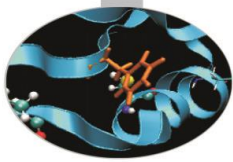
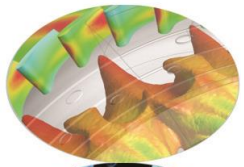
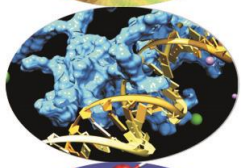
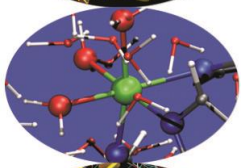
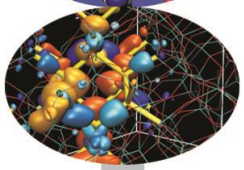
CINECA SCAI
SuperComputing Applications and Innovation



Introduzione alle Procedure

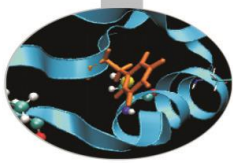





Introduction to modern Fortran

Paolo Ramieri, *CINECA*

Maggio 2017

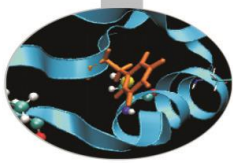
CINECA



Organizzazione del programma

Schema di un'unità di programma Fortran 90

```
PROGRAM, FUNCTION, SUBROUTINE, MODULE nome
   USE (module)
   IMPLICIT NONE
   Dichiarazioni
   istruzioni eseguibili
CONTAINS
   procedure interne
END (PROGRAM, FUNCTION, SUBROUTINE, MODULE nome)
```

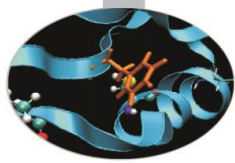


Sottoprogrammi

Il Fortran prevede due tipi di sottoprogrammi:

`FUNCTION` = restituiscono **un solo valore** in funzione di uno o più parametri. La chiamata avviene tramite un'istruzione di assegnazione.

`SUBROUTINE` = possono restituire più valori in funzione di più parametri. La chiamata avviene tramite l'istruzione `CALL`.



FUNCTION

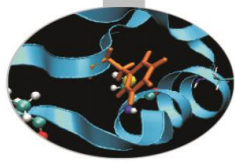
Sintassi per la FUNCTION:

```
tipo FUNCTION nome (lista-argomenti)
     dichiarazioni locali
     istruzioni
     nome = risultato
END FUNCTION nome
```

Chiamata della FUNCTION :

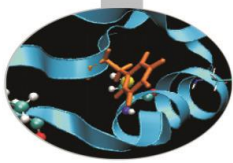
```
PROGRAM principale
  ...
  variabile = nome(lista-argomenti)
  ...
END PROGRAM principale
```

FUNCTION



Esempio:

```
REAL FUNCTION somma(a,b)
   REAL :: a, b
   somma = a + b
END FUNCTION somma

PROGRAM principale
   ...
   r = somma(c,d)
   ...
END PROGRAM principale
```



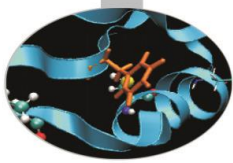
SUBROUTINE

Sintassi per la SUBROUTINE:

```
SUBROUTINE nome (lista-argomenti)
     dichiarazioni locali
     istruzioni
END SUBROUTINE nome
```

Chiamata della SUBROUTINE :

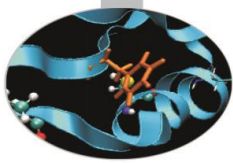
```
PROGRAM principale
   ...
   CALL nome(lista-argomenti)
   ...
END PROGRAM principale
```



SUBROUTINE

Esempio:

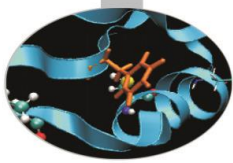
```
SUBROUTINE somma(a,b,c)
   REAL :: a, b, c
   c = a + b
END SUBROUTINE somma

PROGRAM principale
   ...
   CALL somma(c,d,e)
   ...
END PROGRAM principale
```



SUBROUTINE

Può essere utile stabilire più punti di uscita dalla subroutine, per esempio in funzione di un risultato ottenuto al suo interno. Per uscire dalla subroutine e tornare al programma chiamante si usa perciò l'istruzione `RETURN`, che può essere presente anche più volte.

Uscendo dalla subroutine, il valore delle variabili locali viene perso. Se si prevede di usare ancora tali valori alla successiva invocazione della subroutine, si possono preservare con il parametro `SAVE` in fase di dichiarazione.

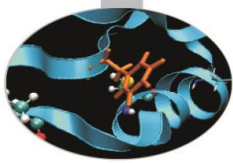


Moduli

I moduli sono uno strumento per:
- definire i dati e le strutture;
- definire tutte le funzioni operanti sui suddetti dati e strutture;
- definire altre procedure utilizzate dal programma.

Possono svolgere il ruolo che in C++ hanno le classi.

L'utilizzo del modulo permette di rendere dati e funzioni accessibili al resto del programma ma consente anche di proteggere i dati ed il codice che non necessitano di manipolazioni esterne.



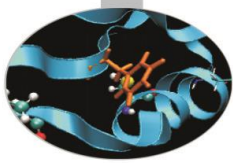
Moduli

Il programma principale e qualunque procedura o modulo possono accedere a ciò che è definito in un modulo con la sintassi:

```
USE nome_modulo
```

da usare prima di ogni altra istruzione, compresa `IMPLICIT NONE`.

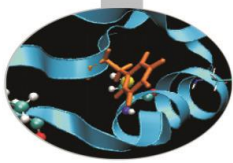
Restrizione: un modulo non può fare la "USE" di se stesso.



Moduli

Un modulo è definito dalla seguente sintassi:

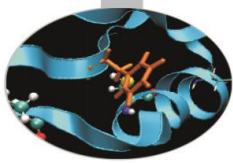
```
MODULE name
      dichiarazioni
      INTERFACE blocks
CONTAINS
      procedure interne del modulo
END MODULE name
```



Moduli - Esempio

```
MODULE Globale
     IMPLICIT NONE
     REAL :: r, s
END MODULE Globale


REAL FUNCTION Calcola()
     USE Globale
     IMPLICIT NONE
     READ(*,*) r, s
     CALCOLA = r * s
     RETURN
END FUNCTION Calcola
```

```
PROGRAM Principale
     IMPLICIT NONE
     REAL c
     c = CALCOLA()
     write (*,*) "il risultato e' ", c
END PROGRAM Principale
```



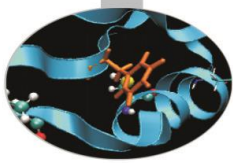
Moduli

Quando in un modulo vengono definite molte procedure, esiste la possibilità di accedere solo ad alcune di queste, specificando l'attributo `ONLY`:

```
USE module-name [, ONLY: { procedure-names } ]
```

I moduli possono contenere:

- entità pubbliche (accessibili da qualunque procedura che utilizza il modulo);
- entità private (visibili solo all'interno del modulo). Le entità private sono dichiarate così usando la parola riservata `PRIVATE`.

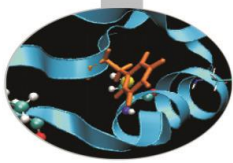


Visibilità

Un programma Fortran è strutturalmente diviso in regioni o aree diverse (*scope* in inglese). Alcune aree sono contenute le une nelle altre, il programma principale le contiene tutte.

In ogni area la visibilità di un'entità dipende da dove questa è stata definita. Le regioni o aree specifiche in un codice Fortran sono:

- programma principale
- interfaccia di procedura
- definizione di tipo personalizzato
- procedura (interna o esterna)
- modulo



Visibilità

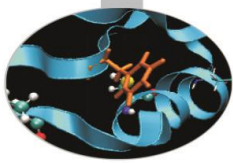
Variabili dichiarate in aree diverse sono sempre diverse.

Entità differenti di una stessa area devono avere nomi diversi.

I nomi delle regioni o aree sono visibili in tutto il codice, perciò devono essere distinti: non possono coesistere in un'unico codice procedure, moduli, tipi derivati con lo stesso nome.

Ogni entità pubblica dichiarata in un modulo è visibile in ogni area che usa quel modulo.

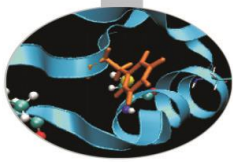
La visibilità di un'etichetta (label) è limitata all'area in cui è contenuta, escludendo ogni area contenuta.



Visibilità - Esempio

```
PROGRAM prova
      IMPLICIT NONE
      INTEGER :: n

11 READ(*,*) n
      CALL calc
      GO TO 11

CONTAINS

  SUBROUTINE calc
       IMPLICIT NONE
       IF (n>0) GO TO 11
       IF (n==0) GO TO 12
       PRINT*," N e' negativo "
       RETURN
   11  PRINT*, " N e' positivo"
       RETURN
   12  PRINT*," N e' nullo"
       RETURN
     END SUBROUTINE calc

END PROGRAM prova
```

Esercizi



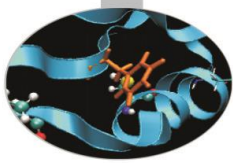
1. Scrivere un programma che data la velocità iniziale `v0` calcoli la gittata (distanza orizzontale del proiettile prima di toccare terra) per tutti i valori dell'angolo iniziale `theta` tra 0° e 90° con step di 1° e ricavi l'angolo `theta` per cui la gittata è massima.

   Formule utili:

   ```
   fattore conversione gradi-->radianti = 0.01745329
   ```

   ```
   gittata = - 2 v0**2 * COS(theta) * SIN(theta) /g
   ```

   Utilizzare una procedura interna per il calcolo.



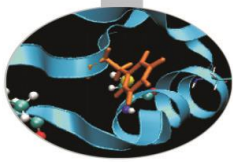
Sottoprogrammi

A loro volta, i sottoprogrammi possono essere:

INTERNI = compaiono nell'ambito del programma principale;

ESTERNI = compaiono in una sezione separata del codice.

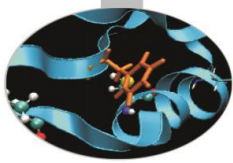
Nel FORTRAN 77 il codice era limitato al programma principale e alle procedure esterne.



Sottoprogrammi

Una procedura può essere esterna o interna, contenuta cioè nel programma principale o in un'altra procedura o in un modulo.

```
PROGRAM Esempio
      . . .
      STOP
  CONTAINS
  SUBROUTINE Procedura_1(...)
      . . .
    RETURN
  END SUBROUTINE Procedura_1
END PROGRAM Esempio
```

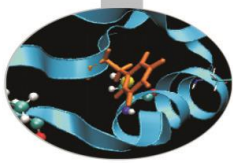


Sottoprogrammi

Ogni programma consiste in più unità di programma: ognuna di queste unità è indipendente dal punto di vista della **memoria** da destinare all'allocazione delle variabili.

Per questo motivo è buona norma definire gruppi di istruzioni in unità di programma separate, in modo da ridurre al minimo i problemi legati a possibili conflitti di memoria.

Procedure interne

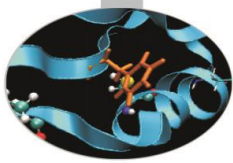


Le procedure interne:

- sono definite dopo l'istruzione `CONTAINS`;
- sono parte di una unità di programma e accedono alle variabili dell'unità di cui fanno parte.

Esempio:

```
PROGRAM contatore
   INTEGER :: i, j, k
   CALL Azzera
CONTAINS
  SUBROUTINE Azzera
        i=0; j=0; k=0;
   END SUBROUTINE Azzera
END PROGRAM contatore
```



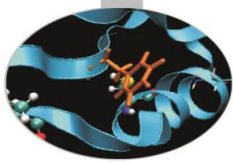
Procedure interne

E' possibile definire FUNCTION in fase di dichiarazione delle variabili, tramite espressioni in una sola linea ("*statement functions*"):

```
nome-funzione (lista-parametri) = espressione
```

Esempio:

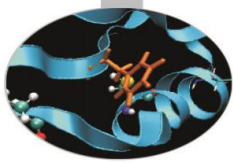
```
PROGRAM Inlinea

   REAL :: a, b, c, somma
   somma (a, b) = a + b

   READ (*,*) a
   READ(*,*) b

   c = somma (a, b)
   WRITE (*,*) c

END PROGRAM Inlinea
```



Procedure esterne

Le procedure esterne sono definite all'esterno del programma principale e dotate di un **propria allocazione di memoria** per le variabili.

Si trovano all'esterno del programma principale: in un file separato o nello stesso file ma non precedute dall'istruzione CONTAINS.

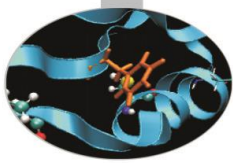


Procedure esterne

La comunicazione tra il programma principale o un sottoprogramma ed una procedura esterna avviene tramite la definizione di una **interfaccia in cui si specifica la lista degli argomenti scambiati e la loro natura.**

Nel caso di procedura esterna, al fine di una corretta compilazione del codice è necessario utilizzare le interfacce (INTERFACE BLOCKS). I blocchi interfaccia devono essere collocati nell'unità di programma chiamante o in appositi moduli.

Interfaccia

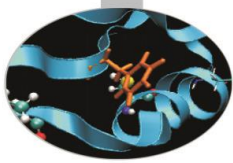


L'interfaccia di una procedura descrive:

- il numero di argomenti da passare;
- il tipo di ogni argomento passato;
- il tipo di ogni dato restituito dalla procedura esterna.

Esempio:

```
INTERFACE
      FUNCTION prova (a)
          REAL, INTENT(IN) :: a
          REAL :: prova
      END FUNCTION prova
END INTERFACE
```



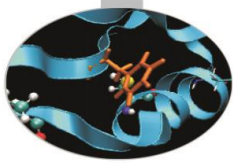
Interfaccia

L'interfaccia di una procedura è utile per permettere il controllo automatico, in fase di compilazione, della correttezza nell'uso della procedura stessa.

In alcuni casi è obbligatorio definire interfacce esplicite.

Le interfacce vanno definite solo per le procedure esterne: le procedure interne hanno interfacce implicite automatiche.

INTENT



E’ un attributo che si associa agli argomenti delle procedure, nella loro dichiarazione. `INTENT` può valere:

`IN` : la variabile è passata in sola lettura, il suo valore non può essere modificato;

`OUT` : la variabile viene calcolata dal sottoprogramma e passata al programma chiamante;

`INOUT` : la variabile viene passata ed eventualmente ricalcolata dal sottoprogramma.