Programmazione su Fortran

Queste dispense sono state scritte per fornire un rapido volume di consulto dei comandi più usati per una programmazione di livello base con Fortran. Sono comprensivi dunque di osservazioni e commenti in funzione dei problemi che ho affrontato nel corso dello studio.
Ringrazio per le spiegazioni sull'uso dei comandi di base il Dott. Robol grazie alle cui dispense ho mosso i primi passi nella programmazione in Fortran.

Per qualsiasi errore o aggiunta contattatemi al mio indirizzo e-mail giulio.pisa@virgilio.it.

Giulio Del Corso

Introduzione (Comandi di base):

Per scrivere il programma si può utilizzare un qualsiasi programma di scrittura (Consiglio GEDIT)

Comando:
gedit nomefile.f90

Una volta salvato dobbiamo semplicemente usare il compilatore per trasformarlo in un eseguibile

Comando:
f95 nomedelfile.f90 -o nome_eseguibile

Osservazione:
Non è necessario che nessuno dei nomi coincida, ne quello del file ne quello dell'eseguibile o quello del programma (A differenza del C).

Per eseguire un file eseguibile compilato in fortran basta aprire la cartella e dare il seguente comando.

Comando:
./nome_eseguibile

Struttura del programma:

PROGRAM nomeprogramma (Non deve necessariamente coincidere con quello del file)
IMPLICIT NONE (Disabilita vecche convenzioni di Fortran)

! Specificazioni (Qui si dichiarano le variabili e i rispettivi tipi)

! Parte esecutiva

! Subroutine (Sottoprogrammi richiamati)

END PROGRAM

Come inserire i commenti:
Utilizzare il ! da li in poi la riga viene trascurata.

I tipi:

Fortran è un linguaggio di programmazione che necessita nella parte dichiarativa la dichiarazione di ogni variabili utilizzata in seguito (Attenzione, compresi gli indici dei cicli DO).

I vari tipi che possono essere dichiarati sono:

Interi:
INTEGER :: nomevariabile , nomevariabile2 , …

Reali:
REAL :: nomevariabile , nomevariabile2 , ...

> **Osservazione 1:**
> Può essere sostituita da DOUBLE PRECISION :: che è una variabile reale con doppia precisione.
>
> **Osservazione 2:**
> Per il linguaggio Fortran non c'è differenza fra i seguenti nomi di variabili:
> Name ; name ; NAme ; NaMe ; NAME
> Quindi evitare di caratterizzare mediante maiuscole una variabile.

Attenzione: così definite le variabili hanno una dimensione standard i 5 byte che è un po' bassa, utilizzeremo dunque il trucco:

```
INTEGER , PARAMETER :: db=KIND(0.DO)
REAL(db) :: real_var
```

Caratteri:
CHARACTER :: nomevariabile , nomevariabile2

La variabile così definita permette di contenere un carattere, l'assegnazione si fa con il comando:
z='t'
Allora la variabile z conterrà il carattere t

Osservazione:
Può essere anche dato come input, allora z memorizzerà la prima lettera inserita.

Osservazione:
La lunghezza di una variabile di questo tipo può essere cambiata nei seguenti modi:
CHARACTER (LEN=5) :: nomevariabile , nomevariabile2
CHARACTER (5) :: nomevariabile , nomevariabile2
CHARACTER (qualsiasi cosa anche nulla) :: nomevariabile , nomevariabile2*i
Assegna la lunghezza *i* a solo le variabili così scritte
CHARACTER (LEN=*) :: nomevariabile , nomevariabile2
Significa che sarà specificato in seguito la lunghezza della stringa

Osservazione:
Se la lunghezza dell'input è maggiore del valore segnato la variabile lo troncherà.
Se la lunghezza dell'input è minore, premendo invio la variabile memorizza quanto scritto.

Attenzione:
Lavorando con CHARACTER se la lunghezza della variabile è maggiore dell'input inserito il programma satura il resto con gli spazi, il problema si presenta dunque quando andiamo a stampare a schermo una di queste variabili perché sarà seguita da una fila di spazi vuoti che rovinano la formattazione.

Osservazione (Estrarre una substring):
Data una variabile CHARACTER x allora il comando:
x(3:11) restituisce una stringa che contiene solamente i valori dal 3° all'11° posto.
Al solito x(:n) è la stringa dall'inizio fino all'n-esimo posto e x(n:) dall'n-esimo alla fine.
Può essere dato come espressione x(h+3:) con h intero.
Può essere usato per rimpiazzare una parte di stringa:
x(3:4)='ab'
(Sostituisce al 3° e al 4° posto i caratteri ab)

Osservazione :
Se la lunghezza di quello da inserire è maggiore allora ne prende la prima parte.
Se la lunghezza della stringa selezionata è maggiore sostituisce al resto degli spazi.

Logico:
LOGICAL :: nomevariabile , nomevariabile2

La variabile così definita può contenere i valori booleani true e false.
Può essere anche inserita mediante il comando READ e in quel caso da tastiera accetta come input esclusivamente t ed f.

Giulio Del Corso

Osservazione generale:
Il valore iniziale di una variabile può essere assegnato già in fase di definizione ponendo y=numero.

Osservazione generale:
Una variabile si può esprimere anche in forma esponenziale
120=12E1

Osservazione generale:
Già in fase di dichiarazione possiamo assegnare valori che sono funzione delle variabili definite precedentemente.

Specificazioni:
TIPOVARIABILE , PARAMETER :: nomeparametro=valore_da_assegnare
(Significa che assegneremo un valore fissato e che questo non verrà modificato nel corso del programma, tipico esempio è pi greco oppure il limite massimo alle iterazioni).

Esempio:
INTEGER , PARAMETER :: limite=30

In pratica non può essere utilizzato per memorizzare un valore, ossia a sinistra di un'espressione.

DIMENSION
Si utilizza questo comando per definire un vettore, la sintassi è la seguente.
TIPOVARIABILE , DIMENSION(...) :: x

Osservazione:
La specificazione DIMENSION fra parentesi deve contenere valori interi che caratterizzano il vettore:
DIMENSION(5) indica un vettore con 5 spazi

Attenzione:
Sono disposti per riga.
DIMENSION(2,3) indica una vettore doppio con rispettivamente 2 e 3 spazi.

Osservazione:
Se vogliamo che i nomi degli spazi siano diversi da quelli standard possiamo utilizzare il comando:
DIMENSION(-3:6) che numera gli spazi con i valori da noi assegnati
(In questo caso -3,-2,-1,0,1,...)

Attenzione:
Se uno spazio non viene inizializzato non assume valore 0 ma rimane semplicemente vuoto.

Osservazione:

La posizione di un vettore può essere richiamata con il comando:

x(n,m) con m che caratterizza quale stringa deve scegliere ed n il posto.

Osservazione importante:

Per poterlo inizializzare in corso d'opera possiamo utilizzare la seguente scrittura in fase dichiarativa:

INTEGER , DIMENSION (:,:,:) , ALLOCATABLE :: x

Ovviamente va bene anche con gli altri tipi.

: per ogni colonna aggiuntiva e le , separano i :

Attenzione:

x dovrà essere coerente con quanto contenuto fra parentesi.

In fase esecutiva, una volta ottenuti i valori che ci servono n,m,k scriviamo il comando:

ALLOCATE(x(n,m,k))

KIND(...)

Può seguire un qualsiasi tipo di variabili.

Utilità del comando stop:

Il comando STOP “stringa di codice” interrompe il programma e restituisce la scritta segnatala, utile per i controlli nel corso di una prima stesura.

Parte esecutiva:

In Fortran non è necessario concludere una frase con (;), questa serve semplicemente a mettere più di un'istruzione sulla stessa riga.

Comandi OPEN/CLOSE

Idea:

Questi comandi servono ad aprire un file da cui ottenere dei dati o sui cui scrivere quello che abbiamo ottenuto.

La sintassi del comando è:

OPEN (UNIT=numero , FILE='nomefile' , ERR=numeroriga , STATUS='status' , ACTION='mode')

Attenzione:

Ricordasi che nomefile, status e mode sono fra virgolette.

Osservazione 1:

UNIT è rappresentato da un numero che caratterizza questa operazione, serve a richiamare rapidamente questa istruzione. (Non va bene il valore 6)

Il comando UNIT=numero può essere direttamente scritto come numero
(UNIT è dato implicitamente).

FILE richiede il nome del file con cui stiamo lavorando
(Attenzione, lo ricerca nella cartella nella quale abbiamo eseguito il programma).
Può anche essergli dato l'intero percorso file (c:\\etc.)

ERR serve nel caso in cui non riesca l'apertura del file, in tal caso il programma salta direttamente alla riga indicata (Serve a fargli evitare le istruzioni che avrebbero riguardato il file non aperto).
Può non essere messo.

STATUS indica cosa fare del file, se si usa il comando UNKNOWN significa che lo sovrascriverà nel caso esista o lo creerà altrimenti.

ACTION può essere READ se è un file di sola lettura o WRITE se è solo di scrittura,
READWRITE se è entrambe.

Osservazione 2:

Per chiudere un file aperto si utilizza il comando:

CLOSE(numero1,numero2,...)

Le due istruzioni più semplici sono READ e WRITE:

READ (*,*) nomevariabile , nomevariabile2 , ...
(Questa legge da schermo un input e lo memorizza nel nome della variabile, deve essere compatibile come tipo, per farglielo leggere basta battere invio dopo ogni numero o lasciare uno spazio fra un numero e l'altro)

Osservazione 1:
È buona convenzione inserire i dati richiesti dal comando:

READ (*,*) x1 , x2 , x3

Con un input del tipo:

4 5 6

Ossia usare le spaziature e non INVIO, questo permette di controllare il numero di variabili inserite riga per riga, una volta usati gli slot infatti ogni altro dato non verrà memorizzato, perciò non rischiamo di occupare gli slot successivi.

Osservazione 2:
Il comando:

READ(*,*)

Fa saltare una riga su terminale, qualsiasi stringa inserita da tastiera verrà ignorata fino al momento in cui viene premuto INVIO

WRITE (*,*) nomevariabile , nomevariabile2 , ...
Stampa su schermo il valore della variabile

Può essere utilizzato per restituire anche una scritta per indicare all'utente cosa deve fare.

WRITE (*,*) “Testo a caso”

Osservazione 0:
“ e ' sono equivalenti

Osservazione 1:
Nel comando WRITE può essere utilizzata al posto di una variabile direttamente un'espressione numerica con il vantaggio di non dover definire una nuova variabile per contenere il valore così calcolato, ad esempio:

WRITE (*,*) a*b
con a e b variabili prima definite.

Osservazione 2:
WRITE (*,*) lascia una riga di spazio se preceduta e seguita da altri comandi WRITE (*,*)

Osservazione 3:
Non sembrano esistere modi intelligenti per scrivere una frase con dentro dei valori ottenuti nel corso del programma.

Esempio:
WRITE (*,*) “Il risultato numerico del prodotto fra “, x , y , “ è: “ , totale

Osservazione 4:
Una volta aperto un file è possibile dire al programma di scrivere il risultato ottenuto sul file. La scrittura è:

WRITE(n,*) con n il numero di UNIT che caratterizza il file.
Attenzione: deve essere aperto in modalità scrittura.
Attenzione:
Per comodità di lettura nel file viene anche scritto quale è il numero dell'outoput nel caso di un ciclo DO.

Osservazione 5:
Il secondo spazio della scrittura WRITE(*,*) indica un'etichetta di un certo tipo di scrittura che non so ancora utilizzare. **DA APPROFONDIRE**

Ciclo Do:
È l'equivalente del ciclo for, la struttura del comando è la seguente:

```
DO index=start,end
        ISTRUZIONI
END DO
```

Osservazione 1:
La variabile index deve essere dichiarate e deve essere di tipo intero, non importa però incrementarla manualmente.

Osservazione 2:
Può essere modificato il passo usando la scrittura:

```
DO index=start,end,step
```

step può essere a sua volta una variabile ma cambiarla nel corso del ciclo DO non altera la lunghezza dello stesso.

Osservazione 3:
Il ciclo DO può essere interrotto mediante il comando EXIT (Lecito, lo rende analogo ad un ciclo while di C).

Ciclo While:
Struttura del comando:

```
DO WHILE (condizione)
        ISTRUZIONI
END DO
```

Attenzione:
Fino a quando la condizione è verificata il ciclo continua, non assegnare dunque condizioni che non possano verificarsi.

IF:
La scrittura rapida per il comando if è:
IF(condizione) ISTRUZIONE

> **Attenzione:**
> Così scritto può ricevere solo un'istruzione.

La scrittura completa invece è:

```
IF (Condizione logica) THEN
        ISTRUZIONI
ELSEIF (Condizione logica) THEN
        ISTRUZIONI
ELSE
        ISTRUZIONI
ENDIF
```

Casi:
Il comando permette di etichettare una serie di casi in maniera rapida, la scrittura è:

```
SELECT CASE (num)
CASE (a,b)     ! Ossia se il caso è esattamente uno di quelli.
        ISTRUZIONI
CASE (c)
        ISTRUZIONI
CASE DEFAULT      ! Equivalente ad ELSE
        ISTRUZIONI
END SELECT
```

Operazioni:

I simboli con cui si indicano le operazioni sono:
Somma: x+y
Sottrazione: x-y
Prodotto: x*y
Divisione: x/y
(Attenzione, la divisione fra interi restituisce il quoziente)
Elevamento a potenza: x**y
(Questo può essere usato anche per fare la radice quadrata elevando alla 0.5)
(Per rappresentare rapidamente le radici n-esime possiamo usare la formulazione:
y=x**(1/k) con k reale)
(Attenzione: -a**b con a reale è sensata mentre (-a)**b non viene accettata)

Osservazione:
Se uno qualsiasi degli operatori è reale allora l'intera operazione verrà spostata sui reali, di conseguenza:
1/0.5=2
Alla stessa maniera se la variabile dentro cui stiamo salvando l'informazione è intera troncherà la parte intera dimenticando quella reale.

Importante (Scelta dell'ordine di esecuzione delle operazioni):

Il Fortran applica la seguente regola:
Quando incontra un'operazione controlla la successiva, se questa ha la stessa importanza esegue la prima, se è di importanza maggiore passa il cursore alla successiva e ripete il controllo.

Ordine di importanza:

Basso:
+ ; -
Medio:
* ; /
Alto:
**

Esempio 1:
3+4*3**2=3+(4*(3**2))

Esempio 2:
3*5*6+2/3**2=15*6+2/3**2=90+2/3**2=90+2/9=90+2.222...=92.222...

Osservazione:
Ovviamente le parentesi cambiano questa regola, in caso di dubbio dunque utilizzarle.

Operazioni logiche:
Possono essere utilizzate per restituire un valore booleano (Attenzione, anche gli input devono essere booleani).
y = x .or. z (Restituisce il booleano corrispondente a x oppure z)

I vari operatori sono:
.or.
.and.
.eqv.
(Confronta SOLO booleani, in pratica a T.eqv.T restituisce T mentre a T.eqv.F restituisce F).
.neqv.
(Restituisce il valore opposto rispetto al precedente)
.not.
(Inverte il valore di verità del booleano. La formulazione è: y = .not.x

Osservazione:
I booleani permettono un confronto fra altri tipi di variabili sfruttando la seguente notazione:
z = x<y (Con z booleano e x,y ad esempio, interi)

Vanno bene come operatori di confronto anche:
< ; > ; <= ; >= ; == ; /= (Diverso)

Attenzione:
Evitare di usare questo comando per confrontare due booleani, per quelli esiste il comando .eqv.

Conversione:

Reali in interi:

INT(x)

(Mantiene la parte intera di x)

NINT(x)

(Intero più vicino al reale x)

FLOOR(x)

(Restituisce il più piccolo intero al di sotto di x)

Interi in reali:

REAL(x)

(Trasforma un intero in un reale, serve a rendere valida un'uguaglianza del tipo y = REAL(x))

Reali in reali:

FRACTION(x)

(Mantiene solo la parte dopo la virgola di x)

Funzioni già presenti:

ABS(x)

SQRT(x)

Possiamo rendere reale un numero intero aggiungendoci .0

SIN(x)

COS(x)

TAN(x)

ASIN(x)

ACOS(x)

ATAN(x)

EXP(x)

LOG(x)

LOG10(x)

(Tranne ABS(x) che accetta anche valori interi tutti gli altri richiedono variabili reali)

Funzioni utili:

MAX(x1,...,x9)

(Restituisce il massimo valore fra queste variabili)

Attenzione:

Possono essere tutti reali o tutti interi ma non alcuni di un tipo e altri di un altro.

Un trucco per confrontarli può essere scrivere il comando:

MAX(REAL(x1),...,REAL(x9))

Osservazione:

L'argomento di una funzione può essere un'espressione o una funzione a sua volta.

Esempio:

MIN(x1,...,x9)

(Restituisce il minimo valore fra queste variabili)

Giulio Del Corso

MOD(x,y)
(Restituisce x-INT(x/y)*y ; nel caso di interi è la classe di resto)

Subroutine:

Sono sottoprogrammi che possono essere richiamati per alleggerire la notazione, formalmente si scrivono come:

```
SUBROUTINE nomesubroutine(parametro)
        ISTRUZIONI
END SUBROUTINE nomesubroutine
```

Osservazione:
Le Subroutine possono essere messe sia dentro il programma che fuori (Consigliabile)

Attenzione:
Fra le istruzioni della Subroutine deve esserci la dichiarazione del tipo della variabile che gli abbiamo passato fra parentesi.

Attenzione:
Stiamo passando con il nome delle variabili dei puntatori, questo significa che non esistono istruzioni di ritorno in quanto si va a modificare il nome del puntatore stesso.
Non è necessario ovviamente che i nomi delle variabili della Subroutine coincidano con i nomi delle variabili che andremo ad inserirci chiamandola.

Attenzione:
Ricordarsi che la Subroutine va chiamata con il comando CALL nomesubroutine.

Ripasso breve dei comandi necessari su Linux:
ls (Mostra quello che c'è all'indirizzo)
ls -l (Idem ma in colonna)

mkdir nomecartella (Crea una cartella con nome indicato)

cd nomecartella (Apre la cartella)
cd (Riconduce alla home)
cd /temp (Per aula informatica, manda alla cartella dei file temporanei)

rm nomefile (rimuove il file indicato)
rm -r nomecartella (Rimuove la cartella)

mv nomeoggetto nomedestinazione (Se è all'interno di quella cartella, altrimenti così sostituisce il nome del file)

Aggiungendo & alla fine di un comando è possibile farlo girare in background

Giulio Del Corso

La dimensione non ho capito come inserirla

Matrici quadrate?
Formato pnm

Giulio Del Corso