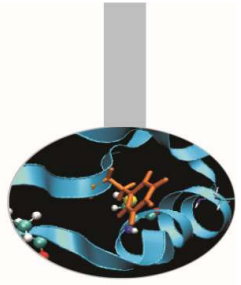
Argomenti di tipo procedura

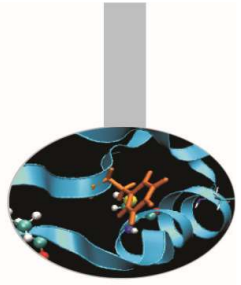


Anche in Fortran è possibile passare procedure in argomento. Le procedure così passate devono essere *esterne* o *contenute in un modulo*.

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.

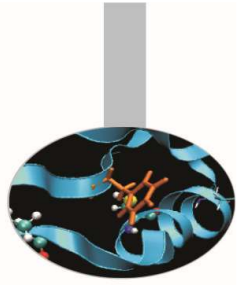
Argomenti di tipo procedura



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.

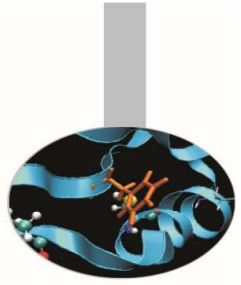
Esempio: FunzioniArgomento.f90

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.

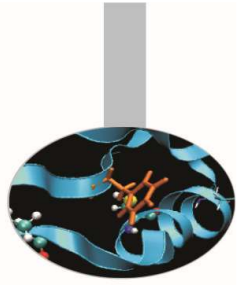
Argomenti allocabili (Fortran 2003)



Nell'esempio dimensiona.f90 la matrice `A` viene passata come oggetto allocabile alla procedura `inizializza` che le assegna memoria e valori numerici; 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.

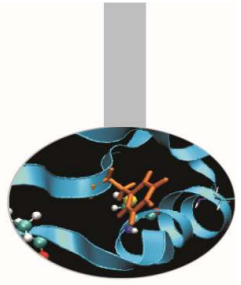
Funzioni vettoriali



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

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

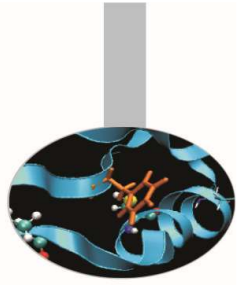
Funzioni vettoriali



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.

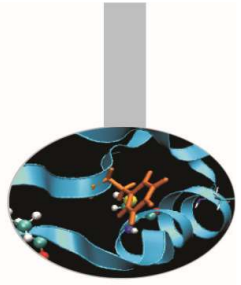
Funzioni vettoriali



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

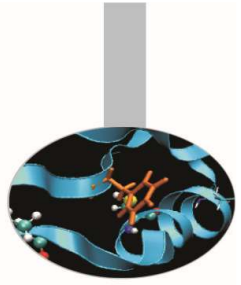
Funzioni vettoriali



Esempio addvetp in sommav.f90:

```
   FUNCTION addvetp(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 :: addvetp
      .   .   .
```

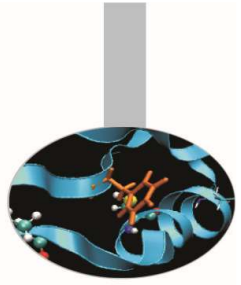
Funzioni vettoriali (Fortran 2003)



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.

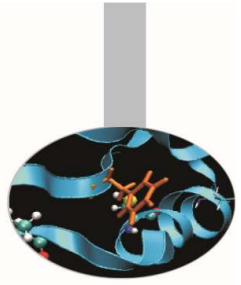
Funzioni vettoriali (Fortran 2003)



Esempio addveta in sommav.f90:

```
   FUNCTION addveta(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 :: addveta
         .   .   .
```

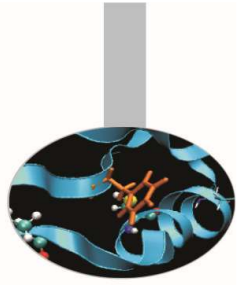
Funzioni ricorsive



Dal Fortran 90 la ricorsione 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

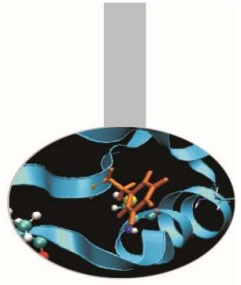


Dallo standard 90 per definire una procedura ricorsiva, è sufficiente dichiararlo anteponendo la parola riservata RECURSIVE davanti al nome.

Esempio:

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

Funzioni ricorsive



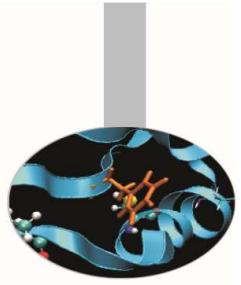
La ricorsione può essere diretta, nel caso la procedura ricorsiva chiami se stessa, o indiretta.

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.

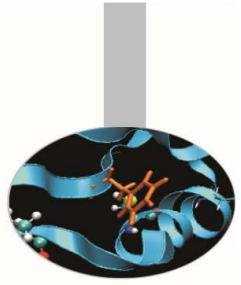
Funzioni ricorsive



Esempio:

```
    FUNCTION addvet(a,b,n) RESULT(somma)
        IMPLICIT NONE
        INTEGER, INTENT(IN) :: n
        REAL, DIMENSION(n), INTENT(IN) :: a, b
        REAL, DIMENSION(n) :: somma
!
        somma = a + b
!
        RETURN
    END FUNCTION addvet
```

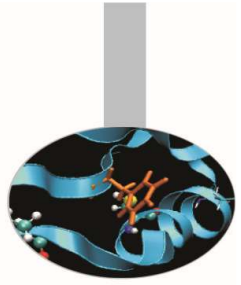
Funzioni ricorsive



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
```

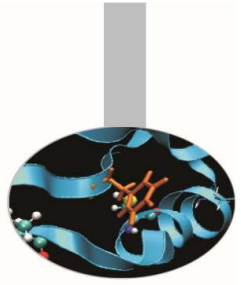
Funzioni ricorsive



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.

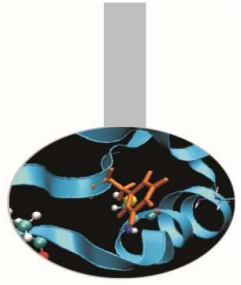
Funzioni ricorsive



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
```

Nomi generici

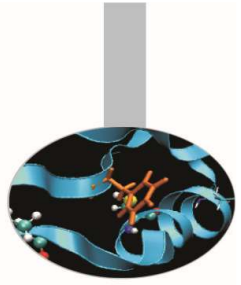


Il Fortran permette di definire un nome generico per richiamare procedure diverse.

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.

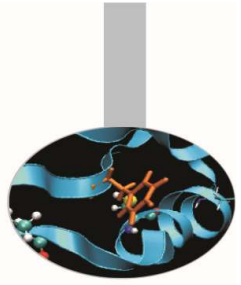
Nomi generici



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

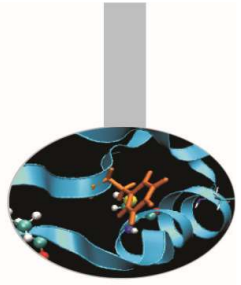
Nomi generici



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
```

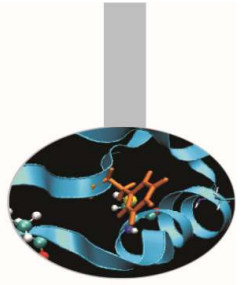
Nomi generici



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

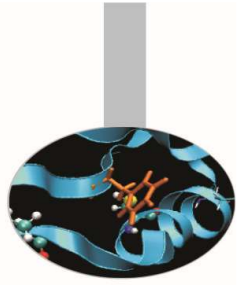


Nell'esempio alle funzioni *sompro_int* e *sompro_real* viene associato il nome generico *sompro*, utilizzabile quindi con argomenti sia `INTEGER` che `REAL`.

Esempio `Generico`:

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

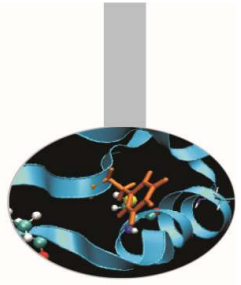
Nomi generici (Fortran 2003)



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)

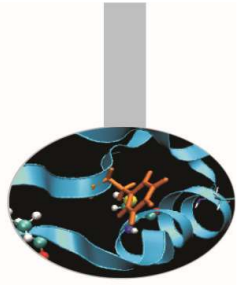


Esempio:

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

Sompro_int e sompro_real possono essere sia esterne che interne

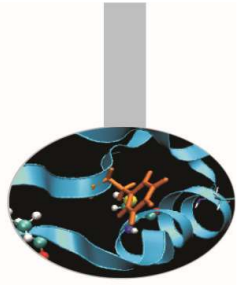
Nomi generici (Fortran 2003)



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.

Procedure intrinseche

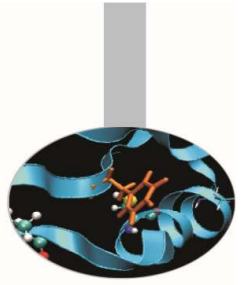


Allo scopo di rendere più chiaro il programma, è buona norma, laddove vengono utilizzate procedure intrinseche, segnalarle con la dichiarazione `INTRINSIC`.

La parola riservata INTRINSIC è seguita dalla lista dei nomi delle procedure intrinseche utilizzate.

Nel caso vengano usate procedure intrinseche esclusive del compilatore, non riconosciute dallo standard, la presenza della dichiarazione INTRINSIC permetterà di evidenziare l'eventuale non disponibilità della funzione.

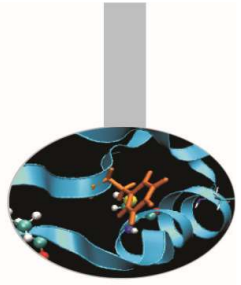
Procedure intrinseche



Esempio:

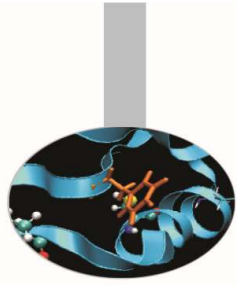
```
   IMPLICIT NONE
   REAL(8) :: x
   INTRINSIC :: SQRT
!
   PRINT*,"Trasmetti un numero"
   READ*,x
   PRINT*,"La radice quadrata di ",x,&
                  " e' ",SQRT(x)
```

Procedure intrinseche



`CALL DATE_AND_TIME([DATE] [,TIME] [,ZONE] [VALUES])`	ritorna data e ora
`CALL RANDOM_NUMBER(HARVEST)`	ritorna una sequenza casuale nell'intervallo [0,1).
`CALL RANDOM_SEED([SIZE] [,PUT] [,GET])`	inizializza il generatore di numeri casuali
`CALL SYSTEM_CLOCK([COUNT] [,COUNT_RATE] [COUNT_MAX])`	ritorna il numero di cicli dell'orologio di sistema
`CALL CPU_TIME(t1)`	ritorna il tempo di CPU in secondi (Fortran 95)

Procedure intrinseche



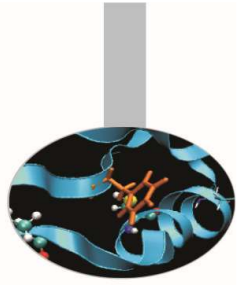
DATE_AND_TIME

Ritorna data e ora dell'istante in cui viene richiamata.

I tempi sono ritornati sia come stringa di caratteri, sia come numeri.

DATE_AND_TIME è una procedura abbastanza complessa, non adatta a misurare tempi di calcolo piccoli; a questo scopo è meglio usare la SYSTEM_CLOCK.

Procedure intrinseche



Sintassi :

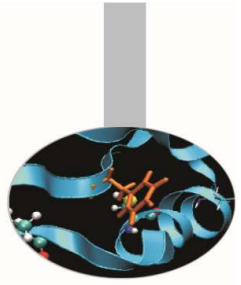
```
CHARACTER :: data*8, ora*10, zona*5
CALL DATE_AND_TIME(DATE=data,TIME=ora &
                   ZONE=zona)
```

`data`: giorno nella forma stringa di caratteri "YYYYMMGG"

`ora`: ora nella forma stringa di caratteri "HHMMSS.mmm"

`zona`: differenza rispetto all'ora di Greenwich nella forma stringa di caratteri "+HHMM"

Procedure intrinseche



Sintassi :

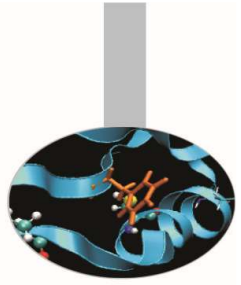
```
INTEGER, DIMENSION(8) :: tempo
CALL DATE_AND_TIME(VALUES=tempo)
```

`tempo(1:3)` : anno, mese, giorno

`tempo(4)` : differenza in minuti rispetto all'ora di Greenwich

`tempo(5:8)` : ora, minuti, secondi, millisecondi

Procedure intrinseche

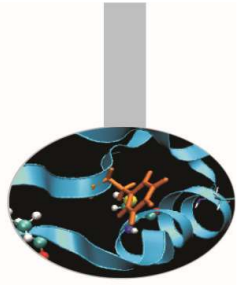


RANDOM_NUMBER

Questa subroutine ritorna una sequenza casuale di numeri, di distribuzione uniforme nell'intervallo [0,1).

La procedura RANDOM_NUMBER è elementale, perciò il suo valore prende la forma dell'oggetto passato in argomento, ovvero se viene passato uno scalare, il risultato è un singolo valore scalare, se vengono passati un vettore o una matrice il risultato è vettore o matrice.

Procedure intrinseche



La sequenza ritornata da RANDOM_NUMBER è pseudo-casuale, ovvero i valori sono calcolati con un algoritmo.

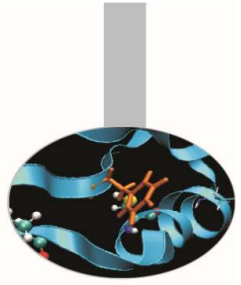
In particolare la sequenza dei valori dipende da un parametro chiamato seme (seed).

Se da un lato questo può rappresentare un problema per certe applicazioni, ha invece il pregio della ripetibilità, garantita sotto precise condizioni.

Algoritmi che si basano sul calcolo di numeri casuali possono richiedere l'utilizzo di librerie più sofisticate, ad esempio PRNG:

http://sprng.cs.fsu.edu/sprng.html

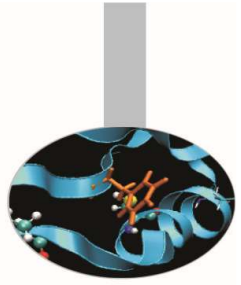
Procedure intrinseche



Esempio:

```
REAL :: casov
CALL RANDOM_NUMBER(casov)
PRINT*,casov
```

Procedure intrinseche

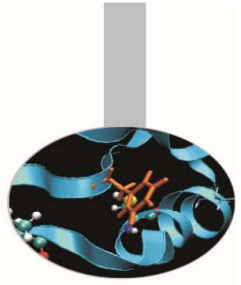


RANDOM_SEED

Questa subroutine ritorna informazioni sul parametro *seed*, che determina la sequenza dei valori prodotti dalla RANDOM_NUMBER.

Essa permette di riassegnare il parametro ogni volta che si vuole ripetere la sequenza.

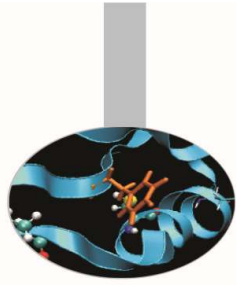
Procedure intrinseche



Uso della RANDOM_SEED:

1. si richiede di avere il valore di *SIZE*, necessario perché il seme è un vettore, la cui lunghezza dipende dal compilatore e dal calcolatore utilizzati
2. sulla base del valore ritornato da *SIZE* si dimensiona opportunamente il vettore che contiene il seme
3. con *GET* si memorizza il valore del seme in un apposito vettore
4. ogni volta che è necessario la sequenza venga ripetuta, si ristabilisce il valore del seme con *PUT*

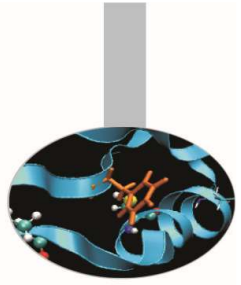
Procedure intrinseche



Esempio (RANDOM_SEED ammette 1 solo argomento alla volta):

```
   INTEGER :: s
   INTEGER, DIMENSION(LEN) :: mioseme, seme
   CALL RANDOM_SEED(SIZE=s)
!     s: output – lunghezza del vettore
   CALL RANDOM_SEED(GET=seme)
!     seme: output - vettore contenente il seme usato
                     dal generatore
   CALL RANDOM_SEED(PUT=mioseme)
!     mioseme: input - vettore contenente un seme
                    alternativo, o quello originale
```

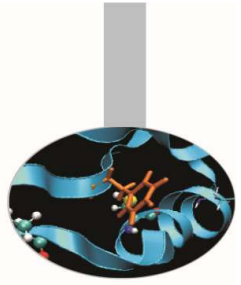
Procedure intrinseche



SYSTEM_CLOCK

In ogni calcolatore le operazioni sono sincronizzate con un orologio i cui battiti sono memorizzati in un apposito registro e da lì possono essere letti utilizzando questa subroutine.
L'operazione impegna poco il calcolatore, quindi la *SYSTEM_CLOCK* è adatta a misurare il tempo di calcolo di porzioni di codice anche abbastanza brevi.

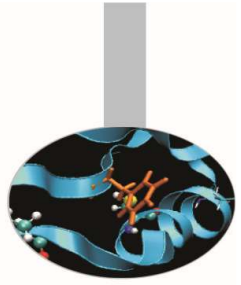
Procedure intrinseche



Esempio:

```
INTEGER :: cicli0, cicli1
CALL SYSTEM_CLOCK(COUNT=cicli0)
        .   istruzioni .
CALL SYSTEM_CLOCK(COUNT=cicli1)
PRINT*,"Battiti misurati: ", &
                        (cicli1-cicli0)
```

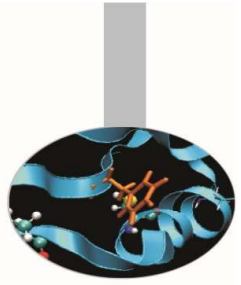
Procedure intrinseche



Una complicazione da tener presente è che il registro ha uno spazio limitato per conservare il numero di battiti dell'orologio, perciò superato un massimo, il valore si riazzera:

```
CALL SYSTEM_CLOCK(COUNT=cicli0, &
                  COUNT_MAX=cicli_massimi)
      .    .    .
CALL SYSTEM_CLOCK(COUNT=cicli1)
IF (cicli1 < cicli0) cicli1 = &
                      cicli1 + cicli_massimi
```

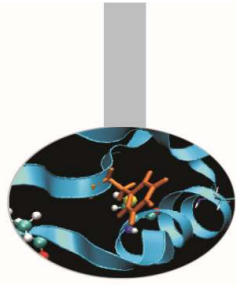
Procedure intrinseche



Se è richiesta una misura temporale più familiare, con l'argomento *COUNT_RATE* è possibile sapere quanti battiti di orologio vengono fatti ogni secondo e calcolare quindi il tempo in secondi:

```
CALL SYSTEM_CLOCK(&
                  COUNT_RATE=cicli_al_secondo)
```

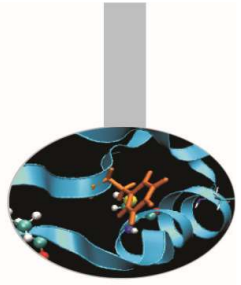
Procedure intrinseche



Esempio:

```
CALL SYSTEM_CLOCK(COUNT=cicli0, &
         COUNT_RATE=cicli_al_secondo, &
         COUNT_MAX=cicli_massimi)
       .    .    .
CALL SYSTEM_CLOCK(COUNT=cicli1)
If ( cicli1 < cicli0 ) &
            cicli1 = cicli1 + cicli_massimi
cicli = (cicli1-cicli0)
frequenza = cicli_al_secondo
PRINT*,"Secondi richiesti dal codice: ", &
          cicli/frequenza
```

Procedure intrinseche

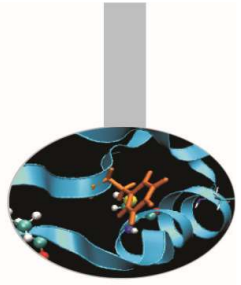


CPU_TIME

Questa subroutine, introdotta dal Fortran 95, ritorna il tempo di CPU in secondi:

```
      REAL: cput, cput0, cput1
         .    .    .
      CALL CPU_TIME(cput0)
         .    .    .
!     Istruzioni da misurare
         .    .    .
      CALL CPU_TIME(cput1)
      cput = cput1 - cput0
```

Procedure intrinseche (Fortran 2003)

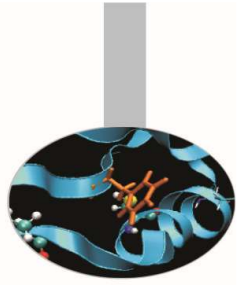


Il Fortran 2003 introduce alcune procedure e costanti utili per standardizzare i rapporti con l'ambiente operativo.

Col Fortran 2003 compare il concetto di modulo intrinseco.

Uno di questi moduli si chiama *ISO_FORTRAN_ENV* e contiene tra l'altro alcune funzioni di cui effettivamente si sentiva la mancanza in Fortran.

Procedure intrinseche (Fortran 2003)



```
COMMAND_ARGUMENT_COUNT()
```

ritorna il numero dei parametri passati al programma

```
CALL GET_COMMAND(comando, lunghezza, stato)
```

ritorna la riga di comando usata per eseguire il programma

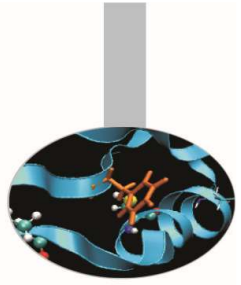
```
CALL GET_COMMAND_ARGUMENT(number, &
                          value, length, status)
```

ritorna il parametro dato a riga di comando, o il nome dell'eseguibile

```
CALL GET_ENVIRONMENT_VARIABLE(name, value,&
                              length, status)
```

ritorna il valore di un parametro di sistema

Procedure intrinseche (Fortran 2003)



Esempio *parametri:*

```
   quanti = COMMAND_ARGUMENT_COUNT()
   IF ( quanti < 1 ) THEN
      PRINT*,"Modalita' di esecuzione: &
                  & fattoriale <n>"
      STOP
   END IF
!  . . . altrimenti si stampa la riga di
!        comando utilizzata
   CALL GET_COMMAND(rg, LENGTH=lun, STATUS=st)
   IF ( st <= 0 ) THEN
      PRINT*,"Hai trasmesso: ",rg(1:lun)
   END IF
```