

INTRODUZIONE AL FORTRAN

G. POLLAROLO

Dipartimento di Fisica Teorica, Università di Torino

Via Pietro Giuria 1, 10125 TORINO (Italy)

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
-

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 devono 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.

secondo,

```
c
c  inizializzazione 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 un5 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

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 $5 \cdot 7$ 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 `avec` 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.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 sé, 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.

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 istruzione 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
```

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 (`STANDARD_INPUT`):

```
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 uno 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'unità logica 5/6 (`STANDARD_INPUT/OUTPUT`) e che i dati devono essere dati in free-format.

.....
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 famigliarizzarvi 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

```
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*float(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 può fare di meglio provateci
c
call gend
c
```

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,i5)', ' 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
c          avvisato
c          affinche modifichi il do-loop.
```

4.4 Ancora algoritmo di Euclide

```

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))
c

```

Il FORTRAN

Il linguaggio FORTRAN (FORMula TRANslation) nato negli anni '50, come un meccanismo per la gestione di librerie (di programmi d'utilit[^]), ed \dot{z} stato il primo linguaggio di programmazione ad alto livello. Dopo le prime versioni, apparve nel 1957 il manuale del FORTRAN II, ad opera di John Backus ed altri /Backus/. Sono stati poi elaborati gli standard di nuove versioni (di cui la pi^u fortunata \dot{z} stata quella del FORTRAN IV /McCracken/), fino agli attuale FORTRAN strutturati /Brained/.

Il FORTRAN si colloca fra i linguaggi imperativi ad alto livello, come uno dei pi^u vicini alla struttura della macchina. Per questo, oltre che per il fatto di essere un linguaggio di largo uso da molti anni, possiede compilatori molto efficienti e "robusti". Inoltre il corredo delle librerie di programmi di utilit[^], venutosi a comporre ed a consolidare nel tempo, \dot{z} senza paragone rispetto ad ogni altro linguaggio della stessa classe. \dot{e} utilizzato soprattutto per elaborazioni di calcolo numerico, dove l'efficienza e la velocit[^] che offre, dovute al suo orientamento alla macchina, gli consentono di non avere molti rivali. \dot{e} facile intuire che la fortuna del FORTRAN \dot{z} destinata a durare, soprattutto in ambienti in cui si richiedono grosse elaborazioni numeriche, anche a causa dell'alto costo che comporta il cambio di un linguaggio in un ambiente di programmazione gi^a esistente.

L'uso specialistico del FORTRAN per elaborazioni numeriche ed il suo orientamento alla macchina costituiscono anche i suoi veri limiti. Nel FORTRAN ogni vincolo sintattico e semantico ha generalmente una propria origine hardware. Per esempio il FORTRAN ha una gestione statica delle risorse, in particolare della memoria. Ci^o significa che non viene effettuata alcuna gestione a tempo di esecuzione. Tutto va fissato al momento di compilazione. I sottoprogrammi non possono essere ricorsivi, le loro variabili locali "ricordano" i valori dell'attivazione precedente, parametri possono essere passati soltanto per riferimento. Le variabili globali sono consentite solo su esplicita dichiarazione dell'area di memoria condivisa dai sottoprogrammi. I soli tipi di dati consentiti sono di tipo numeri

co o booleano ed in generale occorre conoscere e scegliere la rappresentazione interna utilizzata. L'unica struttura dati consentita è l'array di elementi omogenei, generalmente con indice intero positivo e con non più di tre dimensioni. Le operazioni di ingresso/uscita sono complesse ed operano solo su file sequenziali, richiedendo l'esplicita definizione della rappresentazione usata per i dati. Il meccanismo di controllo delle istruzioni è spesso vincolato all'uso dell'istruzione di salto "GOTO" che complica la chiarezza del testo.

1. LA SINTASSI

La sintassi delle istruzioni FORTRAN è rimasta vincolata all'uso delle schede, anche ora che generalmente il programma viene redatto da terminale. Le schede possono contenere perforazioni valide fino alla colonna 72, le colonne 73-80 non vengono lette dall'unità periferica (lettore di schede), e tale limite superiore è valido per ogni istruzione FORTRAN. Le prime cinque colonne rappresentano il campo etichetta, la colonna 6 va riempita con un qualsiasi carattere se l'istruzione precedente continua sulla riga attuale, altrimenti va lasciata bianca, le colonne 7-72 sono utilizzabili per la scrittura delle istruzioni. Una "C" in colonna 1 indica una riga di commento.

Questa rigida sintassi non è stata modificata nonostante il progressivo disuso delle schede ed è sintomatica della residuità delle strutture del linguaggio.

I dati sono costanti e variabili numeriche. Le costanti alfanumeriche sono consentite solo per stampare frasi e sono chiamate costanti Hollerith (dal nome dell'ideatore del primo sistema di codifica alfanumerica su schede); se ne parlerà nel paragrafo relativo all'ingresso/uscita.

Il controllo, cioè l'ordine in cui dovranno essere eseguite le istruzioni nel programma, è sequenziale di riga in riga. Il programmatore lo può alternare specificando, con le istruzioni di salto, la successiva istruzione da eseguire. Le istruzioni su cui è possibile trasferire il controllo devono essere identificate mediante etichette numeriche scritte nella zona iniziale della riga, nel campo etichetta precedentemente

dispensa.txt

e descritto.

Per descrivere la sintassi delle istruzioni verrà utilizzata una Backus Naur Form (BNF) estesa, del tipo di quella descritta in /Welsh/, con le seguenti differenze:

Le categorie sintattiche sono racchiuse fra le parentesi < e >;
i simboli terminali sono scritti in lettere maiuscole.

Non è previsto un ordine di composizione per le istruzioni di un programma FORTRAN. È comunque utile strutturare il programma per migliorarne la flessibilità. In /Batini/ si suggerisce di comporre le dichiarazioni, come parte iniziale del programma, nel seguente ordine:

Dichiarazioni esplicite

DIMENSION*

COMMON*

EQUIVALENCE

FORMAT*

EXTERNAL Istruzioni funzionali

A queste seguirà il resto del programma, composto dalle istruzioni "eseguibili"; l'ultima istruzione deve essere una istruzione END, che segnala al compilatore la fine fisica del programma.

2. TIPI DI DATI

Il FORTRAN consente il trattamento di dati interi, reali, complessi e booleani. I dati possono essere costanti, usate direttamente nelle espressioni senza esplicita dichiarazione, oppure variabili. L'unica primitiva per la strutturazione dei dati è l'array dichiarato con dimensioni definite staticamente.

Costanti

Interi - descritti dalla seguente grammatica:

<intero> ::= [<segno>] <intero-senza-segno>

<segno> ::= "+" "-"

<intero-senza-segno> ::= <cifra> <intero-senza-segno> <cifra>

<cifra> ::= "0" "1" ... "9"

Reali - descritti secondo le due seguenti forme: <reale> ::= <forma-virgola-fissa>

<forma-virgola-mobile> a) nella forma a virgola fissa si usano cif

re decimali ed il punto
decimale \bar{z} obbligatorio

```
<forma-virgola-fissa>::=[<parte-intera>]<parte-decimale>
<parte-intera>::=<intero> <segno>
<parte-decimale>::="."<intero-senza-segno>
```

esempi: .123 45.34 -0.32 -.32 0.

b) nella forma a virgola mobile si utilizza una notazione esponenziale

```
<forma-virgola-mobile>::=<parte-intera><esponente>:
<Forma-virgola-fissa><esponente>
<esponente>::="E"<intero> "D"<intero>
```

esempi:

4.13E12 413E10 4.13*10 413E-10

Le cifre che descrivono un reale (non l'esponente) sono chiamate cifre significative. Il loro numero \bar{z} vincolato alla rappresentazione fisica utilizzata nella memoria. f possibile aumentare questo numero e quindi l'accuratezza (o la precisione) della descrizione usando i reali in doppia precisione. Nella forma a virgola fissa, una costante reale in doppia precisione si distingue unicamente dal numero di cifre; in quella a virgola mobile occorre usare per l'esponente il carattere "D" (l'"E" proprio dei reali in precisione semplice).
Complessi: - sono descritti mediante coppie di reali racchiuse fra parentesi;

```
<complesso>::="("<reale>",<reale>)"
esempi: (3.14,.76342576548D-12) (.3 4.2.)
Booleani - hanno la seguente rappresentazione:
<booleano>::=.TRUE.I.FALSE.'
```

3. LE DICHIARAZIONI

Le dichiarazioni delle variabili possono essere esplicite o implicite. Nelle dichiarazioni implicite il tipo viene definito dalla lettera iniziale del nome della variabile. Cio \bar{z} se il nome inizia con:

I, J, K, L, M, N viene assunto come identificatore d'una variabile intera;

in ogni altro caso viene assunto come l'identificatore d'una variabile reale in singola precisione.

Le dichiarazioni esplicite sono le seguenti:

INTEGER per dichiarare variabili intere; - REAL per variabili reali; - DOUBLE PRECISION per le variabili in doppia precisione; - COMPLEX per le variabili complesse; - LOGICAL per le variabili booleane.

Per esempio il seguente frammento di dichiarazioni definisce le variabili. RESTO di tipo intero, ID1 ed ID2 reali, ACCA e KAPPA reali in doppia precisione e SPETTRO complessa.

```
INTEGER RESTO
REAL ID1, ID2
DOUBLE PRECISION ACCA, KAPPA
COMPLEX SPETTRO
```

La dichiarazione DIMENSION va usata per dichiarare gli array. Questi vanno sempre dichiarati, definendo le dimensioni ed il tipo degli elementi. Il tipo pu` essere definito implicitamente secondo la regola delle iniziali ed in tal caso le dimensioni vanno specificate mediante la dichiarazione DIMENSION. Sia le dichiarazioni esplicite che la DIMENSION specificano il numero delle dimensioni e l'estremo superiore di ognuna, l'estremo inferiore ` sempre 1. L'indice dell'array ` di tipo intero (positivo). Esempi:

```
DIMENSION A(32,32) I(32)
DOUBLE PRECISION MATRIX(4,4,4)
```

La prima dichiarazione riguarda un array A bidimensionale 32x32 con elementi reali, la seconda un array MATRIX tridimensionale di 4x4x4 elementi reali in doppia precisione.

L'uso di dichiarazioni implicite rende il programma meno leggibile e consente di introdurre errori non individuabili dal compilatore. Per esempio immaginando di possedere una variabile di tipo intero dichiarata nel seguente modo:

```
INTEGER PIPPO
```

se durante la stesura del programma si scrivesse la seguente assegnazione:

```
PIPPO = . . .
```

il compilatore FORTRAN assumerebbe PIPPO come identificatore di un

dispensa.txt

a nuova variabile di tipo reale ed a causa della conversione automatica dei tipi nelle assegnazioni (che vedremo nel seguito) cos" non sarebbe rilevato nessun errore. Se il programma fosse composto da migliaia di linee non sarebbe cosa banale individuare l'errore.

Altre due dichiarazioni fondamentali in FORTRAN sono la COMMON e l'EQUIVALENCE. La prima sar` discussa dettagliatamente nel paragrafo relativo ai sottoprogrammi, infatti serve per dichiarare esplicitamente le variabili condivise da pi` sottoprogrammi. La dichiarazione EQUIVALENCE consente di associare la stessa zona di memoria a pi` variabili, in modo da riutilizzare zone di memoria. Le variabili che condividono la stessa area devono appartenere tutte al sottoprogramma o programma in cui viene usata l'EQUIVALENCE. La sintassi e`:

```
EQUIVALENCE ("<lista-var>") "<", ("<lista-var>") ">");
```

ove

<lista-var> e` una lista di nomi di variabili semplici o di singoli elementi di array

Esempio:

```
EQUIVALENCE (A,B,C(1),D(3,1))
```

La stessa zona di memoria viene usata per la rappresentazione delle variabili A, B, dell'array monodimensionale C a partire dal suo primo elemento, nell'array bidimensionale D a partire dal primo elemento della terza riga.

4. LE ESPRESSIONI

Le costanti e le variabili possono comparire nelle espressioni insieme ai seguenti operatori:

OPERATORE	SIGNIFICATO
**	Elevamento a potenza
*	Moltiplicazione
/	Divisione
+	Addizione
-	Sottrazione

La precedente tabella stabilisce anche l'ordine di priorit` di esecuzione delle operazioni (dall'alto verso il basso), inoltre operazioni con lo stesso ordine di priorit` vengono

dispensa.txt

eseguite da sinistra verso destra, eccetto nel caso dell'elevamento a potenza. Per alterare l'ordine di esecuzione si usano le parentesi tonde.

Lo standard del linguaggio consente l'uso di operazioni solo con operandi dello stesso tipo (fuorchè nel caso di reali e complessi), in effetti molti compilatori rispettano il meccanismo di conversione dei tipi rappresentato nella seguente tabella:

+ - * / INTERO REALE DOPPIA-PR. COMPLESSO

INTERO	INTERO	REALE	DOPPIA-PR.	COMPLESSO
REALE	REALE	REALE	DOPPIA-PR.	COMPLESSO
DOPPIA-PR.		DOPPIA-PR.	DOPPIA-PR.	COMPLESSO COMPLESSO
COMPLESSO		COMPLESSO	COMPLESSO	COMPLESSO COMPLESSO

In genere, per l'elevamento a potenza valgono le stesse regole con l'eccezione che un complesso non può mai comparire come esponente.

Gli operatori relazionali del linguaggio sono:

OPERATORE	SIGNIFICATO
.EG.	predicato di uguaglianza
.NE.	predicato di disuguaglianza
.GT.	predicato di maggiorazione
.LT.	predicato di minorazione
.LE.	predicato di minore o uguale a
.GE.	predicato di maggiore o uguale a

Il linguaggio consente anche l'uso dei seguenti operatori logici (la priorità di esecuzione è dall'alto verso il basso):

OPERATORE	SIGNIFICATO
.NOT.	negazione logica
.AND.	prodotto logico
.OR.	somma logica

L'esecuzione di espressioni con operatori eterogenei rispetta la seguente tabella di priorità (dall'alto verso il basso):
operatori aritmetici
operatori relazionali
operatori logici

Esempio: $I1 + I2.GT.100.AND.B.OR.NOT.(A.OR.C.AND.D)$ viene dapprima calcolata la somma fra le due variabili intere $I1$ e $I2$ per confrontarla con la costante intera 100, il valore logico

dispensa.txt

ottenuto viene moltiplicato logicamente con la variabile logica B, il prodotto viene sommato alla negazione dell'espressione logica fra parentesi, cioé la somma della variabile A col prodotto di C e D.

I singoli componenti di un array sono riferiti attraverso il nome dell'array e la specifica dell'indice racchiusa fra parentesi tonde. Il valore dell'indice è dato dal risultato di una espressione intera.

Esempio: gli elementi dell'array quadrato 2x2 di nome A, sono selezionati con le seguenti notazioni:

A(1,1) A(1,2)
A(2,1) A(2,2)

La seguente specifica è corretta purché le variabili I,J,K siano variabili intere:

A(I/J,4-K)

5. I COMANDI

L'assegnazione è l'istruzione basilare che consente di modificare i valori delle variabili.

La sintassi, in FORTRAN, è la seguente:

<assegnazione> ::= <variabile> "=" <espressione>

Dove la <variabile> è il nome di una variabile semplice oppure la selezione di un singolo elemento di array. Non è necessario che la variabile e l'espressione abbiano lo stesso tipo, infatti

prevista una conversione automatica del tipo dell'espressione per adattarlo a quello della variabile. Nel caso che il tipo della variabile sia intero e l'espressione sia reale, viene considerata la parte intera del valore dell'espressione. Se la variabile

reale mentre l'espressione è in doppia precisione, la rappresentazione di quest'ultima viene convertita per troncamento in precisione semplice. Se la variabile è complessa, mentre

l'espressione è reale, verrà modificata solo la parte reale e la parte immaginaria sarà azzerata. Infine, se la variabile è reale o intera e l'espressione di tipo complesso, sarà assegnata solo in parte reale (previa eventuale conversione per troncamento). f consentito

dispensa.txt

inoltre assegnare valori logici a variabili logiche.

Esempio: $N1(3,L)=ALFA(I)+BETA(I,K)*9.81$

All'elemento L-esimo della terza riga di N1, viene assegnato il valore della somma dell'I-esimo elemento di ALFA con il prodotto fra la costante reale e 9.81 e il K-esimo elemento della I-esima riga di BETA.

L'istruzione STOP determina l'arresto dell'esecuzione, cio la terminazione del programma.

Deve comparire almeno una istruzione di questo genere in ogni programma. La sintassi :

STOP

L'istruzione CONTINUE  un'istruzione senza alcun effetto. La sintassi :

CONTINUE

viene di solito usata per specificare la chiusura di un ciclo definito da un'istruzione DO, di cui si parler nel seguito.

In FORTRAN l'ordine di esecuzione delle istruzioni  strettamente sequenziale e tutte le volte che il programmatore lo vuole alterare, pu usare una istruzione di salto che specifiche l'istruzione da eseguire successivamente. Le istruzioni di salto sono di due tipi:

salto incondizionato (vengono eseguiti ogni volta che il flusso del controllo raggiunge l'istruzione;

salto condizionato (sono eseguiti solo al verificarsi di certe condizioni e secondo certe modalit).

GOTO incondizionato

La sintassi : GOTO <n> dove n  una costante intera positiva. L'effetto  di trasferire il controllo all'istruzione di etichetta n.

GOTO calcolato

La sintassi : GOTO (" $<n1>$ ", " $<nK>$ "), " $<variabile-intera>$ dove le parentesi racchiudono una K-upla di costanti intere positive. Il valore della variabile al momento dell'esecuzione deve essere usuale a quello di una delle K costanti e selezioner

dispensa.txt

l'etichetta
dell'istruzione cui sar` trasferito il controllo.

Esempio: GOTO (12,3,23,100),J

IF-logico`

IF ("<espressione-logica>") <istruzione-eseguibile>

f l'istruzione che consente di scegliere lun'alternativa, eseguend
o una data istruzione se
una certa condizione booleana \checkmark
verificata. L'istruzione eseguibile pu` essere: un'assegnazione,

un'istruzione di ingresso/uscita, un GOTO, una STOP o una CONTINUE

L'IF logico non \checkmark una istruzione di salto vera e propria anche se
spesso \checkmark usata per
eseguire salti condizionati.

Esempi (supponiamo che le variabili A e C siano di tipo reale e l'
array B sia composto di
tre elementi booleani):

IF (A.GT.C) GO TO 120 IF (B(1).OR.B(2)) B(3)=.TRUE.

IF-aritmetico`

IF ("<espressione-numerica>") <n>,"<z>","<p>

dove <n>,<z>,<p> sono tre costanti intere.

L'IF aritmetico consente di scegliere fra tre alternative, basando
si sul valore di
un'espressione aritmetica. Se il valore dell'espressione \checkmark minore
di zero, il controllo
verr` passato all'istruzione di etichetta <n>, se invece \checkmark zero,
verr` trasferito
all'istruzione di etichetta <z>, altrimenti all'istruzione di etic
hetta <f>.

Esempio: IF (RICAVO-SPESA) 100,200,300

Esempio: Programma per il calcolo del fattoriale

Presentiamo ora un semplice programma per illustrare alcune delle
nozioni fin qui esposte.

Il programma calcola il fattoriale di un intero positivo.

INTEGER FATT,N

C Si suppone di leggere un intero positivo in
C ingresso e di memorizzarlo in N

dispensa.txt

```
FATT=1
I=0
200 IF (N.GT.O) GOTO 100
```

C Si supponga di stampare il risultato, cioè FATT

```
STOP
```

```
100 N=N-1
    I=I+1
    FATT=FATT*I
    GOTO 200
```

Nel programma le variabili FATT ed N sono dichiarate esplicitamente, mentre la variabile I è dichiarata implicitamente (di tipo intero). Mancano le istruzioni di ingresso-uscita, al loro posto sono stati inseriti dei commenti. Il programma termina la prima volta che la condizione dell'IF logico è falsa.

L'istruzione DO

L'unica istruzione strutturata in FORTRAN è il DO, ovvero il comando per l'esecuzione di un ciclo di istruzioni la cui sintassi è:

```
DO <n> <var-intera>="<espr1>","<espr2["."<passo>]
```

La variabile <var-intera> è utilizzata come contatore per il ciclo, <espr1> è il suo valore iniziale ed <espr2> quello finale, il <passo> è una qualunque espressione intera e se non specificato è considerato unitario. Il DO viene eseguito almeno una volta in ogni caso. Il corpo del DO, cioè le istruzioni da eseguire interattivamente, sono quelle comprese tra l'istruzione successiva al DO e quella etichettata con <n>. Per il DO valgono le seguenti considerazioni:

È possibile che un'istruzione del corpo preveda un salto all'esterno, mentre non è possibile entrare dentro al ciclo se non eseguendo l'istruzione DO.

- Non è possibile all'interno del corpo, riassegnare la variabile indice, né il valore dell'espressione finale e né il passo.

- L'istruzione finale del ciclo deve essere una CONTINUE.

Esempio: il seguente programma è una rielaborazione del precedente e per il calcolo del

dispensa.txt
fattoriale, questa volta utilizzando un DO:
INTEGER FATT,N

C Si supponga di leggere un intero in
C ingresso e di memorizzarlo in N

```
FATT=1
DO 100 I=1,N
FATT=FATT*I
100 CONTINUE
```

C Si supponga di stampare FATT

STOP

6. L'INGRESSO/USCITA

Nelle istruzioni di ingresso/uscita occorre specificare il codice della periferia addetta alle operazioni ed il formato dei dati espresso dalle dichiarazioni di FORMAT.

Dichiarazioni di FORMAT

FORMAT (<lista di specifiche>)

La lista delle specifiche descrive le caratteristiche della rappresentazione dei dati. La dichiarazione individua una registrazione in base alla morfologia dei suoi campi. Le specifiche sono:

- "I" usata per gli interi
- "E" usata per i reali in forma esponenziale
- "D" per la doppia precisione in forma esponenziale
- "F" per i reali
- "X" per lasciare spazi (bianchi)
- "H" per stampare stringhe di caratteri
- "A" per stringhe di caratteri
- "L" per i booleani

Specifica I: "I"<n>

L'<n> è un intero positivo che specifica con quante cifre decimali è rappresentato il dato. Per esempio I4 indica che l'intero è composto di 4 cifre. Esempi di applicazione sono dati nei paragrafi relativi alle istruzioni di lettura e di stampa.

Specifica E: "E"<n>,"<n2>

<n1> è il numero complessivo delle cifre utilizzate nella rappresentazione ed <n2> quello delle cifre della parte decimale. Occorre tener presente

dispensa.txt

te che l'esponente occupa sempre 4 colonne (nella forma E+cc dove i c sono cifre decimali) e che 2 caratteri vanno utilizzati per il punto e per il segno, sicchŽ vale la regola n1>n2+6. In generale valgono le seguenti:

- 1) se il segno Ž + pu~ essere omoesso;
- 2) se il segno dell'esponente Ž presente pu~ essere omessa la lettera E; 3) l'esponente pu~ avere anche una sola cifra;
- 4) non devono essere lasciati spazi bianchi alla destra dell'esponente; 5) in uscita l'esponente viene stampato nella forma E+cc oppure +cc (a seconda del compilatore ed il segno + pu~ essere omoesso).

Specificazione D: "D"<n1>","<n2>

f analoga alla E ed Ž utilizzata per i reali in doppia precisione.

Specificazione F: "F"<n1>","<n2>

f usata per la rappresentazione dei reali, in singola o doppia precisione, in virgola fissa. Gli interi <n1> ed <n2> hanno lo stesso significato che avevano nelle due specifiche precedenti, inoltre n1>n2+2. Il formato F produce uscite pi□ facili da leggere, ma richiede la conoscenza precisa delle dimensioni massime per la rappresentazione del dato. Infatti qualora <n1> non sia sufficiente per la rappresentazione complessiva, il campo verr^ riempito con asterischi. Se in lettura il dato contiene il punto decimale, verr^ interpretato indipendentemente dalla specifica FORMAT. Se invece il punto decimale non Ž presente, viene automaticamente inserito nella posizione indicata dalla specifica.

Specificazione X: <n>"X"

Serve per lasciare n spazi bianchi o per ignorare n colonne.

Specificazione H: <n>"H"

Serve per stampare delle frasi di n caratteri. Dopo la H va scritta immediatamente la frase. Inoltre all'inizio di una riga di stampa, l'uso del descrittore H specifica un eventuale spostamento del carrello della stampante.

Specificazione A: "A"<n>

Usata anch'essa per stringhe alfanumeriche, ma in ingresso. I caratteri letti vengono immagazzinati nelle variabili specificate dalla istruzione di lettura, senza alcuna conversione. Se le variabili verranno poi usate in espressioni aritmetiche, il loro contenuto verrà considerato come quantità numerica, secondo il codice utilizzato dall'elaboratore.

Specifica L: "L"<n>

Serve a caratterizzare la rappresentazione dei booleani. In ingresso è sufficiente che il campo indicato dalla specifica inizi con una T o con una F, per assegnare le rappresentazioni relative. In uscita vengono di solito stampate solo le T o le F.

Nelle dichiarazioni di FORMAT viene di solito usata una barra "/" per indicare che occorre passare ad un successivo record. In tal modo più record possono essere descritti da un unico FORMAT.

Ad ognuna delle specifiche per rappresentazioni numeriche, può essere premesso un intero positivo avente la funzione di ripetitore. Per manipolare complessi si usa una coppia di descrittori per reali.

L'istruzione di lettura

Per leggere un dato qualsiasi occorre specificare:

- 1) Cosa si vuol leggere e dove lo si vuol memorizzare;
- 2) Da quale lettore occorre leggerlo;
- 3) Con quale formato è rappresentato il dato.

Per specificare tutto ciò si usa la seguente istruzione:

READ ("`<codice-lettore>`", "`<etichetta-FORMAT>`") `<lista-variabili>`
Esempio: READ (5,100) K

100 FORMAT (I4)

l'istruzione READ specifica che la lettura verrà effettuata tramite la periferica di codice

"5" e secondo il formato dei dati specificato dal FORMAT di etichetta

dispensa.txt

ttà "100", inoltre la registrazione letta verrà memorizzata nella lista costituita dalla sola variabile intera (supponendo che non sia stata già esplicitamente dichiarata) K. Il FORMAT specifica che la registrazione è composta da un solo intero di 4 cifre. Si noti che se si vuol leggere l'intero 25 occorre digitarlo come in fig.1. Se invece fosse digitato come in fig.2, K verrebbe a contenere la configurazione dell'intero 250.

L'istruzione di scrittura

La sintassi dell'istruzione di scrittura WRITE è analoga a quella dell'istruzione READ:

```
WRITE ("<codice-stampante>", "<etichetta-FORMAT>") "<lista variabili>"
```

La lettura o la stampa di un array possono essere effettuate mediante uno o più DO, o per alcuni compilatori con una sola istruzione. Se l'array ha più indici, nel caso di operazioni di ingresso/uscita effettuate con una sola istruzione, varia rapidamente l'indice di sinistra. Per esempio, nel caso del seguente frammento di programma:

```
INTEGER NVET(3,2)
```

```
WRITE (6,50) NVET
```

```
50  FORMAT (1H ,6(I4,3X))
```

la stampa prodotta seguirà il seguente ordine:

```
NVET(1,1) NVET(2,1) NVET(3,1) NVET(1,2) NVET(2,2) NVET(3,2)
```

Un altro valido metodo per l'ingresso/uscita di array consiste nell'uso di DO impliciti. Per esempio:

```
INTEGER W(3,2)
```

```
WRITE (6,20) ((W(I,J),J=1,2)I=1,3)
```

```
20  FORMAT (1H,2(I4,3X))
```

dispensa.txt

dove viene richiesta l'uscita dell'array W, stampando una coppia di interi su ogni riga e facendo variare più velocemente l'indice di destra (J).

Si noti che il seguente frammento non avrebbe lo stesso comportamento:

```
INTEGER W(3,2)

      DO 10 I=1,3
      DO 10 J=1,2
      WRITE (6,20) W(I,J)
10    CONTINUE
20    FORMAT (1H, 2(I4,3X))
```

infatti, ad ogni passata del DO corrisponde l'uso di un nuovo WRITE e su ogni riga viene stampato un solo intero.

7. I SOTTOPROGRAMMI

Nel FORTRAN sono previsti due tipi di sottoprogrammi compilabili separatamente rispetto al programma principale, le FUNCTION e le SUBROUTINE, e una particolare istruzione funzionale. Quest'ultima consente di ridefinire delle espressioni attraverso un nome.

Inoltre è possibile utilizzare le seguenti funzioni di libreria:

FUNZIONE	TIPO	TIPO
NOME MATEMATICA ARGOMENTI	FUNZIONE	
SIN(X) seno	REAL (radianti)	REAL
COS(X) coseno	" "	"
TAN(X) tangente		
SINH(X) seno iper.		
COSH(X) cos. iper.	" "	"
TANH(X) tan. iper.		
ALOG(X) log.naturale	REAL	"
EXP(X) esponenziale	REAL o INTEGER	
SQRT(X) radice quad.		
CBRT(X) radice cubica		
SGR(X) elev. quadrato		
ABS(X) valore assol.	" "	REAL o INTEGER

Istruzione funzionale

Oltre alle summenzionate funzioni, in FORTRAN è possibile usare nuove funzioni corrispondenti a singole espressioni (aritmetiche o logiche) cui si è voluto associare un nome e dei parametri formali. A tal fine si fa uso della seguente sintassi:

```
<nome-fun> "(("<lista-argomenti>"))"="<>espressione>
```

per esempio: $FN(X,Y)=\sin(X)^2-\cos(X)^2$ dove viene definita la nuova funzione "FN(X,Y)".

Il tipo dei parametri e quello della funzione vengono definiti secondo le usuali regole di dichiarazione implicita ed esplicita. Nel caso che il tipo dell'espressione sia differente da quello della funzione, esso verrà convertito secondo le regole dell'assegnazione già viste precedentemente. I nomi dei parametri sono fittizi, senza alcun rapporto con omonime variabili del programma. Questo tipo di funzioni sono locali ai sottoprogrammi in cui compaiono e vanno inserite prima di ogni altra istruzione eseguibile. L'utilità di questo tipo di istruzioni sta nel sintetizzare forme ricorrenti dell'elaborazione.

La sintassi di questi sottoprogrammi è:

```
<eventuale-tipo> FUNCTION <nome> "(("<lista-argomenti>"))"
```

```
  .
  .
  .
RETURN
  .
  .
END
```

Le FUNCTION sono sottoprogrammi compilabili separatamente ed in grado di restituire un semplice valore (numerico o logico). Vengono attivate all'interno di una espressione attraverso l'uso del loro nome con la specifica dei caratteri attuali. Ogni FUNCTION, oltre alle istruzioni che descrivono il comportamento del sottoprogramma, deve possedere almeno una istruzione RETURN, il cui compito è di trasferire il controllo al programma chiamante.

Illustriamo un semplice esempio di FUNCTION relativo al programma

dispensa.txt

per il calcolo del
fattoriale:

Programma principale:

```
      READ (5,100) I
100   FORMAT (I4)
      IF (I.LT.0) GOTO 300
      J=FATT(I)
      WRITE (6,200) I,J
200   FORMAT (18H,' IL FATTORIALE DI ',I4,4H,' E' ',I4)
300   STOP
      END
```

Il sottoprogramma:

```
      INTEGER FUNCTION FATT (K)
      FATT=1
      DO 100, I=1,K
      FATT=FATT*I
100   CONTINUE
      RETURN
      END
```

Le SUBROUTINE

La sintassi Ĺ:

```
*SUBROUTINE <nome> "("<lista-argomenti>")"
```

```
RETURN
```

```
END*
```

Le SUBROUTINE sono analoghe alle procedure del PASCAL. Come le FUNCTION possono essere composte da un numero qualsiasi di istruzioni, di cui almeno una deve essere una RETURN, e sono compilabili separatamente. Vengono attivate attraverso una esplicita istruzione di chiamata:

```
CALL <nome> "("<lista-parametri-attuali>")"
```

Calcoliamo il fattoriale mediante un sottoprogramma SUBROUTINE. Nel programma principale dell'esempio precedente, cambieremo soltanto l'istruzione J=FATT(I), sostituito dalla seguente:

```
CALL FATT(I,J)
```

Invece il sottoprogramma sar :

```
SUBROUTINE FATT (K,N)
```

dispensa.txt

```
      N=1
      DO 100, I=I,K
      N=N*I
100  CONTINUE
      RETURN
      END
```

Il passaggio dei parametri \bar{z} è effettuato unicamente per riferimento (detto anche per indirizzo). I parametri attuali devono corrispondere in numero ordine e tipo ai parametri formali. Affinchè un sottoprogramma possa essere passato come argomento, deve essere dichiarato mediante una EXTERNAL nel programma chiamante, rispettando la seguente sintassi:

```
EXTERNAL <lista-nomi>
```

Così se volessimo utilizzare la SUBROUTINE PIPPO mediante la seguente chiamata: CALL PIPPO (X,Y,SIN); sarebbe necessario dare, all'interno del programma chiamante, la seguente:

```
EXTERNAL SIN.
```

Non è obbligatorio entrare in un sottoprogramma, passando dalla sua istruzione iniziale, infatti è possibile definire dei punti di ingresso al sottoprogramma con la seguente sintassi:

```
ENTRY <nome> (<lista-argomenti>)
```

Così facendo definiamo, all'interno del sottoprogramma, un nuovo sottoprogramma che ne condivide alcuni o tutti i parametri ed a cui è possibile accedere mediante una chiamata esplicita.

Esempio:

```
      SUBROUTINE SOMMA (A,B,C,X)
```

```
      ENTRY SUCC (A)
```

```
      RETURN
```

```
      END
```

In tal caso sono consentite le seguenti chiamate:

```
      CALL SOMMA (X,Y,Z,T)
      CALL SUCC (N)
```

Le SUBROUTINE e le FUNCTION possono anche essere compilate in altri

dispensa.txt

i linguaggi, per esempio
in ASSEMBLER.

I sottoprogrammi esaminati finora usano solo variabili locali, cos

" non potrebbero
comunicare se non attraverso il passaggio dei parametri. Per ovvia

re a questa limitazione
esiste l'istruzione COMMON.

L'istruzione COMMON

La COMMON è l'analogo della EQUIVALENCE per variabili condivise da
più sottoprogrammi.

Infatti essa assegna la stessa area di memoria principale, detta CO
MMON-area a due o più

variabili usate in diversi sottoprogrammi.

COMMON "/"<blocco>"/"<lista> "/"<bloccoN>"/"<listaN> ^

dove <blocco>...<blocco> sono nomi scelti per designare aree diff
erenti e

<lista>...<listaN> sono liste di variabili condivise. In questa si
tuazione la COMMON-area

viene strutturata a blocchi ed a ogni blocco corrisponde una propr
ia lista di variabili. Il

nome del primo blocco pu` non esserci, in tal caso per la prima li
sta di variabili viene

riservata la porzione iniziale della COMMON-area. Per esempio, se
nel programma principale

compare la seguente:

```
COMMON A,B,C
```

mentre in un sottoprogramma, ad essa correlato, abbiamo:

```
COMMON X,Y,Z
```

l'effetto ottenuto è di far condividere alle variabili A,B,C, ed X
,Y,Z la stessa porzione

iniziale di COMMON-area. Se la COMMON avesse definito dei blocchi,
sarebbe stato necessario

usare nel sottoprogramma il nome del blocco sul quale si effettua
la condivisione.

Nel caso di strutture dati, si pu` usare la COMMON per dichiarare
l'array e/o per definire

la sua appartenenza alla COMMON-area; per esempio:

```
COMMON A(30,5)
```

è equivalente a

```
DIMENSION A(30,5) COMMON A
```

Per` se una variabile strutturata compare con indici in una COMMON
non va ridichiarata in

dispensa.txt

una DIMENSION.

Per la corrispondenza dei tipi valgono le stesse regole del passaggio parametri.

L'istruzione COMMON va posta fisicamente all'inizio del programma, assieme alle altre dichiarazioni.

Breve guida al linguaggio FORTRAN 77

Variabili (valori di default)

Iniziali con lettere da I a N	variabili intere, 4 byte in precisione singola
Iniziali con lettere da A a H oppure da O a Z	variabili reali, 4 byte in precisione singola

Variabili (definizioni)

INTEGER A	La variabile A e' definita intera. Aggiungere *8 o *16 dopo INTEGER per la precisione doppia o quadrupla.
REAL A	La variabile A e' definita reale. Aggiungere *8 o *16 dopo REAL per la precisione doppia o quadrupla.
COMPLEX A	La variabile A e' definita complessa. Aggiungere *16 COMPLEX per la precisione doppia.
CHARACTER*N A	La variabile A e' definita come una variabile stringa. La lunghezza in byte della stringa e' data dal numero intero N.
LOGICAL A	La variabile A e' definita come una variabile logica che puo' assumere solo due valori TRUE o FALSE A=.TRUE. A=.FALSE.

Vettori

DIMENSION A (N1),B(N2,N3)	Le variabili A e B sono definite come vettori di dimensioni N1 e N2 x N3 rispettivamente.
--------------------------------------	---

Operazioni aritmetiche

A + B	somma di A piu' B
A - B	differenza tra A e B
A * B	moltiplicazione di A e B
A / B	divisione di A e B
A**B	elevamento di A alla potenza B

Funzioni matematiche

DO	Ripete i comandi1 fino a che l'indice
INDX=N1,N2,INCR	INDX non supera il valore N2. Il primo valore di INDX (intero) e' N1 (intero) ad ogni ripetizione del comando questo valore viene incrementato del valore INCR (intero) fino a quando non e' maggiore di N2 (intero). Se INCR non e' esplicitato viene utilizzato il valore di default INDX=1. INCR puo' anche assumere valori interi negativi. In questo caso N2 sara' minore di N1 e la ripetizione dei comandi terminera' quando il valore di INDX sara' minore di N2.
comandi1	
ENDDO	
GOTO nmr	Invia l'esecuzione al comando individuato dal numero di linea <i>nmr</i> .

Comandi di input -- output

READ(5,1000) A,B	Leggi dall'unita' 5 i valori delle variabili A e B secondo il formato definito dal comando 1000
READ(5,*) A,B	Come sopra. Il formato adesso e' libero, fissato dalla dichiarazione delle variabili A e B.
READ (5,'(f10.5,1x,f10.5)') A,B	Come sopra. In questo caso il formato di lettura e' specificato nel secondo argomento del comando READ.
WRITE(8,1000) A,B	Scrivi sull'unita' 8 i valori delle variabili A e B secondo il formato definito dal comando 1000. L'utilizzazione dei formati e' analoga a quella del comando READ.
OPEN (UNIT=8,FILE='fn.dat',STATUS='OLD')	Apri l'unita' numero 8, il cui nome e' fn.dat e il cui status OLD dice che esiste gia'. All'unita' puo' essere assegnato qualsiasi numero intero. Il nome del file puo' essere assegnato anche con una variabile stringa. Le possibili opzioni di STATUS sono: OLD, il file gia' esiste, NEW, il file viene creato durante l'esecuzione, UNKOWN, se il file non esiste viene creato.

Formati di lettura e scrittura

FORMAT	E' il comando che indica il formato di scrittura e/o lettura.
I5	Per numeri interi. Il 5 indica che sono allocati 5 spazi per la lettura/scrittura, uno di questi e' utilizzato per il segno.
F10.5	Per numeri reali. Sono allocati dieci spazi di cui 5 per i decimali. Bisogna considerare che in questi dieci spazi sono inclusi il punto ed il segno.
E12.5	Per numeri reali espressi in notazione scientifica. Sono allocati dodici spazi di cui 5 per i decimali. Nei dodici spazi sono inclusi il punto, il segno, il segno dell'esponente, la E dell'esponente e due cifre dell'esponente.

ASSIGN *label* **TO** *variable*
BACKSPACE {*unitspec* |
 ([UNIT=]*unitspec*
 [, ERR=*errlabel*]
 [, IOSTAT=*iocheck*])
BLOCK DATA [*blockdataname*]
CALL *sub* [(*actuals*)]
CHARACTER [*.chars*] *vname* [*.length*](*dim*)
 [, *vname* [*.length*](*dim*)
CLOSE ([UNIT=]*unitspec*
 [, ERR=*errlabel*]
 [, IOSTAT=*iocheck*]
 [, STATUS=*status*])
COMMON [(*cname*) /] *nlist*, [(*cname*) /*nlist*]...
COMPLEX *vnam* [(*dim*)]
 [, *vname* [(*dim*)]]...
CONTINUE
DATA *nlist /clist* [[,] *nlist /clist*]...
DIMENSION *array* ([*lower*:]*upper* [, { [*lower*:]*upper* }])
DO [*label* [,]] *dovar* = *start*, *stop* [, *inc*]
DOUBLE PRECISION *vname* [(*dim*)]
 [, *vname* [(*dim*)]]...
ELSE
statementblock
ELSE IF (*expression*) **THEN**
statementblock
END
END IF
ENDFILE {*unitspec* |
 ([UNIT=] *unitspec*
 [, ERR=*errlabel*]
 [, IOSTAT=*iocheck*] }
ENTRY *ename* [([*formal* [, *formal*]...)]
EQUIVALENCE (*nlist*) [, (*nlist*)]...
EXTERNAL *name* [, *name*]...
FORMAT [*editlist*]
 [*type*] **FUNCTION** *func* ([*formal*] [, *formal*]...)
GOTO *variable* [[,] (*labels*)]
GOTO (*labels*) [,] *n*
GOTO *label*

IF (*expression*) *label1*, *label2*, *label3*

IF (*expression*) *statement*

IF (*expression*) **THEN**

statementblock1

[**ELSE IF** (*expression*) **THEN**

statementblock2]...

[**ELSE**

statementblock3]

END IF

IMPLICIT *type* (*letters*) [, *type* (*letters*)]... **INQUIRE** ([[**UNIT**=]*unitspec* | **FILE**=*file*] [, **ACCESS**=*access*]
[, **BLANK**=*blank*] [, **DIRECT**=*direct*] [, **ERR**=*errlabel*] [, **EXIST**=*exist*] [, **FORM**=*form*]
[, **FORMATTED**=*formatted*] [, **IOSTAT**=*iocheck*]
[, **NAME**=*name*] [, **NAMED**=*named*]
[, **NEXTREC**=*nextrec*] [, **NUMBER**=*num*] [, **OPENED**=*opened*]
[, **RECL**=*recl*] [, **SEQUENTIAL**=*seq*] [, **UNFORMATTED**=*unformatted*])

INTEGER *vname* [(*dim*)]

[, *vname* [(*dim*)]] ...

INTRINSIC *names*

LOGICAL *vname* [(*dim*)]

[, *vname* [(*dim*)]] ...

OPEN ([[**UNIT**=]*unitspec* [, **ACCESS**=*access*] [, **BLANK**=*blanks*]

[, **ERR**=*errlabel*] [, **FILE**=*file*]

[, **FORM**=*form*] [, **IOSTAT**=*iocheck*]

[, **RECL**=*recl*] [, **STATUS**=*status*])

PARAMETER (*name*=*constexpr* [, *name*=*constexpr*]...)

PAUSE [*prompt*]

PRINT { ., | *formatspec* | } [, *iolist*]

PROGRAM *program-name*

READ { *formatspec*, | ([[**UNIT**=] *unitspec* [, [**FMT**=]
formatspec] [, **END**=*endlabel*] [, **ERR**=*errlabel*]
[, **IOSTAT**=*iocheck*] [, **REC**=*rec*])} *iolist*

REAL *vname* [(*dim*)]

[, *vname* [(*dim*)]] ...

RETURN [*ordinal*]

REWIND { *unitspec* |

[[**UNIT**=]*unitspec*

[, **ERR**=*errlabel*]

[, **IOSTAT**=*iocheck*])}

SAVE [*names*]

STOP [*message*]

SUBROUTINE *subr* ([[*formal* [, *formal*]...])

WRITE ([UNIT=] *unitspec*
[, [FMT=] *formatspec*]
[, ERR=*errlabel*]
[, IOSTAT=*iocheck*]
[, REC=*rec*])
iolist