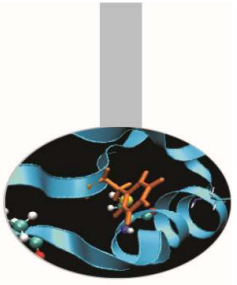
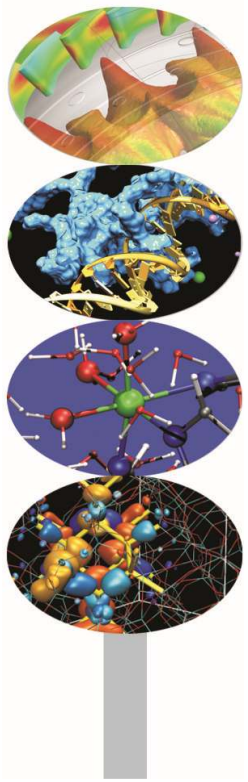
CINECA SCAI
SuperComputing Applications and Innovation



Approfondimenti sulle procedure



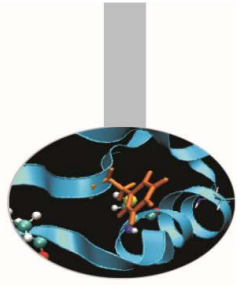
Introduction to modern Fortran

Maurizio Cremonesi, *CINECA*

Maggio 2017

CINECA

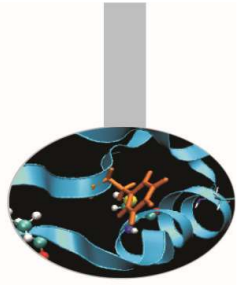
Procedure interne



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**.

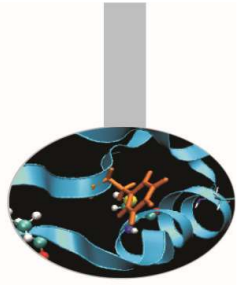
Procedure interne



Sintassi:

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

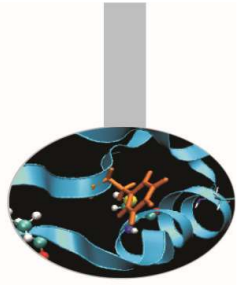
Procedure interne



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
```

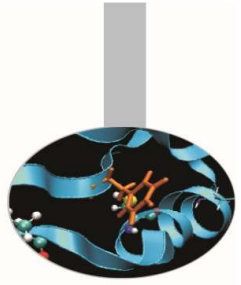
Procedure interne



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
```

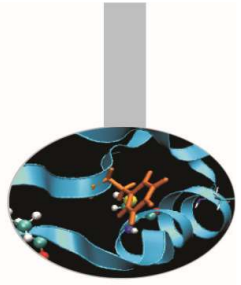
Procedure interne



… 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
```

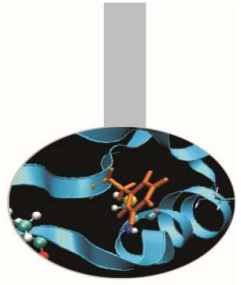
Intento



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*.

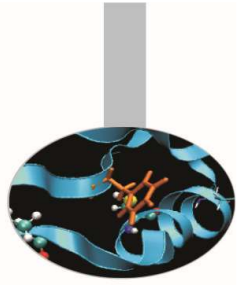
Intento



Esempio:

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

Intento



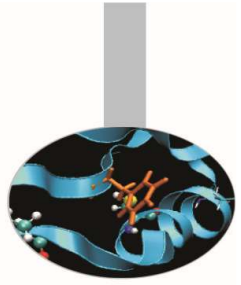
In breve:

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

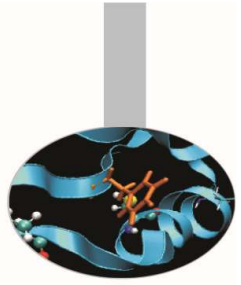
Intento



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

```
SUBROUTINE converte (ore, minuti, secondi, tempo)
    INTEGER, INTENT (IN) &
                    :: ore, minuti, secondi
    INTEGER, INTENT (OUT) :: tempo
    tempo = secondi + minuti*60 + ore*3600
    RETURN
END SUBROUTINE converte
```

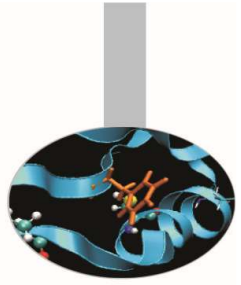
Intento



Esercizio 1:

Implementare il modulo *Base* contenente la funzione *SomPro*, per calcolare il risultato di a+b*c. Scrivere il programma *SomProSca* che richiama quest'ultima funzione.

Procedure esterne e interfacce

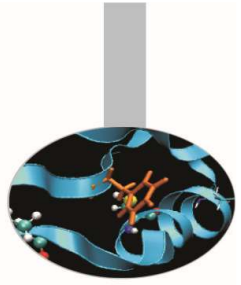


Una procedura è detta *esterna* se non è contenuta in alcun'altra sezione di codice.

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`**.

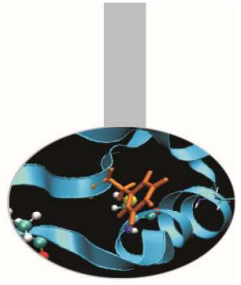
Procedure esterne e interfacce



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
```

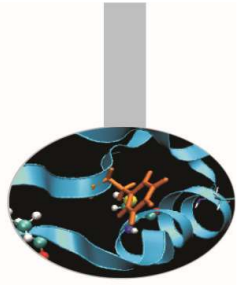
Procedure esterne e interfacce



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

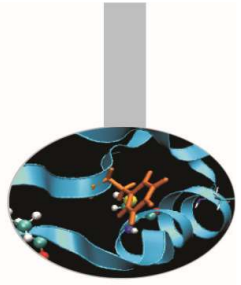
Procedure esterne e interfacce



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

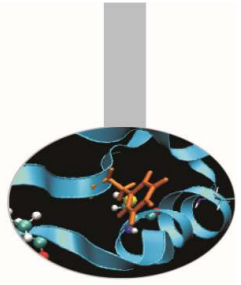
In generale fornire interfacce esplicite per tutte le procedure esterne è comunque vivamente consigliato per generare codici più robusti e chiaramente leggibili.

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.

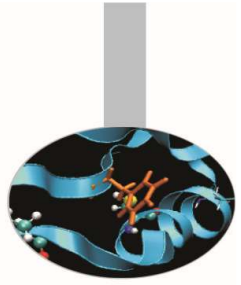
In mancanza di un'interfaccia esplicita, il compilatore genera un'interfaccia implicita, basandosi su come è fatta la chiamata alla procedura.



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)

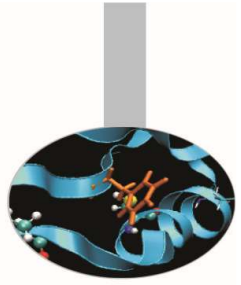


Ad esempio, volendo utilizzare la funzione esterna *incrementa* con l'interfaccia seguente

```
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`.

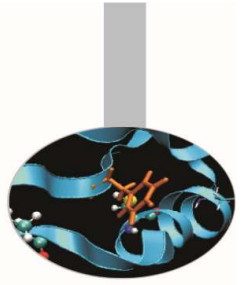
Procedure esterne e interfacce (Fortran 2003)



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
```

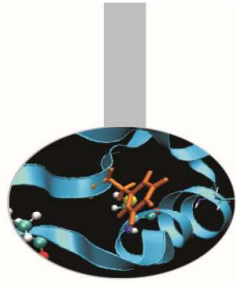
Procedure senza effetti collaterali



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)

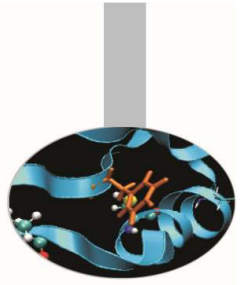
Procedure senza effetti collaterali



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

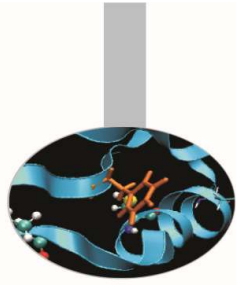


Se ad una procedura è associato l'attributo PURE significa che essa non ha effetti collaterali dannosi, perciò può essere utilizzata in particolari ambiti, ad esempio in costrutti parallelizzabili come FORALL.

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).

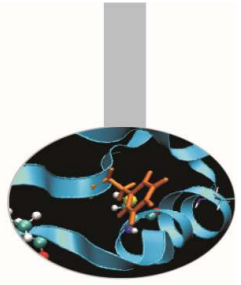
Procedure elementali



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

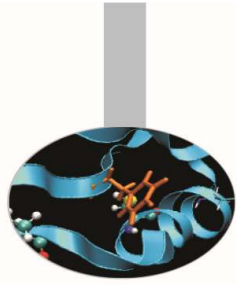


Le funzioni numeriche (ABS, INT, ...) e matematiche (SIN, SQRT, ...) sono elementali.

Esempio:

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

Procedure elementali in Fortran 2003

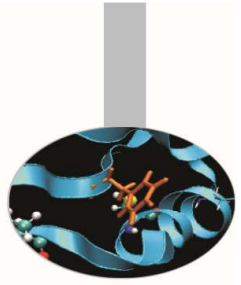


Il Fortran 2003 permette di realizzare facilmente procedure elementali, cosa *non possibile con gli standard precedenti.*

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**.

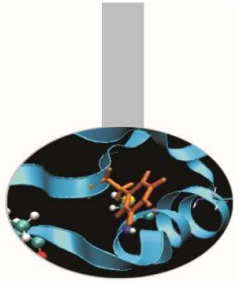
Procedure elementali in Fortran 2003



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*.

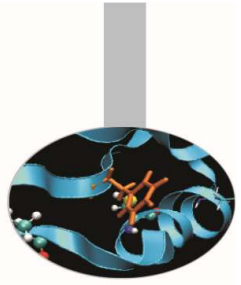
Procedure elementali in Fortran 2003



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
```

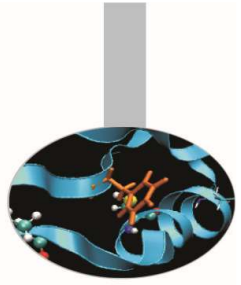
Procedure elementali in Fortran 2003



Esercizio 2:

Verificare che la funzione SomPro dell'esercizio 1 può essere utilizzata come funzione elementale.

Ordine degli argomenti

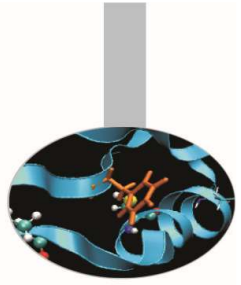


Già nel Fortran 77 esistono esempi sintattici di istruzioni utilizzabili con un ordine non prefissato degli argomenti:

```
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

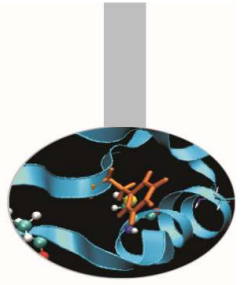


L'unica cosa da fare per chiamare una procedura con gli argomenti in ordine libero è indicare il nome degli argomenti quando si passano i valori.

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



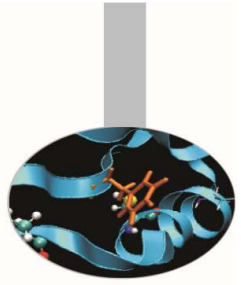
Esempio: la funzione *area* avente interfaccia

```
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

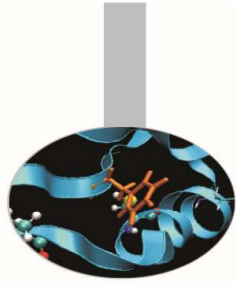


Con il Fortran si possono scrivere procedure che permettono di evitare di passare argomenti non indispensabili.

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.

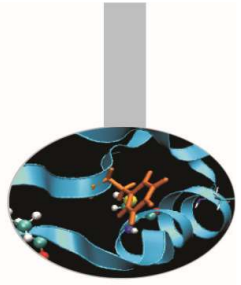
Argomenti facoltativi



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.

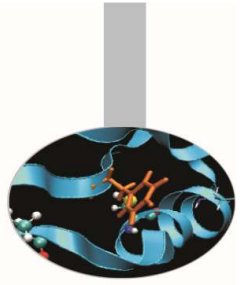
Argomenti facoltativi



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.

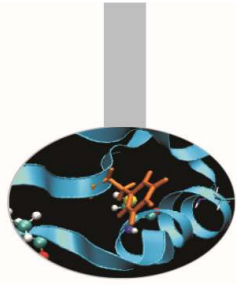
Argomenti facoltativi



```
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
```

Argomenti di tipo procedura

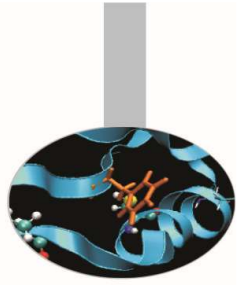


Anche in Fortran è possibile passare procedure in argomento. Le procedure così passate devono essere *esterne* o *contenute in un modulo*.

Non è possibile passare procedure interne.

Si raccomanda di fornire un'interfaccia esplicita alle procedure esterne passate in argomento. Le procedure interne ai moduli hanno già l'interfaccia esplicita.

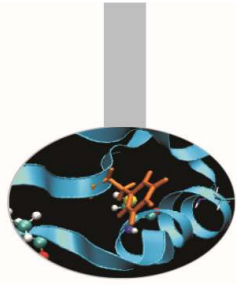
Argomenti di tipo procedura



Nell'esempio le funzioni esterne `somma` e `prodotto`, che definiscono le operazioni da eseguire, vengono passate come funzioni esterne; questo le rende suscettibili di essere riscritte all'occorrenza.

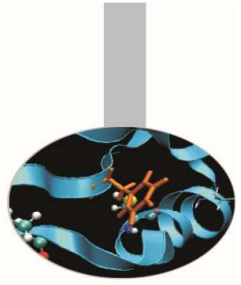
Esempio: FunzioniArgomento.f90

Argomenti allocabili (Fortran 2003)



A partire dal Fortran 2003 è possibile passare a procedure oggetti allocabili (`ALLOCATABLE`), nel senso che nella procedura si può associargli o rilasciare memoria, esattamente come si fa con i sinonimi in Fortran 90. Anche in questo caso è necessario che le procedure, se esterne, abbiano un'interfaccia esplicita.

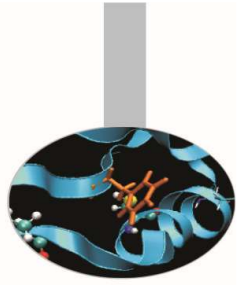
Argomenti allocabili (Fortran 2003)



Nell'esempio dimensiona.f90 la matrice `A` viene passata come oggetto allocabile alla procedura `inizializza` che le assegna memoria e l'azzera; le altre procedure trattano `A` come una matrice usuale.

Poiché `A` è allocabile, la memoria ad essa associata può essere rilasciata anche nel programma principale.

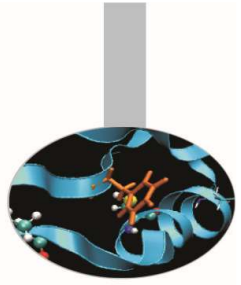
Funzioni vettoriali



È facile definire funzioni che ritornano un vettore di valori. È sufficiente dichiararlo, con l'attributo DIMENSION, quando si definisce la funzione:

```
FUNCTION combvet(a,b,n)
   IMPLICIT NONE
   INTEGER, INTENT(IN) :: n
   REAL, DIMENSION(n), INTENT(IN) :: a, b
   REAL, DIMENSION(n) :: combvet
   .   .   .
```

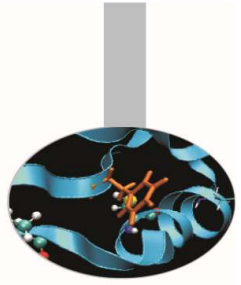
Funzioni vettoriali



Per utilizzare una funzione vettoriale è necessario che questa abbia un'interfaccia esplicita.

Il programmatore deve ricordarsi di scrivere un’interfaccia esplicita se la funzione è esterna.

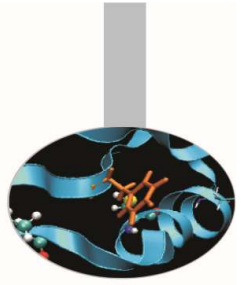
Funzioni vettoriali



Nell'esempio precedente per fare posto al risultato della funzione si è utilizzata l'allocazione automatica, che non permette di controllare preventivamente se la memoria necessaria è effettivamente disponibile.

Per questo motivo è preferibile dichiarare sinonimo (`POINTER`) il risultato della funzione, utilizzando così l'allocazione dinamica esplicita

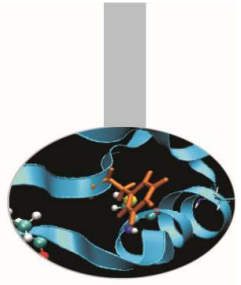
Funzioni vettoriali



Esempio combvetp in combina.f90:

```
   FUNCTION combvetp(a,b,n,f)
      IMPLICIT NONE
!     Allocazione dinamica con sinonimo
      INTEGER, INTENT(IN) :: n
      INTEGER, INTENT(OUT) :: f
      REAL(8), DIMENSION(n), INTENT(IN) :: a, b
      REAL(8), DIMENSION(:), POINTER :: combvetp
      .   .   .
```

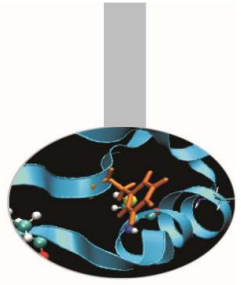
Funzioni vettoriali (Fortran 2003)



Il Fortran 2003 permette di dichiarare il risultato di una funzione con l'attributo ALLOCATABLE in luogo di POINTER.

Anche se i compilatori moderni sono molto ottimizzati rispetto a qualche anno fa, l'utilizzo di variabili ALLOCATABLE facilita la realizzazione di un eseguibile più efficiente; questo perché i POINTER sono oggetti complessi, che dovrebbero essere utilizzati solo quando è effettivamente necessario.

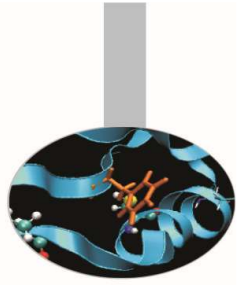
Funzioni vettoriali (Fortran 2003)



Esempio combveta in sommav.f90:

```
   FUNCTION combveta(a,b,n,f)
      IMPLICIT NONE
!     Allocazione dinamica stile Fortran 2003
      INTEGER, INTENT(IN) :: n
      INTEGER, INTENT(OUT) :: f
      REAL(8), DIMENSION(n), INTENT(IN) :: a, b
      REAL(8), DIMENSION(:), ALLOCATABLE :: combveta
          .    .    .
```

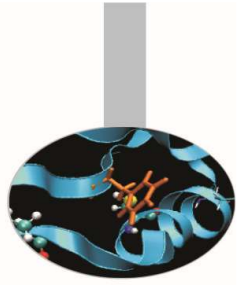
Funzioni ricorsive



Dal Fortran 90 la ricorsività entra a far parte dello standard, con una sintassi semplice e efficace.

Molti ma non tutti i compilatori Fortran 77 permettevano già di definire procedure ricorsive, ma la sintassi dipendeva dal compilatore utilizzato.

Funzioni ricorsive

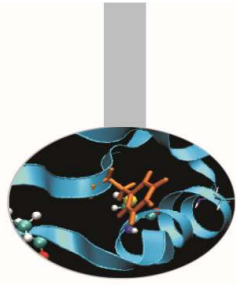


Dallo standard 90 per definire una procedura ricorsiva, è sufficiente dichiararlo anteponendo la parola riservata RECURSIVE davanti al nome.

Esempio:

```
RECURSIVE SUBROUTINE ricors(a,b)
   IMPLICIT NONE
   INTEGER, INTENT(IN) :: a
   INTEGER, INTENT(IN OUT) :: b
      .   .   .
```

Funzioni ricorsive



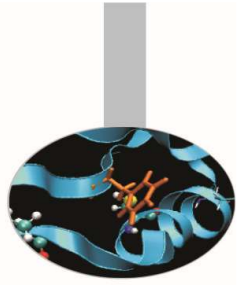
La ricorsività può essere diretta, nel caso la procedura ricorsiva chiami se stessa, o indiretta.

Per definire *funzioni* ricorsive è inoltre necessario aggiungere la clausola RESULT.

In questo modo è possibile indicare con un nome diverso il risultato della funzione.

Questo può essere fatto con qualunque funzione, ne rende più chiaro il codice, ma è obbligatorio solo per le funzioni che chiamano se stesse.

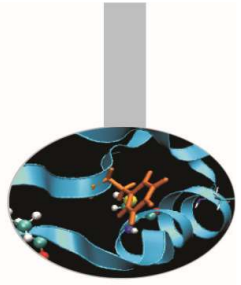
Funzioni ricorsive



Esempio uso di RESULT:

```
   FUNCTION combvet(a,b,n) RESULT(aob)
      IMPLICIT NONE
      INTEGER, INTENT(IN) :: n
      REAL, DIMENSION(n), INTENT(IN) :: a, b
      REAL, DIMENSION(n) :: aob
!
      DO i = 1, n
         IF ( a(i) > b(i) ) aob(i) = a(i)
         ELSE aob(i) = b(i)
      END DO
!
      RETURN
   END FUNCTION combvet
```

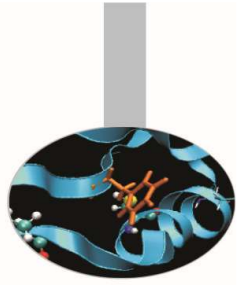
Funzioni ricorsive



Esempio di ricorsione diretta:

```
RECURSIVE FUNCTION fact(n) RESULT(r)
   IMPLICIT NONE
   INTEGER, INTENT(IN) :: n
   INTEGER :: r
    IF ( n == 1 ) THEN
     r = 1
    ELSE
     r = n * fact(n-1)
    END IF
   RETURN
END FUNCTION fact
```

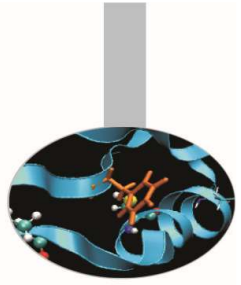
Funzioni ricorsive



Nel caso della ricorsione indiretta la funzione A chiama la funzione B la quale richiama A e così via.

In questo caso per definire le funzioni non è indispensabile utilizzare la parola riservata `RESULT`, ma è comunque consigliabile.

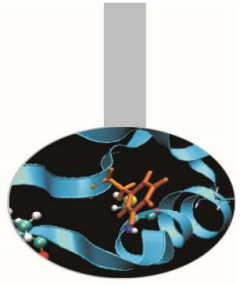
Funzioni ricorsive



Esempio di ricorsione indiretta (fattoriale.f90):

```
RECURSIVE FUNCTION fact(n) RESULT(f)
  .   .   .
   IF (n==1) THEN
     f = 1
   ELSE
     f = risul(n)
   END IF
 END FUNCTION fact
 RECURSIVE FUNCTION risul(n) RESULT(r)
   .   .   .
    r = n*fact(n-1)
 END FUNCTION risul
```

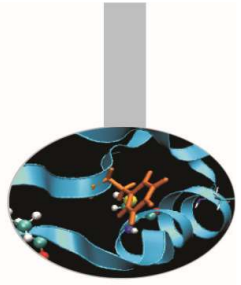
Funzioni ricorsive



Esercizio 3:

Scrivere una funzione, realizzata come ricorsiva, che ritorni il termine N-esimo della successione di Fibonacci, definita come: 1, 1, 2, 3, 5, … N = [N-1] + [N-2]

Nomi generici

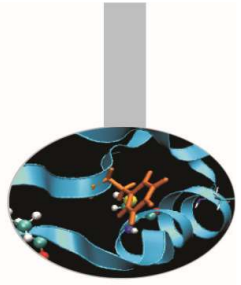


Il Fortran permette di definire un nome generico per richiamare procedure diverse.

Questa possibilità dev'essere usata per richiamare procedure sostanzialmente simili, che agiscono su argomenti di tipo diverso.

Usare uno stesso nome generico per richiamare procedure che fanno cose sostanzialmente diverse confonde la lettura del codice, ostacolandone la comprensione.

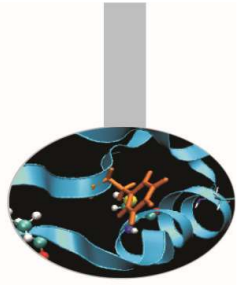
Nomi generici



È possibile associare un nome generico a qualunque insieme di procedure purché:

- siano tutte di tipo `SUBROUTINE` o tutte di tipo `FUNCTION`
- la lista degli argomenti sia diversa in numero (sconsigliato) e/o in tipo

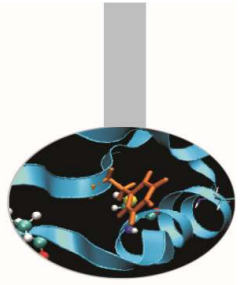
Nomi generici



Definire un nome generico equivale ad assegnare un nome comune all'insieme delle interfacce delle procedure da associarvi, se queste sono esterne:

```
INTERFACE Nome_Generico
   Interfaccia 1
      .   .   .
   Interfaccia N
END INTERFACE
```

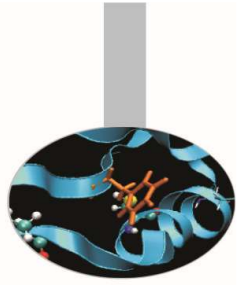
Nomi generici



Se invece le procedure sono interne ad un modulo, è sufficiente in pratica dare la lista dei loro nomi:

```
INTERFACE Nome_Generico
   MODULE PROCEDURE Procedura1
      .   .   .
   MODULE PROCEDURE ProceduraN
END INTERFACE
```

Nomi generici

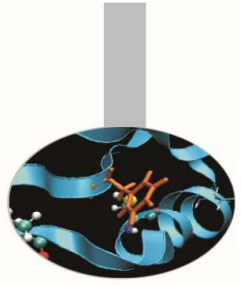


Nell'esempio alle funzioni *sompro_int* e *sompro_real* viene associato il nome generico *sompro*, utilizzabile quindi con argomenti sia `INTEGER` che `REAL`.

Esempio:

```
INTERFACE sompro
   MODULE PROCEDURE sompro_int
   MODULE PROCEDURE sompro_real
END INTERFACE
```

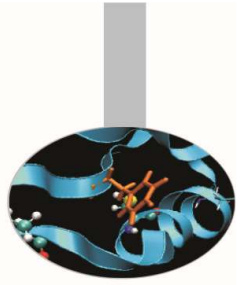
Nomi generici (Fortran 2003)



Il Fortran 2003 permette di semplificare la sintassi appena presentata, purché le procedure da associare al nome generico siano interne, oppure esterne, ma sia già presente la loro interfaccia esplicita.

La nuova sintassi non richiede più l'uso della parola "MODULE".

Nomi generici (Fortran 2003)

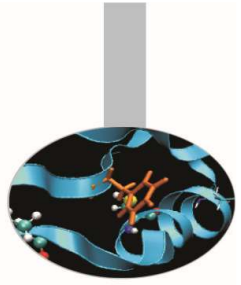


Esempio:

```
INTERFACE sompro
   PROCEDURE sompro_int
   PROCEDURE sompro_real
END INTERFACE
```

Sompro_int e sompro_real possono essere sia esterne che interne

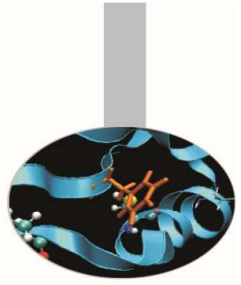
Nomi generici (Fortran 2003)



Anche se in Fortran 2003 non è più obbligatorio, è consigliabile continuare ad usare `MODULE` per le procedure interne al modulo, per maggiore chiarezza.

La possibilità di utilizzare procedure esterne al modulo associate a nomi generici ha implicazioni importanti; ad esempio permette di definire o migliorare il codice della procedura in un secondo momento, ovvero di utilizzare librerie esterne, anche di terze parti.

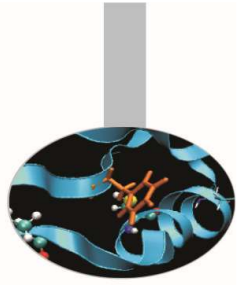
Nomi generici



Esercizio 4:

Scrivere 3 versioni diverse per argomenti di tipo INTEGER, REAL, REAL(8), della funzione SomPro dell'esercizio 2, richiamabili tutte con lo stesso nome generico

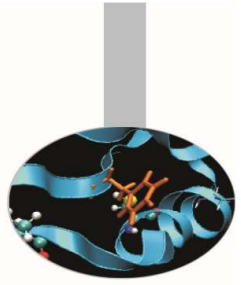
Librerie di funzioni



Nella cartella LifeGame sono presenti 4 files:

- IntData2Mpgm.f90 – definizione della subroutine IntData2Mpgm
- LifeStep.f90 – definizione della subroutine LifeStep
- Lifelib.f90 – interfacce delle due subroutine
- Life.f90 – definizione del program Life

Librerie di funzioni



Ci sono più modi per generare l'eseguibile di questo esempio:

1. con un unico comando di compilazione:

   gfortran -o life IntData2Mpgm.f90 LifeStep.f90 LifeLib.f90 Life.f90

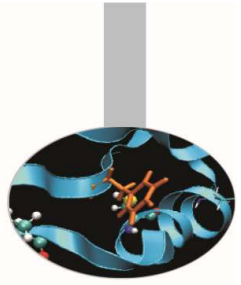
2. compilando le subroutine separatamente:

   gfortran –c IntData2Mpgm.f90

   gfortran –c LifeStep.f90

   gfortran –o life LifeLib.f90 Life.f90 IntData2Mpgm.o LifeStep.o

Librerie di funzioni



3. generando una libreria di subroutine:

```
gfortran –c IntData2Mpgm.f90
gfortran –c LifeStep.f90
ar r libLife.a IntData2Mpgm.o LifeStep.o
gfortran –o life LifeLib.f90 Life.f90 libLife.a
```

4. generando una shared library:

```
gfortran –c –fPIC IntData2Mpgm.f90
gfortran –c –fPIC LifeStep.f90
gfortran –shared –o libLife.so IntData2Mpgm.o LifeStep.o
gfortran –o life LifeLib.f90 Life.f90 –L . -lLife
```