Sottoprogrammi

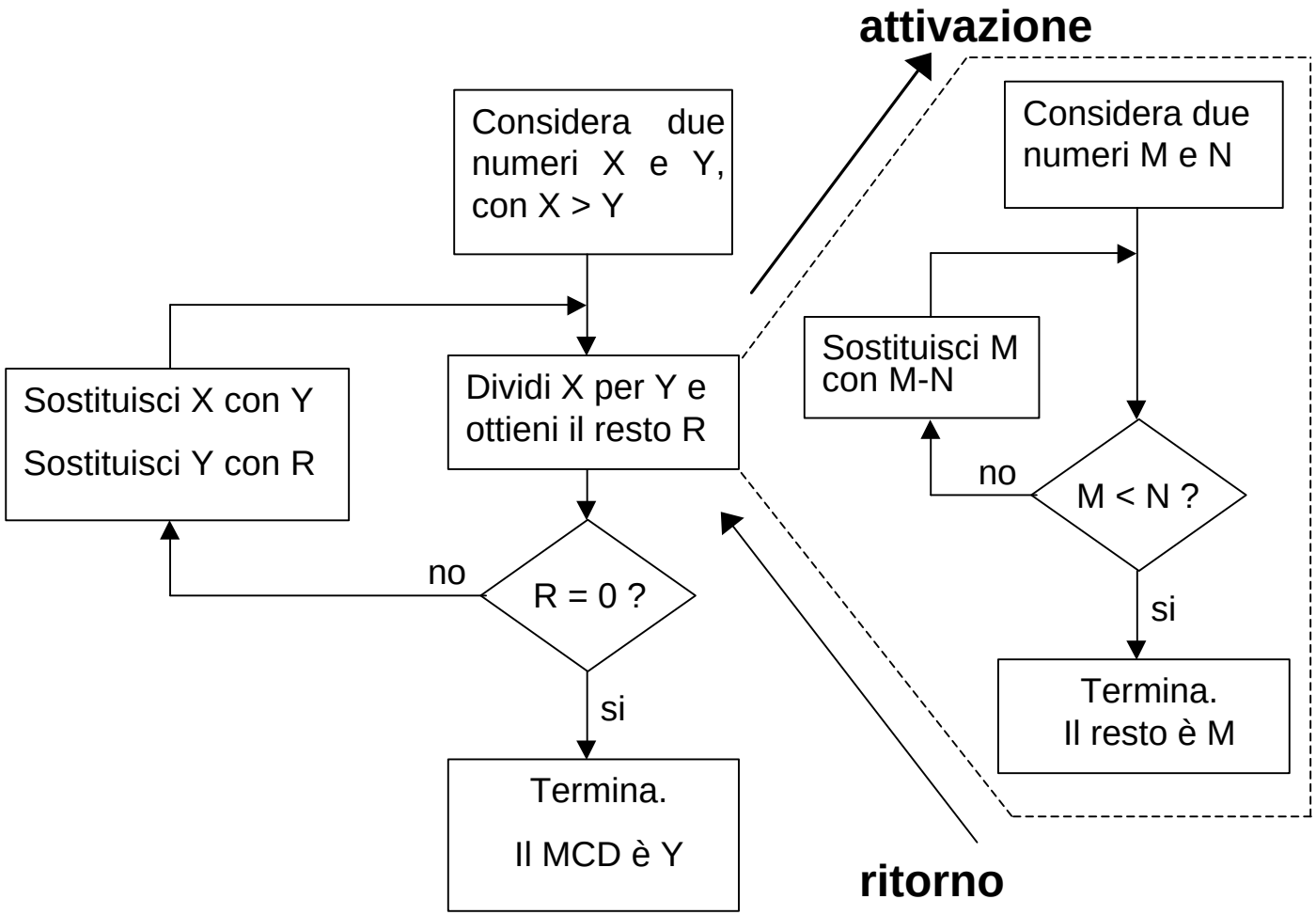
Un sottoprogramma è una particolare unità di codice che non può essere eseguita autonomamente, ma soltanto su richiesta del programma principale (la parte di codice delimitata da `PROGRAM` ed `END PROGRAM`) o di un altro sottoprogramma.

Un sottoprogramma viene realizzato per svolgere un compito specifico (p.es. leggere o stampare gli elementi di un array, calcolare il valore di una particolare funzione matematica, ecc.) per il quale implementa un opportuno algoritmo.

Per questo scopo, il sottoprogramma utilizza variabili proprie, alcune delle quali sono impiegate per scambiare dati con il programma dal quale viene attivato.

Un sottoprogramma può essere attivato più volte in uno stesso programma o anche utilizzato da un programma diverso da quello per cui era stato inizialmente progettato.

Esempio



Scambio dati

$X \rightarrow M$

$Y \rightarrow N$

$R \leftarrow M$

L’esecuzione delle istruzioni di un sottoprogramma è provocata da una particolare istruzione del programma che lo attiva (istruzione di *chiamata*, per cui il programma è anche detto *chiamante*). Ciò determina la sospensione dell’esecuzione delle istruzioni del programma chiamante, che riprenderà dopo l’esecuzione dell’ultima istruzione del sottoprogramma (tipicamente, un’istruzione di *ritorno*).

Il programma chiamante ed il sottoprogramma scambiano dati attraverso una lista di variabili, definite all’interno del sottoprogramma, dette *argomenti* o *parametri formali* del sottoprogramma. Esse sono destinate ad ospitare i dati di ingresso e/o di uscita del sottoprogramma.

Con la istruzione di chiamata, il programma chiamante fornisce al sottoprogramma una lista di *parametri effettivi*, costituiti dai valori effettivi di ingresso su cui il sottoprogramma deve operare e dalle variabili del programma chiamante in cui i valori di uscita del sottoprogramma dovranno essere memorizzati.

La corrispondenza tra parametri effettivi e formali è fissata per ordine.

Funzioni

Una funzione è un particolare sottoprogramma che produce in uscita un valore il quale non è assegnato ad uno dei parametri, ma viene attribuito al nome stesso della funzione.

La chiamata del sottoprogramma non avviene mediante una esplicita istruzione di chiamata, ma inserendo il nome della funzione seguito dalla lista dei parametri effettivi direttamente in altre istruzioni (p.es. in istruzioni di assegnazione).

Alcune funzioni sono già definite all'interno del linguaggio stesso e quindi non richiedono una definizione esplicita da parte dell'utente. Tipici casi sono le funzioni abs, sin, cos, sqrt, log, ecc.

Sottoprogrammi in Fortran

In Fortran, il sottoprogramma viene definito *subroutine* ed ha lo schema seguente:

```
SUBROUTINE nome(lista parametri formali)

  parte dichiarativa
  (contiene anche le dichiarazioni dei parametri formali)

  parte esecutiva

  RETURN

END SUBROUTINE
```

Ogni subroutine è un'unità di programma indipendente ed è compilata separatamente rispetto al programma principale ed agli altri sottoprogrammi.

Per questo motivo, i nomi delle variabili usati in una subroutine possono essere utilizzati anche in altre unità di programma senza che si possa generare confusione.

```
PROGRAM ProvaSubroutine
  IMPLICIT NONE
  REAL :: a,b,c

  WRITE(*,*)"a: "
  READ(*,*) a
  WRITE(*,*)"b: "
  READ(*,*) b

  CALL somma(a,b,c)
  WRITE(*,*)
  WRITE(*,*) "La somma di ",a," e di ",b," e' ",c

  WRITE(*,*) "Premi ENTER per terminare"
  READ(*,*)

END PROGRAM

SUBROUTINE somma(x,y,z)
  IMPLICIT NONE
  REAL :: x,y,z

  z=x+y

  RETURN
END SUBROUTINE
```

Sottoprogrammi in Fortran (2)

Per attivare l'esecuzione della subroutine, il programma chiamante usa l'istruzione `CALL` nella forma:

`CALL` *`nome`*`(`*`lista parametri effettivi`*`)`

La lista dei parametri effettivi deve accordarsi con la lista dei parametri formali per quanto riguarda il numero, il tipo e l'ordine dei parametri.

In Fortran, la corrispondenza tra parametri formali ed effettivi viene realizzata *per riferimento*, per cui <u>ogni modifica fatta sui parametri formali si riflette sui parametri effettivi</u>.

L'istruzione `RETURN` nella subroutine termina l'esecuzione del sottoprogramma e restituisce il controllo al programma chiamante. E' comunque facoltativa e, in sua assenza, il sottoprogramma termina con l'istruzione immediatamente precedente l'`END SUBROUTINE`.

Sottoprogrammi ed array in Fortran

Una variabile di tipo array è un parametro possibile per un sottoprogramma. In questo caso bisogna tenere presente, però, che della variabile array viene effettivamente passato solo l'indirizzo della prima locazione di memoria su cui l'array è stato allocato, mentre nessuna informazione viene data riguardo al cardinalità dell'array.

E' quindi necessario passare esplicitamente la dimensione dell'array come ulteriore argomento ed usare tale argomento nella dichiarazione della variabile array che costituisce il parametro formale.

```
SUBROUTINE leggi_vet(vet,maxnum,num)
  IMPLICIT NONE
  INTEGER :: num,maxnum
  REAL :: vet
  DIMENSION vet(maxnum)
```

```
PROGRAM MaxArray
  IMPLICIT NONE
  INTEGER :: cont,maxnum,riemp,i
  REAL :: vet, max
  PARAMETER(maxnum=10)
  DIMENSION vet(maxnum)

  CALL leggi_vet(vet,maxnum,riemp)
  CALL maxvet(vet,riemp,max)

  WRITE(*,*) "Numero valori letti: ",riemp
  WRITE(*,*)
  WRITE(*,*) "Valori: ",(vet(i),i=1,riemp)
  WRITE(*,*)
  WRITE(*,*) "Il valore massimo e' ",max

  WRITE(*,*) "Premi ENTER per terminare"
  READ(*,*)

END PROGRAM

SUBROUTINE leggi_vet(vet,maxnum,num)
  IMPLICIT NONE
  INTEGER :: maxnum,num,i
  REAL :: vet,x
  DIMENSION vet(maxnum)

  WRITE(*,*)"Numero elementi: "
  READ(*,*) num

  DO WHILE (num>maxnum)
    WRITE(*,*)"Numero elementi eccessivo."
    WRITE(*,*)"Numero elementi: "
    READ(*,*) num
  END DO

  DO i=1,num
    WRITE(*,*)"Valore dell'elemento ",i," : "
    READ(*,*) vet(i)
  END DO
END SUBROUTINE
```

```
SUBROUTINE maxvet(vet,riemp,max)
  IMPLICIT NONE
  INTEGER:: riemp
  REAL :: vet,max
  DIMENSION vet(riemp)
  INTEGER :: i

  max=vet(1)

  DO i=2,riemp
    IF(vet(i)>max) THEN
      max=vet(i)
    END IF
  END DO

  RETURN
END SUBROUTINE
```

Sottoprogrammi ed array in Fortran (2)

Come variabile locale di un sottoprogramma è possibile definire un *array automatico*, la cui cardinalità non è una costante, ma viene definita tramite uno dei parametri formali. L'array automatico non è passato al sottoprogramma, ma viene creato all'interno del sottoprogramma: più precisamente, viene allocato quando viene iniziata l'esecuzione del sottoprogramma per essere deallocato al termine.

```
SUBROUTINE copia_vet(vet,maxnum,num)
  IMPLICIT NONE
  INTEGER :: num,maxnum
  REAL :: vet,temp
  DIMENSION vet(maxnum)  ! array passato alla subroutine
  DIMENSION temp(maxnum) ! array automatico
```

Funzioni in Fortran

Oltre alle subroutine, il Fortran mette a disposizione anche le *function.*

La definizione di una function è simile a quella di una subroutine tranne che per due aspetti:

- è necessario precisare il tipo del valore restituito dalla function
- è necessario assegnare al nome della funzione il valore che si vuole restituire

```
Tipo FUNCTION nome(lista parametri formali)

  parte dichiarativa
  (contiene anche le dichiarazioni dei parametri formali)

  parte esecutiva

  nome=espressione !qui si precisa il valore da restituire
  RETURN

END FUNCTION
```

Funzioni in Fortran (2)

Per attivare una function non viene usata l'istruzione `CALL`. E' sufficiente specificarne il nome (seguito dalla lista dei parametri effettivi) all'interno di un'espressione. La valutazione dell'espressione causerà l'esecuzione della funzione.

Al termine dell'esecuzione, al nome della funzione sarà attribuito un valore che concorrerà alla valutazione dell'intera espressione.

Perché una funzione possa essere usata in un programma, è necessario introdurre una dichiarazione (simile a quella di una variabile) che specifichi il nome della funzione ed il tipo ad essa associato.

```
PROGRAM ProvaFunction
  IMPLICIT NONE
  REAL :: a,b,c
  REAL :: funsomma

  WRITE(*,*)"a: "
  READ(*,*) a
  WRITE(*,*)"b: "
  READ(*,*) b

  c=funsomma(a,b)
  WRITE(*,*)
  WRITE(*,*) "La somma di ",a," e di ",b," e' ",c

  WRITE(*,*) "Premi ENTER per terminare"
  READ(*,*)

END PROGRAM


REAL FUNCTION funsomma(x,y)
  IMPLICIT NONE
  REAL :: x,y

  funsomma=x+y

  RETURN
END FUNCTION
```

```
PROGRAM SincFun
  IMPLICIT NONE
  REAL :: x
  REAL :: sinc

  WRITE(*,*)"x: "
  READ(*,*) x

  WRITE(*,*)
  WRITE(*,*) "Sinc(",x,")= ",sinc(x)

  WRITE(*,*) "Premi ENTER per terminare"
  READ(*,*)

END PROGRAM



REAL FUNCTION sinc(x)

  IMPLICIT NONE
  REAL :: x
  REAL :: eps
  PARAMETER(eps=1.0E-30)

  IF(ABS(x) > eps) THEN
    sinc=sin(x)/x
  ELSE
    sinc=1.
  ENDIF

END FUNCTION
```

Funzioni implicite

Il Fortran mette a disposizione un insieme di funzioni predefinite (dette *implicite*). Alcuni esempi:

```
SIN(x)
COS(x)
TAN(x)
ASIN(x)
ACOS(x)
ATAN(x)
SQRT(x)
LOG(x)
EXP(x)
LOG10(x)
```

```
ABS(a)
MOD(m,n)
ACHAR(n)
IACHAR(c)
```

```
AINT(x)
ANINT(x)
CEILING(x)
FLOOR(x)
DBLE(a)
```

`a`	valore di qualunque tipo numerico
`x`	valore real
`m,n`	valore integer
`c`	valore character

I sottoprogrammi nella progettazione del programma

L'uso dei sottoprogrammi permette di organizzare in modo particolarmente efficace la progettazione di un programma. Infatti, con l'uso dei sottoprogrammi è possibile:

- articolare il programma complessivo in più sottoprogrammi, ognuno dei quali realizza un compito preciso e limitato, rendendo più semplice la comprensione e la manutenzione del programma
- progettare, codificare e verificare ad uno ad uno i singoli sottoprogrammi
- riutilizzare in un programma diverso un sottoprogramma già codificato e verificato
- limitare al minimo gli errori dovuti ad interazioni non previste tra parti diverse del programma (effetti collaterali)

```
PROGRAM OrdinaSelect2
  ! Lettura di un array di valori interi
  ! ed ordinamento degli elementi in
  ! ordine crescente.
  ! Il programma fornisce in uscita l'array ordinato
  ! Versione 2 con l'uso di subroutine

  IMPLICIT NONE
  INTEGER :: maxnum,riemp
  INTEGER :: vet
  PARAMETER(maxnum=10)
  DIMENSION vet(maxnum)

  ! Lettura dell'array
  CALL leggivet(vet,maxnum,riemp)

  ! Ordinamento dell'array
  CALL select(vet,riemp)

  ! Stampa dell'array ordinato
  WRITE(*,*)"Array ordinato: "
  CALL stampavet(vet,riemp)

  WRITE(*,*) "Premere ENTER per terminare"
  READ(*,*)

END PROGRAM
```

```
SUBROUTINE leggivet(v,maxnum,num)
  IMPLICIT NONE
  INTEGER :: maxnum,num,v,i
  DIMENSION v(maxnum)

  WRITE(*,*)"Quanti elementi ?"
  READ(*,*) num
  ! Lettura degli elementi dell'array
  DO i=1,num
    WRITE(*,*)"Valore dell'elemento ",i," :"
    READ(*,*) v(i)
  END DO
END SUBROUTINE
```

```
SUBROUTINE stampavet(v,riemp)
  IMPLICIT NONE
  INTEGER :: v,riemp
  INTEGER k
  DIMENSION v(riemp)

  WRITE(*,*) (v(k),k=1,riemp)

END SUBROUTINE
```

```
SUBROUTINE select(v,riemp)
  IMPLICIT NONE
  INTEGER :: v,riemp
  INTEGER :: i,j,posmin,min,appo
  DIMENSION v(riemp)

  ! Ciclo esterno
  DO i=1,riemp-1
    ! Stampa di controllo
    WRITE(*,*)"Ciclo esterno - indice: ",i
    posmin=i
    min=v(i)

    ! Ciclo interno
    ! Ricerca del minimo tra i e riemp
    DO j=i+1,riemp
      IF(v(j)<min) THEN
        min=v(j)
        posmin=j
      END IF
    END DO

    ! Stampa di controllo
    WRITE(*,*)"Minimo trovato in pos. ",posmin

    ! Aggiornamento dell'elemento i
    IF(posmin/=i) THEN
      ! Stampa di controllo
      WRITE(*,*)"Scambio dell'elemento ",i," con
                 l'elemento ",posmin

      appo=v(i)
      v(i)=v(posmin)
      v(posmin)=appo
    END IF

    ! Stampa di controllo
    WRITE(*,*)"Array al termine del passo ",i," :"
    CALL stampavet(v,riemp)
    WRITE(*,*)
  END DO ! Fine ciclo esterno

END SUBROUTINE
```