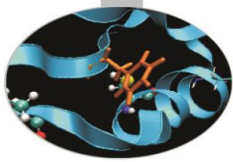
CINECA SCAI
SuperComputing Applications and Innovation

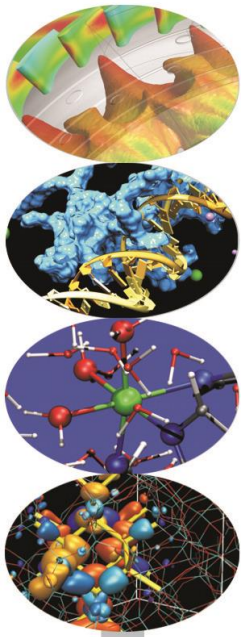


VETTORI E MATRICI
Funzioni Intrinseche

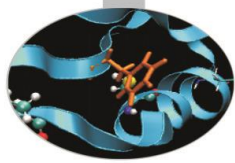
Introduction to Fortran 90

Paolo Ramieri, *CINECA*

Aprile 2014



CINECA



FUNZIONI INTRINSECHE

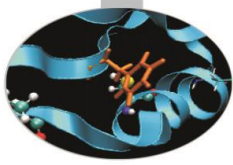
Le funzioni intrinseche relative a vettori e matrici (in Fortran 90) sono numerose e possono essere classificate in base al loro ambito di azione:

- **Calcolo** (*arithmetic*)
- **Riduzione** (*reduction*)
- **Informative** (*inquiry*)
- **Costruttive** (*construction*)
- **Ricerca** (*location*)
- **Manipolazione** (*manipulation*)

Funzioni di calcolo (*arithmetic*)

`DOT_PRODUCT(v,w)`	calcola il prodotto scalare dei vettori V e W
`MATMUL(a,b)`	calcola il prodotto matriciale tra A e B

PRODOTTO DI MATRICI



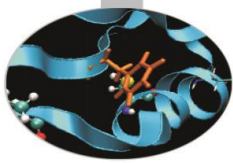
Esempio 1: Prodotto di matrici

Sintassi Fortran 77 (comunque valida)

```
      REAL a(5,10), b(10,20), c(5,20)
         .   .   .
C          Questo e' il prodotto matriciale, in Fortran 77
      DO 100 j = 1, 20
         DO 110 i = 1, 5
            c(i,j) = 0.0
            DO 120 k = 1, 10
               c(i,j) = c(i,j) + a(i,k) * b(k,j)
120         CONTINUE
110      CONTINUE
100   CONTINUE
         .   .   .
```

Prodotto di matrici in Fortran 90

```
   REAL, DIMENSION(5,10) :: a
    REAL, DIMENSION(10,20) :: b
    REAL, DIMENSION(5,20) :: c
         .   .   .
!    Questo e' il prodotto matriciale, in Fortran 90
    c = MATMUL(a,b)
         .   .   .
```



PRODOTTO SCALARE

Esempio 2: Prodotto scalare

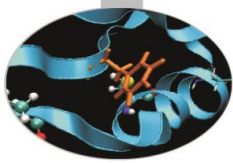
Sintassi Fortran 77 (comunque valida)

```
      REAL a(10), b(10), c
         .   .   .
      c = 0.0
         .   .   .
      DO 100 i = 1, 10
            c = c + a(i) * b(i)
100   CONTINUE
         .   .   .
```

Sintassi vettoriale Fortran 90

```
      REAL, DIMENSION(10) :: a, b
      REAL :: c = 0.0
        .   .   .
      c = DOT_PRODUCT(a,b)
        .   .   .
```

FUNZIONI INTRINSECHE DI RIDUZIONE



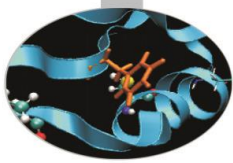
Le funzioni di riduzione si applicano a vettori e matrici. Ritornano un valore scalare, a meno che non venga specificato lungo quale dimensione fare i conti, per cui ritornerebbero un risultato vettoriale o matriciale.

Le funzioni `ALL`, `ANY`, `COUNT` si applicano a matrici di tipo logico e ritornano un valore di tipo logico o numerico.
Le funzioni `MAXVAL`, `MINVAL`, `PRODUCT`, `SUM` si applicano a matrici di tipo numerico e ritornano un valore numerico.

Funzioni di riduzione (*reduction*)

`ALL(MASK[,DIM])`	`.TRUE.` solo se tutti gli elementi sono `.TRUE.`
`ANY(MASK[,DIM])`	`.TRUE.` se almeno un elemento è `.TRUE.`
`COUNT(MASK[,DIM])`	numero degli elementi `.TRUE.`
`MAXVAL(ARRAY[,DIM][,MASK])`	valore del massimo degli elementi
`MINVAL(ARRAY[,DIM][,MASK])`	valore del minimo degli elementi
`PRODUCT(ARRAY[,DIM][,MASK])`	prodotto degli elementi
`SUM(ARRAY[,DIM][,MASK])`	somma degli elementi

RICERCA DEL MASSIMO



Esempio 1: Ricerca del massimo

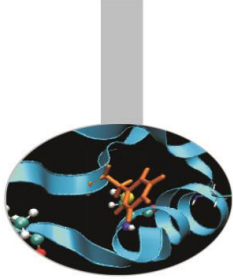
Sintassi Fortran 77 (comunque valida)

```
      REAL a(100,100,100)
        .   .   .
C      Ricerca del valore massimo inferiore a 1.0
       valmax = a(1,1,1)
       DO i = 1, 100
          DO j = 1, 100
             DO k = 1, 100
                IF ( a(k,j,i) .LT. 1.0 ) THEN
                   IF ( a(k,j,i) .GT. valmax ) valmax = a(k,j,i)
                END IF
             END DO
          END DO
       END DO
        .   .   .
```

Sintassi Fortran 90

```
        REAL, DIMENSION(100,100,100) :: a
      .   .   .
!   Ricerca del valore massimo [inferiore a 1.0]
    valmax = MAXVAL( a [,MASK=(a<1.0)] )
```

CALCOLO DEL VALORE MEDIO



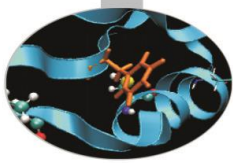
Esempio 2 : Calcolo del valore medio

Sintassi Fortran 77 (comunque valida)

```
      REAL a(100,100,100)
        .   .   .
C       Ricerca del valore medio per gli elementi positivi
        somma = 0.0
        numel = 0
        DO i = 1, 100
           DO j = 1, 100
              DO k = 1, 100
                 IF ( a(k,j,i) .GT. 0.0 ) THEN
                    somma = somma + a(k,j,i)
                    numel = numel + 1
                 END IF
              END DO
           END DO
        END DO
        valmed = somma / numel
```

Sintassi Fortran 90

```
REAL, DIMENSION(100,100,100) :: a
      .   .   .
valmed = SUM(a,MASK=(a>0.0))/COUNT(MASK=(a>0.0))
```

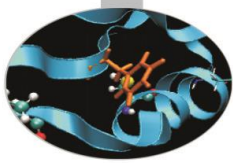


USO DI ALL

Esempio 3 : Uso di `ALL`

```
      LOGICAL :: test, test2, test3
REAL, DIMENSION(3,2) :: a

a= RESHAPE( (/5,9,6,10,8,12/),(/3,2/) )
  .   .   .
test = ALL(a > 5) ! false
test2 = ALL(a < 20) ! true
test3 = ALL(a >= 5 .AND. test2) ! true
  .   .   .
```

FUNZIONI INTRINSECHE INFORMATIVE

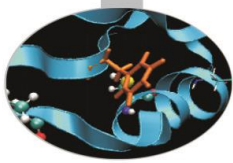


Queste funzioni permettono di **conoscere le proprietà** delle variabili.

`ALLOCATED` si applica a matrici allocabili e sinonimi e ritorna un valore di tipo logico.
`LBOUND`, `UBOUND` si applicano a matrici e ritornano i vettori che specificano i limiti inferiori e superiori della numerazione degli indici. Hanno la stessa dimensione delle matrici cui sono applicati.
`SHAPE` si applica a matrici e scalari e ritorna un vettore di lunghezza pari al numero di dimensioni della matrice, che descrive l'estensione di ogni dimensione. Per gli scalari la lunghezza del risultato è 0.
`SIZE` si applica solo a matrici e ne ritorna la dimensione.

Funzioni informative (*inquiry*)

`ALLOCATED(ARRAY)`	`.TRUE.` se la memoria è associata
`LBOUND(ARRAY[,DIM])`	limiti inferiori per gli indici, per ogni dimensione
`SHAPE(SOURCE)`	forma della matrice (si applica anche agli scalari)
`SIZE(ARRAY[,DIM])`	dimensione della matrice
`UBOUND(ARRAY[,DIM])`	limiti superiori per gli indici, per ogni dimensione



USO DI SIZE

Esempio 1: Uso di `SIZE`

```
REAL, DIMENSION(3,2) :: a
  .    .    .
num = SIZE(a) ! num=6
num = SIZE(a, DIM=1) ! num=3
num = SIZE(a, DIM=2) ! num=2
  .    .    .
```

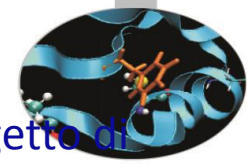
E' possibile usare `SIZE` anche nelle dichiarazioni:

```
REAL, DIMENSION(lda,n) :: a
 REAL, DIMENSION(SIZE(a,1),SIZE(a,2)) :: temp
```

Esempio 2

```
REAL DIMENSION(2:3,2:5) :: a
WRITE(*,*) LBOUND(a) ! (/2,2/)
WRITE(*,*) UBOUND(a) ! (/3,5/)
WRITE(*,*) SHAPE(a) ! (/2,4/)
WRITE(*,*) SIZE(a) ! 8
       .    .    .
```

IL COSTRUTTO WHERE



Il costrutto `WHERE` è utilizzato per controllare quali elementi di un **array** sono oggetto di un'assegnazione sulla base del risultato di una condizione logica.

L'istruzione `WHERE` ha la sintassi:

```
WHERE (matrice_logica) assegnazione_matriciale
```

L'assegnazione matriciale viene eseguita se, elemento per elemento, la condizione matriciale
è vera. **Matrice_logica e assegnazione_matriciale devono essere conformi.**

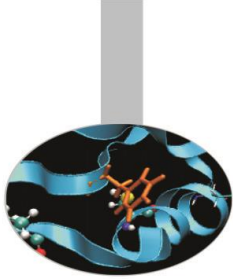
In generale il costrutto `WHERE` ha la sintassi:

```
WHERE (matrice_logica)
   assegnazione_matriciale
ELSEWHERE
   assegnazione_matriciale
END WHERE
```

L'istruzione `ELSEWHERE` all'interno del costrutto consente di fare assegnazioni quando la condizione è falsa.

I costrutti `WHERE` non possono essere annidati in Fortran 90, mentre il Fortran 95 introduce questa possibilità.

IL COSTRUTTO WHERE



Esempio 1: Evitare la divisione per zero (caso 1)

Sintassi Fortran 77 (comunque valida)

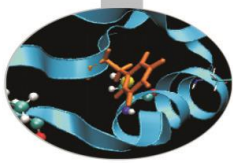
```
      REAL a(100,100), b(100,100)
      .   .   .
    DO 100 i = 1, 100
       DO 100 j = 1, 100
          IF ( b(j,i) .GT. 0.0 ) THEN
             a(j,i) = a(j,i) / b(j,i)
          END IF
```

Sintassi Fortran 90

```
      REAL, DIMENSION(100,100) :: a, b
      .   .   .
   WHERE ( b > 0.0 ) a = a / b
```

Esempio 2: Evitare la divisione per zero (caso 2)

```
      INTEGER, DIMENSION(8,8) :: a
      .   .   .
   WHERE ( a<=0 )
      a = 0
   ELSEWHERE
      a = 1/a
   END WHERE
```



ARRAY DI DIMENSIONE NULLA

Il Fortran 90 permette di trattare **vettori e matrici di dimensione nulla**.
Questo si verifica quando il limite superiore della dimensione è minore del limite inferiore.

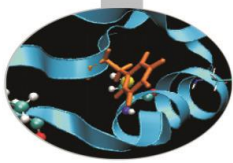
Questo fatto, apparentemente irrilevante, facilita la stesura del codice perché non è più necessario tener conto di molti casi particolari, per esempio dei contorni. Nello specifico può consentirci di non scrivere ulteriore codice per gestire il caso particolare.

Se viene chiamata l'esecuzione di un'istruzione su un array o una sua sezione a dimensione nulla, l'istruzione non viene eseguita e il programma non va in errore: il programmatore non deve quindi occuparsi del problema.

Esempio

```
      DO i = 1, n
         x(i) = b(i) / a(i,i)
!        Lunghezza nulla per I = N: valido comunque
         b(i+1:n) = b(i+1:n) - a(i+1:n,i) * x(i)
       END DO
```

FUNZIONI COSTRUTTIVE



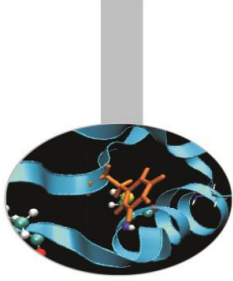
- **MERGE(TSOURCE,FSOURCE,MASK)** si applica a 3 matrici conformi. Le prime 2 devono essere dello stesso tipo, MASK dev'essere di tipo logico. Si genera una matrice conforme a MASK e di tipo uguale alle prime due; il valore di ogni elemento proviene da TSOURCE se l'elemento corrispondente di MASK è .TRUE., altrimenti proviene da FSOURCE.
- **PACK(ARRAY,MASK[,VECTOR])** genera un vettore prendendo gli elementi di ARRAY che corrispondono agli elementi .TRUE. di MASK. MASK può essere lo scalare .TRUE.. Se VECTOR è più lungo, si prende il resto degli elementi da quel vettore.
- **UNPACK(VECTOR,MASK,FIELD)** genera una matrice conforme a MASK prendendo da VECTOR gli elementi corrispondenti ai valori .TRUE. di MASK. Se VECTOR non è abbastanza lungo, si prendono gli elementi della matrice FIELD, che dev'essere conforme a MASK o essere uno scalare.
- **SPREAD(SOURCE,DIM,NS)** genera una matrice duplicando NS volte il vettore SOURCE nella dimensione DIM.

Esempio : Uso di SPREAD

```
REAL, DIMENSION(3):: a=(/2,3,4/)
REAL, DIMENSION(3,3) :: b, c
b=SPREAD(a, DIM=1, NCOPIES=3)
c=SPREAD(a, DIM=2, NCOPIES=3)
          .   .   .
```

```
     2 3 4              2 2 2
b    2 3 4         c    3 3 3
     2 3 4              4 4 4
```

FUNZIONI DI RICERCA



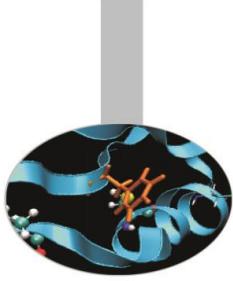
Funzioni di ricerca (*location*):
ritornano la posizione dell'elemento massimo e minimo di una matrice.

- **MAXLOC(ARRAY[,MASK])** genera un vettore con la posizione dell'elemento di valore massimo della matrice `ARRAY`, eventualmente considerando i soli elementi corrispondenti a `.TRUE.` di `MASK`.
- **MINLOC(ARRAY[,MASK])** genera un vettore con la posizione dell'elemento di valore minimo della matrice `ARRAY`, eventualmente considerando i soli elementi corrispondenti a `.TRUE.` di `MASK`.

Alcuni esempi di riepilogo:

```
                0.1  0.2  0.3  0.4     MAXVAL(a) : 40.0
a(3,4) =        1.0  2.0  3.0  4.0     MAXLOC(a) : (/3,4/)
                10.  20.  30.  40.     MAXVAL(a,DIM=2) : (/0.4,4.0,40.0/)

                0.1 0.2 0.3   1.1 1.2 1.3   2.1 2.2 2.3
c(3,3,3) =      1.0 2.0 3.0   1.1 2.1 3.1   1.2 2.2 3.2
                10. 20. 30.   11. 21. 31.   12. 22. 32.

MAXVAL(c) : 32.0

                                2.1  2.2  2.3
MAXVAL(c,DIM=3) :               1.2  2.2  3.2
                                12.  22.  32.
SHAPE(MAXVAL(c,DIM=3)) : (/3,3/)
MAXLOC(MAXVAL(c,DIM=3)) : (/3,3/)
```

FUNZIONI DI MANIPOLAZIONE



Funzioni di manipolazione (*manipulation*)

- **CSHIFT(ARRAY,SHIFT[,DIM])** genera una matrice conforme a ARRAY, spostando gli elementi di ARRAY in modo circolare lungo la dimensione specificata da DIM (o la prima), della quantità indicata in SHIFT.
- **EOSHIFT(ARRAY,SHIFT[,BOUNDARY][,DIM])** genera una matrice conforme a ARRAY, spostando gli elementi di ARRAY lungo la dimensione specificata da DIM (o la prima), della quantità indicata in SHIFT. Gli elementi che finiscono fuori dalle dimensioni di ARRAY sono persi. Le posizioni lasciate libere sono azzerate oppure riempite con gli elementi di BOUNDARY.
- **TRANSPOSE(MATRIX)** genera la trasposta della matrice MATRIX.

Esempi:

```
      1.0 5.0  9.0 13.0
a=    2.0 6.0 10.0 14.0
      3.0 7.0 11.0 15.0
      4.0 8.0 12.0 16.0
```

```
b=cshift(a,2,2):
```

```
 9.0  13.0 1.0 5.0
10.0  14.0 2.0 6.0
11.0  15.0 3.0 7.0
12.0  16.0 4.0 8.0
```

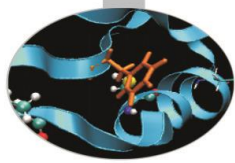
```
c=eoshift(a,2,dim=2):
```

```
 9.0 13.0 0.0 0.0
10.0 14.0 0.0 0.0
11.0 15.0 0.0 0.0
12.0 16.0 0.0 0.0
```

```
d=transpose(a):
```

```
1.0   2.0  3.0  4.0
5.0   6.0  7.0  8.0
9.0  10.0 11.0 12.0
13.0 14.0 15.0 16.0
```

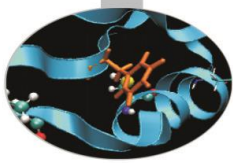
ESERCIZI



1. Generare una matrice A 9x9 i cui elementi siano A(i,j) = ij (es. a(3,4)=34) e stamparla a video e su file (***matrice.f90***).
2. Scrivere un programma che costruisca e stampi una matrice 10 x 10 che rappresenti la Tavola Pitagorica (***tavolapitagorica.f90***).
3. Scrivere un programma che generi una scacchiera 8x8 con "B" e "W" in posizioni alternate, usando la notazione vettoriale (***board.f90***).
4. Dato un vettore v nel piano ed un angolo di rotazione θ calcolare il vettore ruotato sfruttando la matrice di rotazione

   R= | cos(θ) -sin(θ) |
   | sin(θ) cos(θ) |

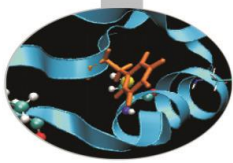
   Utilizzare un modulo per fare tutti i calcoli (***rotazione.f90***).



Esercizi

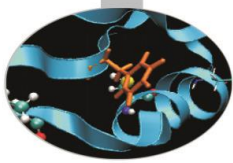
5. Utilizzando il file contenente la matrice dell'esercizio 1, facendo uso del costrutto `WHERE` sostituire con 0 tutti i numeri pari e mantenere nella loro posizione tutti i numeri dispari (***matrice2.f90***).
6. Partendo dal programma board.f90 riscrivere l'algoritmo che genera una tastiera usando `WHERE` (***board1.f90***).
7. Partendo dal programma renum0.f90 trovare valore e posizione del valore massimo del vettore RA(:) e della sua sezione RA(1:3) (***renum.f90***).
8. Riscrivere il programma spread1.f90 così da generare una matrice RA(i,j) = 1.0/REAL(i+j+1) (***spread2.f90***).

SOLUZIONE ESERCIZIO 1
matrice.f90 1/2



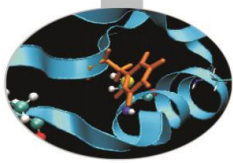
```
PROGRAM matrice
   IMPLICIT NONE
   INTEGER, PARAMETER :: n=9
   REAL, DIMENSION(n,n) :: A, B, C
   INTEGER, DIMENSION(n) :: v
   INTEGER :: i,j

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

   CLOSE(11)
```

SOLUZIONE ESERCIZIO 1
matrice.f90 2/2



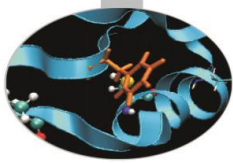
```
   A=0.0
   v=(/(REAL(i),i=1,n)/)

   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
```

SOLUZIONE ESERCIZIO 2
tavolapitagorica.f90 1/2



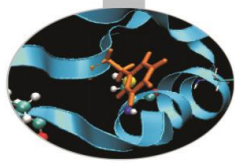
```
PROGRAM Pitagora

   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
```

SOLUZIONE ESERCIZIO 2
tavolapitagorica.f90 2/2



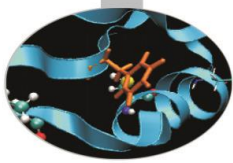
```
   WRITE(*,*) "B = "
   DO i=1,10
       WRITE(*,100) b(i,:)
   END DO

   p=a*b

   WRITE(*,*) "P = "
   DO i=1,10
       WRITE(*,100) p(i,:)
   END DO

100      FORMAT(10(1x,i3))
END PROGRAM Pitagora
```

SOLUZIONE ESERCIZIO 3
board.f90



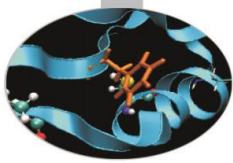
```
PROGRAM chequerboard

  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 (*,*)

  ! 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
```

SOLUZIONE ESERCIZIO 4
rotazione.f90 1/2



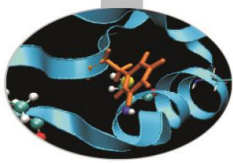
```
MODULE vettori

   IMPLICIT NONE
   REAL, DIMENSION(2) :: v,v1
   REAL :: theta

   CONTAINS
        SUBROUTINE rota (v,theta,v1)
               REAL, DIMENSION(2,2) :: R
               REAL, DIMENSION(2) :: v,v1
               REAL :: theta
!              Costruzione della matrice di rotazione
               R=cos(theta)
               R(1,2)=-sin(theta)
               R(2,1)=sin(theta)

               v1=matmul(R,v)

        END SUBROUTINE rota

END MODULE vettori
```

SOLUZIONE ESERCIZIO 4
rotazione.f90 2/2



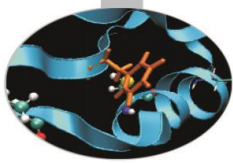
```
PROGRAM rotazione
  USE vettori

  WRITE(*,'(A10)',ADVANCE='no') "theta: "
  READ(*,*) theta

  WRITE(*,'(A10)',ADVANCE='no') "vettore: "
  READ(*,*) v

  CALL rota (v,theta,v1)

  write(*,*) v1
END PROGRAM rotazione
```

SOLUZIONE ESERCIZIO 5
matrice.f90



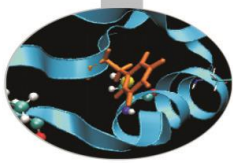
```
PROGRAM matricebis
   IMPLICIT NONE
   REAL, DIMENSION(9,9) :: B
   INTEGER :: i

   OPEN(UNIT=11,FILE='matrice.dat')
   DO i=1,9
        READ(11,*) B(i,:)
   END DO
   CLOSE(11)

   WHERE (MOD(INT(B),2)==0)
        B=0
   END WHERE

   DO i=1,9
        WRITE(*,*) B(i,:)
   END DO

END PROGRAM matricebis
```

SOLUZIONE ESERCIZIO 6
board1.f90



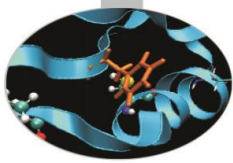
```
PROGRAM chequerboard

  IMPLICIT NONE
  CHARACTER (LEN = 1) , DIMENSION (8,8) :: chboard
  INTEGER :: i, j

  ! 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 (*,*)

  ! Version 2: 3 statements:
  WRITE(*,'(//'' Version 2 - 3 statements: ''//)')
  chboard = 'B'
  WHERE (MOD(RESHAPE( (/ ((I+J,I=1,8),J=1,8) /) ,(/8,8/)),2) == 1)
     chboard = 'W'
  END WHERE

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

  STOP
END PROGRAM chequerboard
```

SOLUZIONE ESERCIZIO 7
renum.f90 1/2



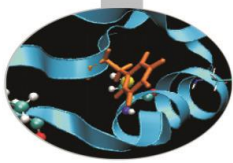
```
PROGRAM renum

  IMPLICIT NONE

  REAL, DIMENSION (-3:4) :: ra
  INTEGER, DIMENSION (1) :: locmax1, locmax2

  ! Assign elements of array ra:
  ra = (/ 1.2, 3.4, 5.4, 11.2, 1.0, 3.7, 1.0, 1.0 /)

  WRITE(*,*) " LBOUND(ra), UBOUND(ra) = ",LBOUND(ra), UBOUND(ra)
  WRITE(*,*) " RA(:,:) = ",ra
  WRITE(*,*) ""
  ! Find location of maximum value of whole array:
  ! (Note value given assumes first element has location 1.)
  locmax1 = MAXLOC(ra)
  WRITE(*,*) ' MAXLOC(ra) = ', locmax1(1)

  ! Find location of maximum value of array section:
  locmax2 = MAXLOC(ra(1:3))
  WRITE(*,*) ' MAXLOC(ra(1:3)) = ', locmax2(1)
```

SOLUZIONE ESERCIZIO 7
renum.f90 2/2



```
  WRITE(*,*)

  ! Find value of element with maximum value of whole array from location:
  WRITE(*,*) 'Maximum value of whole array is: ', &
             ra(LBOUND(ra) + locmax1(1) - 1)

  ! Check with MAXVAL:
   WRITE(*,*) 'Maximum value of whole array using MAXVAL is: ', MAXVAL(ra)

  WRITE(*,*)

  ! Find value of element with maximum value of array section from
    location:
  WRITE(*,*) 'Maximum value of ra(1:3) is: ', ra(locmax2(1))
  ! ( Value is ra(1 + locmax2(1) - 1) )

  ! Check with MAXVAL:
  WRITE(*,*) 'Maximum value of ra(1:3) using MAXVAL is: ', MAXVAL(ra(1:3))

  STOP
END PROGRAM renum
```

SOLUZIONE ESERCIZIO 8
spread2.f90



```
PROGRAM spread2

  IMPLICIT NONE
  INTEGER, PARAMETER :: n = 5
  INTEGER :: i, j
  REAL, DIMENSION (n,n) :: ra

  WRITE(*, '("Using SPREAD:"//)')
  ra = 1.0 / REAL( SPREAD( (/ (i, i = 1, n) /), DIM = 1, NCOPIES = n)  &
           + SPREAD( (/ (j, j = 1, n) /), DIM = 2, NCOPIES = n) + 1 )
  WRITE(*, '(5F8.2)') (ra(:, j), j = 1, n)

  ! Using Array Constructor:

  WRITE(*,'(//"Using Array Constructor:"//)')
  ra = RESHAPE ( (/ ((1.0/(i+j+1), i=1, n), j=1, n) /), SHAPE=(/ n, n /) )
  WRITE(*,'(5F8.2)') (ra(:, j), j = 1, n)

  ! Note the simpler method with FORALL statement in Fortran 95:
  WRITE(*,'(//"Using FORALL statement (Fortran 95):"//)')
  FORALL (i = 1: n, j = 1: n) ra(i, j) = 1.0/(i+j+1)
  WRITE(*,'(5F8.2)') (ra(:, j), j = 1, n)

  STOP
END PROGRAM spread2
```