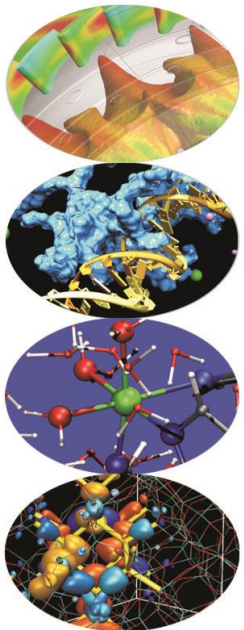


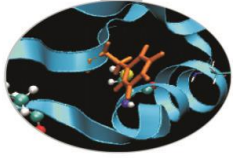
# Tipi derivati

## *Introduction to modern Fortran*

Paride Dagna, *CINECA*

*Maggio 2017*

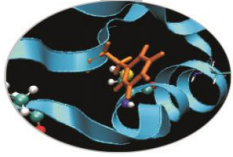




# Tipi derivati

## A COSA SERVONO:

- permettono di raggruppare in un unico oggetto dati eterogenei;
- possono essere usati con operatori nuovi e operatori intrinseci;
- la realizzazione di tipi derivati e la ridefinizione degli operatori costituiscono il punto di partenza per una programmazione orientata agli oggetti.



# Tipi predefiniti

Tipi predefiniti:

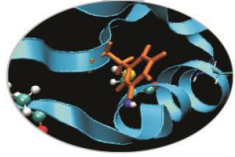
```
INTEGER :: i, j, k, ivett(10)
```

```
REAL(8) :: a(10,10), b(10,20)
```

```
COMPLEX(16) :: z, w, x(100,100)
```

```
CHARACTER :: intro = "La modalita' e': "
```

```
LOGICAL :: vero = .T., falso = .F., cond
```



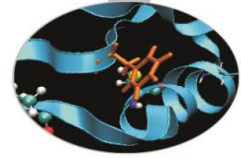
# Tipi derivati

Tipi derivati:

```
TYPE data
    INTEGER :: g, m, a
END TYPE data
```

Possono contenere diverse componenti di vario tipo;

```
TYPE nascita
    CHARACTER(132) :: luogo
    INTEGER :: g, m, a
END TYPE
```

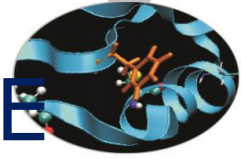


# Tipi derivati

Possono anche contenere oggetti di tipo personalizzato:

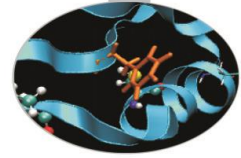
```
TYPE nascita  
    CHARACTER(132) :: luogo  
    TYPE(data) :: giorno  
END TYPE nascita
```

```
TYPE(nascita) :: a, b, c
```



# Tipi derivati - Istruzione SEQUENCE

- Garantisce l'esatto ordinamento in memoria delle entità che compongono il tipo derivato
- Se il tipo personalizzato è composto da altri tipi derivati ed uno di questi contiene l'istruzione `SEQUENCE`, è allora necessario che anche gli altri tipi derivati contengano l'istruzione `SEQUENCE`.
- Utile nel caso in cui occorra chiamare funzioni scritte in C/C++
- Per migliorare l'interoperabilità con il C/C++ dal fortran 2003 è stato introdotto il modulo intrinseco `ISO_C_BINDING` che non richiede più l'uso dell'istruzione `SEQUENCE`.



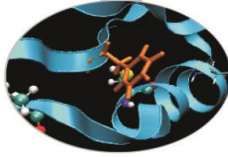
# Tipi derivati

## Inizializzazione:

```
TYPE nascita
  CHARACTER(132) :: luogo
  TYPE(data) :: giorno
END TYPE nascita

TYPE(nascita) :: a, b, c

a%luogo = "NewYork"
a%giorno%g = 16
a%giorno%m = 8
a%giorno%a = 1998
```



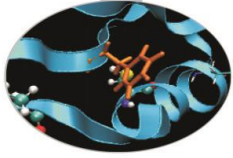
# Tipi derivati

## Assegnazione:

Tra gli operatori *predefiniti* solo l'assegnazione è applicabile ai tipi derivati:

```
TYPE data
    INTEGER :: g, m, a
END TYPE data
TYPE nascita
    CHARACTER(132) :: luogo
    TYPE(data) :: giorno
END TYPE nascita
TYPE(nascita) :: n
n = nascita("Citta' del Capo", data(29, 2, 2004))
```



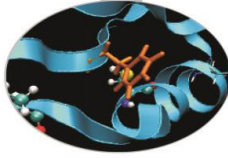


# Tipi derivati

## Stampa:

Il Fortran gestisce direttamente la presentazione dei dati relativi ai tipi derivati:

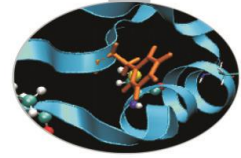
```
TYPE data
    INTEGER :: g, m, a
END TYPE data
TYPE nascita
    CHARACTER(132) :: luogo
    TYPE(data) :: giorno
END TYPE nascita
TYPE(nascita) :: n
n = nascita("Citta' del Capo",data(29,2,2004))
PRINT*,"Sono nato a ",n
```



# Costruttori (Fortran 2003)

Il Fortran 2003 estende la sintassi per la definizione dei tipi derivati.

In particolare non è più necessario specificare tutti i valori, poichè è possibile inizializzare il tipo derivato con valori di default.



# Costruttori (Fortran 2003)

Ad esempio, se il tipo derivato *punto* viene così definito:

```
TYPE punto
  REAL(8) :: x=0, y=0 ! inizializzazione
END TYPE punto
TYPE segmento
  TYPE(punto) :: a, b
END TYPE segmento
```

Si può inizializzare il segmento *s* passando esplicitamente il valore solo del primo membro:

```
s = segmento(punto(x, y))
```

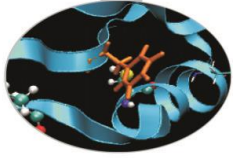
! Non è indispensabile passare esplicitamente il secondo punto, se è (0,0)



# Costruttori (Fortran 2003)

Il Fortran 2003 permette inoltre di definire, in un tipo derivato, componenti pubbliche (accessibili all'esterno del modulo in cui è definito) e componenti private (non accessibili al di fuori del modulo). Esempio:

```
MODULO Geom2D
  TYPE punto
    REAL(8) :: x=0, y=0 ! Inizializzazione
    INTEGER, PRIVATE :: nd = 2 ! Non accessibile fuori dal
                                ! modulo Geom2D
  END TYPE punto
  TYPE segmento
    TYPE(punto) :: a, b
  END TYPE segmento
END MODULE Geom2D
```



# Costruttori (Fortran 2003)

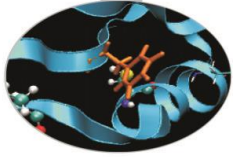
L'assegnazione di un valore al segmento  $s$  con il costruttore nella sintassi 2003 è possibile anche in questo caso; ovviamente non è possibile assegnare esplicitamente un valore alla componente privata  $nd$ :

```
USE geom2D
```

```
...
```

```
x = 1; y = 1; w = 2; z = 3;
```

```
s = segmento(punto(x,y), punto(w,z))
```



# Costruttori (Fortran 2003)

Inoltre, anche al costruttore è possibile passare i valori dei membri in ordine diverso da quanto definito nel tipo derivato, purché se ne indichi il nome:

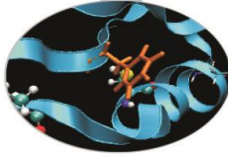
```
USE geom2D
```

```
...
```

```
x = 1; y = 1; w = 2; z = 3;
```

```
s = segmento(b = punto(y = w))
```

Nell'esempio al primo membro di segmento ( $a$ ) non viene assegnato un valore esplicitamente, perciò il valore di default (0,0) viene considerato; nel secondo membro ( $b$ ) viene assegnato esplicitamente il valore alla sola componente  $y$ .



# Esercizi

1. Dato il tipo nascita, come dagli esempi precedenti, si scriva un programma che, dopo aver definito il tipo “nascita” e aver inizializzato alcune variabili di tipo “nascita”, ne stampi i valori sia in modalità nativa (semplice write del tipo derivato), sia per mezzo di una funzione appositamente generata che ne stampi esplicitamente le componenti.
2. Scrivere un programma che definisca il tipo personalizzato “punto” e il tipo personalizzato “cerchio” (contenente il centro come “punto” ed avente anche la componente raggio), inizializzi alcune variabili di questi tipi e le stampi.