Subroutine e funzioni

Porzioni di codice possono essere scritte (e compilate) separatamente e richiamate in ogni punto del programma.

Il Fortran fornisce vari modi

- Subroutine
- Function
- Module

Sintassi di SUBROUTINE

```
SUBROUTINE nomeroutine (par1,...,parN)
   IMPLICIT NONE
   ⋮
   dichiarazioni di variabili
   istruzioni
   ⋮
END SUBROUTINE nomeroutine
```

- I parametri che compaiono nell'intestazione si chiamano parametri formali e devono essere dichiarati all'interno della subroutine
- I parametri vengono passati per riferimento

Chiamata di una subroutine

```
PROGRAM nomeprogramma
   IMPLICIT NONE
   ⋮
   CALL nomeroutine (par1,...,parN)
   ⋮
END PROGRAM nomeprogramma
```

- La chiamata si effettua con il comando `CALL`
- I parametri di chiamata (parametri attuali) possono essere variabili o espressioni e devono coincidere in tipo e numero con i parametri formali.

Sintassi di FUNCTION

```
FUNCTION nomefunction (par1,...,parN)
   IMPLICIT NONE
   ⋮
   dichiarazioni di variabili
   istruzioni
   nomefunction=valore
   ⋮
END SUBROUTINE nomeroutine
```

- I parametri devono essere dichiarati all'interno della funzione così come lo stesso nome della funzione
- Alla funzione va assegnato un valore

Chiamata di una funzione

```
PROGRAM nomeprogramma
   IMPLICIT NONE
   ⋮
   quellochemipare=nomefunzione (par...)
   ⋮
END PROGRAM nomeprogramma
```

- `nomefunzione` va dichiarato e può essere assegnato a una variabile del suo stesso tipo o usato in un'espressione

Le funzioni ricorsive

```
RECURSIVE FUNCTION nomefun (par1,...)
RESULT (risultato)
    ⋮
   istruzioni
    ⋮
END SUBROUTINE nomeroutine
```

- Occorre dichiarare il tipo di `risultato`
- La funzione chiamerà se stessa con `nomefun` ma restituirà un valore con `risultato`
- Ci deve essere un caso in cui la funzione non chiama se stessa

Tipi di tutti i tipi

- Tipi semplici (reali, interi, ecc.)
- Tipi strutturati (array e tipi derivati)
- Tipi puntatore (puntano ad altri tipi)

Array

Gli array sono aggregati di variabili dello stesso tipo identificate da uno o più indici.

La dichiarazione avviene aggiungendo al nome del tipo la dimensione

```
REAL,DIMENSION(3) ::  vettore
```

dichiara un vettore di 3 reali, accessibili singolarmente come `vettore(1)`, `vettore(2)`, `vettore(3)`

```
REAL,DIMENSION(4,6) ::  a
```

dichiara una matrice 4×6, i cui elementi sono accessibili singolarmente come `a(i,j)`

Array allocabili

È possibile definire array la cui dimensione viene stabilita dinamicamente a tempo di esecuzione

```
REAL,ALLOCATABLE,DIMENSION(:)::vett
```

```
REAL,ALLOCATABLE,DIMENSION(:,:)::a
```

dichiarano rispettivamente un vettore `vett` e una matrice a di dimensione indefinita.

Gli array allocabili sono utili perché permettono di ottimizzare l'uso della memoria

Array allocabili

Prima di utilizzarli è necessario scrivere un comando che alloca la memoria che conterrà gli array

```
ALLOCATE(vett(k),a(m,n))
```

k, m e n possono essere variabili il cui valore non è noto a tempo di compilazione

Dopo l'utilizzo sarebbe opportuno liberare la memoria con il comando

```
DEALLOCATE(vett(k),a(m,n))
```

Comandi su dati aggregati

Il FORTRAN possiede una grossa quantità di comandi e funzioni che permettono di operare su vettori o parti di vettori in modo semplice. Alcuni esempi:

- `SUM(vett)` somma tutti gli elementi del vettore
- `2*a+1` restituisce una matrice i cui elementi sono moltiplicati per 2 e sommati a 1
- `v(1:2)` restituisce il sottovettore contenente gli elementi `v(1)` e `v(2)`
- `MATMUL(a,v)` restituisce il prodotto riga per colonna tra `a` e `v` purché abbiano dimensioni compatibili

Tipi derivati

È possibile costruire nuovi tipi tramite il comando

```
TYPE nometipo
   tiposemplice1 ::  campo1
   tiposemplice2 ::  campo2,campo3
   ⋮
END TYPE nometipo
```

E si può dichiarare una variabile del tipo `nometipo` tramite il comando

```
TYPE(nometipo) ::  miavariabile
```

Tipi derivati

Un tipo derivato può rappresentare un record di una base di dati

```
TYPE persona
   character(20) ::  nome,cognome
   integer ::  cellulare
END TYPE nometipo
```

Definisce un tipo tramite cui si può costruire una struttura

```
TYPE(persona),DIMENSION(100) ::  rubrica
```

`rubrica` è un array di 100 persone

Tipi derivati

L’accesso ai campi avviene tramite %
Per assegnare una variabile occorre assegnare tutti i campi:

```
rubrica(1)%nome="Carlo Azeglio"
rubrica(1)%cognome="Ciampi"
rubrica(1)%cellulare=3333333333
```

Crea l’elemento 1 della mia rubrica

Utilizzo di file esterni

Per leggere o scrivere su un file occorre aprirlo

```
OPEN(unit=N,iostat=ios,file=nome,action=r
```

- `N` è un numero positivo a piacere che si associa in modo univoco al file (es. 1,2,80)
- `ios` è un intero che assume il valore 0 se e solo se tutto è andato bene
- `nome` è una stringa che contiene il nome del file da aprire (es. matrice.spa)
- `action="read"` indica che si vuole aprire il file in lettura, `action="write"` in scrittura

Utilizzo di file esterni

Il *rituale* completo è

```
OPEN(unit=N,iostat=ios,file=nome,action=r
IF (ios/=0) THEN
   WRITE(*,*) "Error:  can't open ",nome
   STOP
ENDIF
```

dove si è effettuato un controllo su `ios`, se è diverso da 0 il programma si interrompe e viene visualizzato un messaggio di errore

Leggere da file

Il comando per leggere è

```
READ(unit,format) variabile
```

Il primo parametro indica l'unità

- * indica il dispositivo di standard input (tastiera)
- un numero N indica che occorre leggere dal file associato in apertura all'unità N

Il secondo parametro indica il formato
* indica un formato generico

Scrivere su file

Il comando per scrivere è

```
WRITE(unit,format) var, cost o espr
```

Il primo parametro indica l'unità

- * indica il dispositivo di standard output (terminale)
- un numero N indica che occorre scrivere sul file associato in apertura all'unità N

Il secondo parametro indica il formato
* indica un formato generico