Programmazione in Fortran:

Lezione 6

A.A. 2009/2010

Ing. A . Siviglia

nunzio.siviglia@ing.unitn.it

stanza:

Laboratorio didattico di modellistica idrodinamica (2° piano)

Tel 2440

INTTRODUZIONE ALLE PROCEDURE

Il FORTRAN possiede uno speciale meccanismo di progettazione di **sottoprogrammi** semplice da sviluppare e da farne il debug prima di costruire l'intero programma. E' possibile codificare ogni singola parte (sottoprogramma) in modo indipendente. Ogni singola unita' di programma e' chiamata **external procedure.**

Il FORTRAN ha 2 tipi di **external procedure:**

SUBROUTINE

FUNCTION

INTTRODUZIONE ALLE PROCEDURE

Procedure ben progettate riducono notevolmente gli sforzinella stesura di un programma lungo e complesso: Il loro utilizzo ha dei benefici:

- **Test indipendente di ogni singola parte del codice**

  Ogni singola parte puo' essere codificata e testata separatamente prima di essere incorporata nel programma principale

- **Riutilizzo dei singoli sottoprogrammi**

  Spesso l'utilizzo di uno stesso sotto-programma puo' essere riutilizzato in programmi diversi. Es: il calcolo della media di una serie di numeri puo' venir bene sia per calcolare la deviazione standard, sia per calcolare la media delle portate giornaliere e mensili di una serie di dati

Subroutines

Una subroutine e' una procedura del Fortran che viene richiamata dal attraverso una CALL (chiamata). La chiamata puo' essere effettuata sia dal programma principale, sia da altre procedure:

```
SUBROUTINE nome_subroutine ( lista_argomenti)
.......
   (sezione delle dichiarazioni)
.....
   (istruzione di esecuzione)


RETURN
END SUBROUTINE [nome_subroutine]
```

Subroutines

ES: calcolo dell'ipotenusa di un triangolo noti i due cateti

```
SUBROUTINE calcola_ipotenusa (cateto_1,cateto_2,ipotenusa )

IMPLICIT NONE

!Dichiarazione di variabili comuni

REAL, INTENT(IN)  :: cateto_1,cateto_2  !lunghezza cateti

REAL, INTENT(OUT) :: ipotenusa          !lunghezza ipotenusa

!Dichiarazione di variabili locali

REAL :: temp          !variabile temporanea



!Calcolo ipotenusa

temp = cateto_1**2 + cateto_2**2

ipotenusa = sqrt(temp)

END SUBROUTINE calcola_ipotenusa
```

Subroutines

```
PROGRAM test_ipotenusa

IMPLICIT NONE

!Dichiarazione di variabili

REAL :: s1,s2  !lunghezza cateti

REAL :: ipot   !lunghezza ipotenusa

!Acquisizione lunghezza cateti

WRITE(*,*) 'Introduci lunghezza cateti s1 e s2'

READ(*,*) s1,s2

!Calcolo ipotenusa

CALL calcola_ipotenusa(s1,s2,ipot)

WRITE(*,*) 'ipotenusa', ipot

END PROGRAM test_ipotenusa
```

L'attributo INTENT

Gli argomenti della subroutine possono avere un attributo di intenzione INTENT.

Si dice al compilatore quali sono le intenzioni del programmatore rispetto a certe variabili. L'attributo INTENT puo' assumere le seguenti forme:

```
INTENT(IN)     Gli argomenti della subroutine
               sono utilizzati per passare dati
               di input alla subroutine
INTENT(OUT)    Gli argomenti della subroutine
               sono utilizzati per ritornare
                risultati al programma chiamante
INTENT (INOUT) Gli argomenti della subroutine
                sono utilizzati per passare
                dati di input sia per far
                uscire risultati
```

Passare array ad una subroutine

Esistono 3 diversi modi per passare array a una subroutine.

Vi consiglio di passare i bounds di ogni singola dimensione dell'array come argomento della subroutine e dichiarare gli array argomenti della subroutine di quella lunghezza:

es:

```
SUBROUTINE process (array_in,array_out,n,n_dati)

IMPLICIT NONE

INTEGER, INTENT(IN)  :: n,n_dati

REAL, INTENT(IN), DIMENSION(n)  :: array_in  !forma esplicita

REAL, INTENT(OUT), DIMENSION(n) :: array_out  !forma esplicita

INTEGER:: i   !indice do loop

IF (n_dati > n) THEN

  WRITE(*,*) 'ATTENZIONE BOUND CHECK SUPERATO'

END IF

DO i = i,n_dati

 array_out(i) = 3.*array_in(i)

END DO

END SUBROUTINE process
```

Condividere dati usando MODULI

Abbiamo visto che i programmi scambiano dati con le subroutines attraverso la lista di argomenti. **Ogni singolo dato nella chiamata deve avere un corrispondente nella subroutine.**

Oltre alla lista degli argomenti le varie parti di un programma Fortran possono scambiare dati attraverso moduli. Un **modulo** e' una unita' di programma che viene compilata a parte e che contiene le definizioni e i valori iniziali di dei dati che vogliamo condividere tra le varie unita' del programma.

I moduli sono quindi un modo per condividere dati tra le varie unita' del programma

Condividere dati usando MODULI

```
MODULE dati_condivisi
!
!Fine: condividere dati fra 2 subroutines
!
IMPLICIT NONE
SAVE
INTEGER, PARAMETER :: num_dati=5 !max dati in array
REAL, DIMENSION(num_dati):: array
END MODULE dati_condivisi
```

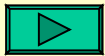
Condividere dati usando MODULI

```
PROGRAM test_modulo

USE dati_condivisi    !Rende visibili i dati del modulo

IMPLICIT NONE

REAL, PARAMETER :: PI=3.141592   !PI

array = PI*(/ 1.,2.,3.,4.,5. /)

CALL sub1

END PROGRAM test_modulo
```

Condividere dati usando MODULI

```
SUBROUTINE sub1

USE dati_condivisi    !Rende visibili i dati del modulo

IMPLICIT NONE

WRITE(*,*) array

END SUBROUTINE sub1
```

La sobroutine non ha nessun argomento!!!!!



FORTRAN FUNCTIONS

La FUNCTION e' una procedura il cui risultato e' un singolo numero, valore logico, stringa di caratteri o array.

Queste espressioni devono apparire alla destra di un'assegnazione.

Esistono due tipi di funzioni:

- INTRINSIC FUNCTION (sin(x), log(x) ecc.)
- FUNCTION SUBPROGRAMS (sono user defined)

FUNCTIONS

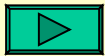
```
FUNCTION nome ( lista_argomenti)

......

   (sezione delle dichiarazioni)

.....

   (istruzione di esecuzione)

nome = espressione

RETURN

END FUNCTION [nome]
```

FUNCTIONS

```
REAL FUNCTION quadf (x,a,b,c)

IMPLICIT NONE

REAL, INTENT(IN):: x,a,b,c

!calcolo espressione

quadf = a*x**2 + b*x+c

RETURN

END FUNCTION [nome]
```

FUNCTIONS

```
PROGRAM test_quadf

IMPLICIT NONE

REAL x,a,b,c,risultato

WRITE(*,*) “Inserisci x,a,b,c”

READ(*,*) x,a,b,c

risultato = quadf(x,a,b,c)

WRITE(*,*) 'quadf',risultato

WRITE(*,*) 'quadf', quadf(x,a,b,c)


RETURN

END FUNCTION [nome]
```

ESERCIZIO

Utilizzare il codice della lezione 5 che calcola media e deviazione standard e riscriverlo in modo che utilizzi due subroutine diverse una per la media e l'altra per il calcolo della deviazione standard.
La versione 1 del programma deve condividere i dati attraverso una lista di argomenti, mentre la versione 2 deve condividere dati attraverso un module.