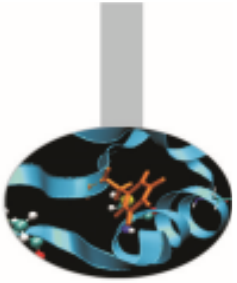
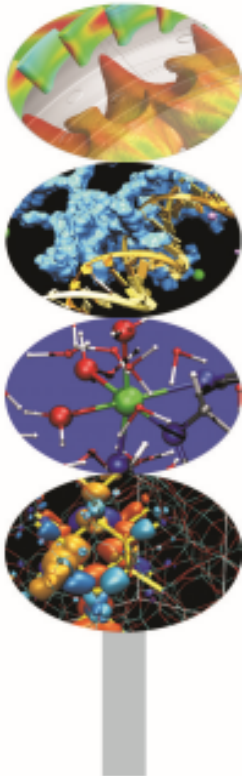
CINECA SCAI
SuperComputing Applications and Innovation



Sinonimi



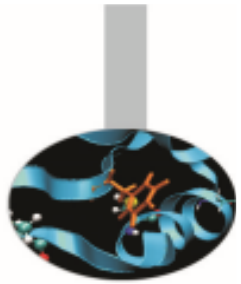
Introduction to Fortran 90

Maurizio Cremonesi, *CINECA*

Marzo 2015

CINECA

SINONIMI: SINTASSI DI BASE



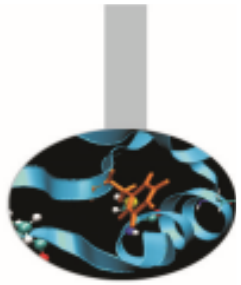
I sinonimi, o `POINTER`, non sono altro che nomi cui è associato un tipo ed eventualmente un rango.

Ad un sinonimo corrisponde uno spazio in memoria *solo dopo che è stato associato* ad un altro oggetto. Per questo motivo abbiamo tradotto con la parola sinonimi i `POINTER` Fortran.

I sinonimi possono essere associati ad altri sinonimi o ad altre entità dello stesso tipo, purché abbiano l'attributo `TARGET`.

```
TYPE(punto), POINTER :: pt, pq
TYPE(punto), TARGET :: p1, p2

pt => p1
pq => p2
```

SINONIMI: SINTASSI DI BASE

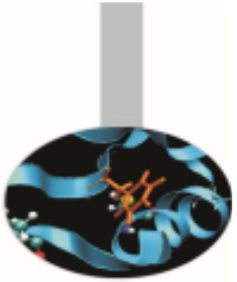


Un sinonimo viene associato ad un'altra entità con l'operatore =>.

Dopo questa associazione il "bersaglio" può essere indicato col nome del sinonimo.

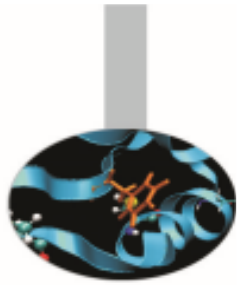
```
TYPE(punto), POINTER :: pt, pq
TYPE(punto), TARGET :: p1, p2

pt => p1    ! pt è un nuovo nome per p1
pq => p2    ! pq è un nuovo nome per p2

pq = p1 ! Ora p1 e p2 hanno lo stesso valore
```

SINONIMI: ESEMPI



```
REAL, DIMENSION(:), POINTER :: p
REAL, DIMENSION(-4:5), TARGET :: a
p => a

! ATTENZIONE: p(-4:5) è sostanzialmente
! un nuovo nome per a(-4:5)

p => a(:)
! in questo caso INVECE p(1:10) e' un
! alias di a(-4:5)

p => a(-4:5:2)
! Ora p(1:5) si può usare in luogo
! di a(-4:5:2)
```

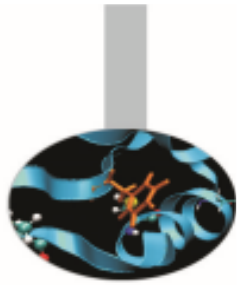
SINONIMI: ESEMPI



Si deve notare che se un sinonimo punta a una sezione di un vettore o matrice, il primo elemento per ogni dimensione ha sempre indice 1.

```
REAL, DIMENSION(:), POINTER :: p
REAL, DIMENSION(:,:), POINTER :: q
REAL, DIMENSION(5,6), TARGET :: b

p => b(3,:)
! p(1:6) è sinonimo per la terza riga
! di b(:,:)

q => b(2:4,2:4)
! q(1:3,1:3) si può usare in luogo
! di b(2:4,2:4))
```

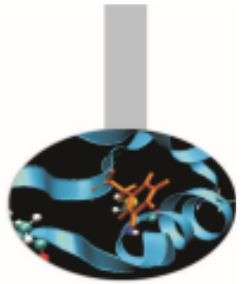
SINONIMI: ESEMPI



Non è possibile associare un sinonimo a una sezione di array definita con un altro vettore:

```
REAL, DIMENSION(:), POINTER :: pv1
REAL, DIMENSION(-3,5), TARGET :: tv1
INTEGER, DIMENSION(3) :: v=(/-3,1,4/)


pv1 =>  tv1(v)    ! Non è corretto
```

SINONIMI: SINTASSI



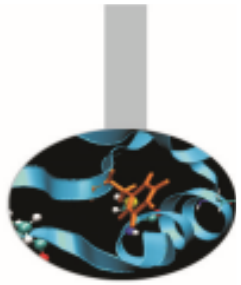
Un sinonimo dopo la dichiarazione ha stato ***indefinito***. In pratica è come una variabile cui non è stato associato ***nessun valore***.

Un sinonimo viene azzerato (nullified in inglese) con l'istruzione `NULLIFY`.

E' possibile azzerare i puntatori già nella dichiarazione, per mezzo della funzione intrinseca `NULL`:

```
REAL, POINTER :: p(:)=>NULL()
TYPE stringa
   CHARACTER, POINTER :: car(:)=>NULL()
END TYPE
```

SINONIMI: SINTASSI

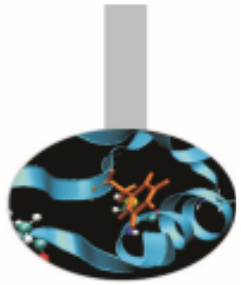


Un sinonimo si dice associato quando lo si mette in relazione ad un'altra entità con l'operatore =>, oppure gli si associa memoria con `ALLOCATE`.

Lo stato di un sinonimo può essere verificato con la funzione intrinseca `ASSOCIATED`:

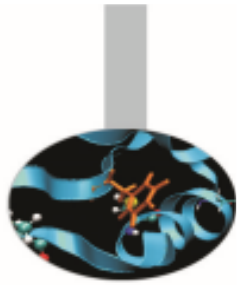
```
REAL, DIMENSION(:,:), POINTER :: q
REAL, DIMENSION(5,6), TARGET :: b

WRITE(*,*) ASSOCIATED(q)  ! Non valido: Q e' indefinito

NULLIFY(q); WRITE(*,*) ASSOCIATED(q)   ! .FALSE.

q => b; WRITE(*,*) ASSOCIATED(q)   ! .TRUE.
```

SINONIMI: SINTASSI



Un sinonimo si può usare come qualsiasi altra entità dello stesso tipo e rango purché gli si associ memoria con `ALLOCATE`, analogamente a quanto si fa con i puntatori del C.

```
REAL, DIMENSION(:,:), POINTER :: q
REAL, DIMENSION(5,6), TARGET :: b
WRITE(*,*) ASSOCIATED(q) ! Non valido: Q è indefinito
NULLIFY(q); ALLOCATE(q(5,6),STAT=st) ! Assegna memoria
WRITE(*,*) ASSOCIATED(q)   ! .TRUE.
! Si può usare q come una normale matrice:
q(:,:) = b(:,:)
DEALLOCATE(q) ! Si libera la memoria assegnata
WRITE(*,*) ASSOCIATED(q)   ! .FALSE.
```

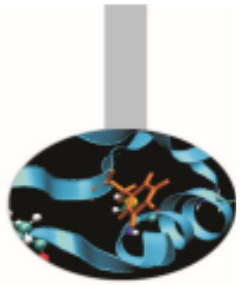


SINONIMI E TIPI PERSONALIZZATI

In Fortran 90 i vettori allocabili non possono essere usati come componenti di un tipo personalizzato.

E' necessario allora usare i sinonimi.

```
TYPE dati
    INTEGER :: lda, ldb
    INTEGER, DIMENSION(:,:), POINTER :: indici
    REAL(8), DIMENSION(:,:), POINTER :: valori
    REAL(8), DIMENSION(:,:), POINTER :: coordinate
END TYPE
```

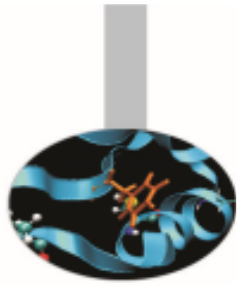


SINONIMI E MEMORIA

Quando si usano i sinonimi per gestire la memoria dinamicamente occorre fare particolare attenzione a liberare la memoria assegnata (con il comando `DEALLOCATE`) appena non è più necessaria, altrimenti si rischia di lasciare in giro "buchi" di memoria inutilizzabile.

```
REAL, DIMENSION(:,:), POINTER :: q

ALLOCATE(q(500,600),STAT=st)
! Più di 1 MB associato a q
NULLIFY(q)   ! Ora quel MB non è più accessibile
```

SINONIMI E MEMORIA



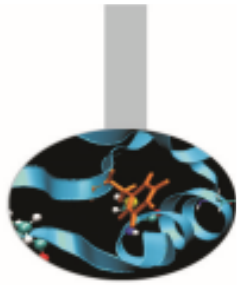
Un altro pericolo da evitare è quello del cosiddetto *sinonimo sospeso* o *dangling pointer*:

```
REAL, POINTER :: p1, p2
ALLOCATE (p1)
p1 = 3.4
p2 => p1
DEALLOCATE (p1)
```

Dopo che `p1` è stato "deallocato", la variabile alla quale puntava `p2` è scomparsa: `p2` diventa un sinonimo sospeso e ogni riferimento a `p2` produrrà risultati inattendibili.

In questi casi la soluzione è azzerare `p2` subito dopo la deallocazione di `p1`.

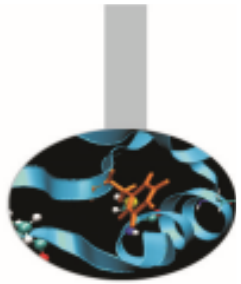
ESERCIZIO



1. Prevedere quali valori verrebbero stampati dal codice seguente:

```
PROGRAM sinonimi
REAL, POINTER :: p, q
REAL, TARGET :: t1 = 3.4, t2 = 4.5
p => t1
PRINT *,“Dopo p => t1, ASSOCIATED(p) = ", ASSOCIATED(p)
q => t1
PRINT *,“Dopo q => t1, ASSOCIATED(q) = ", ASSOCIATED(q)
PRINT *,"ASSOCIATED(p, q) = ", ASSOCIATED(p, q)
PRINT *,“ma ASSOCIATED(p, t2) = ", ASSOCIATED(p, t2)
NULLIFY(p)
PRINT *,“Dopo NULLIFY, ASSOCIATED(p) = ", ASSOCIATED(p)
END PROGRAM sinonimi
```

VETTORI DI SINONIMI: SINTASSI



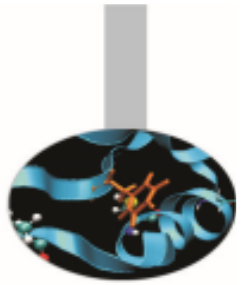
Sembrerebbe immediato definire un vettore di sinonimi usando la sintassi

```
INTEGER, DIMENSION(10,10), POINTER :: tabella
```

ma questa è **in concorrenza** con la sintassi con la quale si dichiara un sinonimo: infatti, per definizione, un sinonimo non ha dimensioni predefinite.

Di conseguenza, in Fortran 90 non è possibile definire in modo diretto un vettore di sinonimi.

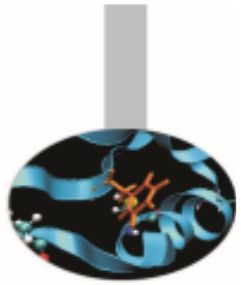
VETTORI DI SINONIMI: SINTASSI



Si aggira il problema definendo in modo opportuno un tipo personalizzato:

```
TYPE tabella
  INTEGER, DIMENSION(:), POINTER :: lista
END TYPE tabella

TYPE(tabella), DIMENSION(10,10) :: elenchi
```

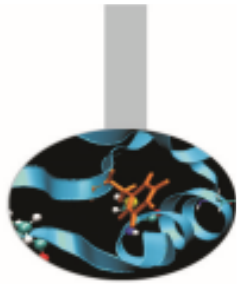
dove *ogni elemento* della matrice `elenchi` è un dato di tipo `tabella` e di conseguenza è un sinonimo.



VETTORI DI SINONIMI: SINTASSI

L'esempio appena mostrato permette di realizzare in Fortran qualcosa che si avvicina ad un vettore di sinonimi, ma deve essere completato riservando la memoria necessaria ad ogni oggetto:

```
TYPE tabella
   INTEGER, DIMENSION(:), POINTER :: lista
END TYPE tabella

INTEGER, DIMENSION(100) :: a
TYPE(tabella), DIMENSION(10,10) :: elenchi

ALLOCATE(elenchi(j,i)%lista(100),STAT=st)

elenchi(j,i)%lista(n) = a(n)
```

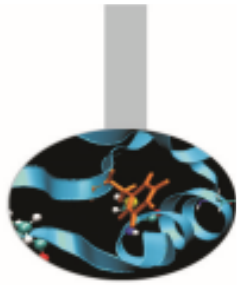


VETTORI DI SINONIMI: ESEMPIO

Può accadere in un programma di dover gestire una porzione di matrice, per esempio le metà inferiore o superiore separatamente.

```
d s s s s s s s s s
i d s s s s s s s s
i i d s s s s s s s
i i i d s s s s s s
i i i i d s s s s s
i i i i i d s s s s
i i i i i i d s s s
i i i i i i i d s s
i i i i i i i i d s
i i i i i i i i i d
```

Nello schema le metà superiore, inferiore e la diagonale sono indicate con le lettere `s`, `i` e `d` rispettivamente.



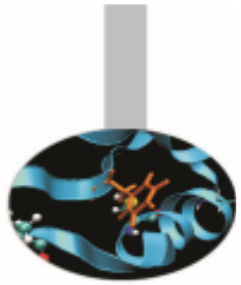
VETTORI DI SINONIMI: ESEMPIO

Se è necessario ad esempio considerare solo la metà inferiore di una matrice, può essere inutile o addirittura impossibile memorizzare l'intera matrice. In questo caso si pone il problema di come memorizzare la sola porzione che interessa senza dover sprecare inutilmente la memoria.

Una tecnica possibile fa uso di un vettore `V` che memorizza i valori ed un indice `I` che registra dove inizia ogni nuova riga:

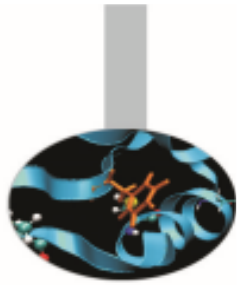
```
a11
a21 a22
a31 a32 a33
a41 a42 a43 a44

V = a11 a21 a22 a31 a32 a33 a41 a42 a43 a44
I = 1   2       4           7
```

VETTORI DI SINONIMI: ESEMPIO



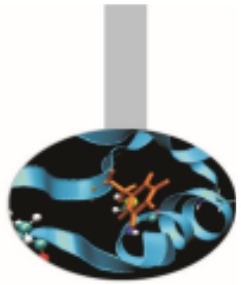
```
PROGRAM matrice_1
    IMPLICIT NONE
    INTEGER, PARAMETER :: Nda = 10  ! Numero di righe
    INTEGER, PARAMETER :: Ndat = Nda + ( Nda * Nda ) / 2
    REAL(8), DIMENSION(Ndat) :: ati ! vettore dati
    INTEGER, DIMENSION(Nda) :: i_ati ! vettore indici
    INTEGER :: ir, ic, i, ind

    Ati(:) = 0.0d0  ! Vettore dei dati
    i_ati(:) = 0        ! Vettore degli indici
    ind = 0
```

VETTORI DI SINONIMI: ESEMPIO



```
    DO ir = 1, Nda
      DO ic = 1, ir  !  Si definiscono i valori per ATI
            ind = ind + 1
            IF ( ic == 1 ) THEN     !    Inizio riga
                  i_ati(ir) = ind
            ENDIF
            ati(ind) = ir + ic
      END DO
    END DO
    DO ir = 1, Nda
      WRITE(*,*) "ATI(",ir,",:) = ",&
      &      ( ati(i_ati(ir) + ic), ic = 0, (ir - 1) )
    END DO
END PROGRAM matrice_1
```

VETTORI DI SINONIMI: ESEMPIO

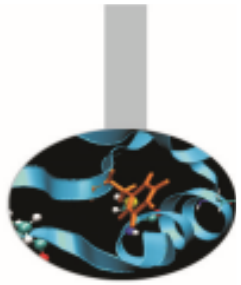


Con la sintassi Fortran è possibile strutturare i dati in modo più espressivo, senza rinunciare a risparmiare memoria:

```
TYPE riga
   REAL(8), DIMENSION(:), POINTER :: col
END TYPE riga

INTEGER, PARAMETER :: nr=1000
TYPE (riga), DIMENSION(nr) :: matA_rig

ALLOCATE(matA_rig(i)%col(nc), STAT=st)
```

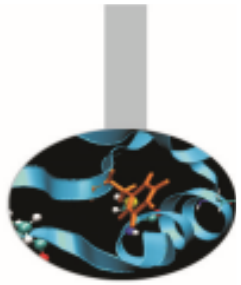
Grazie a questa struttura, l'elemento `matA_rig(i)%col(j)` corrisponderebbe all'elemento `A(i,j)`.

VETTORI DI SINONIMI: ESEMPIO



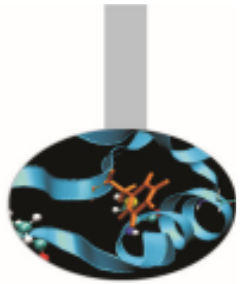
```
PROGRAM matrice_2
      IMPLICIT NONE
      INTEGER, PARAMETER :: Nda = 10

!  Matrice matA, implementata con i tipi derivati
      TYPE riga
         REAL(8), DIMENSION(:), POINTER :: col
      END TYPE riga
      TYPE (riga), DIMENSION(:), POINTER :: matA_rig
      INTEGER :: ir, ic

! Si riserva la memoria necessaria
      ALLOCATE(matA_rig(Nda), STAT=ic)
      DO ir = 1, Nda
         ALLOCATE(matA_rig(ir)%col(ir), STAT=ic)
         matA_rig(ir)%col = 0.0D0
      END DO
```

VETTORI DI SINONIMI: ESEMPIO



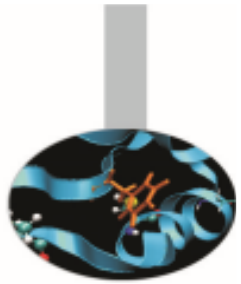
```
! Si definiscono i valori per matA
      DO ir = 1, Nda
         DO ic = 1, ir
            matA_rig(ir)%col(ic) = ir + ic
         END DO
      END DO
       DO ir = 1, Nda
         print*,"matA_rig(",ir,") = ",matA_rig(ir)%col(:)
      END DO
 END PROGRAM matrice_2
```

SINONIMI E PROCEDURE



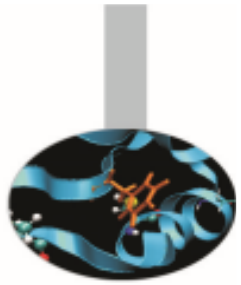
- I sinonimi possono essere passati in argomento alle procedure.
- In questo caso la memoria può essere associata e disassociata ai sinonimi anche **nella procedura**.
- Una procedura con argomenti dummy dichiarati `POINTER` o `TARGET` deve avere un'**interfaccia esplicita**.
- Ad una procedura con argomento sinonimo gli deve essere passato un sinonimo dello stesso tipo e rango.
- Un argomento sinonimo dummy non può avere l'attributo `INTENT`.
- Passare ad una procedura un sinonimo come argomento **non dichiarato** `POINTER` equivale a passare alla procedura il "bersaglio" del sinonimo.

SINONIMI E PROCEDURE



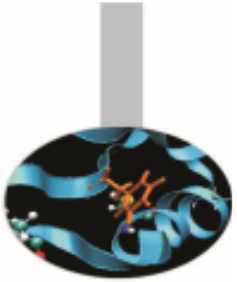
- Se in una procedura si dichiara che qualche argomento è `POINTER`, ciò significa che si intende utilizzare quell'argomento per associargli della memoria, che quindi sarà conservata in uscita.
- Come sempre, occorre avere cura che il tipo ed il rango degli oggetti passati ad una procedura siano compatibili con quanto dichiarato nella procedura stessa.
- Può essere sorprendente il fatto che un argomento sinonimo non può avere attributo `INTENT`, tuttavia occorre fare attenzione a cosa significherebbe questo: se un argomento `POINTER` ha attributo `INTENT(IN)` significa che non si può cambiare l'oggetto associato (o la memoria), oppure non si può cambiare il valore dell'oggetto bersaglio?

SINONIMI E PROCEDURE



- Il Fortran 2003 avrebbe risolto la questione in favore delle limitazioni nell'operazione di associazione. Perciò, per lo standard 2003, ad un argomento `POINTER` con `INTENT(IN)` non gli si può cambiare l'oggetto associato, né azzerarlo con `NULLIFY`, ma si può cambiare il valore all'oggetto bersaglio.

SINONIMI E PROCEDURE: ESEMPIO



```
PROGRAM Sinonimi
      INTERFACE
         SUBROUTINE sub2(b)
            REAL, DIMENSION(:,:), POINTER :: b
         END SUBROUTINE sub2
      END INTERFACE
      REAL, DIMENSION(:,:), POINTER :: p
      ALLOCATE (p(50, 50))
      CALL sub1(p)
      CALL sub2(p)
END PROGRAM Sinonimi
```

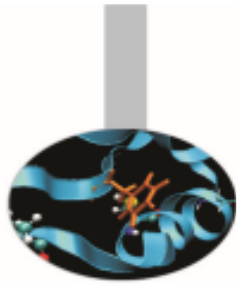
Nell'esempio illustrato se la procedura `SUB1` non ha un'interfaccia esplicita, il suo unico argomento sarà dichiarato nel seguente modo:

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

quindi come una matrice di forma presunta. Non può avere l'attributo `POINTER`, ma può avere un `INTENT` esplicito.

Viceversa l'argomento di `SUB2` può essere dichiarato come illustrato, ma non può avere un `INTENT`.

FUNZIONI SINONIMO

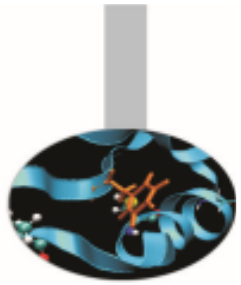


Il risultato di una funzione può essere un sinonimo.

E' utile definire una funzione di questo tipo se non si può prevederne l'occupazione di memoria.

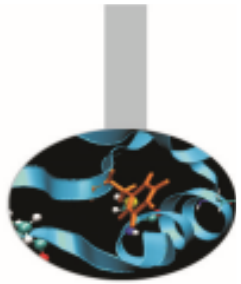
```
FUNCTION NumeriPrimi(max) RESULT(lista)
      IMPLICIT NONE
      INTEGER, INTENT(INOUT) :: max
      INTEGER, DIMENSION(:), POINTER :: lista

         .    .    .

      RETURN
END FUNCTION NumeriPrimi
```

FUNZIONI SINONIMO



L'interfaccia di una funzione sinonimo deve essere esplicita.

```
PROGRAM primi
      IMPLICIT NONE
      INTEGER :: n
      INTERFACE
         FUNCTION NumeriPrimi(max) RESULT(lista)
            IMPLICIT NONE
            INTEGER, INTENT(INOUT) :: max
            INTEGER, DIMENSION(:), POINTER :: lista
         END FUNCTION NumeriPrimi
      END INTERFACE

        . . .

END PROGRAM primi
```

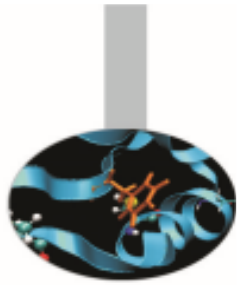


ESEMPIO: il crivello di Eratostene

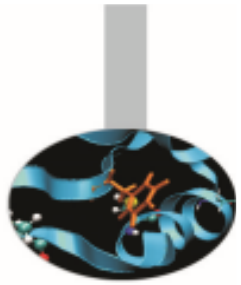
```
PROGRAM primi
    IMPLICIT NONE
    INTEGER :: n
    INTERFACE
       FUNCTION NumeriPrimi(max) RESULT(lista)
           IMPLICIT NONE
           INTEGER, INTENT(INOUT) :: max
           INTEGER, DIMENSION(:), POINTER :: lista
       END FUNCTION NumeriPrimi
    END INTERFACE

    DO
    PRINT *, "Digita un intero"
    READ *, n
    IF ( n .LE. 0 ) EXIT

    WRITE(*,*) NumeriPrimi(n)
    END DO
    STOP
END PROGRAM primi
```

ESEMPIO: il crivello di Eratostene

(continuazione)



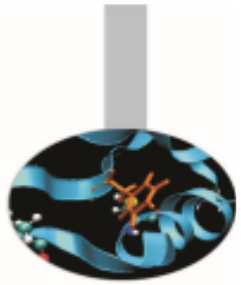
```
FUNCTION NumeriPrimi(max) RESULT(lista)
IMPLICIT NONE
INTEGER, INTENT(INOUT) :: max
INTEGER, DIMENSION(:), ALLOCATABLE :: primi
INTEGER, DIMENSION(:), POINTER :: lista
INTEGER :: i, n, quanti, st

ALLOCATE(primi(max), STAT=st)
IF ( st .NE. 0 ) THEN
   PRINT *, "errore allocazione PRIMI(",max,")"
ELSE
   primi = (/ (i, i = 1, max) /)
   quanti = 0
   DO i = 2, max
      IF ( primi(i) .GT. 0 ) THEN
         DO n = i + 1, max
            IF ( primi(n) .GT. 0 .AND. MOD(primi(n), i) .EQ. 0 ) THEN
               primi(n) = 0; quanti = quanti + 1
            END IF
         END DO
      END IF
   END DO
END IF
```



ESEMPIO: il crivello di Eratostene

(continuazione)

```
ALLOCATE(lista(max - quanti), STAT=st)
IF ( st .NE. 0 ) THEN
   PRINT *, "errore allocazione LISTA(",(max - quanti),")"
ELSE
   lista = 0
   i = 0
   DO n = 1, max
        IF ( primi(n) .GT. 0 ) THEN
                i = i + 1
                IF ( i .GT. (max - quanti) ) THEN
                        PRINT *, " I > ",(max - quanti)
                ELSE
                        lista(i) = primi(n)
                END IF
        END IF
   END DO
END IF
DEALLOCATE(primi)
RETURN
END FUNCTION NumeriPrimi
```



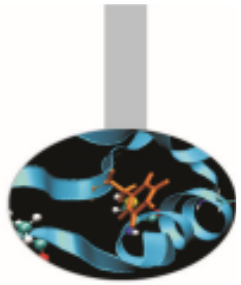
OPERATORI SINONIMO

(Fortran 2003)

Lo standard FORTRAN 2003 permette di associare un nuovo nome agli operatori personalizzati definiti in un modulo.

```
MODULE vettori
   INTERFACE OPERATOR (.turn.)
      MODULE PROCEDURE rigira
   END INTERFACE
   .   .   .
END MODULE vettori

PROGRAM giriamo
  USE vettori, OPERATOR(.gira.) => OPERATOR(.turn.)
  .   .   .
END PROGRAM giriamo
```

LISTE PUNTATE



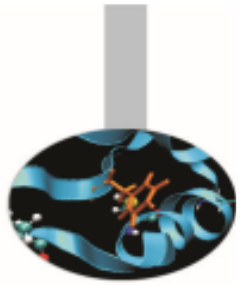
Una lista puntata è un insieme di oggetti, l'uno legato al successivo.

In generale ad una lista si possono aggiungere e togliere elementi in ordine casuale.

In Fortran è facile generare liste con tipi personalizzati contenenti sinonimi ad oggetti dello stesso tipo.

```
TYPE nodo
      TYPE(dati_reali) :: valore
      TYPE(nodo), POINTER :: succ, prec
END TYPE nodo
```

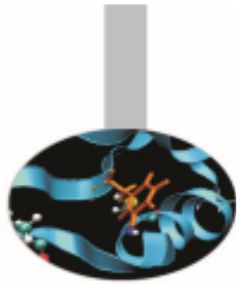
CATASTE



Una catasta è una lista con la particolarità che gli elementi si possono estrarre solo in ordine inverso all'ordine di immissione, ovvero dal più giovane al più vecchio.

```
TYPE elemento
      TYPE(dati_reali) :: valore
      TYPE(elemento), POINTER :: prec
END TYPE elemento

TYPE(elemento) :: corr, temp
```

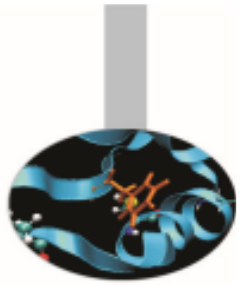
CODE



Viceversa una coda è una lista in cui gli elementi si possono estrarre in ordine di immissione, ovvero dal più vecchio al più giovane.

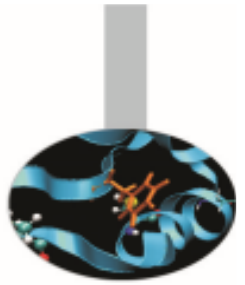
```
TYPE elemento
      INTEGER :: valore
      TYPE (elemento), POINTER :: prox
END TYPE elemento

TYPE (elemento), POINTER :: primo, ultimo, corr
```

Esempio di coda



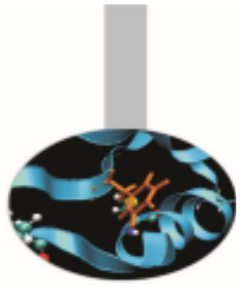
```
PROGRAM coda
IMPLICIT NONE
TYPE elemento
   INTEGER :: valore
   TYPE (elemento), POINTER :: prox
END TYPE elemento

TYPE (elemento), POINTER :: primo, ultimo, corr
INTEGER :: num, st, nr

NULLIFY(primo, ultimo, corr)
PRINT *, "Trasmetti una serie di interi, 0 per
   terminare"
```

Esempio di coda



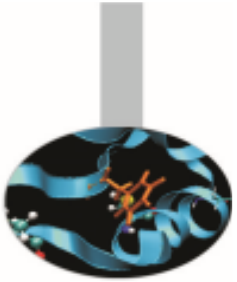
```
nr = 0
READ *, num

IF (num .GT. 0) THEN
   nr = nr + 1
   ALLOCATE(primo, STAT = st)
   IF (st > 0) THEN
       PRINT *, "Errore allocazione elemento N. ", nr
       STOP
   END IF
  primo%valore = num
   NULLIFY(primo%prox)
   ultimo => primo
```

Esempio di coda

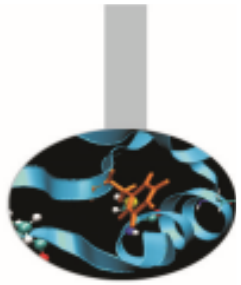


```
DO
    READ *, num
    IF (num == 0) EXIT
    ALLOCATE(corr, STAT = st)
    nr = nr + 1
    IF (st > 0) THEN
    PRINT *, "Errore allocazione elemento N. ", nr
    STOP
    END IF
    corr%valore = num
    NULLIFY(corr%prox)
    ultimo%prox => corr
    ultimo => corr
END DO
```

Esempio di coda



```
END IF  ! Num > 0


PRINT *, "Si presentano i numeri in ordine di
  immissione:"
corr => primo
DO
  IF (.NOT. ASSOCIATED(corr)) EXIT
  PRINT *, corr%valore
  corr => corr%prox
END DO
END PROGRAM coda
```



ESERCIZI

Realizzare un esempio di catasta