LABORATORIO DI PROGRAMMAZIONE
Corso di laurea in matematica

15 – IL LINGUAGGIO FORTRAN

Dipartimento di Matematica e Applicazioni
Universita' degli Studi di Napoli Federico II

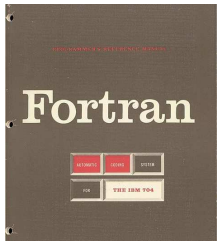
wpage.unina.it/lapegna

1

Il FORTRAN

il **FORTRAN** (FORmula TRANslation) e' il primo esempio di linguaggio di programmazione ad alto livello

- FORTRAN I (1956) *
- FORTRAN II e FORTRAN III (1958)
- FORTRAN IV (1961)
- FORTRAN 66 *
- FORTRAN 77 *
- FORTRAN 90 *
- FORTRAN 95
- FORTRAN 2003 *
- FORTRAN 2008
- FORTRAN 2018



Continue evoluzioni per adattarsi a nuove e piu' sofisticate esigenze del calcolo scientifico (puntatori, allocazione dinamica, programmazione a oggetti, programmazione concorrente)

2

Il FORTRAN

- sintassi pulita (semplice da imparare)
- linguaggio compilato (veloce ed efficiente)
- numerose librerie matematiche efficienti disponibili (NAG, LAPACK, …)

Ancora oggi e' uno dei principali e piu' utilizzati

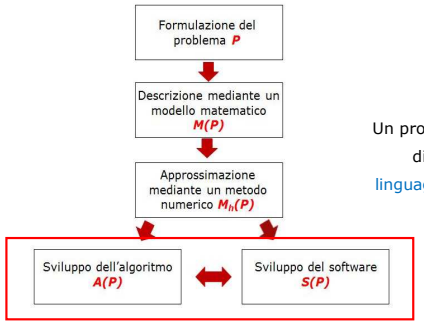
linguaggi di programmazione per le applicazioni scientifiche

Forniremo una **introduzione** alla sintassi della versione

Fortran 2003 (oggi lo standard de facto)

3

dall'algoritmo al programma



Un programma e' la traduzione di un algoritmo in un linguaggio di programmazione

obiettivo: **introdurre la sintassi** per tradurre un algoritmo descritto in **Pascal-like** in un programma scritto in linguaggio **Fortran**

4

inizio e fine di un programma

- Ogni programma FORTRAN e' compreso tra le istruzioni PROGRAM e END
- **L'istruzione PROGRAM da' il nome al programma**
- Ogni programma termina con **END**

```
begin nomealgoritmo
...
...
...
end nomealgoritmo
```

→

```
PROGRAM NOME
…
…
…
END
```

esempio di traduzione dal Pascal-like al FORTRAN

5

dichiarazione delle variabili

- Il FORTRAN e' **Case Unsensitive** (non fa differenza tra maiuscole e minuscole)
- Possibile (**ma non consigliato**) non dichiarare le variabili
- Nomi di max 32 caratteri e **iniziano sempre con una lettera** (A-Z)

```
var M, N: integer
var X, Y: real
var A, B: character
var R,T: boolean
var  W, Z: complex
```

→

```
INTEGER:: M, N
REAL:: X, Y
CHARACTER:  A, B
LOGICAL:: R, T
COMPLEX:: W, Z
```

rappresentazione tipo **INTEGER** : **complemento a 2**
rappresentazione tipo **REAL** : **IEEE 754**
rappresentazione tipo **CHARACTER**: **codice ASCII**

6

approfondimenti

- E' possibile specificare il tipo (KIND) di un variabile. **Corrisponde al numero di byte** occupato dalla variabile

```
INTEGER (1) :: I
INTEGER (2) :: J
INTEGER (4) :: K
REAL (4) :: X
REAL (8) :: Y
REAL (16) :: Z
```

KIND possibili per il tipo **INTEGER**
default : KIND = 4 (32 bit)

KIND possibili per il tipo **REAL**
default : KIND = 4 (32 bit)

- Il codice ASCII utilizza 1 byte per carattere. In caso di nomi (stringhe di caratteri) **e' possibile specificare la lunghezza massima**

```
CHARACTER (LEN = 10) :: NOME
```

dichiara una variabile capace di contenere al piu' 10 caratteri

7

approfondimenti

- E' possibile non dichiarare le variabili (sconsigliato!). Esse sono comunque **dichiarate implicitamente** con la seguente regola:

iniziali del nome: **A** … **H** **I** … **N** **O** … **Z**

variabili reali — **variabili intere** — **variabili reali**

```
IMPLICIT NOME
```

Impedisce le dichiarazioni implicite

- E' possibile specificare **caratteristiche supplementari** (attrbuti):

esempio:

```
REAL, PARAMETER:: PI = 3.1415926
```

Definisce dichiara **PI** come numero reale che non puo' essere cambiato durante l'esecuzione (errore in compilazione)

8

commenti

- I commenti sono linee di testo inserite nel codice che non vengono tradotte dal compilatore
- Hanno lo scopo di migliorare la leggibilita' dei programmi, descrivendo le sezioni del programma o anche gruppi di istruzioni
- E' buona norma inserire numerose linee di commento (anche fino al 50% delle linee dl programma)
- In Fortran i commenti iniziano con il **carattere !**

```
begin nomealgoritmo
...
(questo e' un commento)
...
end nomealgoritmo
```

```
PROGRAM NOME
…
! questo e' un commento
…
END
```

9

approfondimenti

- E' possibile inserire piu' istruzioni su una sola riga, separate da un **;**
- esempio: scambio di 2 variabili

```
...
T = A
A = B
B = T
...
```

e' equivalente a

```
...
T = A ; A = B ; B = T
...
```

10

lettura, definizione e stampa di variabili

- **READ*** definisce una o piu' variabili mediante l'unita' di input (es. la tastiera)
- **PRINT*** stampa il contenuto di una variabile sull'unita' di output (es. il monitor)
- L'operatore **=** indica una operazione di assegnazione di un valore ad una variabile

```
...
read A, B, C
X = A+B+C
print X
...
```

```
…
READ*, A, B, C
X = A+B+C
PRINT*, C
…
```

11

struttura di selezione

- Esatta corrispondenza con il Pascal Like
- Disponibile anche senza il ramo ELSE

```
...
if (condizione) then
        ...
else
        ...
endif
...
```

```
…
IF (CONDIZIONE) THEN
        …
ELSE
        …
ENDIF
…
```

12

operatori logici e relazionali

- le condizioni da verificare nelle strutture di selezione sono la composizione di operazioni logiche e relazionali
- vengono eseguiti prima gli operatori **relazionali**

> < == /= >= <=

- poi quelli **logici**

.NOT. .AND. .OR.

```
...
if ( a > b and x==y) then
        ...
endif
...
```

```
…
IF ( A > B .AND. X == Y) THEN
        …
ENDIF
…
```

13

approfondimenti

Strutture di selezione innestate

Invece di

```
IF ( condizione1 ) THEN
    blocco istruzioni 1
ELSE
    IF (condizione2) THEN
      blocco istruzioni 2
    ELSE
      IF (condizione3) THEN
            blocco istruzioni 3
      ELSE
            blocco default
      ENDIF
    ENDIF
ENDIF
```

E' possibile

```
IF ( condizione1 ) THEN
    blocco istruzioni 1
ELSE   IF (condizione2) THEN
       blocco istruzioni 2
ELSE   IF (condizione3) THEN
       blocco istruzioni 3
ELSE
       blocco default
ENDIF
```

14

struttura di iterazione for

- tre campi per specificare valore **iniziale**, valore **finale** e **incremento** dell'indice
- possono essere **variabili** (non necessariamente costanti)
- Il campo per **l'incremento e' opzionale**, se assente si sottointende **1**
- Piu' strutture innestate richiedono **indici distinti**

```
...
for i = 1 to N step K
        ...
endfor
...
```

```
…
DO I = 1, N, K
        …
ENDDO
…
```

15

un esempio completo

```
PROGRAM MASSIMO_N_NUMERI
! dichiarazione delle variabili
INTEGER:: I, N
REAL:: MASSIMO, NUMERO
! inizio esecuzione
READ*, N
DO I = 1, N
       READ NUMERO
       IF ( NUMERO > MASSIMO ) THEN
              MASSIMO = NUMERO
       ENDIF
ENDDO
PRINT MASSIMO
END
```

16

struttura di iterazione while

- Ripete piu' volte le istruzioni tra **DO WHILE** e **ENDDO**
- Quanto la condizione risulta **vera** la struttura di iterazione **continua**
- analogo al pascal like

```
...
while ( condizione )
        ...
        ...
endwhile
...
```

```
…
DO WHILE ( CONDIZIONE )
        …
        …
ENDDO
…
```

17

struttura di iterazione repeat

- Ripete piu' volte le istruzioni tra repeat e until
- Quanto la condizione risulta **vera** la struttura di iterazione **termina**
- Non esiste una traduzione "diretta"
- Realizzato mediante struttura di **DO-ENDDO + condizione di uscita**

```
...
repeat
        ...
        ...
until (condizione)
...
```

```
…
DO
        …
        …
IF ( CONDIZIONE ) EXIT ; ENDDO
…
```

18

gli array

- in fase di dichiarazione e' necessario **specificare la dimensione**
- successivamente **non e' possibile modificare** la dimensione
- e' possibile fare riferimento a ciascuna componente tramite **un indice**
- e' possibile **utilizzare solo una parte** dell'array
- in memoria le componenti di un array occupano **locazioni di memoria consecutive**

```
var x(10) : integer
...
read N
for i = 1 to N
        read*, X(i)
endfor
```

```
INTEGER, DIMENSION(10) :: X
…
READ*,N
DO I = 1, N
        READ*, X(I)
ENDDO
```

19

gli array

Gli array 2-D sono memorizzati **per colonne**
in locazioni di memoria consecutive

Esempio: $A = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix}$

```
INTEGER, DIMENSION(4,4) :: A
…
N = 3
DO I = 1, N
        DO J = 1, N
                READ*, A(I,J)
        ENDDO
ENDDO
```

1	A(1,1)
4	A(2,1)
7	A(3,1)
	A(4,1)
2	A(1,2)
5	A(2,2)
8	A(3,2)
	A(4,2)
3	A(1,3)
6	A(2,3)
9	A(3,3)
	A(4,3)
	A(1,4)
	A(2,4)
	A(3,4)
	A(4,4)

memorizzazione della matrice A

20

sottoprogrammi SUBROUTINE

- la **SUBROUTINE** sono la traduzione in FORTRAN delle **procedure**
- istruzione **CALL** passa il controllo dell'esecuzione alla subroutine
- la SUBROUTINE riceve i valori di input e restituisce quelli di output attraverso una **lista di argomenti**
- L'esecuzione del programma chiamante riprende al termine della subroutine
- Gli argomenti nella testata rappresentano l'unico strumento di comunicazione tra programma chiamante e subroutine

```
begin main
var A, B, C: integer
read A, B
somma (in A, B; out C)
print C
end main
```

```
PROGRAM MAIN
INTEGER:: A, B, C
READ *, A, B
CALL SOMMA (A, B, C)
PRINT*, C
END
```

21

sottoprogrammi SUBROUTINE

- inizia con la parola chiave **SUBROUTINE**
- E' possibile utilizzare **nomi diversi** nelle subroutine e nel programma chiamante
- variabili non presenti nella testata non sono visibili dal programma chiamante

```
procedure somma(in X, Y; out Z)
var X, Y, Z: integer
Z = X+Y
end somma
```

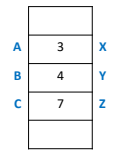
```
SUBROUTINE SOMMA(X, Y, Z)
INTEGER:: X, Y, Z
Z = X + Y
END
```

22

osservazione importante

- il passaggio delle informazioni e' **per indirizzo** (della variabile)
- la subroutine opera sulle **stesse locazioni di memoria** del programma chiamante
- **non c'e' differenza** tra parametri di input e di output.
- Ogni modifica fatta dalla SUBROUTINE sugli argomenti **si ripercuote nelle corrispondenti variabili** del programma chiamante

```
PROGRAM MAIN
INTEGER:: A, B, C
READ *, A, B
CALL SOMMA (A, B, C)
PRINT*, C
END
```



memorizzazione degli argomenti nelle locazioni di memoria

```
SUBROUTINE SOMMA(X, Y, Z)
INTEGER:: X, Y, Z
Z = X + Y
END
```

23

subroutine ed array

Nel caso di array viene passato solo l'indirizzo della prima componente

- array memorizzati in locazioni contigue
- la subroutine non usa ulteriore memoria per gli argomenti

Da queste informazioni la **subroutine e' in grado di "ricostruire" tutto l'array**

Nelle SUBROUTINE e' possibile dichiarare gli array **evitando di specificare una dimensione fissata**

```
procedure sommavet(in N, X, Y; out Z)
var i, N, X(N), Y(N), Z(N): integer
for i = 1 to N
        Z(i) = X(i) + Y(i)
endfor
end somma
```

```
SUBROUTINE SOMMAVET(N, X, Y, Z)
INTEGER:: I, N
INTEGER, DIMENSION(N) :: X, Y, Z
DO I = 1, N
        Z(I) = X(I) + Y(I)
ENDDO
END
```

24

osservazione importante

array con 2 indici (memorizzati per colonna)

Viene passato alla subroutine solo l'indirizzo di A(1,1)

Se programma chiamante e subroutine dichiarano l'array con **numero di righe differente**, fanno riferimento alla stessa variabile **con indici diversi**

```
PROGRAM MAIN
INTEGER:: N
REAL, DIMENSION(3,3):: A
…
N = 2
CALL SUBMAT(A, N)
…
END
```

memoria

A(1,1)		X(1,1)
A(2,1)		X(2,1)
A(3,1)		X(1,2)
A(1,2)		X(2,2)
A(2,2)		
A(3,2)		
A(1,3)		
A(2,3)		
A(3,3)		

```
SUBROUTINE SUBMAT(X,M)
INTEGER:: M
REAL, DIMENSION(M,M):: X
…
…
…
END
```


soluzione

Passare alla subroutine anche l'effettivo numero di righe dell'array

Leading Dimension (LD)

- Numero di **righe con cui si e' dichiarato l'array** nel programma chiamante
- Utilizzato **solo nella dichiarazione dell'array nella subroutine**
- Permette il **corretto allineamento degli indici** tra le due unita' di programma

```
PROGRAM MAIN
INTEGER:: N, LD
REAL, DIMENSION(3,3):: A
…
N = 2
LD = 3
CALL SUBMAT(A, N, LD)
…
END
```

memoria

A(1,1)		X(1,1)
A(2,1)		X(2,1)
A(3,1)		X(3,1)
A(1,2)		X(1,2)
A(2,2)		X(2,2)
A(3,2)		X(3,2)
A(1,3)		X(1,3)
A(2,3)		X(2,3)
A(3,3)		X(3,3)

```
SUBROUTINE SUBMAT(X,M,LD)
INTEGER:: M, LD
REAL, DIMENSION(LD,M):: X
…
…
…
END
```


sottoprogrammi FUNCTION

- Procedura Fortran **invocata direttamente attraverso il nome**
- Alternativa alle subroutine
- **Restituisce un valore attraverso il nome**
- Utile per definire funzioni matematiche
- Va dichiarata come una variabile

```
begin main
var  X, Y: real
read X
fun(in: X; out: Y)
print Y
end somma
```

```
PROGRAM MAIN
REAL: X, Y, FUN
READ*, X
Y = FUN(X)
PRINT*, Y
END
```


sottoprogrammi FUNCTION

- nella testata e' necessario **dichiarare il tipo del valore di ritorno**
- **Il valore di ritorno e' assegnato ad una variabile con lo stesso nome della function**
- Stesso meccanismo di passaggio di argomenti delle subroutine

```
procedure fun(in: X; out: Y)
var  X, Y: real
Y = X*X
end somma
```



```
REAL FUNCTION FUN(X)
REAL:: X
FUN = X*X
END
```

approfondimento

- E' possibile passare una function come argomento ad una subroutine
- Necessario usare la dichiarazione EXTERNAL nel programma chiamante

```
PROGRAM MAIN
REAL:: X, Y, FUN
EXTERNAL FUN
READ*, X
CALL SQUAREF(X,FUN,Y)
PRINT*, Y
END
```

```
SUBROUTINE SQUAREF(X, FUN, Y)
REAL:: X, Y, FUN
Y = FUN(X) * FUN(X)
END


REAL FUNCTION FUN(X)
REAL:: X
FUN = 1./(X-1)
END
```

programma chiamante e subroutine
per calcolare il quadrato di una funzione
(nell'esempio y=1/(x-1))

29

Il Fortran dispone di una libreria di function per le piu' comuni funzioni matematiche

- Y = SQRT(X)
- Y = ABS(X)
- Y = TAN(X)
- Y = SIN(X)
- Y = COS(x)
- Y = LOG(X)
- Y = LOG10(X)
- Y = EXP(X)
- Y = MAX(A1, A2, A3, …)
- Y = MIN(A1, A2, A3, …)
- Y = MOD(A, P)
- Y = INT(X)

- E tante altre…

```
PROGRAM MAIN
REAL:: X, Y, PI, H

PI = ACOS(-1)
H = PI/10

DO I = 0, 10
        X = I*H
        Y = SIN(X)
        PRINT*, X, Y
ENDDO
END
```

Semplice programma per il calcolo di
11 valori di sin(x) nell'intervallo [0,3.14]

30

un po' di storia (15)

Richard Stallman (1953)

- Americano, si laurea in Fisica ad Harvard nel 1974 e' il principale esponente del movimento per il software libero, nato alla fine degli anni '70 a seguito delle leggi americane sul copyright
- Il software libero e' un software distribuito con una licenza che ne permette la distribuzione, lo studio e la modifica da parte di altri (licenza GPL)
- Fonda la Free Software Foundation per contrastare ogni forma di proprieta' sul software e sviluppare nuovo software libero.
- Nel 1983 inizia il progetto GNU, una versione di Unix con licenza GPL. Linus Torval utilizzera' nel 1990 il sistema GNU come base del s.o. Linux
- Ha ricevuto numerosi premi, lauree honoris causa e cattedre onorarie

FREE SOFTWARE
FOUNDATION
www.fsf.org

31