

## **Sottoprogrammi**

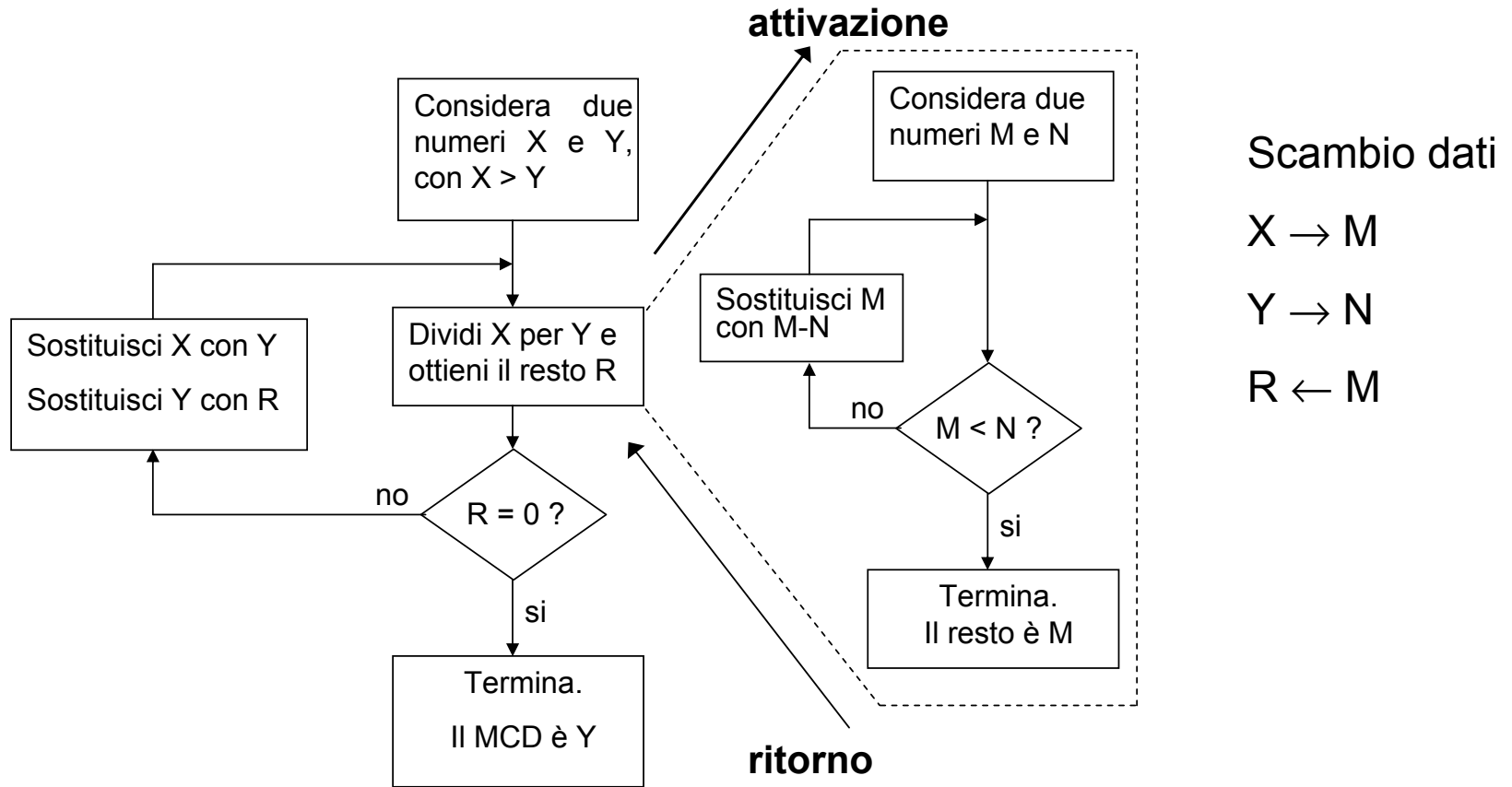
Un sottoprogramma è una particolare unità di codice che non può essere eseguita autonomamente, ma soltanto su richiesta del programma principale (la parte di codice delimitata da `PROGRAM` ed `END PROGRAM`) o di un altro sottoprogramma.

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



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 l'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.

## 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 ogni modifica fatta sui parametri formali si riflette sui parametri effettivi.

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)
```

## 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.

## Funzioni implicite

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

SIN ( x )

ABS ( a )

AINT ( x )

COS ( x )

MOD ( m , n )

ANINT ( x )

TAN ( x )

ACHAR ( n )

CEILING ( x )

ASIN ( x )

IACHAR ( c )

FLOOR ( x )

ACOS ( x )

DBLE ( a )

ATAN ( x )

SQRT ( x )

LOG ( x )

EXP ( x )

LOG10 ( x )

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)