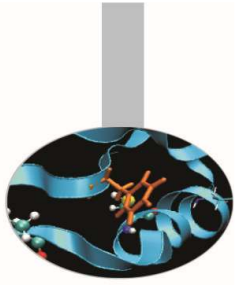
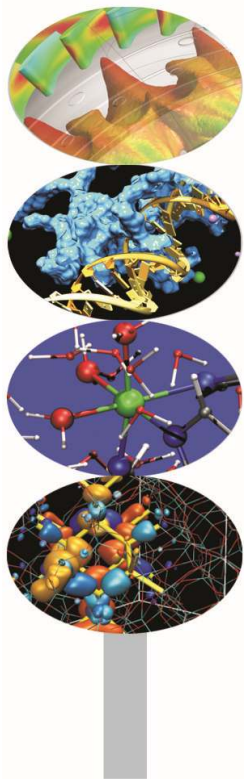
CINECA SCAI
SuperComputing Applications and Innovation



Approfondimenti sulle procedure



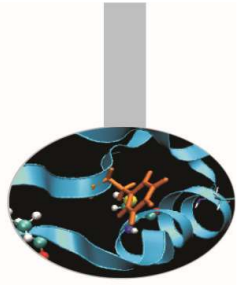
Introduction to modern Fortran

Maurizio Cremonesi, *CINECA*

Maggio 2016

CINECA

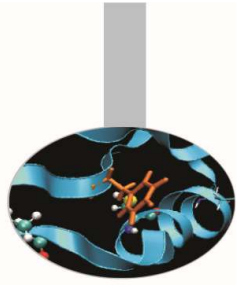
Procedure interne



Una procedura è detta interna, se è contenuta in un blocco (o sezione) del programma, ovvero nel *PROGRAM*, in un'altra *procedura* o in un *modulo*.

Sintatticamente una procedura interna è inserita dopo l'istruzione **CONTAINS**.

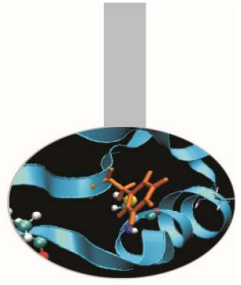
Procedure interne



Sintassi:

```
PROGRAM Esempio
   IMPLICIT NONE
         . . .
   STOP
   CONTAINS
      SUBROUTINE Procedura (...)
      IMPLICIT NONE
          . . .
      RETURN
   END SUBROUTINE Procedura
 END PROGRAM Esempio
```

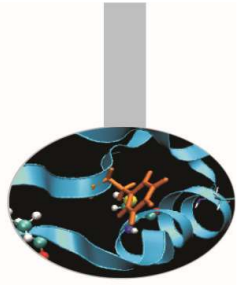
Procedure interne



Esempio 1: inizializzazione di tutti i contatori usati dal programma principale.

```
PROGRAM contatori
   INTEGER :: i, j, k, l, m, n
   CALL azzera
CONTAINS
   SUBROUTINE azzera
         i=0; j=0; k=0; l=0; m=0; n=0;
   END SUBROUTINE azzera
END PROGRAM contatori
```

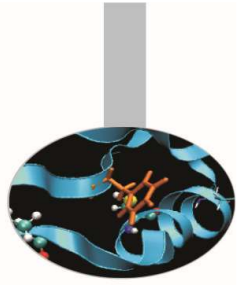
Procedure interne



Esempio 2: il nome *somma* può essere utilizzato per rappresentare 2 oggetti distinti: uno nel programma principale, l'altro nella funzione interna

```
PROGRAM somme
   INTEGER :: i, j, k, somma
   INTEGER, DIMENSION(100,100) :: matr
      .  .  .
   somma = 0
   DO i = 1, 100
      somma = (sommariga(i) * k) / (100 + 1 - i)
   END DO
      .  .  .
CONTAINS
```

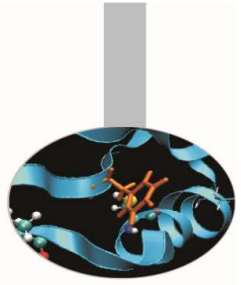
Procedure interne



… il nome *somma* viene utilizzato nella funzione interna per rappresentare un altro oggetto

```
   FUNCTION sommariga(r)
      INTEGER :: sommariga
      INTEGER, INTENT(IN) :: r
      INTEGER :: j, somma
      somma  = 0
      DO j = 1, 100
         somma = Matr(r,j) + somma
      END DO
      sommariga = somma
      RETURN
   END FUNCTION sommariga
END PROGRAM somme
```

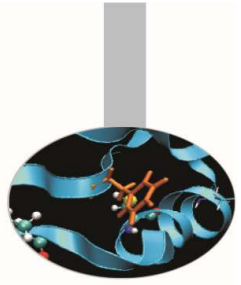
Intento



L'intento (INTENT in Fortran) serve a specificare se l'argomento passato ad una procedura è di solo lettura, solo scrittura o lettura/scrittura. *Questo permette al compilatore di controllare se una procedura è richiamata correttamente, ad esempio che il valore di un argomento di sola lettura non venga alterato nella procedura.*

L'utilizzo esplicito dell'intento porta in generale allo sviluppo di codici *più robusti*.

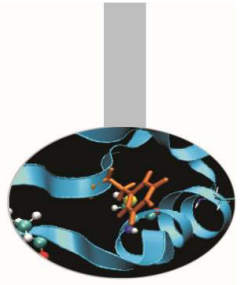
Intento



Esempio:

```
FUNCTION funzione(a,b,c)
   IMPLICIT NONE
   REAL :: funzione
   REAL, INTENT(IN) :: a, b
   REAL, INTENT(INOUT) :: c
END FUNCTION funzione
```

Intento



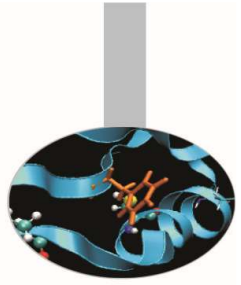
In breve:

IN viene usato per gli argomenti di sola lettura

OUT viene usato per le variabili calcolate all'interno della procedura

INOUT per gli argomenti forniti alla procedura e ricalcolati al suo interno

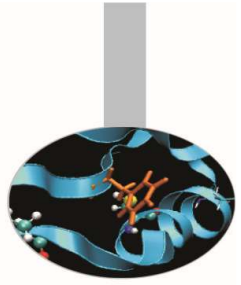
Intento



Esempio 3: la subroutine *converte* calcola in secondi un tempo fornito in ore, minuti e secondi.

```
INTERFACE
   SUBROUTINE converte (ore, minuti, &
                                secondi,tempo)
      INTEGER, INTENT (IN) &
               :: ore, minuti, secondi
      INTEGER, INTENT (OUT) :: tempo
   END SUBROUTINE converte
END INTERFACE
```

Procedure esterne e interfacce

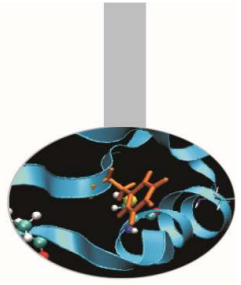


Una procedura è detta *esterna* se non è contenuta in alcun'altra sezione di codice.

Il corretto utilizzo delle procedure esterne viene controllato automaticamente dal compilatore se sono disponibili *interfacce esplicite*.

Come suggerisce il nome, un'interfaccia esplicita dev'essere inserita appositamente nel codice e messa a disposizione del compilatore mediante l'istruzione **`INTERFACE`**.

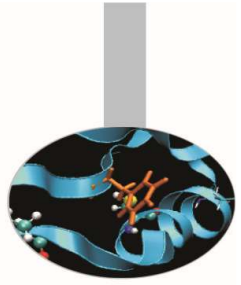
Procedure esterne e interfacce



I blocchi interfaccia hanno la sintassi:

```
INTERFACE
     FUNCTION funzione(a,b,c)
          IMPLICIT NONE
          REAL :: funzione
          REAL, INTENT(IN) :: a, b
          REAL, INTENT(OUT) :: c
     END FUNCTION funzione
 END INTERFACE
```

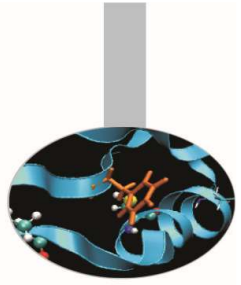
Procedure esterne e interfacce



Nell'interfaccia si devono specificare esattamente e solamente il tipo ed il nome della procedura e degli argomenti.

Le procedure interne (alle procedure o ai moduli) sono già visibili al compilatore, perciò non devono avere un'interfaccia esplicita

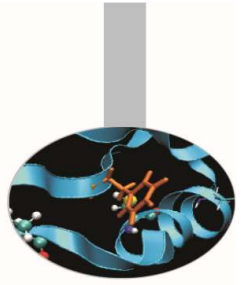
Procedure esterne e interfacce



I casi in cui è obbligatorio che una procedura abbia un'interfaccia esplicita verranno evidenziati nel seguito.

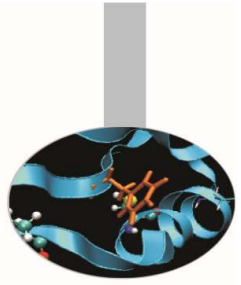
In generale fornire interfacce esplicite per tutte le procedure esterne è comunque vivamente consigliato per generare codici più robusti e chiaramente leggibili.

Procedure esterne e interfacce



Le interfacce esplicite permettono al compilatore di controllare la correttezza formale degli argomenti passati alle procedure evitando spesso errori banali ma potenzialmente pericolosi.

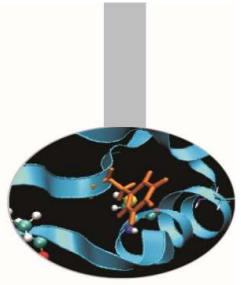
In mancanza di un'interfaccia esplicita, il compilatore genera un'interfaccia implicita, basandosi su come è fatta la chiamata alla procedura.



Procedure esterne e interfacce (Fortran 2003)

Una limitazione delle interfacce consiste nel fatto che, se esplicitate all'interno di un modulo, non hanno visibilità di variabili dichiarate come `PARAMETER` e di tipi derivati ivi definiti. In questi casi è necessario utilizzare l'istruzione `IMPORT`, introdotta dallo standard Fortran 2003.

Procedure esterne e interfacce (Fortran 2003)

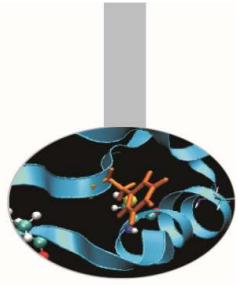


Ad esempio, volendo utilizzare la funzione esterna *incrementa* con l'interfaccia seguente

```
SUBROUTINE incrementa(r,v)
   USE dati
   IMPLICIT NONE
   INTEGER(tipo), INTENT(OUT) :: r
   INTEGER(tipo), INTENT(IN) :: v
END SUBROUTINE incrementa
```

in un programma che usa il modulo *dati*, dove *tipo* è definito, è necessario esplicitarne l'interfaccia, che richiede perciò l'uso dell'istruzione `IMPORT`.

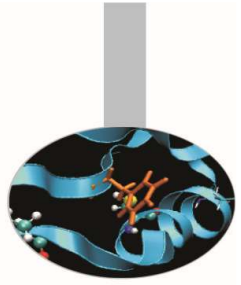
Procedure esterne e interfacce (Fortran 2003)



Perciò nel PROGRAM l'interfaccia dovrà essere riscritta così:

```
INTERFACE
    SUBROUTINE incrementa(r,v)
       IMPORT :: tipo
       IMPLICIT NONE
       INTEGER(tipo), INTENT(OUT) :: r
       INTEGER(tipo), INTENT(IN) :: v
    END SUBROUTINE incrementa
END INTERFACE
```

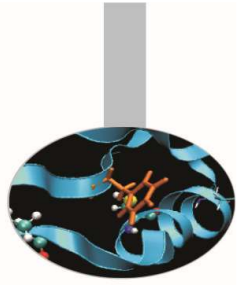
Procedure senza effetti collaterali



Se una procedura ha le caratteristiche seguenti:

1. non altera il valore degli argomenti, se è una funzione
2. non contiene entità con l'attributo SAVE
3. non contiene entità inizializzate (con DATA o nella dichiarazione)
4. non contiene argomenti senza INTENT
5. non altera il valore di entità globali (in COMMON o in MODULE)

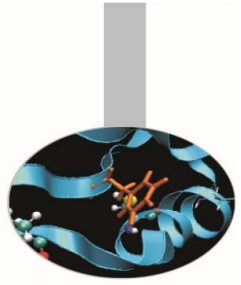
Procedure senza effetti collaterali



6. non usa sinonimi associati a entità globali o argomenti INTENT(IN)
7. non contiene istruzioni di lettura o scrittura su unità esterne
8. non contiene le istruzioni PAUSE o STOP
9. non richiama procedure che non siano dichiarate PURE

ad essa si può associare l'attributo *PURE*.

Procedure senza effetti collaterali

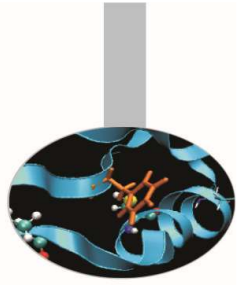


Se ad una procedura è associato l'attributo PURE significa che essa non ha effetti collaterali dannosi, perciò può essere utilizzata in particolari ambiti, ad esempio in costrutti parallelizzabili come FORALL.

Una FUNCTION dichiarata PURE ha argomenti con solo INTENT(IN) e non effettua operazioni che modifichino variabili globali.

Una SUBROUTINE dichiarata PURE ha le stesse limitazioni ma alcuni suoi argomenti possono avere INTENT(IN OUT).

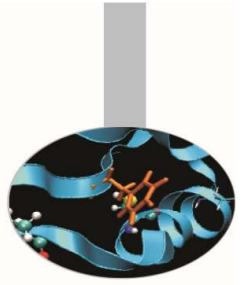
Procedure elementali



Una procedura si dice *elementale* se è ben definita per argomenti scalari, ma il risultato si conforma all'oggetto passato in argomento: diventa vettore per argomenti vettoriali, matrice per argomenti matriciali.

Il risultato è calcolato *elemento per elemento*.

Procedure elementali

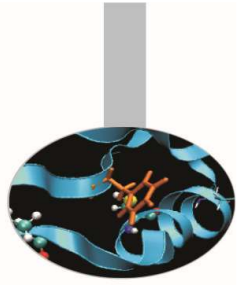


Le funzioni numeriche (ABS, INT, ...) e matematiche (SIN, SQRT, ...) sono elementali.

Esempio:

```
REAL(8) :: a, s
REAL(8), DIMENSION(120,32) :: vs, va
s = SIN(a)
vs = SIN(va)
```

Procedure elementali in Fortran 2003

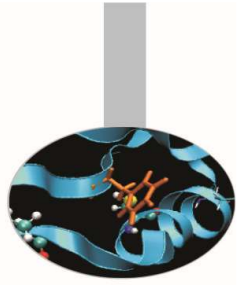


Il Fortran 2003 permette di realizzare facilmente procedure elementali, cosa *non possibile con gli standard precedenti.*

Per realizzare una procedura elementale è necessario che questa abbia le caratteristiche di una procedura PURE.

Inoltre è necessario che *tutti gli argomenti*, oltre al risultato della funzione, siano dichiarati come variabili **scalari**.

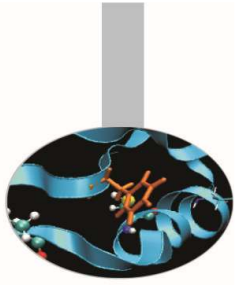
Procedure elementali in Fortran 2003



Quando ci sono i presupposti, basta esplicitare l'attributo ELEMENTAL nella dichiarazione della procedura per poterla utilizzare anche con oggetti non scalari. Si ricorda che una procedura elementale è considerata necessariamente PURE, perciò deve averne tutte le caratteristiche.

Per utilizzare una procedura non intrinseca come elementale è necessario che questa abbia *un'interfaccia esplicita*.

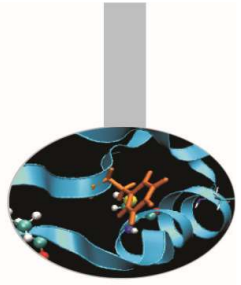
Procedure elementali in Fortran 2003



Esempio:

```
ELEMENTAL SUBROUTINE swap(a,b)
    IMPLICIT NONE
    INTEGER, INTENT(INOUT) :: a, b
    INTEGER :: k
    k = a
    a = b
    b = k
    RETURN
END SUBROUTINE swap
```

Ordine degli argomenti

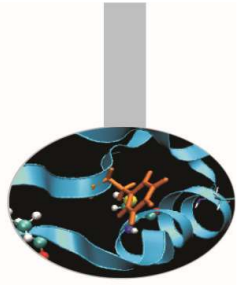


Già nel Fortran 77 esistono esempi sintattici di istruzioni utilizzabili con un ordine non prefissato degli argomenti:

```
OPEN(FILE='pippo.txt',UNIT=11, &
                              IOSTAT=ios)
READ(UNIT=11,FMT=100,END=99) n
```

Il Fortran 90 formalizza meglio e generalizza il concetto.

Ordine degli argomenti

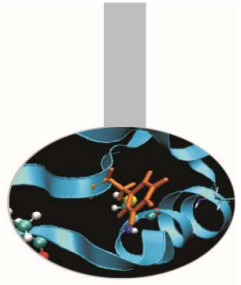


L'unica cosa da fare per chiamare una procedura con gli argomenti in ordine libero è indicare il nome degli argomenti quando si passano i valori.

I nomi degli argomenti sono quelli dichiarati nella procedura; perciò è **necessario conoscere l'interfaccia esplicita** della procedura.

L'unica limitazione consiste nel fatto che, a partire dal primo argomento passato indicandone il nome, gli argomenti successivi devono essere passati allo stesso modo.

Ordine degli argomenti



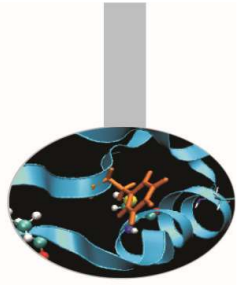
Esempio: la funzione *area* avente interfaccia

```
FUNCTION area(inizio,fine,tol)
   IMPLICIT NONE
   REAL :: area
   REAL, INTENT(IN) :: inizio, fine, tol
END FUNCTION area
```

può essere richiamata nei modi seguenti:

```
a = area(0.0, 100.0, 0.01)
b = area(inizio=0.0, tol=0.01, fine=100.0)
c = area(0.0, tol=0.01, fine=100.0)
```

Argomenti facoltativi

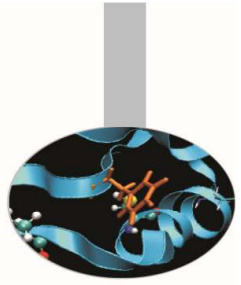


Con il Fortran si possono scrivere procedure che permettono di evitare di passare argomenti non indispensabili.

Esempi sintattici in FORTRAN 77 sono le funzioni di lettura/scrittura (I/O).

Per rendere *non obbligatorio* un argomento è necessario specificare la parola riservata OPTIONAL nella dichiarazione.

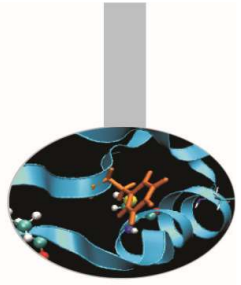
Argomenti facoltativi



L'uso degli argomenti facoltativi è necessariamente legato all'uso della funzione intrinseca `PRESENT()`.

Se un argomento è stato dichiarato facoltativo, è importante ricordare che, nel caso in cui non viene passato, è *come se l'argomento non esistesse*, quindi il suo nome non può essere utilizzato, neppure per dargli un valore di default.

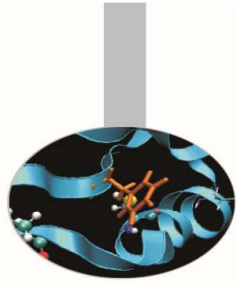
Argomenti facoltativi



Una soluzione pratica può essere l'uso di una variabile locale, con un valore di default, in cui è copiato il valore dell'argomento opzionale se presente.

Si illustra un esempio in cui la variabile `tol` ha l'attributo `OPTIONAL`, perciò viene utilizzata la funzione intrinseca `PRESENT()` per definire l'insieme di istruzioni da eseguire nel caso `tol` non compaia quando la funzione `area` viene richiamata.

Argomenti facoltativi



```
FUNCTION area(inizio,fine,tol)
      IMPLICIT NONE
      REAL :: area
      REAL, INTENT(IN) :: inizio, fine
      REAL, INTENT(IN), OPTIONAL :: tol

      IF ( PRESENT(tol) ) THEN
         .   .   .
      ELSE
         .   .   .
      END IF
         .   .   .
      RETURN
END FUNCTION area
```