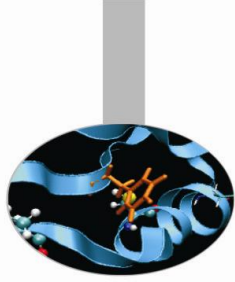
```
PROGRAM array
 IMPLICIT NONE
 REAL, ALLOCATABLE, DIMENSION(:,:) ::a
 REAL :: res
 INTEGER ::n1,n2
 INTERFACE
  SUBROUTINE sub(b,res)
   REAL,INTENT(OUT) :: res
   REAL, DIMENSION (:,:),INTENT(IN)::b
   REAL,DIMENSION (SIZE(b,1)),SIZE(b,2)) :: work
  END SUBROUTINE sub
 END INTERFACE
 READ (*,*)n1,n2
 ALLOCATE(a(n1,n2))
 CALL sub(a,res)
END PROGRAM array
   SUBROUTINE sub(b,res) !la subroutine è esterna
     IMPLICIT NONE
     REAL,INTENT(OUT) :: res
     REAL, DIMENSION (:,:),INTENT(IN)::b
     REAL,DIMENSION (SIZE(b,1)),SIZE(b,2)) :: work
      res=b(...)
   END SUBROUTINE sub
```

Esercizi



1. Implementare il seguente algoritmo per trovare i numeri primi:
   a) definire un vettore "Prime" di dimensione n
   b) inizializzare "Prime" di modo che Prime(i) = i per i =1,n
   c) porre i = 2
   d) per j>i (i+1<j<n) se Prime(j) è esattamente divisibile per i allora porre Prime(j) = 0
   e) incrementare i
   f) se i = n allora terminare il programma
   g) se Prime(i) è nullo allora tornare al punto e, altrimenti ripetere dal punto d.
   h) Infine scrivere tutti i valori non nulli di Prime (ovvero i numeri primi)

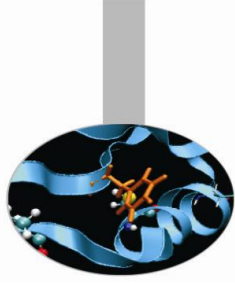
Esercizi



2. Scrivere un programma che contenga una funzione interna che calcoli la deviazione standard dal valor medio di un vettore di valori reali secondo la seguente formula:

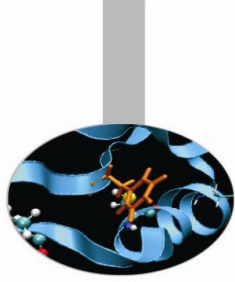
$$s = \sqrt{\frac{\sum_{i=1}^{n} (x_i - m)^2}{n}}$$

Esercizi



3. Scrivere un programma che calcoli "i" volte il prodotto elemento per elemento (o elementale) fra due matrici conformi A e B di dimensione NxN utilizzando sempre gli stessi elementi per la matrice A e senza usare l'attributo SAVE.La matrice A può essere costruita in modo che ogni elemento sia dato dalla somma degli indici corrispettivi, ovvero `a(j,m) = j+m.`
   Ogni elemento della matrice B sarà dato dalla somma degli indici corrispettivi più lo step "i" indice di quante volte ho già eseguito il prodotto scalare fra A e B, ovvero `b(j,m) = j+m+i.`
   A tale scopo conviene utilizzare una SUBROUTINE interna.

Esercizi



4. Verificare ora il miglioramento delle prestazioni del programma dell'esercizio precedente nella velocità di esecuzione utilizzando l'attributo SAVE. In questo caso le matrici vengono allocate solo la prima volta.