Programmare in Fortran - II

By Luigi Vetrano

Istruzioni di diramazione

- IF logico
  - IF (< espressione-logica >) <istruzione_eseguibile >
- Se il risultato dell'espressione-logica è VERO allora esegue l'istruzione a destra. se è FALSO procede in sequenza senza eseguire l'istruzione a destra.
- Esempio:
  1. max = y
  2. IF (x .GT. y) max = x ! IF (x > y) max = x
  3. …… quanto vale qui max ?
- Se x è maggiore di y allora max = x altrimenti max = y

Istruzioni di diramazione

- Costruzione IF …. THEN …. ELSE

  IF (< espressione_logica >) THEN

  < blocco di istruzioni 1 >

  ELSE

  < blocco di istruzioni 2 >

  ENDIF

- Se l'espressione_logica è vera viene eseguito il blocco di istruzioni 1 altrimenti viene eseguito il blocco di istruzioni 2.
- Se il blocco 2 è vuoto allora si può omettere l'else.
- Vi possono essere più IF annidati ma non si possono intersecare.
- Si può uscire da un IF ma non si può entrare se non dall'istruzione IF.

Esempio

Soluzione di una equazione di secondo grado

$$ax^2+bx+c=0$$

```
Delta= b*b - 4.*a*c)
IF (Delta <  0.) THEN
        WRITE(*,*) "L'equazione ha due radici complesse"
END IF
```

Istruzioni di diramazione

Costruzione IF … THEN … ELSEIF

```
IF ( < espressione_logica_1 > ) THEN
        < blocco_1 >
ELSEIF ( < espressione_logica_2 > ) THEN
        < blocco_2 >
    .............
ELSE
        < blocco_else >
ENDIF
```

<blocco_*> può contenere ulteriori IF, o altre istruzioni di diramazione o di ciclo

Esempio: equazione di 2° grado

```
Delta = b*b-4.*a*c
IF (Delta > 0.) THEN
    WRITE(*,*) 'L'equazione ha due radici reali distinte'
ELSE IF ( Delta == 0.) THEN
    WRITE(*,*) 'L'equazione ha due radici reali coincidenti'
ELSE
    WRITE(*,*) 'L'equazione ha due radici complesse'
END IF
```

Select case

```
SELECT CASE  (espressione_case )
      CASE(selettore_case_1)
            < blocco_1 >
      CASE(selettore_case_2)
            < blocco_2 >
      CASE DEFAULT
            < blocco_2 >
END SELECT
```

espressione_case è di tipo integer, logical o character

selettore_case deve essere dello stesso tipo di espressione_case e può indicare:

1. min: VERO se selettore_case >= min
2. A:B VERO se A <= selettore_case<= B
3. :max VERO se selettore_case <= max
4. Value VERO se selettore_case==value

Esempio di SELECT CASE

```
SELECT CASE( valore )
      CASE(1,3,5,7,9)
         write(*,*) ' valore è dispari'
      CASE(2,4,6,8,10)
         write(*,*) ' valore è pari'
     CASE(11:)
         write(*,*) ' valore è troppo alto'
      CASE DEFAULT
         write(*,*) 'il  valore è 0 o negativo'
END SELECT
```

Esercitazione

- Scrivere un codice che assegnati i coefficienti a,b,c calcoli le radici reali di un'equazione di secondo grado. Nel caso le radici siano complesse deve restituire un messaggio di errore su video
  - (provare caso a=1/3, b=2, c=3)
- Scrivere un codice utilizzando il costrutto select case che, dopo aver letto un valore T di temperatura, scriva sullo schermo
  - T < 0 → siamo sotto zero
  - T = 0 → punto di congelamento
  - 0 < T <= 10 → freddo
  - 10<T<25 → temperatura ottimale

Cicli a conteggio

- Permette di ripetere delle istruzioni per un numero definito di volte.

```
DO iter_ciclo=val_init,val_fin,increment
    Istruzione_1
    Istruzione_2
    .
    .
    Istruzione_n
END DO
```

- iter_ciclo è una variabile numerica intera (contatore)
- val_init, val_fin, increment sono variabili intere

Cicli a conteggio

- Il ciclo parte da val_init e prosegue fino a val_fin, incrementandosi di increment
- Il numero di iterazioni è fissato a priori
- Se il numero di iterazioni è < 0, le istruzioni all'interno del ciclo non vengono mai eseguite
- Il ciclo può effettuare conteggi decrescenti se val_fin< val_init e increment<0
- Se increment non è specificatio, viene assunto pari a 1

Esempi

5 iterazioni	0 iterazioni	3 iterazioni
DO i=4,24,5 write(*,*) i END DO	DO i=7,5 write(*,*) i END DO	DO i=12,6,-3 write(*,*) i END DO
Output	**Output**	**Output**
4 9 14 19 24		12 9 6

Cicli condizionati: DO… END DO

```
DO
                < blocco_istr_1 >
IF ( < espr_logica_1 > )  EXIT
                [ < blocco_istr_2 > ]
IF ( < espr_logica_2 > ) CYCLE
                [< blocco_istr_3 > ]
END DO
```

Cicli condizionati: DO… END DO

```
DO i=1,5
      IF( i == 3)  EXIT
      WRITE(*,*)  i
END DO
WRITE(*,*)  'Fine del ciclo'
```

Output→ 1 2

Fine del ciclo

```
DO i=1,5
      IF( i == 3)  CYCLE
      WRITE(*,*)  i
END DO
WRITE(*,*)  'Fine del ciclo'
```

Output→ 1 2 3 4 5

Fine del ciclo

Cicli condizionati: DO WHILE..END DO

```
DO WHILE ( < espr_logica > )
        < blocco_istr_1 >
END DO
```

- Il ciclo (quindi < blocco_istr>) viene ripetuto fintantoché viene rispettata la condizione < espr_logica_1 > (=VERA!).
- Utile se non si conosce a priori quanti cicli servono ma si ha una precisa condizione di uscita (es. raggiunta una certa tolleranza).

Esempio

```
i=1

DO WHILE (i<20)
      i=i*2
      WRITE(*,*)  i
END DO

WRITE(*,*)  'Fine del ciclo'
```

Risultato:

```
2
4
8
16
Fine del ciclo
```

Funzioni intrinseche

- Il Fortran include numerose funzioni intrinseche per eseguire operazioni:
  1. Di Conversione (INT, DBLE, ...)
  2. Matematiche (SIN , LOG, ...)
  3. Numeriche (SUM , CEILING, ...)
  4. Di tipo carattere (INDEX , TRIM, ...)
  5. Per la manipolazione di bit (IAND, IOR, ...)
  6. Di indagine (ALLOCATED, SIZE, ...)
  7. Di trasformazione (REAL, TRANSPOSE, ...)
- Diverse Funzioni e Subroutine non elementari come SYSTE_CLOCK o DATE_AND_TIME

Utilizzo delle funzioni intrinseche

- Le funzioni intrinseche possono essere usate ovunque all'interno dell'algoritmo.
- Utilizzo:

function_name(input1,input2)

- Gli input delle funzioni sono chiamati **argomenti**
- Una funzione può restituire reali, interi, o altri valori dipendenti dalle sue caratteristiche

Alcune funzioni veramente utili

Function	Argument Type	Result Type	Comment
`sqrt(x)`	real	real	Square root
`abs(x)` or `abs(i)`	real integer	real integer	Absolute value
`sin(x)`	real	real	Sine
`cos(x)`	real	real	Cosine
`tan(x)`	real	real	Tangent
`exp(x)`	real	real	Exponential
`log(x)`	real	real	Natural logarithm
`log10(x)`	real	real	Base-10 logarithm
`asin(x)`	real	real	Inverse sine
`acos(x)`	real	real	Inverse cosine
`atan(x)`	real	real	Inverse tangent
`max(a,b)`	real/integer	real/integer	Maximum of A & B
`min(a,b)`	real/integer	real/integer	Minimum of A & B
`nint(x)`	real	integer	Nearest integer

Esercitazione

- Scrivere un codice che calcoli la media di n numeri reali positivi (massimo 50) inseriti da tastiera.
- Caso (1) Se si trova il numero -999, vuol dire che tutti i numeri sono stati inseriti e si deve interrompere il calcolo.
  - (usare EXIT);
- Caso (2) Se si trova un numero negativo lo si scarta.
  - (usare CYCLE)

Formati e Formattazione

- Finora abbiamo letto valori da tastiera e scritto sullo schermo usando le due istruzioni READ e WRITE.
- Il formato che abbiamo usato è detto free-format ed è specificato dal secondo * nelle espressioni

READ(*,*) e WRITE(*,*)

- Come si è visto, il risultato dell'utilizzo del free-format, fa apparire spazi non necessari.

Formati e Formattazione

```
WRITE(*,100) i,risultato

100 FORMAT ('Iterazione numero',I3,1x,'vale', F7.3)
```

L'istruzione FORMAT contiene le istruzioni per la formattazione utilizzata dall'istruzione WRITE
I3 e F7.3 sono descrittori di formato associati alle variabili i e risultato

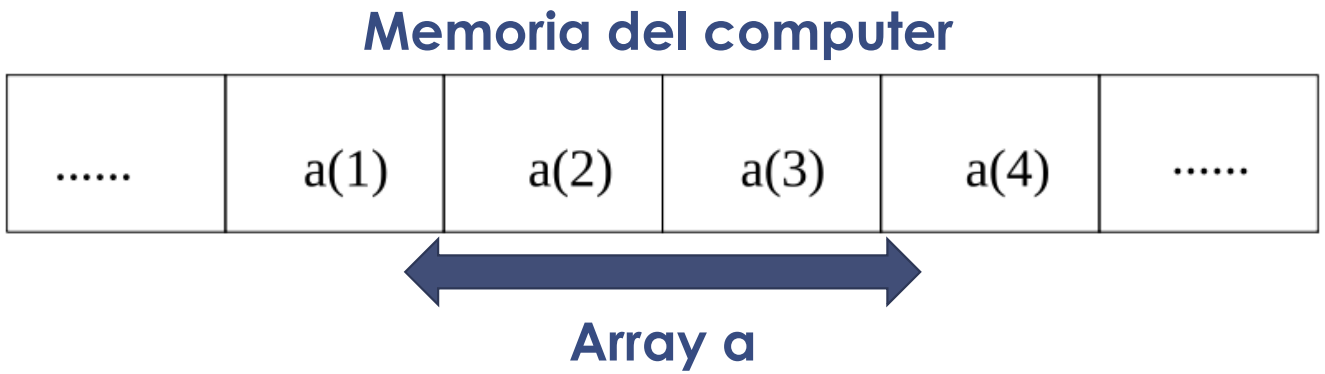
```
Iterazione numero 3 vale 3.142      Output formattato

3 3.141593                          Output Non Formattato
```

Vettori o array

- Alcuni problemi di programmazione necessitano l'utilizzo di un'aggregazione di valori, piuttosto che di uno solo. Questo significa che è conveniente indicare l'insieme di valori con una sola variabile piuttosto che tante variabili.
- In FORTRAN questa problema è risolto utilizzando degli array.

Vettori o array

- Un array è un gruppo di variabili o costanti , tutte dello stesso tipo, che si riferiscono ad un singolo nome.
- Ogni singolo valore dell'array è detto elemento dell'array, ed è identificato dal nome dell'array e da un indice che punta ad una particolare posizione all'interno dell'array.



Vettori e Matrici (Terminologia)

- Un array (vettore o matrici a più dimensioni) permette di specificare con un nome ed un insieme di indici una serie di elementi in modo molto semplice. Ogni array ha un tipo ed ogni elemento è di tale tipo.
- Si possono definire le seguenti proprietà:
  - RANGO (RANK) - Numero di dimensioni
  - LIMITI (BOUNDS) - Limite superiore ed inferiore di ogni dimensione
  - ESTENSIONE (EXTENT) - Numero di elementi in ogni dimensione
  - GRANDEZZA (SIZE) - Numero totale di elementi

Dichiarazione

```
INTEGER, DIMENSION(100)   ::  vettore_i
INTEGER                   ::  vettore_i(100)

REAL, DIMENSION(100)      ::  vettore_r
REAL                      ::  vettore_r(100)

CHARACTER(len=5), DIMENSION(100)  :: cognome
CHARACTER(len=5)          ::  cognome(100)
```

Visualizzazione degli array

- REAL, DIMENSION(15) :: A

A(1)	A(2)	A(3)	…	…	A(13)	A(14)	A(15)

Gli elementi degli array sono semplici variabili

```
INTEGER, DIMENSION(10) :: index
REAL, DIMENSION(3)     :: temp
index(1) = 5
temp(3) = REAL(index(1))/5.
WRITE(*,*) "index(1)",index(1)
```

Inizializzazione degli elementi degli array

- Inizializzazione con delle istruzioni di assegnazione...

```
REAL, DIMENSION(10) :: array
DO i=1,10
  array(i) = REAL(i)
END DO
```

- Oppure, in alternativa, con la seguente istruzione:

```
array=(/1.,2.,3.,4.,5.,6.,7.,8.,9.,10./)
```

È anche possibile scrivere

```
array= 0.0        !azzera tutti gli elementi dell'Array
```

Inizializzazione degli elementi degli array

- Inizializzazione all'interno delle dichiarazioni

  INTEGER, DIMENSION(5) :: array2 = (/1,2,3,4,5/)

- DO implicito

  /(arg1,arg2,....,index = istart,iend,incr)/

  INTEGER, DIMENSION(5) :: array2 = (/(i,i=1,5)/)

Cambiare il limite inferiore del range di un array

REAL, DIMENSION(5) :: a1

REAL, DIMENSION(-2:2) :: b1

REAL, DIMENSION(5:9) :: c1

a1, b1, c1

sono array di 5 elementi!!

Superamento del range di un array (out of bounds)

```
INTEGER :: i

REAL,DIMENSION(5):: a=(/1.,2.,3.,4.,5./)

REAL,DIMENSION(5):: b=(/10.,20.,30.,40.,50./)

DO i = 1,6

   write(*,*) a(i)

END DO
```

Uso delle costanti nella dichiarazione di array

- INTEGER, PARAMETER :: max_size = 1000
- REAL :: array1(max_size)
- REAL :: array2(max_size)
- REAL :: array3(2*max_size)

Operazioni sull'array globale

condizione necessaria: gli array devono avere la stessa forma

```
INTEGER :: i
REAL, DIMENSION(4) :: a = (/1.,2.,3.,4./)
REAL, DIMENSION(4) :: b = (/5.,6.,7.,8./)
REAL, DIMENSION(4) :: c,d
! somma elemento per elemento
DO i=1,4
   c(i) = a(i) + b(i)
END DO
write(*,*) c
```

Operazioni sull'array globale

- condizione necessaria: gli array devono avere la stessa forma

```
INTEGER :: i
REAL, DIMENSION(4) :: a = (/1.,2.,3.,4./)
REAL, DIMENSION(4) :: b = (/5.,6.,7.,8./)
REAL, DIMENSION(4) :: c,d
! somma sull'array globale
d = a + b
write(*,*) d
```

Operazioni sull'array globale

- posso eseguire operazioni con costanti a livello globale

```
INTEGER :: i
REAL, DIMENSION(4) :: a = (/1.,2.,3.,4./)
REAL :: cost=5.
! somma sull'array globale
DO i=1,4
   a(i) = a(i) +/* cost
END DO
! equivalente a
   a = a +/* cost
```

Quando usare gli array?

- Non risolvere tutti i problemi con gli array!!!!
- Si usano quando tutti, o la maggior parte dei dati, devono essere tenuti in memoria per risolvere in modo efficiente un problema.
- Utilizzare array richiede memoria!!!

Esercizio

1. Assegnato un numero di dati N, calcolare la media e la deviazione standard

$$\hat{x} = \frac{1}{N}\sum_{i=1}^{N} x_i = \frac{x_1+x_2+x_3+\ldots+x_N}{N}$$

$$\sigma = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(x_i - \hat{x})^2}$$

2. Trovare il max e il min di una serie di dati e la loro posizione (senza utilizzare le funzioni intrinseche MAXVAL e MAXLOC)
3. Ordinare la serie di dati in senso crescente
4. Scrivere gli output con tre cifre significative!