Guida Rapida al Fortran77

D. Tambuchi

SOMMARIO. Questo documento vuole introdurre i principali comandi del linguaggio di programmazione Fortran77. Non ha la pretesa di presentare tutti gli aspetti del linguaggio; per ulteriori approfondimenti si rimanda ai testi citati in bibliografia.

1. Aspetti generali del Fortran77

Il linguaggio Fortran77 é adatto principalmente per programmi scientifici, ovvero per programmi di calcolo numerico. Per applicazioni grafiche, elaborazione di testi e gestionali é consigliabile utilizzare altri linguaggi (ad esempio C, C++, Java).

2. Impaginazione del codice sorgente

L'impaginazione del codice, che puó essere scritto con qualsiasi editor di testi, non é libera. Infatti, in un programma scritto in Fortran77 dobbiamo riservare le prime 6 colonne per usi particolari. Piú precisamente, si ha la seguente utilizzazione:

- La prima colonna viene utilizzata per settare delle *righe di commento*; mettendovi un asterisco o la lettera C, tutta la riga diventa una linea di commento.
- Dalla seconda alla quinta linea é possibile inserire dei numeri interi positivi, detti *etichette* utilizzate per salti o per riferimenti
- Nella sesta colonna, si mette un simbolo di continuazione (normalmente il carattere +, oppure delle cifre $1, 2, \ldots$) per indicare che una istruzione si estende su piú righe.
- Dalla settima alla settandaduesima colonna, vengono scritte le istruzioni.
- Le colonne dalla settantatreesima all'ottantesima possono essere utilizzate per brevi commenti

Vediamo ora un esempio:

```
      PROGRAM SALUTO
C        NOME DEL PROGRAMMA (IN FORTRAN77 IL NOME DEL PROGRAMMA
C        NON PUO' SUPERARE I 6 CARATTERI DI LUNGHEZZA)
```

```
C        OSSERVIAMO INOLTRE CHE SE IL PROGRAMMA VIENE SCRITTO CON
C        LETTERE MAIUSCOLE, ESSO PUO' ESSERE COMPILATO CON
C        QUALSIASI COMPILATORE FORTRAN77. L'USO DELLE LETTERE
C        MINUSCOLE RIDUCE QUESTA POSSIBILITA'.

C        QUESTO PROGRAMMA VISUALIZZA UN MESSAGGIO DI SALUTO
C        SUL VIDEO, CHE VIENE VISUALIZZATO 10 VOLTE

      INTEGER I
C        VARIABILE INTERA USATA COME CONTATORE DEL NUMERO
C        DI VISUALIZZAZIONI

      DO 100, I = 1, 10
C        CICLO DEFINITO CHE VIENE ESEGUITO 10 VOLTE.
C        LE ISTRUZIONI DEL CICLO SONO COMPRESE TRA
C        L'ISTRUZIONE DO 100 ... E L'ISTRUZIONE
C        CONTINUE, INDIVIDUATA DALL'ETICHETTA 100
          WRITE (FMT=*, UNIT=*)
     +       ' UN SALUTO A TUTTI GLI UTILIZZATORI DI'
     +       ' QUESTO PROGRAMMA '
 100  CONTINUE

      STOP
C        QUESTA ISTRUZIONE TERMINA IL PROGRAMMA

C        IL TERMINE DEL FILE SORGENTE DEVE ESSERE INDICATO
C        DALL'ISTRUZIONE END
      END
```

3. Compilazione ed esecuzione

Dopo aver scritto il programma con qualsiasi editor (ad esempio si puó utilizzare il `vi`), ed aver salvato il codice sorgente in un file con estensione `.f` o `.for`, si puó scegliere un qualsiasi compilatore `Fortran77`. Si puó ad esempio utilizzare il compilatore gratuito `g77`, disponibile ad esempio in ambiente `LINUX`. Supponiamo che il codice sorgente del programma illustrato nel paragrafo 2 sia stato salvato con il nome `saluto.f`; é allora possibile ottenere il codice eseguibile digitando il comando:

```
g77 saluto.f -o saluto
```

Se non vengono segnalati errori in compilazione, il programma puó essere eseguito con il comando

```
./saluto
```

4. Variabili e costanti

I tipi di variabile del `Fortran77` sono:

- `INTEGER` - Per variabili che devono contenere numeri interi.
- `REAL` - Per variabili che devono contenere numeri reali.

- `DOUBLE PRECISION` - Ancora numeri reali, ma con una maggiore precisione (maggior numero di cifre decimali)
- `COMPLEX` - Per variabili che devono contenere numeri complessi
- `CHARACTER` - Caratteri alfanumerici
- `LOGICAL` - Variabili logiche, che possono assumere solo i valori `.TRUE.` (vero) o `.FALSE.` (falso).

Se si vuol lavorare con stringhe di caratteri (ovvero con variabili contenenti piú di un carattere, e quindi in grado ad esempio di memorizzare dei nomi), occorre dichiarare la lunghezza massima della stringa; ad esempio la dichiarazione:

```
CHARACTER*20 NOME
```

permette di dichiarare una stringa capace di contenere al piú 20 caratteri.

Vediamo ora alcuni tipi di dichiarazione e di assegnazione[1]:

```
REAL X                - dichiarazione di una variabile reale
REAL Y                - dichiarazione di un'altra
                        variabile reale
INTEGER I, J          - dichiarazione di due variabili intere
LOGICAL Q             - dichiarazione di una variabile logica
CHARACTER*14 NOME     - dichiarazione di una variabile stringa
CHARACTER*25 NOME     - un'altra variabile stringa
CHARACTER CAR         - dicharazione di un singolo carattere
COMPLEX Z             - dichiarazione di una variabile
                        complessa

X = 10.0              - assegna ad X il valore 10.0
Y = 12.1              - assegna ad Y il valore 12.1
X = X * Y             - calcola il prodotto tra X ed Y e
                        mette il risultato in X
I = 2                 - assegna il valore 2 ad I
J = 3.14              - ERRORE: NON SI PUO' ASSEGNARE UN
                        VALORE REALE AD UNA VARIABILE INTERA
X = I                 - corretto: l'intero I viene convertito
                        in un numero reale, ed X vale 2.0
J = I**3              - eleva I al cubo e mette il risultato
                        in J
X = X**4              - eleva X alla quarta
Q = .TRUE.            - assegna il valore .TRUE. a Q
Z = (0.0, 1.0)        - Z assume il valore 0.0 + j 1.0
NOME = 'PAMELA ANDERSON'
                      - ERRORE: LA VARIABILE NOME PUO'
                        CONTENERE AL PIU' 14 CARATTERI
NOME = 'LINUS TORVALDS'
                      - ok!
CAR = 'y'             - assegna il carattere 'y' a CAR
CAR = 'no'            - ERRORE: CAR pu\'o contenere un
                        solo carattere
AUTORE = 'GIOSUE'' CARDUCCI'
```

[1] le annotazioni a fianco, precedute da un trattino non sono commenti in stile `Fortran77`

```
                        - il carattere accento ' viene
                          rappresentato con un doppo apice: ''
```

Notiamo che i caratteri o le stringhe assegnati alle variabili `CHARACTER` devono essere racchiusi tra apici. I nomi di variabile devono essere composti da lettere dell'alfabero e da numeri; il primo carattere del nome deve essere una lettera. La massima lunghezza ammessa, come per il nome del programma, é di 6 caratteri. Sono ad esempio errate le seguenti dichiarazioni:

```
INTEGER 2X3          - INIZIA CON UNA CIFRA
REAL PRESSIONE       - NOME TROPPO LUNGO
LOGICAL X_1          - NON VA' BENE IL CARATTERE _
```

Se le variabili non sono dichiarate, sono automaticamente considerate intere se iniziano con i caratteri `I, J, ..., N` e reali se iniziano con un'altra lettera. Ad esempio, é possibile scrivere `AREA = 12.4` e `INDICE = 20` senza dichiarare `AREA` ed `INDICE`. Questa procedura é tuttavia sconsigliata, in quanto la dichiarazione delle variabili rende il programma piú leggibile.

Vediamo ora come si dichiara una costante; a tal scopo occorre specificarne il tipo (come per una variabile), e poi fissarne il valore con l'istruzione `PARAMETER`

```
INTEGER VMAX          - voto massimo di maturita'
PARAMETER (VMAX = 100)

REAL PI               - definizione della costante pi-greco
PARAMETER (PI = 3.14159265)

INTEGER MAXC          - numero massimo di caratteri
                        in una stringa
PARAMETER (MAXC = 30)
```

Tutte le grandezze costanti dovrebbero essere sempre dichiarate in questo modo, e tali dichiarazioni dovrebbero trovarsi subito dopo l'istruzione `PROGRAM`; ció permette di poter facilmente cambiare il valore delle costanti in modo facile in caso di necessitá.

Ad esempio, se volessimo modificare il programma del paragrafo 2 in modo da fargli visualizzare il messaggio per 5 volte (anziché 10), dovremmo cercare l'istruzione `DO` e cambiarla in

```
DO 100, I = 1, 5
```

Se invece viene dichiarata una costante che definisce il numero di esecuzioni del ciclo, ogni modifica al programma risulta immediata. Ció é particolarmente importante per programmi di grandi dimensione, ove é impossibile rintracciare un valore in mezzo a centinaia di migliaia di righe di codice. Il programma andrebbe riscritto come segue (ho tralasciato, per brevitá, i commenti).

```
PROGRAM SALUTO

INTEGER NCICLI
PARAMETER (NCICLI = 5)

INTEGER I

DO 100, I = 1, NCICLI
```

```
      WRITE (FMT=*, UNIT=*)
     +       ' UN SALUTO A TUTTI GLI UTILIZZATORI DI'
     +       ' QUESTO PROGRAMMA '
 100  CONTINUE

      STOP
      END
```

Considerazioni analoghe valgono per tutti gli altri tipi di variabile. Ad esempio, la lunghezza delle stringhe puó essere fissata come una costante:

```
      INTEGER LMAX
      PARAMETER (LMAX = 20)
C        LUNGHEZZA DELLE STRINGHE DI CARATTERI

      CHARACTER*LMAX NOME

      NOME = 'LINUX TORVALDS'
```

5. Operatori aritmetici

I principali operatori aritmetici sono:

- $+$: per la somma di due variabili (intere, reali o complesse)
- $-$: come sopra, per la sottrazione (puó essere utilizzato anche come segno meno davanti ai numeri negativi)
- $\star$: come sopra, per la moltiplicazione
- $/$: come sopra, per la divisione
- $\star\star$: come sopra, per l'elevamento a potenza
- $=$: operatore di assegnazione

Si possono usare le parentesi tonde per cambiare le precedenze degli operatori, secondo le consuete leggi della matematica (in cui ad esempio l'operatore prodotto ha precedenza sull'operatore di somma). Esempi:

```
      INTEGER I, J, K
      REAL X, Y, Z
      DOUBLE PRECISION D, E
      COMPLEX A, B

      I = 2
      J = 2 ** 3        - J vale 8 dopo questa istruzione
      K = (J - I) / I   - K vale 3 dopo questa istruzione

      X = 4.0
      Y = 1.2E-3       - notazione esponenziale; Y vale 0,0012
      D = -0.3443D2    - notazione esponenziale per variabili
                         in doppia precisione; D vale -34.43
      E = 1.0D0        - per le variabili in doppia precisione
                         e' sempre necessario il suffisso D, anche
                         se l'esponente e' 0; E vale 1.0
      Z = X * Y        - Z vale 0,0048 dopo questa istruzione
```

```
A = (1.0, 2.0)   - A = 1.0 + j 2.0
B = (-1.0, 0.8)  - B = -1.0 + j 0.8
A = A - B        - sottrazione tra complessi
                   il risultato viene assegnato alla
                   variabile A che vale 2.0 + j 1.2
```

6. Operatori logici e di confronto

Puó capitare di dover confrontare due variabili, ad esempio per stabilire se due numeri sono uguali, se il primo é maggiore del secondo, eccetera. Gli operatori di confronto piu' utilizzati sono i seguenti:

- `.GT.` greather than, ovvero $>$
- `.GE.` greather equal, ovvero $\geq$
- `.LT.` less than, ovvero $<$
- `.LE.` less equal, ovvero $>$
- `.EQ.` equal, ovvero l'operatore di uguaglianza (notare la differenza con l'operatore di assegnazione $=$; in questo caso `.EQ.` confronta due variabili)
- `.NE.` not equal, ovvero $\neq$

Tutti questi operatori confrontano due variabili, ed il risultato é `.TRUE.` o `.FALSE.` a seconda del valore delle due variabili.

Esempi di utilizzo:

```
INTEGER I, J
LOGICAL L

I = 2
J = 5

L = I .GT. J     - L vale .FALSE. dopo questa istruzione
L = I .LT. J     - L vale .TRUE. dopo questa istruzione
L = I .EQ. J     - L vale .FALSE. dopo questa istruzione
L = I .NE. J     - L vale .FALSE. dopo questa istruzione
```

Gli operatori logici `.AND.` ed `.OR.` agiscono su due variabili di tipo `LOGICAL`, e danno come risultato `.TRUE.` o `.FALSE.`, secondo le regole descritte qui sotto. L'operatore `.NOT.` agisce su una sola variabile. Piú precisamente, si ha:

- `.AND.` Il risultato é `.TRUE.` se e solo se entrambi le variabili logiche valgono `.TRUE.`
- `.OR.` Il risultato é `.FALSE.` se e solo se entrambi le variabili logiche valgono `.FALSE.`
- `.NOT.` Nega la variabile. Il risultato é `.TRUE.` se la variabile é `.FALSE.` e viceversa.

Essi si applicano solo alle variabili logiche; diamo qui qualche esempio:

```
LOGICAL A, B, C

A = .FALSE.
B = .TRUE.

C = A .AND. B    - C vale .FALSE. dopo questa istruzione
A = A .AND. B    - A vale .TRUE. dopo questa istruzione
```

```
B = .NOT. A      - ora B vale .FALSE.
C = (.NOT. B) .AND. A
                 - ora C vale .TRUE.
```

7. Operatori sui caratteri e sulle stringhe

Per quanto riguarda le operazioni sulle stringhe, esaminiamo le operazioni di concatenazione e di estrazione di una stringa.

L'operazione di concatenazione é assai semplice: due stringhe possono essere legate assieme mediante l'operatore //. Ad esempio, se

```
CHARACTER*6 NOME
CHARACTER*5 COGNOM
CHARACTER*20 AUTORE

NOME = 'DONALD'
COGNOM = 'KNUTH'
```

allora l'istruzione:

```
AUTORE = NOME // COGNOM
```

assegna il valore

```
'DONALDKNUTH'
```

alla variabile `AUTORE`. Notiamo che se desideriamo uno spazio tra il nome ed il cognome, occorre concatenarlo tra i due, nel modo seguente:

```
AUTORE = NOME // ' ' // COGNOM
```

otteniamo allora il seguente valore della variabile `AUTORE`:

```
'DONALD KNUTH'
```

L'estrazione di una stringa é una operazione abbastanza semplice; ad esempio sia

```
CHARACTER*20 AUTORE
CHARACTER*20 NOME
CHARACTER*20 COGNOM
CHARACTER INIZ
AUTORE = 'GIUSEPPE PEANO'
```

Allora specificando la posizione del primo e dell'ultimo carattere da estrarre all'interno di una parentesi tonda, separati dal simbolo :, é possibile estrarre il nome ed il cognome. Infatti, con le istruzioni:

```
NOME = AUTORE(1:8)
COGNOM = AUTORE(10:15)
INIZ = AUTORE(1:1)
```

si ottiene:

```
NOME vale 'GIUSEPPE'
COGNOM vale 'PEANO'
INIZ vale 'G'
```

8. Input ed output

I dispositivi di ingresso e di uscita *standard* sono costituiti dal *video* e dalla *tastiera*.

8.1. Output. Per scrivere sul video, si utilizza l'istruzione `WRITE`, per leggere dei dati introdotti per mezzo della tastiera si utilizza l'istruzione `READ`. Ad esempio, utilizzando le seguenti istruzioni:

```
      INTEGER I
      I = 20

      CHARACTER*8 NOME
      NOME = ' GREGORY'

      WRITE (UNIT=*, FMT=*) I
C        VISUALIZZA IL VALORE DELLA VARIABILE I
      WRITE {UNIT=*, FMT=*) NOME
C        VISUALIZZA IL VALORE DELLA VARIABILE NOME
      WRITE (UNIT=*, FMT=*) NOME, ' HA', I, ' ANNI'
C        VISUALIZZA SIA NUMERI CHE STRINGHE
```

si ottiene su video:

```
 20
 GREGORY
 GREGORY HA 20 ANNI
```

La descrizione `UNIT=*` dice al calcolatore di scrivere sullo *standard output*, ovvero sul video, mentre la descrizione `FMT=*` indica al calcolatore che non vi sono particolari condizioi di formattazione (cioé di impaginazione) del testo.

8.2. Input. Vediamo ora come leggere un numero da tastiera; rimandiamo la lettura dei caratteri e delle stringhe al paragrafo 9

L'istruzione usata é `READ (UNIT=*, FMT=*) variabile`, come da esempi che seguono:

```
      INTEGER I
      REAL X
      DOUBLE PRECISION D
      COMPLEX Z

      READ (UNIT=*, FMT=*) I    - legge la variabile intera I;
                                  l'utente puo' introdurre il
                                  numero intero seguito dalla
                                  pressione del tasto INVIO
      READ (UNIT=*, FMT=*) X    - legge la variabile reale X;
                                  l'introduzione avviene come nel
                                  caso precedente
      READ (UNIT=*, FMT=*) D    - legge la variabile in doppia
                                  precisione D; l'utente deve
                                  introdurre il dato nella forma
                                  esponenziale, esempio 1.2D1
                                  e premere INVIO
      READ (UNIT=*, FMT=*) Z    - legge la variabile complessa Z;
                                  l'introduzione delle parti reali
                                  ed immaginaria avviene racchiuden-
                                  dole tra parentesi; ad esempio
```

```
                                scrivendo (2.0, 3.0) seguito da
                                INVIO si assegna a Z il valore
                                complesso 2 + j 3
```

Nelle istruzioni `READ` e `WRITE` é possibile abbreviare le specificazioni in parentesi togliendo le parole `UNIT=` e `FORMAT=`, scrivendo ad esempio

```
WRITE (*,*) Z
READ (*,*) D
```

La scelta é lasciata al lettore.

9. Formattazione degli input e degli output

9.1. Formattazione degli output. Ad ogni istruzione `READ` puó essere associata (mediante una etichetta) una istruzione `FORMAT`, che specifica la modalitá di impaginazione dei dati in uscita. Ad esempio, se si vuol visualizzare un numero intero, riservandogli spazio per al piú 5 cifre, si possono utilizzare le istruzioni:

```
      INTEGER I
      ...
      WRITE (UNIT=*, FMT=200) I
200   FORMAT (1X, I5)
```

ove il *descrittore* `I5` riserva 5 spazi a disposizione del numero intero `I`; se esso é composto da meno di 5 cifre viene incolonnato a destra. Ad esempio se `I = 234` la visualizzazione sará del tipo

```
__234
```

ove il carattere _ rappresenta uno spazio vuoto. L'istruzione `FORMAT` puó essere posta in qualsiasi punto del programma. Si consiglia di porla subito dopo l'istruzione `WRITE` corrispondente, oppure di porre tutte le istruzioni `FORMAT` dopo l'istruzione `STOP`. Vediamo ora in dettaglio i principali descrittori:

- `I`n, ove n é un numero intero maggiore di zero. Questo descrittore viene usato per riservare n spazi ad un numero intero.
- `F`$n.d$, ove $n > 0$ e $d \geq 0$ sono numeri interi. Questo descrittore viene usato per un numero reale, e riserva n spazi al numero, che verrá inoltre visualizzato con d cifre decimali.
- `E`$n.d$, ove $n > 0$, $d \geq 0$. Utilizzato per visualizzare un numero reale in notazione esponenziale (detta anche notazione scientifica), con n spazi ad esso riservato e con d cifre decimali.
- `D`$n.d$ é analogo al precedente, ma si riferisce ai numeri in doppia precisione. É un descrittore obsoleto, che é stato mantenuto per compatibilitá con le precedenti versioni del `Fortran`, e pertanto al suo posto é possibile utilizzare il descrittore `E`.

Per visualizzare un numero complesso, occorre visualizzarne sia la parte reale che quella immaginaria; a tal scopo si utilizzano due descrittori di tipo `E` o `F`. Come esempio, si consideri il seguente frammento di programma, in cui a fianco di ogni istruzione `WRITE` é indicato il corrispondente output:

```
      INTEGER I
      REAL X
      DOUBLE PRECISION Y
      COMPLEX C
```

```
      I = -12
      X = 1.2311
      Y = 2.32E-4                - si noti l'uso del descrittore
                                   D in assegnazione (obbligatorio)
      C = (2.0, -3.1)

      WRITE (UNIT=*, FMT=200) I
 200  FORMAT (1X, I5)           - output: __-12
      WRITE (UNIT=*, FMT=210) X
 210  FORMAT (1X, F8.3)         - output: ___1.231
      WRITE (UNIT=*, FMT=220) X
 220  FORMAT (1X, SP, F7.2)     - output: __+1.23
      WRITE (UNIT=*, FMT=230) Y
 230  FORMAT (1X, E9.1)        - output: ___2.3E-4
      WRITE (UNIT=*, FMT=240) C
 240  FORMAT (1X, 'RE=', F6.2, ' IM=', F6.2)
                               - output: RE=__2.00 IM=_-3.10
      WRITE (UNIT=*, FMT=250) X, X**2
 250  FORMAT (2(1X, SP, F7.2)) - output: __+1.23 __+1.52
```

Osserviamo innanzitutto che in tutte le istruzioni `FORMAT` compare il descrittore `1X`; esso equivale ad uno spazio vuoto, ed il suo utilizzo sará chiarito fra breve. Osserviamo inoltre che l'utilizzo (facoltativo) del descrittore `SP` forza l'utilizzo del segno + nell'output. Infine, alla riga 250 si é scritto

```
2(1X, SP, F7.2)
```

come abbreviazione di:

```
(1X, SP, F7.2, 1X, SP, F7.2)
```

Osserviamo inoltre che conviene porre tutte le istruzioni `FORMAT` dopo l'istruzione `STOP` se si vuole utilizzare una di esse per piú di una istruzione `WRITE`.

Passiamo ora all'esame del *controllo dell'interlinea*; a tal scopo il primo carattere di una istruzione `FORMAT` é solitamente scelto tra i seguenti:

- `1X` (oppure `' '`) per stampare sulla linea successiva
- `0` per avanzare di due linee
- `1` per avanzare di una linea
- `+` per scrivere sulla stessa linea di un precedente `WRITE`

La necessitá di usare il primo carattere come controllo dell'interlinea spiega perché nel visualizzare una stringa costante, ad esempio la parola `CIAO` occorre scrivere:

```
WRITE (UNIT=*, FMT=*) ' CIAO'
```

mettendo uno spazio vuoto (equivalente al descrittore `1X` davanti alla lettera `C`. Lo stesso controllo dell'interlinea puó essere usato nelle operazioni di lettura (ad esempio di un file); in questo caso ciascuna istruzione `FORMAT` é associata, mediante una etichetta, ad una o piú istruzioni `READ`.

Passiamo infine al descrittore `A`n, utilizzato per descrivere la modalitá di visualizzazione di una stringa. Il numero n indica il numero di caratteri visualizzati. Si hanno i seguenti casi:

```
CHARACTER*7 NOME
NOME = 'VALERIA'
```

```
      WRITE (UNIT=*, FMT=200) NOME
 200  FORMAT (1X, A7)               - output: VALERIA
      WRITE (UNIT=*, FMT=210) NOME
 210  FORMAT (1X, A4)               - output: VALE
      WRITE (UNIT=*, FMT=220) NOME
 220  FORMAT (1X, A10)              - output: ___VALERIA
```

Se interessa stampare tutta la stringa, senza preoccuparsi del numero di caratteri, si puó utilizzare la seguente formattazione, piú semplice:

```
      ...
      WRITE (UNIT=*, FMT=230) NOME
 230  FORMAT (1X, A)                - output: VALERIA
```

9.2. Formattazione degli input. Anche ad ogni istruzione `READ` puó essere associata una istruzione `FORMAT`, con gli stessi descrittori. Ad esempio, scrivendo:

```
      INTEGER I

      READ (UNIT=*, FMT=300) I
 300  FORMAT (I4)
```

si chiede all'utente di introdurre un numero intero da tastiera costituito da 4 cifre, o dal segno piú 3 cifre. Se l'utente vuole introdurre un numero minore di cifre, deve lasciare spazi davanti al numero; ad esempio per introdurre il numero 12 deve digitare __12. Questa procedura é assai scomoda, ed é pertanto poco utilizzata. Viene invece utilizzata, a volte, per leggere dei dati da un file.

L'unico utilizzo consigliato ad un principiante per la formattazione degli input é quello relativo all'immissione di una stringa: ad esempio si scrive:

```
      CHARACTER*20 NOME

      READ (UNIT=*, FMT=200) NOME
 200  FORMAT (A)
```

tutte le volte che si desidera leggere una stringa. Se si vuole fissare il numero massimo di caratteri (ad esempio 20 caratteri) si puó utilizzare il descrittore An (in questo caso `A20`).

10. Istruzioni di salto

L'istruzione `GOTO` permette di saltare in un punto qualsiasi del programma. Tale punto deve essere indicato con un numero intero che deve essere riportato nella riga di programma che si desidera raggiungere, nelle posizioni riservate alle etichette. Ad esempio:

```
      PROGRAM STUPID

      INTEGER I
      I = 1

      GOTO 200
C        SALTA ALL'ISTRUZIONE NUMERATA DALL'ETICHETTA 200
```

```
      I = I + 1
C        QUINDI QUESTA ISTRUZIONE NON SARA' MAI ESEGUITA

 200  WRITE (UNIT=*, FMT=*) I
C        E QUINDI QUESTA ISTRUZIONE VISUALIZZERA' IL NUMERO 1

      STOP
      END
```

Questa modalitá di utilizzo dell'istruzione `GOTO` é detta *incondizionata*. Vedremo nel paragrafo successivo un'altra modalitá di utilizzo. L'istruzione `GOTO` viene di solito usata per realizzare dei cicli indefiniti, come vedremo nel paragrafo 12.

11. Istruzioni condizionali

Esistono tre tipi di istruzioni condizionali:

- La struttura `IF` semplice
- La struttura `IF ... THEN ... ELSE ... ENDIF`
- La struttura `GOTO` condizionale

La prima puó essere utilizzata se si vuol eseguire una istruzione se e solo se é verificata una certa condizione. Ad esempio, il seguente programma chiede di introdurre un numero intero, e ne calcola il quadrato. Se il numero introdotto é negativo, visualizza altresí il messaggio `NUMERO NEGATIVO`.

```
      PROGRAM ELEVA

      INTEGER NUMERO
      INTEGER QUADR

      WRITE (UNIT=*, FMT=*) ' INTRODUCI UN NUMERO:'
      READ (UNIT=*, FMT=*) NUMERO

      IF (NUMERO .LT. 0) WRITE (UNIT=*, FMT=*) ' NUMERO NEGATIVO'

      QUADR = NUMERO**2
      WRITE (UNIT=*, FMT=100) NUMERO, QUADR

      STOP

 100  FORMAT (1X, 'IL QUADRATO DI ', I5, 'E'' ', I5)

      END
```

Se si vogliono eseguire piú di una istruzione qualora una certa condizione é verificata, occorre utilizzare una struttura piú complessa, ovvero racchiudere le istruzioni tra le istruzioni `IF (condizione) THEN` ed `ENDIF`. Il seguente frammento di programma é un esempio:

```
      INTEGER I
      ...
      IF (I .GT. 0) THEN
         WRITE (UNIT=*, FMT=*) ' NUMERO NEGATIVO'
```

```
         I = -1 * I
C  SE IL NUMERO I E' NEGATIVO VISUALIZZA UN MESSAGGIO E LO
C        CAMBIA DI SEGNO
      ENDIF
```

Se si deve eseguire qualcosa quando la condizione é falsa, occorre inserire le istruzioni da eseguire dopo l'istruzione ELSE, come da esempio seguente:

```
      INTEGER I
      ...
      IF (I .EQ. 0) THEN
         WRITE (UNIT=*, FMT=*) ' IL NUMERO E' UGUALE A ZERO'
      ELSE
         WRITE (UNIT=*, FMT=*) ' IL NUMERO E' DIVERSO DA ZERO'
      ENDIF
```

Le strutture IF ... ENDIF possono essere *annidate* una dentro l'altra, come nel seguente frammento di programma:

```
      INTEGER I
      ...
      IF (I .EQ. 0) THEN
         WRITE (UNIT=*, FMT=*) ' IL NUMERO E' UGUALE A ZERO'
      ELSE IF (I .GT. 0) THEN
         WRITE (UNIT=*, FMT=*) ' IL NUMERO E' MAGGIORE DI ZERO'
      ELSE
         WRITE (UNIT=*, FMT=*) ' IL NUMERO E' MINORE DI ZERO'
         I = -1 * I
      ENDIF
```

Si osservi come la condizione piú complessa (quella con maggior numero di istruzioni da eseguire) é messa per ultima; ció garantisce una migliore leggibilitá del programma.

L'ultima struttura condizionale, che consente una selezione multipla tra casi, si ottiene con l'istruzione GOTO, utilizzata nella cosiddetta *modalitá condizionale*. In questo caso viene valutata una espressione intera, messa a fianco delle parentesi in cui compare una lista di etichette, come nel seguente esempio:

```
      INTEGER GIORNO
      ...
      GOTO (100, 110, 120, 130, 140, 150, 160) GIORNO

      WRITE (UNIT=*, FMT=*) ' GIORNO NON VALIDO'
      GOTO 200

 100  WRITE (UNIT=*, FMT=*) ' LUNEDI'''
      GOTO 200
 110  WRITE (UNIT=*, FMT=*) ' MARTEDI'''
      GOTO 200
 120  WRITE (UNIT=*, FMT=*) ' MERCOLEDI''
      GOTO 200
 130  WRITE (UNIT=*, FMT=*) ' GIOVEDI'''
      GOTO 200
```

```
 140  WRITE (UNIT=*, FMT=*) ' VENERDI'''
      GOTO 200
 150  WRITE (UNIT=*, FMT=*) ' SABATO'
      GOTO 200
 160  WRITE (UNIT=*, FMT=*) ' DOMENICA'
      GOTO 200

 200  ...
```

Se la variabile intera `GIORNO` é compresa tra 1 e 7, salta alla riga indicata dalla prima, dalla seconda, ..., dalla settima etichetta. Se invece `GIORNO` non é compreso tra 1 e 7, allora l'istruzione `GOTO` non compie alcun salto, e viene visualizzato il messaggio `GIORNO NON VALIDO`.

12. Cicli

Molte volte, é necessario eseguire una istruzione, o piú di una istruzione, ripetitivamente per diverse volte. Si parla allora di un *ciclo*. I cicli possono essere di due tipi: *definiti* ed *indefiniti*.

12.1. Cicli definiti. Un ciclo é detto *definito* se viene eseguito un numero determinato di volte. Ad esempio, é definito un ciclo che visualizza la somma dei primi n termini della successione di numeri:

$$1, \frac{1}{2}, \frac{1}{3}, \ldots, \frac{1}{n}$$

Un ciclo definito puó essere realizzato con una struttura del tipo:

```
      INTEGER I, N
      REAL SOMMA
      SOMMA = 0.0
      N = 10

C     CICLO CHE SOMMA I PRIMI 10 TERMINI DELLA SUCCESSIONE
C     1, 1/2, 1/3, ...

      DO 100, I = 1, N
C        LA CIFRA 100 DOPO IL DO INDICA L'ETICHETTA CHE
C        DELIMITA IL BLOCCO DI ISTRUZIONI DA ESEGUIRE NEL
C        CICLO; LA VARIABILE I, CHE ASSUME TUTTI I VALORI
C        INTERI TRA 1 ED N=10 E' DETTA CONTATORE DEL CICLO
C        E SERVE PER CONTARE IL NUMERO DI ISTRUZIONI
C        DA ESEGUIRE (IN QUESTO CASO 10)
            SOMMA = SOMMA + 1.0/N
C        QUESTA ISTRUZIONE VIENE ESEGUITA 10 VOLTE
  100  CONTINUE
```

E' inoltre possibile scegliere l'incremento da dare al contatore del ciclo; tale incremento puó essere un intero arbitrario; ad esempio il programmma seguente visualizza tutti gli interi pari da 10 a 0 compreso.

```
      INTEGER I

      DO 100, I = 10, 0, -2
```

```
C        IL PARAMETRO -2 E' L'INCREMENTO DATO ALLA VARIABILE I AD
         OGNI ESECUZIONE DEL CICLO
            WRITE (UNIT=*, FMT=*) I
 100  CONTINUE
```

12.2. Cicli indefiniti. Un ciclo é *indefinito* se non é noto il numero di esecuzioni del ciclo stesso. Esistono due tipi di ciclo indefinito, la prima con controllo di uscita all'inizio del ciclo, e la seconda con controllo posto alla fine del ciclo. Scrivendone la struttura in italiano, essi appaiono del tipo:

```
nnn  RIPETI
        blocco di istruzioni da eseguire
     SE condizione vera SALTA ALLA RIGA nnn

ppp  SE condizione falsa SALTA ALLA RIGA mmm
        blocco di istruzioni da eseguire
     SALTA ALLA RIGA ppp
nnn  ...
```

Essi possono essere realizzati nel modo seguente:

```
 100  CONTINUE
         blocco di istruzioni da eseguire
      IF (condizione) GOTO 100

 120  IF (condizione) GOTO 200
         blocco di istruzioni da eseguire
      GOTO 120
 200  CONTINUE
```

ove l'istruzione `CONTINUE` non fa nulla, e serve solo per indicare una riga a cui saltare. Vediamo ora due esempi; il primo si riferisce alla richiesta di introdurre un numero positivo da tastiera, tale che l'introduzione venga ripetuta se il numero é minore o uguale a zero, sino all'introduzione di un numero positivo.

```
      INTEGER I

 100  CONTINUE
         WRITE (UNIT=*, FMT=*) ' INTRODUCI UN NUMERO POSITIVO'
         READ (UNIT=*, FMT=*) I
      IF (I .LE. 0) GOTO 100
```

Nel secondo caso un numero reale viene diviso per due, sino a quando non si ottiene un numero minore di 1; se il numero é giá minore di 1, non serve eseguire il ciclo nemmeno una volta, e pertanto é preferibile porre l'istruzione di controllo dell'uscita dal ciclo all'inizio dello stesso.

```
      REAL X
      ...
 100  IF (X .LT. 1) GOTO 110
         X = X / 2.0
         GOTO 100
 110  CONTINUE
```

13. Gestione dei file

Il linguaggio `Fortran77` permette facilmente di reindirizzare le operazioni di input/output verso altri dispositivi diversi dal video e dalla tastiera. Supponiamo ad esempio, di voler scrivere su un file, anziché sul video. Sia `USCITA.TXT` il nome del file su cui vogliamo scrivere. In questo caso basta *aprire* un canale di comunicazione tra il programma ed il file, con l'istruzione:

```
      OPEN (UNIT=1, NAME='USCITA.TXT', ERR=800)
```

ove `NAME` é il nome del file, a cui viene associato un numero intero a scelta mediante l'istruzione `UNIT`. Se vengono aperti piú di un file contemporaneamente, a nomi diversi dei file devono corrispondere numeri distinti. Il numero associato ad `ERR` ed é una etichetta corrispondente ad una riga di programma; in caso non sia possibile aprire il file, il programma salta a questa riga, utilizzata normalmente per visualizzare un messaggio di errore. Lo stesso comando `OPEN` permette di scrivere su un file. Per leggere o scrivere su un file, si utilizzano le istruzioni `READ` e `WRITE`, specificando a fianco di `UNIT=` il numero associato al file. Terminate le operazioni di lettura e/o di scrittura, occorre chiudere il file con l'istruzione:

```
      CLOSE (UNIT=1)
```

ove il numero a fianco di `UNIT=` é identico a quello usato nell'istruzione `OPEN`. Il seguente programma legge dei numeri reali memorizzati nel file `TEMPER.TXT`, che rappresentano le temperature, e scrive sul file `DELTAT.TXT` le differenze tra due temperature consecutive. L'etichetta `END=500` salta alla riga 500 quando non vi sono piú dati sul file di input (ovvero si é raggiunta l'*end of file*). L'uso dell'`END` é facoltativo e puó essere utile per terminare la lettura dei dati quando si é raggiunta la fine del file.

```
      PROGRAM TEMP

      INTEGER N
C         CONTATORE DEL NUMERO DI LETTURE EFFETTUATE

      REAL T, OLDT, DT
C         TEMPERATURA DEL GIORNO N, TEMPERATURA DEL GIORNO
C         PRECEDENTE, DIFFERENZA DI TEMPERATURA TRA DUE
C         GIORNI CONSECUTIVI

      N = 1

      OPEN (UNIT=1, NAME='TEMPER.TXT', ERR=800)
      OPEN (UNIT=2, NAME='DELTAT.TXT', ERR=810)

 100  CONTINUE
         READ (UNIT=1, END=500, FMT=*) T
C           LEGGE UNA TEMPERATURA DAL FILE DI INPUT
         IF (N .GT. 1) THEN
C              SE HA LETTO ALMENO DUE TEMPERATURE NE CALCOLA
C              LA DIFFERENZA E LA SCRIVE SUL FILE DI OUTPUT
            DT = T - OLDT
            WRITE (UNIT=1, FMT=200) N, DT
```

```
            N = N + 1
            OLDT = T
          ENDIF
          GOTO 100
500   CONTINUE

      CLOSE (UNIT=2)
      CLOSE (UNIT=1)

      WRITE (UNIT=*, FMT=*) ' FINE LAVORO'

      STOP

800   CLOSE (1)
      WRITE (UNIT=*, FMT=*) ' ERRORE DI GESTIONE DEL FILE DI INPUT'
      STOP

810   CLOSE (2)
      WRITE (UNIT=*, FMT=*) ' ERRORE DI GESTIONE DEL FILE DI OUTPUT'
      STOP

      END
```

14. Sottoprogrammi

Puó capitare che una parte di codice possa essere presente in piú punti di un programma. Ad esempio, consideriamo un programma che legge due numeri positivi da tastiera, chiedendone la reimmissione se sono minori o uguali a zero, e ne faccia la media. In questo caso il frammento di codice relativo all'introduzione ed al controllo del primo numero é identico al frammento relativo alle stesse operazioni effettuate sul secondo numero, a parte il nome della variabile. Conviene pertanto introdurre dei *sottoprogrammi* (detti anche *subroutine*), che possono essere richiamati nel programma principale quando occorre, risparmiando righe di codice. Un sottoprogramma é specificato da un nome (di al piú 6 caratteri), preceduto dalla parola `SUBROUTINE`, e da una lista di parametri tra parentesi. Ad esempio, il sottoprogramma seguente legge un numero reale da tastiera, chiedendone la reintroduzione se il numero introdotto é minore o uguale a zero:

```
      SUBROUTINE INTROD(NUMBER)
      REAL NUMBER
C       NUMBER E' IL PARAMETRO DELLA SUBROUTINE ED IL SUO
C       TIPO (INTERO, REALE, ECC.) DEVE COINCIDERE CON IL
C       TIPO DEL PARAMETRO PASSATO DAL PROGRAMMA

 100  WRITE (UNIT=*, FMT=*) ' INTRODUCI UN NUMERO POSITIVO'
      READ (UNIT=*, FMT=*) NUMBER
      IF (NUMBER .LE. 0.0) GOTO 100

      RETURN
C       QUESTA ISTRUZIONE RITORNA AL PROGRAMMA PRINCIPALE
```

```
      END
```

Questa subroutine puó essere chiamata nel programma principale quante volte occorre; ad esempio se si vuol fare la media di due numeri positivi il programma principale puó essere scritto come:

```
      PROGRAM MEDIA
      REAL N1, N2, VALMED

      CALL INTROD(N1)
C        PRIMA CHIAMATA DELLA SUBROUTINE, PER INTRODURRE N1
      CALL INTROD(N2)
C        SECONDA CHIAMATA, PER INTRODURRE N2

      VALMED = (N1 + N2) / 2.0

      WRITE (UNIT=*, FMT=*) ' MEDIA = ', VALMED

      STOP
      END
```

L'istruzione `CALL` seguita dal nome del sottoprogramma esegue il codice in essa contenuto; ad esempio `CALL INTROD(N1)` utilizza la subroutine, in cui il parametro `N1` passato tra parentesi alla subroutine ed il parametro `NUMBER` dichiarato nella subroutine sono sinonimi. Il valore di `NUMBER` viene ricopiato, dopo il ritorno al programma principale (ottenuto con l'istruzione `RETURN`) nella variabile `N1`. Analogamente, nella seconda chiamata `CALL INTROD(N2)`, `NUMBER` é un sinonimo di `N2`.

La subroutine ed il programma principale possono essere scritti in due file separati; ció consente di poter utilizzare la subroutine in altri programmi. Se ad esempio la subroutine é stata salvata con il nome `introd.f` ed il programma con il nome `media.f` é possibile creare il file eseguibile compilando entrambi i file, con vl comando:

```
g77 media.f introd.f -o media
```

ed eseguire il programma con il comando `./media`.

La seguente subroutine riceve come parametri quattro numeri reali, corrispondenti alle coordinate (x_1, y_1), x_2, y_2 di due punti nel piano, nonché due parametri m, q corrispondenti ai coefficienti dell'equazione della retta passante per i due punti, e li calcola, ritornando i valori al programma principale.

```
      SUBROUTINE RETTA (X1, Y1, X2, Y2, M, Q)
      REAL X1, Y1, X2, Y2, M, Q

      M = (Y2 - Y1) / (X2 - X1)
      Q = Y1 - M * X1

      RETURN
      END
```

e puó essere chiamata dal programma principale con l'istruzione

```
      CALL RETTA (XA, YA, XB, YB, CM, CQ)
```

Notiamo come in entrambi gli esempi i nomi dei parametri passati dalla `CALL` e quelli indicati nella parentesi della subroutine possono essere anche differenti; in ogni caso il primo parametro `XA` é sinonimo di `X1`, il secondo parametro passato `YA` é sinonimo di `Y1` e cosí via.

15. Funzioni

Una funzione é una corrispondenza tra due insiemi A e B tale che ad ogni elemento $a \in A$ corrisponde uno ed un solo elemento $b \in B$. In `Fortran77` é possibile definire delle funzioni; ad esempio la seguente funzione calcola la media (reale) tra due numeri reali:

```
      REAL FUNCTION AVER (X1, X2)
      REAL X1, X2
C        PARAMETRI DELLA FUNZIONE

      AVER = (X1 + X2) / 2.0
C        IL VALORE RITORNATO DALLA FUNZIONE DEVE COINCIDERE
C        CON IL NOME DELLA FUNZIONE

      RETURN
      END
```

Essa puó essere chiamata dal programma principale, che in questo caso utilizza anche la subroutine `INTROD` esaminata nel paragrafo 14.

```
     PROGRAM MEDIA

     REAL AVER
C       IL TIPO DI FUNZIONE (OSSIA IL TIPO DI VALORE CALCOLATO
C       DALLA FUNZIONE) DEVE ESSERE DICHIARATO.
C       E' AMMESSA ANCHE LA DICHIARAZIONE IMPLICITA, COME PER LE
C       VARIABILI; TUTTAVIA ESSA E' SCONSIGLIATA, PER OVVIE
C       RAGIONI DI CHIAREZZA

     REAL X, Y
C       NUMERI DA INTRODURRE PER FARNE LA MEDIA
     REAL M
C       MEDIA DEI DUE NUMERI

     CALL INTROD(X)
     CALL INTROD(Y)

     M = MEDIA (X, Y)
C        UTILIZZO DELLA FUNZIONE
C        I PARAMETRI VENGONO PASSATI COME PER LE SUBROUTINE
C        IL VALORE CALCOLATO VIENE ASSEGNATO AD M

     WRITE (UNIT=*, FMT=*) ' MEDIA = ', M

     STOP
     END
```

16. Funzioni implicite

In FORTRAN77 esistono delle funzioni giá pronte per l'uso (dette *implicite*) che agiscono su numeri interi, reali, complessi, ed anche su stringhe e su caratteri. Esaminiamole rapidamente nei prossimi paragrafi, distinguendo tra funzioni matematiche e funzioni operanti su stringhe e caratteri.

16.1. Funzioni matematiche. Le funzioni qui menzionate lavorano su numeri; a fianco di ciascuna di esse é elencato il significato. Indichiamo con:

- `I` un argomento intero
- `R` un argomento reale
- `D` un argomento reale in doppia precisione
- `C` un argomento complesso
- $\star$ una grandezza dello stesso tipo dell'argomento

che rappresenteranno, nella tabella seguente, i tipi di argomenti permessi. Ad esempio, scrivendo `I = INT(IRDC)` intenderemo che la funzione `ANINT` puó avere come variabile indipendente, una variabile intera, o reale, o in doppia precisione, o complessa, e che la variabile indipendente (detta anche valore di ritorno) é un numero intero.

Funzione	Descrizione
`R = ABS(X)`	Calcola il modulo di `X`
$\star$ `= ACOS(XD)`	Calcola $\arccos x$, ed il risultato é compreso tra 0 e π.
`R = AIMAG(C)`	Determina la parte reale di `C`
$\star$ `= ANINT(RD)`	Arrotonda all'intero piú vicino Il risultato é dello stesso tipo dell'argomento (ad esempio 12.122 viene arrotondato a 12)
$\star$ `= ASIN(XD)`	Calcola $\arcsin x$
$\star$ `= ATAN2(RD,RD)`	Calcola $\arctan \frac{a}{b}$, ove a é il primo argomento e b é il secondo argomento. Il risultato é compreso tra $-\pi$ e π.
`C = CMPLX(IRDX, IRD`	Converte una coppia di argomenti in un numero complesso. Il secondo argomento é opzionale.
`C = CONJG(C)`	Calcola il complesso coniugato di `C`
$\star$ `= COS(RDC)`	Calcola il coseno dell'argomento
`D = DBLE(IRDC)`	Converte l'argomento in doppia precisione
$\star$ `= DIM(IRD, IRD)`	Calcola la differenza tra i due argomenti a e b e se essa é maggiore di zero ritorna $a - b$, altrimenti ritorna 0
`D = DPROD(R,R)`	Calcola il prodotto in doppia precisione di due numeri reali
$\star$ `= EXP(RDC)`	Calcola e^x
`I = INT(IRDC)`	Converte l'argomento in un intero troncandolo. Ad esempio 14.7 é convertito in 14

`⋆ = LOG(RDC)`	Calcola $\log x$ (in base e)
`⋆ = LOG10(RDC)`	Calcola $\log_{10} x$
`⋆ = MAX(IRD, IRD, ...)`	Ritorna il massimo degli argomenti (che possono essere piú di due)
`⋆ = MIN(IRD, IRD, ...)`	Come sopra, per il minimo
`⋆ = MOD(IRD, IRD)`	Ritorna il resto della divisione tra i due argomenti
`R = REAL(IRDC)`	Converte l'argomento in un numero reale
`⋆ = SIGN(IRD, IRD)`	Se il secondo argomento é negativo ritorna il primo argomento cambiato di segno; in caso contrario ritorna il primo argomento. Ad esempio, `I = REAL(2, 3)` assegna 2 ad `I`, mentre `R = REAL(2.3, -1.1)` assegna il valore -2.3 ad `R`
`⋆ = SIN(RDC)`	Calcola $\sin x$
`⋆ = SQRT(RDX)`	Calcola la radice quadrata dell'argomento
`⋆ = TAN(RD)`	Calcola $\tan x$

Osserviamo la potenzialitá del FORTRAN77; molte sue funzioni operano anche sui numeri complessi, e cosí é possibile, ad esempio, calcolare il logaritmo di un numero complesso (che sará un numero complesso).

Il seguente esempio utilizza alcune dele funzioni descritte; esso visualizza i valori delle funzioni $y = \sin x$, $z = \cos x$ e $t = \tan x$, ove x é un numero reale immesso da tastiera, e determina inoltre il piú grande ed il piú piccolo tra y, z, t.

```
PROGRAM TRIGON

REAL X, Y, Z, T, MASSIM, MINIM

WRITE (UNIT=*, FMT=*) ' INTRODUCI UN NUMERO REALE:'
READ (UNIT=*, FMT=*) X

Y = SIN (X)
Z = COS (X)
T = TAN (X)

MINIM = MIN(X, Y, T)
MASSIM = MAX(X, Y, T)

STOP
END
```

16.2. Funzioni correlate alle stringhe. Se si vuol determinare la lunghezza di una stringa, si utilizza la funzione `LEN`; essa restituisce un numero intero positivo pari al numero di caratteri della stringa, oppure restituisce 0 se la stringa non contiene alcun carattere, Ad esempio:

```
INTEGER LUN
CHARACTER*10 STR
```

```
STR = 'CIAO'
LUN = LEN(STR)   - LUN vale 4 dopo questa istruzione

STR = ''         - STR non contiene alcun carattere
LUN = LEN(STR)   - ora LUN vale 0
```

Esistono inoltre delle funzioni che restituiscono un valore logico operando sulle stringhe, in modo da poter confrontare due stringhe secondo l'ordine lessicografico. Ció puó venire utile se si devono ordinare dei nomi secondo l'ordine alfabetico. Esse sono simili agli operatori di confronto utilizzati per i numeri (ovvero `.GT.`, `.GE.`, ecc.) sono i seguenti:

- `LGT(S1,S2)` Ritorna `.TRUE.` se la stringa `S1` precede la stringa `S2` nell'ordine.
- `LGE(S1,S2)` Ritorna `.TRUE.` se la stringa `S1` precede la stringa `S2` nell'ordine, o se le due stringhe coincidono.
- `LLT(S1,S2)` Ritorna `.TRUE.` se la stringa `S1` segue la stringa `S2` nell'ordine.
- `LLE(S1,S2)` Ritorna `.TRUE.` se la stringa `S1` segue la stringa `S2` nell'ordine, o se le le due stringhe sono uguali

Ecco, come al solito, alcuni esempi:

```
CHARACTER*20 NOME1, NOME2, NOME3
LOGICAL L

NOME1 = 'CICERONE'
NOME2 = 'PLATONE'
NOME3 = 'PLATONE'

L = LLT(NOME1, NOME2)  - L vale .TRUE. perche'
                         'CICERONE' precede 'PLATONE'
L = LLE(NOME2, NOME3)  - L vale .TRUE. perche' le due
                         stringhe sono uguali
```

La seguente tabella riporta inoltre altre funzioni legate alle stringhe ed ai caratteri (i significati di `R`, `C`, ecc. sono gli stessi del paragrafo 16.1; indicheremo inoltre con `A` un parametro di tipo carattere o stringa).

Funzione	Descrizione
`A = CHAR(I)`	Il carattere `A` é individuato dal codice (generalmente ASCII) `I`
`I = ICHAR(A)`	Ritorna il codice (generalmente ASCII) del carattere `A`
`I = INDEX(A, A)`	Calcola la posizione di inizio della seconda stringa all'interno della prima, e se non la trova ritorna 0. Ad esempio, `I = INDEX('CAVOLO', 'VOLO')` assegna 3 ad `I`
`I = LEN(A)`	Calcola la lunghezza della stringa `A`

17. Le tabelle

In molti problemi di calcolo numerico, occorre lavorare con delle tabelle di valori. Vediamo come poterle utilizzare all'interno di un programma.

Una *tabella* (od *array*) é una zona di memoria riservata ad un gruppo di variabili, dello stesso tipo. Ad esempio, se si vogliono memorizzare gli anni di nascita degli impiegati di una azienda con al massimo 100 dipendenti, é possibile dichiarare una tabella di 100 elementi, anziché 100 distinte variabili. Ció puó essere fatto con la seguente dichiarazione:

```
      INTEGER ANNI(100)
```

Sarebbe meglio fissare la dimensione dell'array con una costante, come nel seguente esempio:

```
      INTEGER MAXDIP
      PARAMETER (MAXDIP = 100)
C        NUMERO MASSIMO DI DIPENDENTI

      INTEGER ANNI(MAXDIP)
```

L'elemento `ANNI(1)` indicherá l'etá del primo dipendente, `ANNI(2)` del secondo, e cosí via.

Il seguente programma legge un intero positivo da tastiera e crea una tabella costituita dei 10 numeri primi maggiori del numero letto, che viene visualizzata. Utilizza la subroutine `INTROD` definita nel paragraf 14.

```
      PROGRAM NUMPRM

      LOGICAL PRIMO
C        FUNZIONE LOGICA PER TESTARE SE UN INTERO E' PRIMO

      INTEGER DIMTAB
      PARAMETER (DIMTAB=10)

      INTEGER PRIMI(DIMTAB)
C        TABELLA DEI NUMERI PRIMI

      INTEGER NLETTO
C        NUMERO LETTO DA TASTIERA

      INTEGER I, N
C        I E' UN CONTATORE DI UN CICLO ED N E' UN
C        NUMERO PROGRESSIVO MAGGIORE DI NLETTO

      CALL INTROD(NLETTO)
C        LEGGE UN NUMERO DA TASTIERA

      N = NLETTO

      DO 100, I = 1, 10
C          CICLO PER RIEMPIRE LA TABELLA
 110     N = N + 1
         IF (PRIMO(N)) GOTO 120
         GOTO 110
 120     PRIMI(I) = N
```

```
         WRITE (UNIT=*, FMT=*) PRIMI(I)
 100  CONTINUE

      STOP
      END
```

Questo programma utilizza la funzione `PRIMO` che ritorna il valore `.TRUE.` se e solo se il parametro ricevuto é un numero primo. La funzione puó essere implementata nel modo seguente:

```
      LOGICAL FUNCTION PRIMO(NUMERO)

      INTEGER NUMERO

      INTEGER J
      LOGICAL RISULT

      RISULT = .TRUE.

      DO 200, J = 2, NUMERO/2
         IF (MOD(NUMERO, 2) .EQ. 0) RISULT = .FALSE.
C           SE LA DIVISIONE HA RESTO ZERO IL NUMERO NON E' PRIMO
200   CONTINUE

      PRIMO = RESULT

      RETURN
      END
```

In pratica, per determinare se un numero n é primo, basta dividerlo per tutti gli interi compresi tra 2 ed $\frac{n}{2}$ ed esaminare il resto della divisione.

18. Tabelle multidimensionali

Una tabella puó avere piú di una dimensione. Ad esempio la dichiarazione `INTEGER TABLE(10,10)` dichiara una tabella di 100 elementi, distribuiti su 10 righe e 10 colonne. Il seguente programma crea una tabella pitagorica in un array a due dimensioni:

```
      PROGRAM PITAG

      INTEGER DM
      PARAMETER (DM = 10)
C       DIMENSIONE DELLA TAVOLA PITAGORICA

      INTEGER TABLE(DM, DM)

      INTEGER I, J
C        CONTATORI DI CICLI

C     CICLI PER LA CREAZIONE DELLA TABELLA
      DO 100, I = 1, 10
```

```
         DO 110, J = 1, 10
            TABLE(I,J) = I * J
 110     CONTINUE
 100  CONTINUE

C     VISUALIZZAZIONE DI UN ELEMENTO SCELTO DELLA TABELLA
      WRITE (UNIT=*, FMT=*) ' INTRODUCI RIGA E COLONNA'
      READ (UNIT=*, FMT=*) I, J
      IF ((I.GE.0).AND.(I.LE.DM).AND.(J.GE.DM).AND.(J.LE.DM)) THEN
          WRITE (UNIT=*, FMT=900) TABLE(I,J)
      ELSE
          WRITE (UNIT=*, FMT=*) ' RIGA E COLONNA ERRATI '
      STOP

 900  FORMAT (1X, I5)

      END
```

19. Input ed output di tabelle

L'introduzione dei dati di una tabella (ad esempio da tastiera) non deve necessariamente essere fatta elemento per elemento, ma ad esempio le istruzioni:

```
      INTEGER TBL(5)
      ...
      READ (UNIT=*, FMT=100) TBL
 100  FORMAT (5I3)
```

permette di lettere 5 valori dalla riga di ingresso, sistemati nelle prime 15 colonne, riservando 3 caratteri per ogni numero. Ció é sicuramente piú comodo che scrivere:

```
      READ (UNIT=*, FMT=100) TBL(1), TBL(2), TBL(3), TBL(4), TBL(5)
 100  FORMAT (5I3)
```

Una matrice a piú dimensioni puó essere introdotta nello stesso modo, tenendo presente che l'indice piú a sinistra all'interno delle parentesi é quello che varia piú velocemente. Ad esempio, le istruzioni:

```
      INTEGER TBL(2,2)
      ...
      READ (UNIT=*, FMT=100) TBL
 100  FORMAT (4I3)
```

sono equivalenti a:

```
      READ (UNIT=*, FMT=100) TBL(1,1), TBL(2,1), TBL(1,2), TBL(2,2)
 100  FORMAT (4I3)
```

Le stesse considerazioni valgono per l'istruzione `WRITE`; ad esempio le istruzioni:

```
      INTEGER I(2,2)
      ...
      WRITE (UNIT=*, FMT=200) I
 200  FORMAT (4(1X, I5))
```

visualizza nell'ordine `I(1,1)`, `I(2,1)`, `I(1,2)`, `I(2,2)`. Se non si vogliono visualizzare tutti gli elementi di una tabella, o se si vuol cambiare l'ordine di visualizzazione

(o l'ordine di lettura) si puó utilizzare una istruzione derivata dai cicli definiti, e detta pertanto DO *implicito*. Ad esempio, per visualizzare solo gli ultimi 5 elementi di una tabella di 10 elementi, si puó scrivere:

```
      INTEGER TBL(10)
      INTEGER I
      ...
      WRITE (UNIT=*, FMT=100) (TBL(I), I = 6, 10)
 100  FORMAT (5(1X, I4))
```

Analogamente, se si vuole invertire l'ordine di visualizzazione degli elementi di una matrice bidimensionale, possiamo scrivere:

```
      INTEGER T(4,4)
      INTEGER I, J
      ...
      WRITE (UNIT=*, FMT=100) ((T(I,J), J=1,4), I=1,4)
 100  FORMAT (16(1X, I2))
```

20. Passaggio delle tabelle a sottoprogrammi e funzioni

Passiamo ora all'esame di come passare delle tabelle, unidimensionali e monodimensionali a subroutine e funzioni. Le considerazioni fatte per le subroutine valgono, pari pari, per le funzioni. Il modo piú semplice di passaggio di una tabella ad una subroutine consiste nello specificarne la dimensione sia all'interno del programma chiamante che della funzione:

```
      SUBROUTINE CAOS(A)
      REAL A(100)
      ...
      RETURN
      END

      PROGRAM TEST
      REAL MATRICE(100)
      ...
      CALL CAOS(MATRICE)
      ...
      STOP
      END
```

Questo metodo ha l'inconveniente di non poter riutilizzare la funzione per tabelle di dimensione diversa da 100 elementi. Un metodo alternativo consiste nel passare alla subroutine anche la dimensione della tabella, come nel seguente esempio:

```
      SUBROUTINE CAOS(DIM, A)
      INTEGER DIM
      REAL A(DIM)
      ...
      RETURN
      END

      PROGRAM TEST
      INTEGER NUMEL
```

```
PARAMETER (NUMEL=100)
REAL MATRICE(NUMEL)
...
CALL CAOS(NUMEL, MATRICE)
...
STOP
END
```

Questo metodo é da preferirsi al primo, e puó essere esteso anche a matrici multidimensionale; ad esempio il seguente programma passa una tabella a 3 dimensioni ad una funzione:

```
REAL FUNCTION PROVA(D1, D2, D3, TABLE)
INTEGER D1, D2, D3
REAL TABLE(D1, D2, D3)
...
RETURN
END

PROGRAM TEST
INTEGER DM1, DM2, DM3
PARAMETER (DM1=10, DM2=20, DM3=5)
REAL TAB(DM1, DM2, DM3)
REAL RISULT
...
RISULT = PROVA(DM1, DM2, DM3, TAB)
...
STOP
END
```

Osserviamo infine che il passaggio di una stringa puó essere effettuato nello stesso modo del passaggio di una tabella monodimensionale, dato che una stringa non é altro che una tabella di caratteri.

21. Considerazioni finali

In questa dispensa sono stati esaminati gli aspetti fondamentali del linguaggio, con lo scopo di poter permettere la scrittura e l'interpretazione di semplici programmi. Il linguaggio presenta altri aspetti interessanti, che tuttavia non possono essere trattati per ragioni di brevitá, e per i quali si rimanda agli ottimi testi citati in bibliografia. Segnaliamo infine che esiste una versione piú attuale del linguaggio, nota come `Fortran90`, che costituisce una estensione del `Fortran77`. Osserviamo tuttavia che il `Fortran77` non é affatto superato per quanto riguarda l'enorme affidabilitá dei compilatori, per la facilitá di apprendimento, e per l'ottimizzazione in termini di velocitá e di compattezza di codice. Esistono inoltre numerose librerie di funzioni e sottoprogrammi scritti in `Fortran77`, ad esempio `LAPACK`, liberamente disponibile ad esempio sotto `LINUX`.

Riferimenti bibliografici

[1] A. Celentano. *Fortran77 per Applicazioni Non Numeriche*, Clup, Milano, (1983).
[2] R. Kumar. *Programming with Fortran77*, Tata McGraw-Hill publishing Company Limited, New Delhi, (1986).
[3] C.G. Page. *Professional Programmer's Guide to Fortran77*, file disponibile liberamente su *internet*.

TYPESET BY LaTeX 2ε UNDER LINUX

COMO, 8 FEBBRAIO 2004