

Tutorial sul Fortran 77

A cura di G. Pollarolo.

INDICE

1. INTRODUZIONE
2. IL PRIMO PROGRAMMA
3. IL LESSICO DEL FORTRAN
 1. Programma
 2. Istruzioni
 3. Commenti
 4. Variabili
 5. Vettori e Matrici
 6. Operazioni aritmetiche e funzioni intrinseche.
 7. Istruzioni Logiche e di Controllo
 8. Strutture di controllo.
 1. Iterazione, DO-loops
 2. Strutture IF
 3. Istruzioni di iterazione (di salto) GOTO.
 9. Istruzioni di INPUT-OUTPUT
4. UNA LISTA DI PROGRAMMI MOLTO SEMPLICI
 1. Una farfalla da aggiungere alla collezione
 2. Un bel tappeto caotico
 3. Algoritmo di Euclide.
 4. Ancora algoritmo di Euclide
 5. Rappresentazione di un numero reale in base binaria

1 - INTRODUZIONE.

Iniziamo queste nostre lezioni di **FISICA NUMERICA** con una breve introduzione al linguaggio **FORTRAN** (*formula translator*). Abbiamo scelto questo linguaggio per la sua duttilità nella implementazione degli algoritmi che si incontrano nella risoluzione dei problemi numerici che sono comuni a moltissime discipline scientifiche. Lo scopo di questa introduzione è di mostrarvi gli elementi essenziali del linguaggio di programmazione senza avere la pretesa di essere completi o anche esatti (ovviamente i diversi esempi che faremo spero lo siano). Quello che si vuole è familiarizzarvi al più presto con il linguaggio stesso in modo che possiate scrivere subito programmi utili. Per fare questo ci dobbiamo concentrare sugli aspetti fondamentali del **FORTRAN**: variabili, costanti, operazioni aritmetiche, controllo di flusso, function, subroutine e i rudimenti dell'input e dell'output. La parte più raffinata del linguaggio, utile per la stesura di programmi complessi, viene lasciata da parte, si cercherà di descriverla man mano che diventa necessaria nel prosieguo del nostro corso. Ovviamente questa scelta comporterà delle inevitabili ripetizioni che spero non siano di tedio ma servano di approfondimento. Spero anche che questa introduzione vi porti ad una comprensione del **FORTRAN** tale da poter consultare con profitto i manuali dove sono descritte in grande dettaglio tutte le varie possibilità del linguaggio.

2 - IL PRIMO PROGRAMMA.

C'è un unico modo per imparare un nuovo linguaggio di programmazione è, scrivendo programmi. Il primo programma che si scrive è sempre lo stesso in tutti i linguaggi: cioè richiedere al calcolatore di scrivere le parole `hello, benvenuti a FISICA NUMERICA`

La stesura di questo programma, seppur semplicissimo, comporta la conoscenza di tutti quei passaggi fondamentali che dovete imparare a maneggiare subito.

Per poter richiedere al vostro calcolatore di eseguire questo semplice ordine dovete essere in grado di scrivere da qualche parte il testo del programma (questo comporta l'aver accesso ad un terminale e saper usare un **EDITOR**, dovete saperlo **compilare** correttamente (cioè ordinare al calcolatore di controllare l'ortografia del testo del programma), saper fare il **LINK** (cioè controllarne la struttura con il sistema operativo della macchina stessa) e per ultimo saperlo eseguire e scoprire così dove la scritta compare (cioè dove va a finire l'**OUTPUT**).

Trovato un amico (il cosiddetto local expert) che vi spieghi questi dettagli, propri della struttura di calcolo che dovete usare, la scrittura del programma è estremamente semplice. Usando appunto il linguaggio **FORTRAN** il programma per scrivere `hello, benvenuti a FISICA NUMERICA` è semplicemente:

```
print *, 'hello, benvenuti a FISICA NUMERICA'  
stop  
end
```

Il come eseguirete questo programma dipenderà dal sistema su cui state lavorando, per essere specifici supporremo di lavorare su una macchina **VAX** con il sistema operativo **VMS**. Su questo sistema invocando l' **EDITOR**, per esempio con il comando:

```
$ edit/edt      !($ è il prompt del VMS)
```

create una file che contenga il sorgente (source) del programma e che abbia un nome con la estensione **.for**, per esempio `benvenuto.for`. Fatto questo con il comando

```
$ for benvenuto
```

eseguitene la compilazione, se il compilatore non protesta, cioè se non vi inonda lo schermo con messaggi di errore, potete proseguire alla creazione dell'elemento eseguibile con il comando

```
$ link benvenuto
```

Se anche il link non protesta troverete nella vostra directory un nuovo file `benvenuto.exe` che può essere eseguito con il comando

```
$ run benvenuto
```

Se tutti gli steps descritti sono eseguiti correttamente sul, vostro schermo (sulla sinistra in basso) troverete le parole:

```
hello, benvenuti FISICA NUMERICA  
[FORTRAN STOP]
```

Come esercizio provate a riscrivere il summenzionato programma commettendo deliberamente degli errori durante la fase di editing e leggete attentamente i messaggi di errore che compariranno sul vostro schermo durante la fase di compilazione, questo servirà a famigliarizzarvi con il linguaggio del calcolatore stesso.

Prima di finire, alcune considerazione sul programma stesso. Un programma **FORTRAN**, indipendentemente dalla sua dimensione, è costituito da un insieme di più *functions* e/o *subroutines* che specificano le diverse operazione che devone essere fatte. Il nome di queste *routines* è lasciato libero al programmatore eccetto che per il *main* che viene definito come quella *routine* che contiene le istruzioni `stop` ed `end` (le altre *routines* terminano con l'istruzione `return`). Tutti i programmi **FORTRAN** iniziano la loro esecuzione dal *main*. è il *main* che distribuisce poi i vari compiti alle *subroutine*. La chiamata ad una *routine*, sia questa intrinseca o scritta dal programmatore, avviene normalmente nominandola nel programma e facendola seguire da una serie di argomenti che ne costituiscono l' *input* e ne determinano il risultato *output* (anche questo può, in generale, essere contenuto negli argomenti).

Ritornando al nostro esempio vediamo che questo programma è costituito da una sola *routine* il *main* che contiene una chiamata alla *routine* intrinseca del **FORTRAN** `print`. Questa *routine* ha come argomenti una stella (`*`) e la stringa di caratteri che deve scrivere, notate che questa è stata delimitata da apici (`'`). La stellina (`*`) definisce il formato con cui gli argomenti che seguono devono essere stampati (in questo caso in *free*

format), ne approfondiremo il significato più avanti. Le ultime due istruzioni del programma sono già state commentate.

3 - IL LESSICO DEL FORTRAN.

Riassumiamo brevemente cosa abbiamo imparato dall'esempio precedente. Il **FORTRAN**, permettendo la stesura di procedure (programmi) per impartire istruzioni alla CPU del calcolatore, costituisce una importante interfaccia utente macchina. Al contrario del forse più noto BASIC, non è direttamente eseguibile ma necessita di una prima fase di compilazione per il controllo della ortografia e di una successiva fase di mapping che ne controlla la sua struttura con il sistema operativo della macchina stessa. Dopo questi due passaggi si ottiene un modulo che è direttamente eseguibile.

Il **FORTRAN** permette una programmazione STRUTTURATA e per la sua struttura algebrica è particolarmente adatto alle applicazioni scientifiche. Il riconoscimento di questo è molto importante in quanto a noi interessa non tanto sapere quello che un programma fa ma vogliamo sapere come lo fa in quanto vogliamo essere in grado di rileggerlo ed eventualmente modificarlo in tempi successivi per renderlo adatto alle nostre nuove esigenze.

Quando scriviamo un programma dobbiamo sempre tener presente che questo dovrà poter essere letto da altri (o da noi stessi in tempi successivi) per cui dovranno essere facilmente riconoscibili le sue diverse parti ed il modo con cui gli algoritmi usati sono stati implementati. Questo per capire se un dato programma di cui siamo venuti in possesso soddisfa le nostre esigenze e se gli algoritmi sono stati implementati correttamente ed in modo efficiente.

La struttura di un programma non si manifesta solo nella sua suddivisione in SUBROUTINE e FUNCTION ma nella presenza di una certa gerarchia nelle diverse parti che lo compongono. Al livello più basso troviamo l'insieme dei caratteri ASCII, poi le costanti, gli operatori e semplici istruzioni quali per esempio:

```
root(j) = .5 * (-b + sqrt(delta)) / a
```

Al livello successivo si trovano gruppi di istruzioni che hanno un certo significato quando sono considerate in blocco, per esempio:

```
aux=a(i)
a(i)=b(i)
b(i)=aux
```

vengono interpretate da ogni programmatore come lo scambio di due variabili mentre il gruppo di istruzioni

```
sum=0
ifial=1
answer=0.
n=1
```

significa probabilmente la inizializzazione di variabili per alcuni procedimenti iterativi. Il buon programmatore avrà cura di mettere commenti che descrivono il significato di questi blocchi di istruzioni, così scriverà cambio di variabili nel primo caso ed inizializzazione nel secondo,

```
c
c    inizilizzazione per l'intergrazione....
sum=0
ifial=1
answer=0.
n=1
```

Al livello successivo abbiamo poi le strutture che sono governate da comandi di controllo come il DO o l' IF che modificano l'esecuzione sequenziale di un programma e che verranno spiegate dovutamente in seguito.

Da ultimo abbiamo le strutture fornite dalle subroutine e dalle function . Il loro significato e la loro importanza nella corretta costruzione di un programma sarà trasparente solo quando si implementeranno i vari algoritmi che studieremo nella risoluzione pratica dei problemi fisici. Per prepararci a capire questo cominciamo con il vedere un po' di lessico del **FORTRAN**.

3.1 Programma

La traduzione di un metodo di calcolo (algoritmo) con una sequenza di istruzioni che il calcolatore può interpretare e che termina con la parola end definisce una struttura di calcolo cioè un programma. Il programma principale (main), le subroutine e le function costituiscono le unità di programma.

3.2 Istruzioni

Queste si possono suddividere in due categorie:

- eseguibili, cioè che descrivono una azione che il programma può eseguire
- non eseguibili, cioè che definiscono delle variabili o delle strutture di dati o delle dichiarazioni,....
-

Le istruzioni, cioè i diversi passi in cui un programma è suddiviso, devono essere solitamente contenute in una linea di **72 caratteri**, partendo sempre dalla colonna **7**. Le prime sei colonne sono usate per definire la *label* di una data istruzione, per dire se questa linea di programma deve essere considerata di commento o per dire se essa deve essere vista come una continuazione di quella precedente. Se l'istruzione che si vuole scrivere richiede più dei caratteri permessi si può continuare nella linea successiva avendo cura di mettere un carattere (non sono permessi lo zero (0) e lo spazio (blank)) in colonna 6.

3.3 Commenti

Sono linee che si inseriscono durante la stesura di un programma per documentazione. Queste non alterano la sequenza di calcolo e possono essere inserite in qualsiasi punto del programma identificandole con un carattere speciale in colonna 1 (uno). Sono permessi i seguenti caratteri di commento:

- la lettera `c` in colonna 1
- l'asterisco `*` in colonna 1
- il punto esclamativo `!` si ignora tutto ciò che sta alla destra

Per facilitare la lettura di un programma si raccomanda l'uso delle linee di commento che devono illustrare le procedure adottate. Si raccomanda altresì l'uso della `*` in colonna uno (1) per definire le linee di commento, questo perché sulle macchine vettoriali la lettera `c` in prima colonna viene interpretata diversamente dal **FORTRAN** implementato su questo tipo di macchine.

3.4 Variabili.

Nel **FORTRAN** una variabile è definita quando appare sul lato sinistro di una istruzione. Per esempio:

```
x=3.
```

definisce la variabile `x`, questa può in seguito essere usata in successive istruzioni che possono definire nuove variabili come per esempio:

```
y=1.+x+3.*x**2
```

si noti che il segno di uguale (`=`) non va interpretato con il suo significato matematico ma piuttosto rappresenta l'operazione di rimpiazzamento, così l'espressione:

```
x=x+1.
```

non ha senso in matematica mentre in programmazione significa rimpiazzare la `x` con il valore di `x+1`

Il nome di una variabile è una sequenza al più di 31 caratteri che possono essere lettere (maiuscole e minuscole) numeri e caratteri speciali quali il `$` e `_` (underscore). Le variabili non possono iniziare con un numero. Su alcune vecchie macchine le variabili non possono eccedere la lunghezza di sei (6) caratteri.

Il **FORTRAN** permette variabili di diverso tipo, le più importanti sono:

- variabili intere (per default il compilatore considera intere le variabili che iniziano con le lettere `i, j, k, l, m, n` che possono assumere valori da -32768 a $+32767$ (`integer *2`))
- variabili reali (per default tutte le variabili che iniziano con gli altri caratteri permessi) che assumono valori nell'intervallo $-0.29 \cdot 10^{-38}$ ed $1.7 \cdot 10^{38}$ con una precisione sulle prime sette (7) cifre (`real *4`)
- variabili complesse (come due variabili reali)
- variabili logiche
- caratteri alfanumerici (dette anche stringhe)
- byte (cioè otto (8) bit)

3.5 Vettori e Matrici.

L'istruzione:

```
dimension avec(12),amat(5,7)
```

oppure

```
real*4 avec(12),amat(5,7)
```

definisce che la variabile `avec` deve essere considerata come un insieme di 12 variabili reali identificabili con l'indice $i=(1-12)$, in modo analogo `amat` definisce un insieme di $5*7$ variabili reali. Ovviamente l'introduzione di vettori e matrici non costituisce semplicemente una economia di nomi di variabili ma permette l'implementazione di raffinati algoritmi matematici. I nomi `ivec` e `imat` indicheranno vettori (matrici) di numeri interi.

3.6 Operazioni aritmetiche e funzioni intrinseche.

Le operazioni aritmetiche elementari che si possono fare sono:

- `x+y` somma
- `x-y` differenza
- `x*y` prodotto
- `x/y` divisione
- `x**m` potenza m-esima di `x` (`m` intero)
- `x**y` esponenziazione

dove con `x` e `y` abbiamo indicato il nome di variabili o di espressioni. Oltre alle operazioni fondamentali viste sopra si possono richiamare un certo numero di funzioni intrinseche (*build-in functions*). Le più comuni che troverete sono:

- `sin, cos, tan` le funzioni trigonometriche del seno, coseno e tangente
- `asin, acos` le funzioni trigonometriche inverse
- `log` logaritmo naturale
- `exp` esponenziale
- `sqrt` radice quadrata
- `abs` valore assoluto

Queste funzioni intrinseche possono intervenire nella definizione di espressioni del tipo `3.+2.*sqrt(5.)` oppure del tipo `2.*acos(0)` (quest'ultima per definire il numero π). Si noti che l'argomento va racchiuso tra parentesi e segue il nome della funzione chiamata. Per ottenere la lista completa delle funzioni intrinseche consultate il manuale del FORTRAN che state usando.

3.7 Istruzioni Logiche e di Controllo.

Sono relazioni che intercorrono fra variabili logiche. Siano `x` and `y` due variabili logiche, allora:

- `.not .x` falsa se `x` è vera e vera se `x` è falsa,
- `x.and.y` vera se `x` ed `y` sono vere contemporaneamente, falsa altrimenti,
- `x.or.y` vera se la `x` o la `y` sono vere, falsa altrimenti.

Se `x` ad `y` sono due espressioni o due variabili (interi o reali) le seguenti istruzioni permettono di confrontarle fra loro:

- `x.eq.y` vera se `x` è uguale a `y`, falsa altrimenti
- `x.ne.y` vera se `x` e `y` sono diversi, falsa altrimenti
- `x.gt.y` vera se `x` è più grande di `y`, falsa altrimenti
- `x.lt.y` vera se `x` è più piccola di `y`, falsa altrimenti
- `x.ge.y` vera se `x` è più grande od uguale a `y`, falsa altrimenti
- `x.le.y` vera se `x` è più piccola od uguale a `y`, falsa altrimenti.

Si notino i punti (.) che intervengono sempre nella definizione degli operandi logici.

3.8 Strutture di controllo.

Un programma viene eseguito sequenzialmente una istruzione dopo l'altra nell'ordine in cui queste sono state scritte (il compilatore può a volte alterare questo ordine se lo ritiene necessario per ottimizzare lo svolgimento del programma). Le istruzioni che modificano l'ordine con cui le istruzioni vengono eseguite si chiamano *istruzioni di controllo*. Queste non hanno un particolare significato in sè, il significato che assumono deriva dal blocco di istruzioni che governano. La corretta implementazione di queste istruzioni è essenziale per una buona programmazione strutturata.

3.8.1 Iterazione, DO-loops .

Nel **FORTRAN** la semplice iterazione viene effettuata con il `do-loop` per esempio se si vuole calcolare la funzione $f(x)=\sin(x)\exp(-x/a)$ per i valori di `x` compresi fra `x=0` ad `x=2.` in step di `dx=0.1` si procede come segue:

```

dimension a(200)
.....
flam=0.65
xmin=0.
xmax=2.
dx=0.1
x=xmin
npoint=(xmax-xmin)/dx+1.0001 !(nota il fattore 1.0001)
do n=1,npoint
  a(n)=sin(x)*exp(-x/flam)
  x=x+dx
enddo
.....

```

Notare che nella scrittura delle istruzioni governate dal do-loop si è avuto cura di indentarle. Questo modo di scrivere il source di un programma non interessa il compilatore ma aiuta molto i vostri occhi quando devono rileggere il programma. Purtroppo non tutti i calcolatori accettano come ortografia del do-loop quella appena vista, ma preferiscono la seguente:

```

dimension a(200)
.....
flam=0.65
xmin=0.
xmax=2.
dx=0.1
x=xmin
npoint=(xmax-xmin)/dx+1.0001 !(nota il fattore 1.0001)
do 10 n=1
  a(n)=sin(x)*exp(-x/flam)\endline
  x=x+dx
10 continue
.....

```

come si vede in questo secondo caso si è dovuto ricorrere all'uso di numeri **labels** che identificano una data istruzione in questo caso il `continue`. Questa è una istruzione muta che passa l'esecuzione alla istruzione successiva.

Oltre al semplice do-loop visto sopra si possono avere i cosiddetti NESTED do-loop come:

```

dimension a(30,25)
.....
do n=1,25\endline
  .....
  do m=1,30
    .....
    a(m,n)=.....
  .....
  enddo
enddo
.....

```

si noti ancora l'importanza della indentazione per identificare il gruppo di istruzioni che sono interessate alla iterazione.

In generale il do-loop si scrive:

```

.....
do n=n1,n2,nstep
  .....
  a(n)=.....
  .....
enddo
.....

```

dove i valori di `n1` ed `n2` specificano l'intervallo entro cui l'indice `n` del do-loop deve essere iterato ed `nstep` indica con che passo questa iterazione debba avvenire. Se `nstep` viene omissso, come negli esempi precedenti, il compilatore assume `nstep=1`. È importante sapere che l'indice del do-loop viene ad assumere il valore (`n2+nstep`) al termine della iterazione e che il controllo di fine loop avviene prima della esecuzione del blocco di istruzioni interno. Questa considerazione pu' risultare importante in molte applicazioni che vedremo in seguito.

3.8.2 Strutture IF.

Gli esempi che seguono illustrano l'uso della struttura `if`, è importante capirla bene in quanto è essenziale nella implementazione degli algoritmi che vedremo in seguito. In tutti gli esempi che seguono con `e`, `e1`, `e2`, ... si intenderanno delle espressioni logiche.

Esempio 1.

```
if (e) then
    .....
    .....
endif
```

È chiaro che il blocco di istruzioni (indicato con `)` è eseguito solo se la condizione `e` è vera.

Esempio 2.

```
if (e1) then
    .....
    .....
else
    .....
    .....
endif
```

Il blocco 1 di istruzioni è eseguito se la condizione `e1` è vera, in caso contrario viene eseguito il blocco 2 di istruzioni.

Esempio 3.

```
if (e1) then
    .....
    .....
else if (e2) then
    .....
    .....
endif
```

Se la condizione `e1` è verificata viene eseguito il blocco 1 mentre se è verificata la condizione `e2` viene eseguito il blocco 2. Se le condizioni logiche `e1` ed `e2` non sono verificate allora nessuno dei blocchi viene eseguito.

Esempio 4.

```
if (e1) then
    .....
    .....
else if (e2) then
    .....
    .....
else
    .....
    .....
endif
```

Le istruzioni indicate genericamente con `)` vengono eseguite solo se le condizioni `e1` ed `e2` non sono verificate. Le strutture `if` degli esempi 3 e 4 possono essere iterate anche per più condizioni logiche. L'importanza di queste strutture è di per se ovvia e la si apprezzerà appieno in seguito durante le applicazioni pratiche. Ancora una volta vi ricordo l'importanza della corretta indentazione del blocco di istruzioni interessato dalla istruzione `IF`.

3.8.3 Istruzioni di iterazione (di salto) GOTO.

In molte applicazioni si ha la necessità di interrompere alcune sequenze di iterazione quando si verificano certe condizioni su alcuni parametri che governano l'iterazione stessa. Per illustrare come queste istruzioni si usano vediamo come si può riscrivere il primo esempio dell'applicazione del `do-loop` appunto usando il `goto`.

```

dimension a(200)
.....
flam=0.65
xmin=0.
xmax=2.
dx=0.1
x=xmin
npoint=(xmax-xmin)/dx+1.0001      !(nota il fattore 1.0001)
n=1
10 continue
a(n)=sin(x)*exp(-x/flam)
x=x+dx
n=n+1
if(n.gt.npoint) goto 20
goto 10
20 continue
.....

```

Si deve notare che la istruzione `goto` deve sempre fare riferimento ad una istruzione del programma con una `label`.

L'adozione di questa istruzione di salto al di fuori di un suo uso corretto deve essere in ogni caso molto parsimonioso impone infatti al programma di avere diversi `labels` (si ricordi che in un programma i `labels` devono essere unici) che pregiudicano la lettura del programma stesso in quanto ne alterano la sequenzialità. Un uso esasperato dei `goto` porta inevitabilmente a commettere molti errori nella stesura del programma, fortunatamente questi sono quasi sempre rilevati dal compilatore derivando per lo più da un cattivo uso delle `labels`. Ovviamente non è qui il caso di fare una lista di tutti i possibili errori che si possono commettere è meglio discuterli man mano che si presentano raccomandando però di leggere attentamente il commento che il compilatore scrive a fianco dell'errore rilevato.

3.9 Istruzioni di INPUT-OUTPUT .

Le istruzioni di `input-output`, di cui abbiamo visto un esempio, nella nostra introduzione, sono molto importanti in quanto costituiscono lo strumento con cui il programma comunica con il mondo esterno. Questa comunicazione avviene sia quando il programma richiede (`input`) dati per l'elaborazione sia per comunicare i risultati della sua elaborazione (`output`).

Per le operazioni di `input` si possono invocare i comandi:

```

read
accept      (preferibile non usare)

```

mentre per quelle di `output` i comandi:

```

write
print
type      (preferibile non usare)

```

Questi comandi devono essere specificati attraverso due campi di opzioni; il primo contiene le opzioni di controllo mentre l'altro conterrà la lista per l'i/o. La lista di i/o contiene chiaramente la lista delle variabili che il programma deve elaborare o che ha elaborato ed il cui valore deve essere comunicato all'utilizzatore esterno. La lista di controllo da uno o più parametriche specificano:

- unità logica su cui devono agire
- file interno su cui devono operare
- se i dati da leggere o da scrivere devono essere formattati (cioè se su questi si devono fare delle operazioni di *editing*)
- il numero di una istruzione a cui devono fare riferimento nel caso in cui un errore od un `end of data` intervenga durante la fase di i/o.

Una struttura tipica che interviene nei programmi che devono richiedere dei dati in `input` è per esempio:

```

print *, ' Enter zmin,zmax,dz (*) '
read *, z1,z2,dz

```

Con queste istruzioni il calcolatore richiede i dati scrivendo sullo schermo (`STANDAR_DINPUT`):

```

Enter zmin,zmax,dz (*)

```

e quindi posizionando il cursore sul primo carattere della linea successiva aspetta che voi scriviate i valori dei parametri richiesti. In questo particolare esempio l'asterisco (`*`) ricorda che questi dati devono essere entrati in `free format` cioè i tre numeri devono essere separati o da una virgola (`,`) o da una spazio (blank). In queste due istruzioni la lista di controllo è costituita semplicemente dall' asterisco (`*`) intendendo con questo che si vogliono usare i

default del **FORTRAN** stesso, cioè che l'input/output avvenga attraverso l'unit'a logica 5/6 (STANDARD_INPUT/OUTPUT e che i dati devono essere dati in free-format.

Senza ricorrere ai valori di default le istruzioni precedenti posso riscriversi:

```
write(6,98)
read(5,99)z1,z2,dz
98 format(' Enter zmin,zmax,dz[3f10.4]')
99 format(3f10.4)
```

oppure

```
write(6,'(a)') 'Enter zmin,zmax,dz'
read(5,'(3f10.4)')z1,z2,dz
```

Si osservi che in questo caso i dati devono essere inseriti rispettando le regole di editing come specificato dalle istruzioni *format*. In questo caso per i dati (sono in floating-point, cioè numeri reali) si sono riservati 3 campi di dieci (10) caratteri di cui gli ultimi quattro (4) per le cifre significative dopo la virgola.

Come altro esempio supponiamo di dover leggere una serie di dati (sia per esempio la tabulazione di una funzione) che sono contenuti in un file di nome *mydata.dat*. Le seguenti istruzioni servono allo scopo:

```
dimension x(50),y(50)
character*50 filename
100 continue
print *,' Enter name of the file with da data'
read(*,'(a)',err=100) filename
c
open(unit=20,file=filename,status='old',err=100)
c
do n=1,1000
read(20,*,end=110,err=100) x(n),y(n)
enddo
110 continue
close 20
npoint=n-1
.....
stop
end
```

si noti che la lista di controllo nella istruzione *read* contiene due campi aggiuntivi che permettono l'uno di gestire l'errore passando il controllo alla istruzione con label 100, l'altro di gestire l' *end_ of_ file* passando il controllo alla istruzione con label 110 e quindi interrompe la sequenza del *do-loop*. Si noti che con questo metodo siamo in grado di determinare il numero di punti in cui la funzione è definita.

Qui di seguito vediamo tre modi equivalenti per ottenere lo stesso scopo in fase di scrittura, cioè l' *output* dei valori della due variabili *a* e *b*.

Esempio 1.

```
.....
write(6,97) ' dr =',a,'R =',b
97 format(a,1f5.2,2x,a,1f8.2)
.....
```

Esempio 2.

```
.....
write(6,'(a,1f5.2,2x,a,1f8.2)')' dr =',a,'R =',b
.....
```

Esempio 3.

```
.....
print '(a,1f5.2,2x,a,1f8.2)', ' dr =',a,'R =',b
.....
```

se $a=0.25$ e $b=10.18$ si ottiene il seguente risultato:

```
dr = 0.25 R = 10.18
```

Mi pare ovvio rimandare una più approfondita esposizione dei vari metodi di *input/output* durante la stesura dei programmi veri e propri e di consigliarvi la lettura delle appropriate pagine dei manuali.

4. UNA LISTA DI PROGRAMMI MOLTO SEMPLICI

Qui di seguito una lista di semplicissimi programmi iniziali che sono commentati durante le lezioni che servono per familiarizzarvi con il linguaggio. Nella loro stesura si è anche cercato di completare alcune informazioni sul FORTRAN stesso.

4.1 Una farfalla da aggiungere alla collezione

```
program farfalla
  implicit real*8(a-h,o-z)
  print *, ' enter number of point, number of turn'
  read *, npoint, nturn
  pi=2.d0*acos(0.d0)
  dth=2.d0*pi*dfloat(nturn)/dfloat(npoint)
  call ginit
  th0=0.d0
  call window(-4.d0,-3.d0,4.d0,4.d0)
  do i=0,npoint-1
    th=th0+dth*dfloat(i)
    r=exp(cos(th))-2.d0*cos(4.d0*th)+sin(th/12.d0)**5
    x1=r*cos(th+pi/2.d0)
    y1=r*sin(th+pi/2.d0)
c
    th=th0+dth*dfloat(i+1)
    r=exp(cos(th))-2.d0*cos(4.d0*th)+sin(th/12.d0)**5
    x2=r*cos(th+pi/2.d0)
    y2=r*sin(th+pi/2.d0)
c
    call line(x1,y1,x2,y2)
  enddo
c
c   si puo' fare di meglio provateci
c
  call gend
c
  stop
end
```

4.2 Un bel tappeto caotico

```
program caotico
  include 'fgraph.fi'
  include 'fgraph.fd'
  record /videoconfig/ vc
  record /xycoord/ xy
  integer*2 dummy,xt(4),yt(4),x0,y0,xmedio,ymedio
  data xt/ 320,590, 50,320/
  data yt/ 5,450,450, 5/
c
  print *, ' enter number off calls'
  read *,ncalls
  call getvideoconfig (vc)
  if(vc.adapter.eq.$vga) idummy=setvideomode($vres2color)
c
c   inizializza lo schermo grafico
  call getvideoconfig (vc)
c
c   disegna il triangolo
  call moveto(xt(1),yt(1),xy)
  do i=2,4
    dummy=lineto(xt(i),yt(i))
  enddo
```

```

c
x0= 0      ! starting point
y0= 0      ! starting point
do n=1,ncalls
  call random(ran)
  ip=3.*ran+1
  xmedio=(x0+xt(ip))/2.
  ymedio=(y0+yt(ip))/2.
  call moveto(xmedio,ymedio,xy)
  dummy=lineto(xmedio,ymedio)
  x0=xmedio
  y0=ymedio
enddo
if (ncalls.gt.49999) then
  call settextposition(2,2,xy)
  call outtext(' The CAOS game')
  call settextposition(3,2,xy)
  call outtext(' Sierpinski gasket')
else
  call settextposition(2,2,xy)
  call outtext(' End Iteration')
endif
read (*,*)
c
idummy=setvideomode($defaultmode)
c
stop
end

```

4.3 Algoritmo di Euclide.

```

c  program euclid.for
c
c  this program implements the Euclid algorithm to calculate the MCD
c  between two integer number (m,n).
c
c  integer*4 n,m,r                      ! (1)
c
c  print *, ' scrivi n,m (*)'
c  read *,n,m
c  if(m.eq.n) stop 'm=n'
c  do i=1,100
c    r=n-(n/m)*m                        ! (2)
c    if(r.eq.0)then
c      print '(a,li5)', ' Massimo Comun Divisore (MCD) ', m
c      goto 101
c    else
c      n=m
c      m=r
c    endif
c  enddo
c  print *, ' passi: troppo pochi'      ! (3)
101 continue
c  stop
c  end
c
c  (1) questa istruzione serve per dichiarare al compilatore che
c  tutte le variabili in questa lista devono essere considerate
c  come variabili integer*4 (cioe' 32 bit).
c
c  (2) E' piu' conveniente richiamare la funzione del Fortran
c  MOD(m,n)
c  Questa funzione calcola i modulo di m in base n

```

```

c         cioe' ritorna come valore il resto del divisione di m per n,
c         in formula  $\text{mod}(n,m)=n-(n/m)*m$ .
c
c         (4) questa scritta comparira' qualora il programma faccia piu' di
c         100 iterazioni senza arrivare al risultato. Chiaramente quando
c         questa evenienza si verifica il programmatore deve essere
avvisato
c         affinche modifichi il do-loop.

```

4.4 Ancora algoritmo di Euclide

```

c  program euclid1.for
c
c  this program implements the Euclid algorithm to calculate the MCD
c  between two integer number (m,n).
c
c      integer*4 n,m,r
c
c      print *, ' scrivi n,m (*)'
c      read *,n,m
c      r=421616
c      do while(r.ne.0)
c          r=mod(m,n)
c          print *,r
c          m=n
c          n=r
c      enddo
c      print '(a,li5)', ' Massimo Comun Divisore (MCD) ', m
c
c      stop
c      end

```

4.5 Rappresentazione di un numero reale in base binaria

```

c  program chbf
c  this program transforms the real number fnum in base 10
c  in its rapresentation in base 2.
c
c      implicit real*8(a-h,o-z)                ! (1)
c      dimension ires(100),kres(100)
c
c      input the number and set to zero's ires
c      nba=2
100 continue
c      print *, ' Enter number (*), ctrl_z to exit'
c      read (*,*,end=101) fnum
c      n1=int(fnum)
c      dec=fnum-float(n1)
c
c      zeros ires (integer part) kres (decimal part)
c      do n=1,100
c          ires(n)=0
c          kres(n)=0
c      enddo
c
c      change of base starts for the integer part
c      do n=1,100
c          n2=n1/nba
c          ires(n)=n1-n2*nba
c          n1=n2
c          if(n1.eq.0) goto 90
c      enddo

```

```

90 continue
   ndint=n
c
c   printing of the result for the real part only
   iaux=ndint/8
   if((8*iaux).lt.n)then
       nmax=8*(ndint/8+1)
   else
       nmax=iaux*8
   endif
   print *, ' Integer part'
   print '(1x,4(8i1,1x))', (ires(n),n=nmax,1,-1)      ! (2)
   if(dec.eq.0.) goto 100
c
c   change of base starts for the decimal part
   nbit=48
   x=dec
   do n=1,nbit
       aux=0.5**n
       dif=x-aux
       if(dif.ge.0.) then
           kres(n)=1
           x=dif
       else
           kres(n)=0
       endif
   enddo
c
c   Printing the results for the dec part
   print *, ' Decimal part'
   print '(1x,6(8i1,1x))', (kres(n),n=1,nbit)          ! (2)
   goto 100
c
101 continue
c
   end

```

```

c-----
c
c   Note al programma CHBF.FOR
c
c   (1) Con questa istruzione implicir real*8 si dichiarano in doppia
c   precisione (cioe' di 64 bit) tutte le variabili che iniziano
c   per una qualsiasi lette dalla a alla h e dall o all z.
c   Le variabili che iniziano per i,j,k,l,m ed n non sono modificate
c   dalla dichiarazione per cui per esse valgono i valori di defaults
c   del FORTRAN cioe' sono variabili intere.
c
c   (2) Si noti cove viene richiesta la stampa di tutte le componeti del
c   vettore ires(i) (oppure kres(i))

```