Sottoprogrammi

- Un sottoprogramma è una particolare unità di codice che non può essere eseguita autonomamente, ma soltanto su richiesta del programma principale 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.

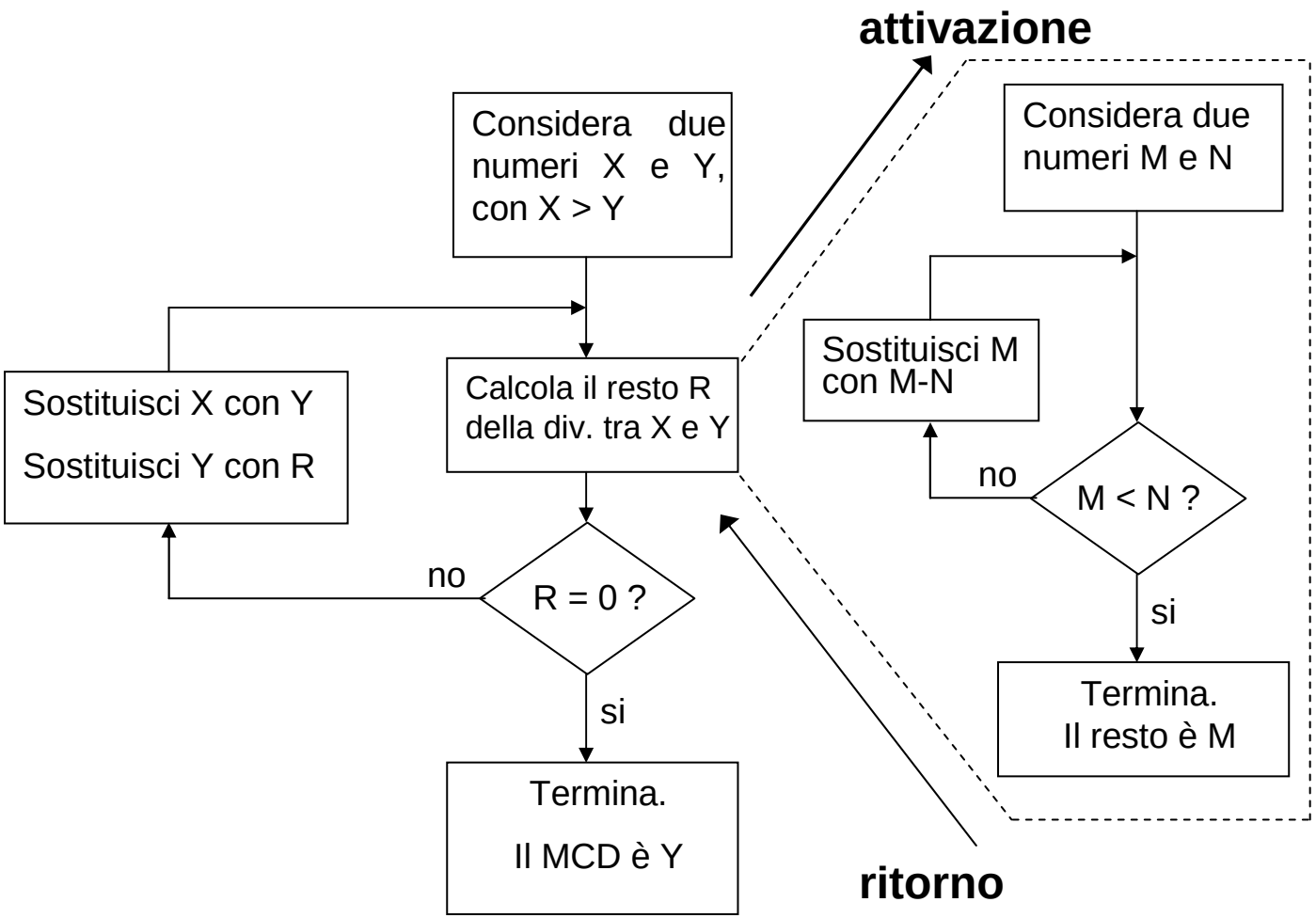
Sottoprogrammi

- 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.

Sottoprogrammi: definizione

- Nel definire un sottoprogramma è quindi necessario precisare:
  - Quale operazione esso realizza
  - Qual è il flusso di dati tra il sottoprogramma ed il codice che lo ha attivato ed, in particolare:
    - Quali sono i dati in ingresso al sottoprogramma
    - Quali sono i dati in uscita dal sottoprogramma

Sottoprogrammi: esempio



Scambio dati

$X \rightarrow M$

$Y \rightarrow N$

$R \leftarrow M$

Sottoprogrammi: attivazione

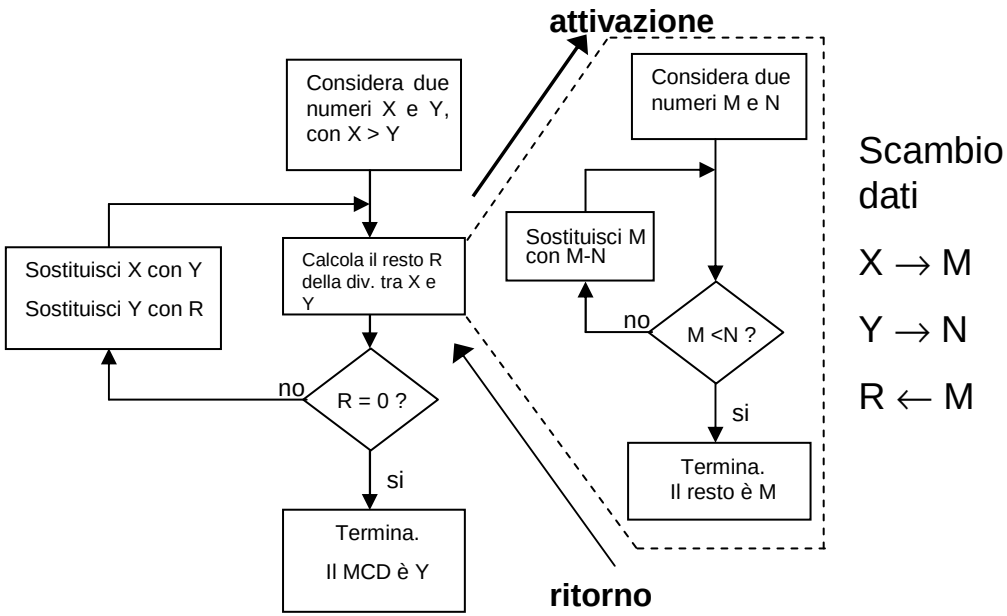
- L'esecuzione delle istruzioni di un sottoprogramma è provocata da una particolare istruzione del programma che lo attiva (istruzione di *chiamata*, per cui il programma è anche detto *chiamante*). Ciò determina la sospensione dell'esecuzione delle istruzioni del programma chiamante, che riprenderà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

Sottoprogrammi: flusso di dati

- 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.

Sottoprogrammi: proprietà

- Il programma chiamante vede realizzato il compito richiesto, ma non conosce i dettagli di come questo venga fatto.
- L'esecuzione del chiamante viene sospesa all'atto della chiamata e ripresa al ritorno dal sottoprogramma
- Prima dell'esecuzione del sottoprogramma, i suoi parametri formali (M ed N) vengono inizializzati con i valori dei parametri effettivi (X e Y) forniti dal chiamante.
- Al ritorno dal sottoprogramma, il chiamante vede modificate alcune sue variabili (R)
- Da un punto di vista progettuale, il sottoprogramma è completamente autonomo rispetto al programma chiamante.



Ambienti e visibilità

- L'insieme delle variabili definite in un sottoprogramma può dividersi in due insiemi:
  - Parametri formali: utilizzati per gestire il flusso di dati con il chiamante
  - Variabili locali: utilizzate per implementare l'algoritmo nel sottoprogramma (es. indici, variabili di appoggio, ecc.
- L'insieme di queste variabili viene definito *ambiente* del sottoprogramma.
- Analogamente, l'insieme delle variabili definite nel chiamante costituisce l'ambiente del chiamante.

Ambienti e visibilità

- Quale relazione esiste tra ambiente del chiamante e ambiente del sottoprogramma ?
- In altre parole, il sottoprogramma ha la *visibilità* (cioè, può fare uso) delle variabili del chiamante e viceversa ?

Ambienti e visibilità

- Sono **due ambienti distinti** per cui:
  - Le variabili del chiamante non sono visibili dal sottoprogramma e viceversa.
  - Nei due ambienti possono quindi esistere variabili con lo stesso nome, ma sono due variabili distinte e separate.
  - L'unico canale per scambiarsi dati è quindi fornito dallo scambio di parametri.

Tecniche di scambio di parametri

- Esistono due tecniche principali per lo scambio di parametri:
  - **Scambio per valore**
  - **Scambio per riferimento**

Scambio per valore

- Nello scambio per valore, il valore del parametro effettivo viene copiato nel parametro formale.
- Il parametro formale costituisce quindi una copia locale del parametro effettivo.
- Ogni modifica fatta sul parametro formale non si riflette sul parametro effettivo.

Scambio per riferimento

- Nello scambio per riferimento, al parametro formale viene assegnato l'indirizzo del parametro effettivo.
- In questo modo, al sottoprogramma è possibile accedere al registro che ospita il parametro effettivo e fare delle modifiche che saranno poi visibili al programma chiamante.

Il ruolo dei sottoprogrammi nella progettazione dei programmi

- 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)

Sottoprogrammi in Matlab

- I sottoprogrammi in Matlab sono realizzati tramite le *function*
- Una function è definita come :

Parametri formali di ingresso

Parametri formali di uscita

function z = somma(x,y)

blocco_istruzioni

Nome della funzione

Sottoprogrammi in Matlab: Nome della funzione

- Il nome identifica univocamente la funzione
- I nomi delle funzioni hanno gli stessi vincoli dei nomi delle variabili. Il nome deve cominciare con una lettera che può essere seguita da una combinazione di lettere, cifre, underscore.

Sottoprogrammi in Matlab: Gestione dei parametri

- Lo scambio parametri avviene <u>per valore</u>
- Nel caso ci siano più parametri di uscita, questi sono elencati tra parentesi quadre [ ].
- La corrispondenza tra parametri formali ed effettivi avviene secondo l'ordine di elencazione.

Sottoprogrammi in Matlab: Blocco istruzioni

- Il blocco istruzioni contiene tutto il codice che implementa le operazioni che la funzione deve realizzare.
- Le istruzioni possono essere costrutti di qualunque tipo (calcolo e assegnazione, I/O, commenti, linee vuote, chiamate di altre funzioni). Tra le istruzioni presenti nel blocco, devono essere comprese le assegnazioni ai parametri formali di uscita

Sottoprogrammi in Matlab: esempio

```
function z=somma(x,y)
   % calcola la somma di x e y

   % variabili usate
   % s: variabile di appoggio

   s = x+y;
   z = s;
```

Invocazione della funzione

- La function viene invocata da un'istruzione che contiene il nome della funzione, tipicamente a destra di un'istruzione di calcolo e assegnazione.
- Nell'invocazione della funzione, in corrispondenza dei parametri formali, sono presenti delle espressioni (per i parametri di ingresso) e delle variabili (per i parametri di uscita) che, insieme, formano i parametri effettivi.

Sottoprogrammi in Matlab: esempio

```
function main
    % programma chiamante
    % variabili usate
    % a,b,c,d: variabili di
    % input
    % e,f: variabili di output
    a = input('a: ');
    b = input('b: ');
    c = input('c: ');
    d = input('d: ');
    e = somma(a,b);
    f = somma(c,d);
    fprintf('e: %g   f: %g\n',e,f);
```

```
function z=somma(x,y)
    % calcola la somma di x e y

    % variabili usate
    % s: variabile di appoggio

    s = x+y;
    z = s;
```

programma

Sottoprogrammi in Matlab

- Si implementino come sottoprogrammi alcuni degli algoritmi sugli array visti finora:
  - Lettura di un array
  - Stampa di un array
  - Ricerca del minimo in un array

```
function main
% programma chiamante

% variabili usate
% a,b,c,d: variabili di input
% e,f: variabili di output

% input
a = input('a: ');
b = input('b: ');
c = input('c: ');
d = input('d: ');

% invocazione della funzione
e = somma(a,b);
f = somma(c,d);

fprintf('e: %g   f: %g\n',e,f);
% fine programma principale



function z=somma(x,y)

% variabili usate
% s:    variabile di appoggio

s = x+y;

z = s;
```

```
function [p,c]=cercaocc(vet,num,val)
% restituisce il numero e gli indici delle occorrenze
% del valore val nel vettore vet

% parametri di ingresso
% vet: array su cui effettuare la ricerca
% num: numero degli elementi dell'array
% val: valore di cui cercare le occorrenze

% parametri di uscita
% p: array contenente gli indici delle occorrenze di val
% c: numero di occorrenze trovate

% variabili locali
% i:   indice per scorrere l'array

% dimensionamento dell'array p
p=zeros(num,1);

c=0;
for i=1:num
   if(vet(i)==val)
      c=c+1;
      p(c)=i;
   end
end
% fine funzione cercaocc
```

```
function m=cercamin(vet,num)
% restituisce il minimo del vettore vet

% parametri di ingresso
% vet: array su cui cercare il minimo
% num: numero di elementi nell'array

% parametri di uscita
% m: minimo trovato

% variabili locali
% i:   indice per scorrere l'array

m=vet(1);
for i=2:num
    if(vet(i)<m)
        % trovato nuovo minimo
        m=vet(i);
    end
end
% fine funzione cercamin
```

```
function stampaarray(vet,num)
% Stampa gli elementi dell'array vet

% parametri di ingresso
% vet: array da stampare
% num: numero degli elementi dell'array

% parametri di uscita
% nessuno

% variabili usate
% i:   indice per scorrere l'array

for i=1:num
   fprintf('%g\n',vet(i));
end
% fine funzione stampaarray
```

```
function vet=leggiarray(num)
% legge un array di num elementi

% parametri di ingresso
% num: numero di elementi da leggere

% parametri di uscita
% vet: array letto

% dimensionamento array
vet=zeros(num,1);

% ciclo di lettura
for i=1:num
   fprintf('Valore %d: ',i);
   vet(i)=input('');
end
% fine funzione leggiarray
```

```
function [p,c]=cercaocc(vet,num,val)
% restituisce il numero e gli indici delle occorrenze
% del valore val nel vettore vet

% parametri di ingresso
% vet: array su cui effettuare la ricerca
% num: numero degli elementi dell'array
% val: valore di cui cercare le occorrenze

% parametri di uscita
% p: array contenente gli indici delle occorrenze di val
% c: numero di occorrenze trovate

% variabili locali
% i:   indice per scorrere l'array

% dimensionamento dell'array p
p=zeros(num,1);

c=0;
for i=1:num
   if(vet(i)==val)
      c=c+1;
      p(c)=i;
   end
end
% fine funzione cercaocc
```

```
function vet=leggiarray(num)
% legge un array di num elementi

% parametri di ingresso
% num: numero di elementi da leggere

% parametri di uscita
% vet: array letto

% dimensionamento array
vet=zeros(num,1);
% ciclo di lettura
for i=1:num
    fprintf('Valore %d: ',i);
    vet(i)=input('');
end
% fine funzione leggiarray

function stampaarray(vet,num)
% Stampa gli elementi dell'array vet

% parametri di ingresso
% vet: array da stampare
% num: numero degli elementi dell'array

% parametri di uscita
% nessuno

% variabili usate
% i:   indice per scorrere l'array

for i=1:num
    fprintf('%g\n',vet(i));
end
% fine funzione stampaarray
```

```
function main
% lettura di un array, ricerca delle posizioni
% delle occorrenze di un valore

% variabili utilizzate
% v:    array in input
% n:    dimensione dell'array fornito in input
% val:  variabile contenente il valore da cercare
% pos:  array contenente gli indici delle occorrenze
% cont: variabile contenente il numero di occorrenze trovate

% input dimensione
n=input('Numero elementi: ');

% input array
v = leggiarray(n);

% ricerca del minimo
val = input('Valore: ');

% ricerca delle occorrenze
[pos,cont] = cercaocc(v,n,val);

% stampa dei risultati
fprintf('\nArray letto:\n');
stampaarray(v,n);

if(cont==0)
   fprintf('\nIl valore %g non è presente nell''array.\n',val);
else
   fprintf('\nIl valore %g è presente %d volte nelle seguenti posizioni:\n',val,cont);
   stampaarray(pos,cont);
   fprintf('\n');
end
%%%%%% fine main %%%%%%%%%%%%%%%%%
```

```
function stampaarray(vet,num)
% Stampa gli elementi dell'array vet

% parametri di ingresso
% vet: array da stampare
% num: numero degli elementi dell'array

% parametri di uscita
% nessuno

% variabili usate
% i:   indice per scorrere l'array

for i=1:num
   fprintf('%g\n',vet(i));
end
% fine funzione stampaarray
```

```
function [p,c]=cercaocc(vet,num,val)
% restituisce il numero e gli indici delle occorrenze
% del valore val nel vettore vet

% parametri di ingresso
% vet: array su cui effettuare la ricerca
% num: numero degli elementi dell'array
% val: valore di cui cercare le occorrenze

% parametri di uscita
% p: array contenente gli indici delle occorrenze di val
% c: numero di occorrenze trovate

% variabili locali
% i:   indice per scorrere l'array

% dimensionamento dell'array p
p=zeros(num,1);

c=0;
for i=1:num
    if(vet(i)==val)
        c=c+1;
        p(c)=i;
    end
end
% fine funzione cercaocc
```

```
function vet=leggiarray(num)
% legge un array di num elementi

% parametri di ingresso
% num: numero di elementi da leggere

% parametri di uscita
% vet: array letto

% dimensionamento array
vet=zeros(num,1);
% ciclo di lettura
for i=1:num
    fprintf('Valore %d: ',i);
    vet(i)=input('');
end
% fine funzione leggiarray


function m=cercamin(vet,num)
% restituisce il minimo del vettore vet

% parametri di ingresso
% vet: array su cui cercare il minimo
% num: numero di elementi nell'array

% parametri di uscita
% m: minimo trovato

% variabili locali
% i:   indice per scorrere l'array

m=vet(1);
for i=2:num
    if(vet(i)<m)
        % trovato nuovo minimo
        m=vet(i);
    end
end
% fine funzione cercamin
```

```
function main
% lettura di un array, ricerca del minimo e ricerca
% delle posizioni delle occorrenze del minimo

% variabili utilizzate
% v:    array in input
% n:    dimensione dell'array fornito in input
% min:  variabile contenente il minimo
% pos:  array contenente gli indici delle occorrenze
% cont: variabile contenente il numero di occorrenze trovate

% input dimensione
n=input('Numero elementi: ');

% input array
v = leggiarray(n);

% ricerca del minimo
min = cercamin(v,n);

% ricerca delle occorrenze
[pos,cont] = cercaocc(v,n,min);

% stampa dei risultati
fprintf('\nArray letto:\n');
stampaarray(v,n);

% stampa del minimo e delle occorrenze
fprintf('\nIl valore minimo nell''array è %g.\n',min);
fprintf('E'' presente %d volte nelle seguenti posizioni:\n',cont);
stampaarray(pos,cont);
fprintf('\n');
%%%%%% fine main %%%%%%%%%%%%%%%%
```