

Funzioni in C

Modello cliente/servitore

Servitore:

- Un qualunque ente computazionale capace di nascondere la propria organizzazione interna
- **presentando ai clienti una precisa *interfaccia*** per lo scambio di informazioni

Cliente:

- qualunque ente in grado di **invocare uno o più servitori** per svolgere il proprio compito



Comunicazione cliente/servitore

- Lo scambio di informazioni tra un cliente e un servitore può avvenire
 - in *modo esplicito* tramite le *interfacce* stabilite dal servitore
 - in *modo implicito* tramite *aree-dati* accessibili ad entrambi, ossia l'ambiente condiviso

Interfaccia di una funzione

- L'**interfaccia** (o firma o *signature* o *prototipo*) di una funzione comprende
 - *nome della funzione*
 - *lista dei parametri*
 - *tipo del valore da essa denotato*
- *Esplicita il contratto di servizio* fra cliente e servitore
- Cliente e servitore comunicano quindi mediante
 - i *parametri* trasmessi dal cliente al servitore all'atto della chiamata
 - il *valore restituito* dal servitore al cliente

Funzioni: esempio C

```
int max (int x, int y ){  
    if (x>y) return x ;  
    else return y;  
}
```

- Il simbolo **max** denota il nome della funzione
- Le variabili intere **x** e **y** sono i parametri della funzione
- Il valore restituito è un intero **int**

Comunicazione cliente/servitore

- Il cliente passa informazioni al servitore mediante una serie di ***parametri***

Parametri **formali**:

- sono specificati nella *dichiarazione* del servitore
- esplicitano *il contratto* fra servitore e cliente
- indicano *che cosa il servitore si aspetta dal cliente*

Parametri **attuali**:

- sono *trasmessi dal cliente* all'atto della chiamata
- devono corrispondere ai parametri formali in *numero, posizione e tipo*

Funzioni: esempio parametri

Parametri formali

```
int max (int x, int y) {  
    if (x>y) return x ;  
    else return y;  
}
```

SERVITORE

Definizione
della funzione

```
main() {  
    int z = 8;  
    int m;  
  
    m = max (z, 4);  
}
```

CLIENTE

Chiamata
della funzione

Parametri attuali

Comunicazione cliente/servitore

Legame tra parametri attuali e parametri formali:

- effettuato *al momento della chiamata*,
in modo dinamico

Tale legame:

- vale solo per l'invocazione corrente
- vale solo per la durata della funzione

Esempio

Parametri formali

```
int max (int x, int y) {  
    if (x>y) return x ;  
    else return y;  
}
```

```
main() {  
    int z = 8;  
    int m1, m2;  
    m1 = max (z, 4);  
    m2 = max (5, z);  
}
```

All'atto di questa
chiamata della funzione,
si effettua un legame tra:

x e z

y e 4

Esempio

Parametri formali

```
int max (int x, int y) {  
    if (x>y) return x ;  
    else return y;  
}
```

```
main() {  
    int z = 8;  
    int m1, m2;  
    m1 = max (z, 4);  
    m2 = max (5, z);  
}
```

All'atto di questa
chiamata della funzione,
si effettua un legame tra:

x e 5

y e z

Information hiding

- La *struttura interna* (corpo) di una funzione è *completamente inaccessibile dall'esterno*
- Così facendo si garantisce *protezione dell'informazione* (*information hiding*)
- Una funzione è accessibile **solo** attraverso la sua interfaccia

Definizione di funzione in C

```
<definizione-di-funzione> ::=  
    <tipoValore> <nome>(<parametri-formali>) {  
        <corpo>  
    }
```

La forma base è:
return <espressione>;

<parametri-formali>

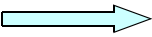
- o una **lista vuota**: **void**
- o una **lista di variabili** (separate da virgole)
visibili solo entro il corpo della funzione

<tipoValore>

- deve coincidere con il tipo del valore restituito dalla funzione
- Può non esservi valore restituiti, in tal caso il tipo è **void**

Definizione di funzione in C

```
<definizione-di-funzione> ::=  
    <tipoValore> <nome>(<parametri-formali>) {  
        <corpo>  
    }
```



La forma base è:
return <espressione>;

- Nella parte **corpo** possono essere presenti definizioni e/o dichiarazioni locali (*parte dichiarazioni*) e un insieme di istruzioni (*parte istruzioni*)
- I dati riferiti nel corpo possono essere **costanti**, **variabili**, oppure **parametri formali**
- All'interno del corpo, i parametri formali vengono trattati come variabili

Funzioni: nascita e morte

-
- All'atto della chiamata,
**l'esecuzione del cliente viene sospesa
e il controllo passa al servitore**
 - Il servitore “*vive*” solo per il tempo necessario a svolgere il servizio
 - Al termine, il servitore “*muore*”,
e l'esecuzione torna al cliente

Chiamata di funzione

- La chiamata di funzione è un'espressione della forma

`<nomefunzione> ( <parametri-attuali> )`

- dove:

`<parametri-attuali> ::=`
`[ <espressione> ] { , <espressione> }`

Funzioni: esempio

Parametri formali

```
int max (int x, int y) {  
    if (x>y) return x ;  
    else return y;  
}
```

SERVITORE

Definizione
della funzione

```
main() {  
    int z = 8;  
    int m;  
  
    m = max (z, 4);  
}
```

CLIENTE

Chiamata
della funzione

Parametri attuali

Risultato di una funzione

- L'istruzione **return** provoca la *restituzione del controllo al cliente*, unitamente al valore dell'espressione che la segue
- Eventuali istruzioni successive alla **return** *non saranno mai eseguite*

```
int max (int x, int y ){  
    if (x>y) return x ;  
    else return y;  
}
```

Funzioni: esempio

Parametri formali

```
int max (int x, int y ){  
    if (x>y) return x ;  
    else return y;  
}
```

SERVITORE

Definizione
della funzione

```
main() {  
    int z = 8;  
    int m;  
  
    m = max (z, 4);  
}
```

CLIENTE

Chiamata
della funzione

Risultato

Riassumendo

All'atto dell'invocazione di una funzione:

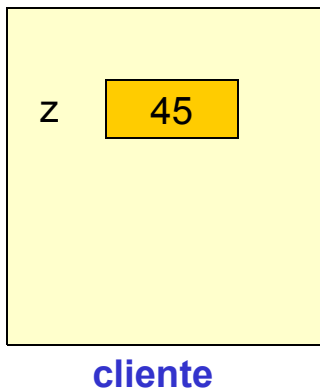
- si crea una **nuova attivazione** (*istanza*) del servitore
- si alloca la **memoria per i parametri** (e le eventuali variabili locali)
- si trasferiscono i parametri al servitore
- si trasferisce il controllo al servitore
- si esegue il codice della funzione

Passaggio dei parametri

- In generale, un parametro può essere trasferito dal cliente al servitore:
 - per **valore** o **copia** (*by value*)
 - si trasferisce **il valore** del parametro attuale
 - per **riferimento** (*by reference*)
 - si trasferisce **un riferimento** al parametro attuale

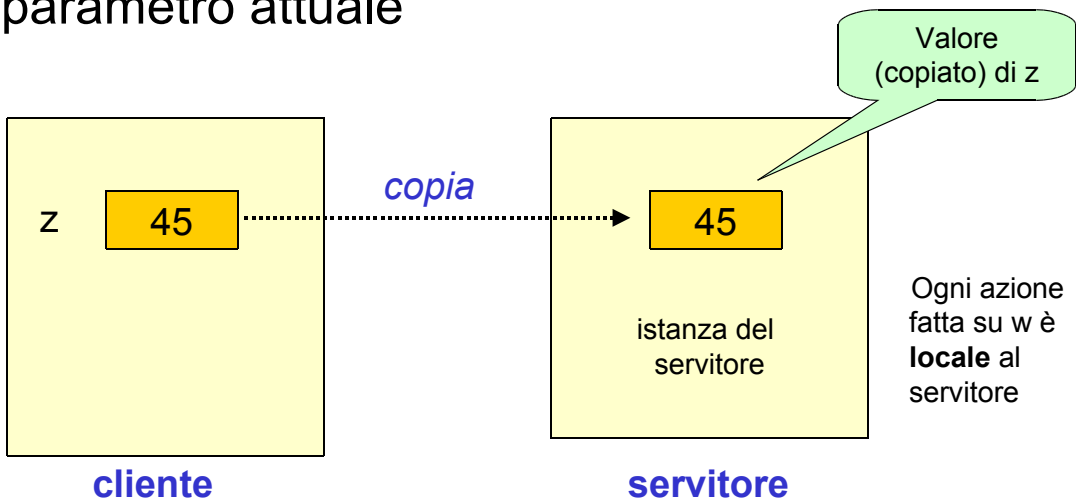
Passaggio per valore

- Si trasferisce **una copia del valore** del parametro attuale



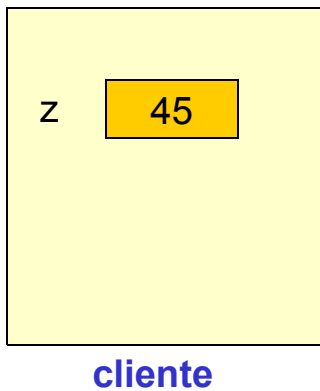
Passaggio per valore

- Si trasferisce **una copia del valore** del parametro attuale



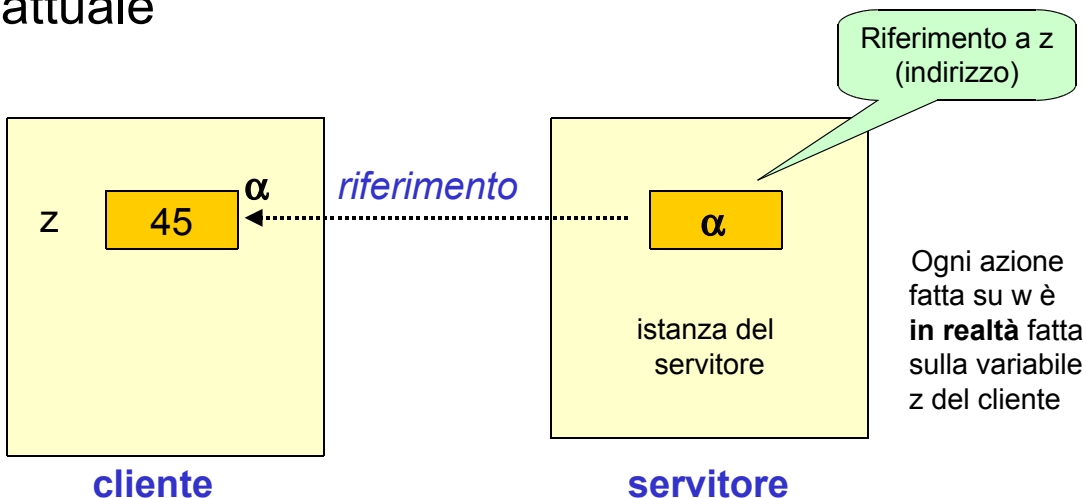
Passaggio per riferimento

- Si trasferisce **un riferimento** al parametro attuale



Passaggio per riferimento

- Si trasferisce **un riferimento** al parametro attuale



Passaggio dei parametri in C

- In C, i parametri sono trasferiti sempre e solo **per valore** (*by value*)
 - si trasferisce **una copia** del parametro attuale, *non l'originale*
 - tale copia è *strettamente privata e locale a quel servitore*
 - il servitore potrebbe quindi alterare il valore ricevuto, *senza che ciò abbia alcun impatto sul cliente*

Passaggio dei parametri in C

- In C, i parametri sono trasferiti sempre e solo **per valore** (*by value*)

Conseguenza:

- è impossibile usare un parametro per *trasferire informazioni verso il cliente*
- per trasferire un'informazione al cliente si sfrutta il **valore di ritorno** della funzione

Esempio

Perché il passaggio per valore non basta?

Problema:

scrivere una procedura che *scambi i valori di due variabili intere*

Specifica:

Dette A e B le due variabili, ci si può appoggiare a una *variabile ausiliaria T*, e svolgere lo scambio in *tre fasi*

Frammento di codice:

```
int a,b,t;  
...  
t = a; a = b; b = t;  
...
```

Esempio

-
- Supponendo di utilizzare, senza preoccuparsi, il passaggio per valore usato finora, la codifica potrebbe essere espressa come segue:

```
void scambia(int a, int b) {  
    int t;  
    t = a; a = b; b = t;  
    return; /* può essere omessa */  
}
```

Esempio

Il cliente invocherebbe quindi la procedura così:

```
main() {  
    int y = 5, x = 33;  
    scambia(x, y);  
    /* ora dovrebbe essere  
       x=5, y=33 ...  
       MA NON È VERO  
    */  
}
```

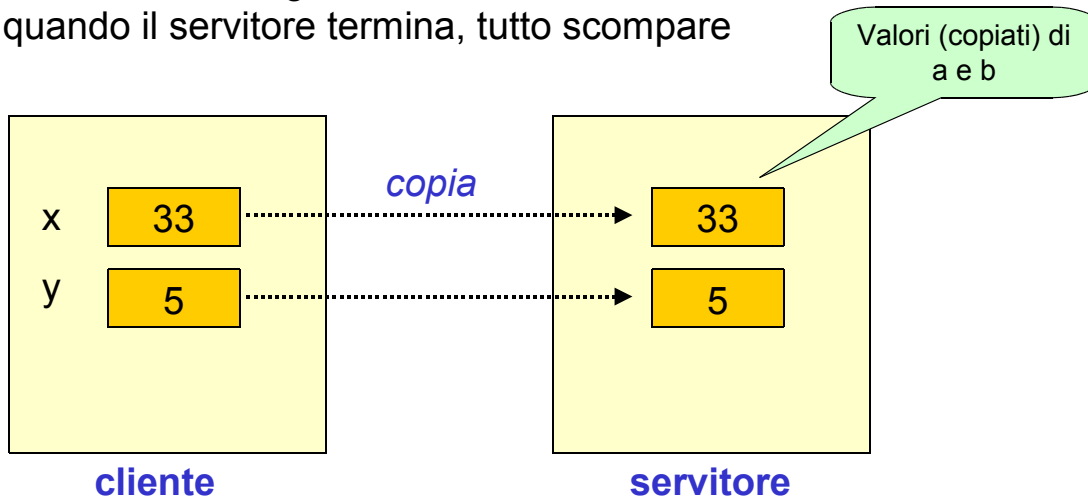
Perché non funziona?

Esempio: cosa è successo?

-
- La procedura ha *effettivamente scambiato* i valori di A e B *al suo interno*
 - *ma questa modifica non si è propagata al cliente*, perché sono state scambiate *le copie locali alla procedura, non gli originali*
 - al termine della procedura, le sue variabili locali *sono state distrutte*, quindi *nulla è rimasto* del lavoro svolto dalla procedura

Esempio: cosa è successo?

- Ogni azione fatta su **a** e **b** è **strettamente locale** al servitore. Quindi **a** e **b** vengono scambiati, ma quando il servitore termina, tutto scompare



Passaggio dei parametri in C

Il C adotta sempre il passaggio per valore!

- È sicuro: le variabili del cliente e del servitore sono disaccoppiate
- ... ma non consente di scrivere componenti software il cui scopo sia diverso dal calcolo di una espressione
- Per superare questo limite occorre il passaggio per riferimento (by reference)

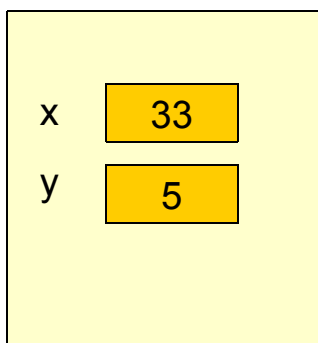
Passaggio per riferimento

Il passaggio per riferimento (*by reference*):

- non trasferisce una copia del valore del parametro attuale
- *ma un riferimento al parametro*, in modo da dare al servitore **accesso diretto** al parametro in possesso del cliente
- il servitore, quindi, **accede direttamente** al dato del cliente e **può modificarlo**

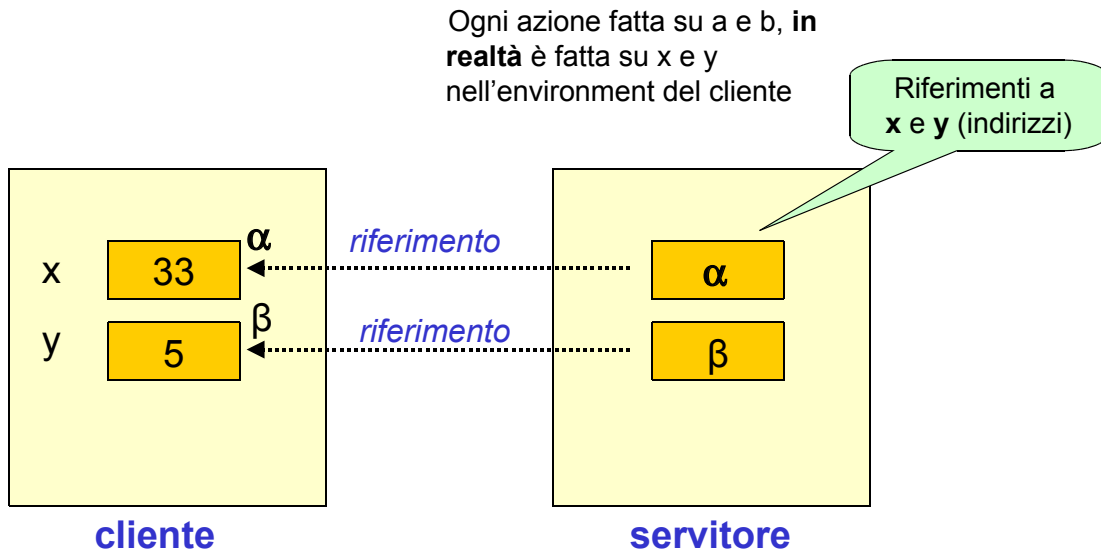
Passaggio per riferimento

- Si trasferisce un riferimento ai parametri attuali (cioè i loro indirizzi)

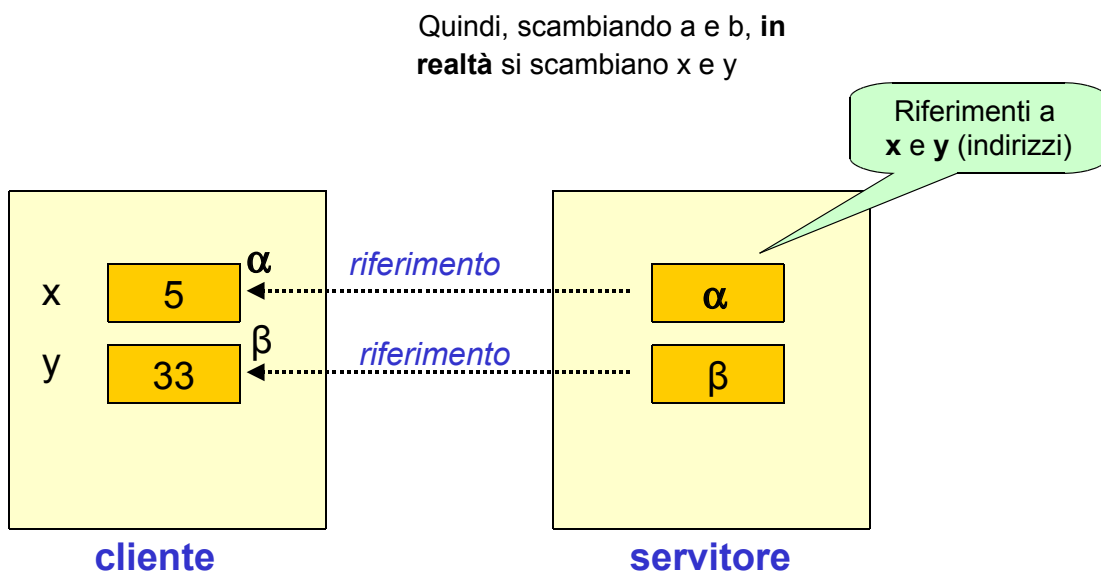


cliente

Passaggio per riferimento



Passaggio per riferimento



Realizzare il passaggio per riferimento in C

- Il C *non* fornisce *direttamente* un modo per attivare il passaggio per riferimento -> a volte occorre *costruirselo*

È possibile costruirlo? Come?

- Poiché passare un parametro per riferimento comporta la capacità di manipolare *indirizzi di variabili* ...
- ... gestire il passaggio per riferimento implica la capacità di *accedere, direttamente o indirettamente, agli indirizzi* delle variabili

Realizzare il passaggio per riferimento in C

- In particolare occorre essere capaci di:
 - ➔ **ricavare l'indirizzo** di una variabile
 - ➔ **dereferenziare un indirizzo** di variabile, ossia “recuperare” il valore dato l'indirizzo della variabile
- Nei linguaggi che offrono direttamente il passaggio per riferimento, *questi passi sono effettuati* in modo trasparente all'utente
- In C il **programmatore deve conoscere gli indirizzi** delle variabili e quindi accedere alla macchina sottostante

Indirizzamento e dereferencing

- Ricordiamoci che il C offre a tale scopo *due operatori*, che consentono di:
 - **ricavare l'indirizzo di una variabile**
 - operatore *estrazione di indirizzo* **&**
 - **dereferenziare un indirizzo di variabile**
 - operatore *di dereferenzamento* *****

Realizzare il passaggio per riferimento in C

- il cliente deve *passare esplicitamente gli indirizzi*
- il servitore deve *prevedere esplicitamente dei **puntatori** come parametri formali*

```
void scambia(int* a, int* b) {  
    int t;  
    t = *a; *a = *b; *b = t;  
}  
  
main() {  
    int y=5, x=33;  
    scambia(&x, &y);  
}
```

Osservazione

- Quando un puntatore è usato per realizzare il passaggio per riferimento, *la funzione non dovrebbe mai alterare il valore del puntatore*

- Quindi, se **a** e **b** sono due puntatori:

***a = *b** **SI**

a = b **NO**

- In generale una funzione può modificare un puntatore, ma *non è opportuno che lo faccia se esso realizza un passaggio per riferimento*

Comunicazione tramite environment globale

- Una procedura può anche comunicare con il cliente mediante *aree dati globali*: ad esempio, *variabili globali*:
 - sono allocate nell'area dati globale (*fuori da ogni funzione*)
 - esistono *prima* della chiamata del *main*
 - sono *inizializzate automaticamente a 0* salvo diversa indicazione
 - possono essere *nascoste* in una funzione da una variabile locale omonima
 - sono visibili, previa dichiarazione *extern*, in tutti i file dell'applicazione
- Il loro uso è fortemente **SCONSIGLIATO** poichè:
 - Rendono le funzioni non autosufficienti, in quanto legate a variabili esterne
 - La modifica delle variabili utilizzate dalla funzione è possibile anche da parte di altre funzioni
 - Minore leggibilità

Esempio

- **Esempio:** Divisione intera x/y con calcolo di quoziente e resto. Occorre calcolare *due* valori che supponiamo di mettere in due variabili globali

```
int quoziente, int resto;
```

Variabili globali **quoziente** e **resto** visibili in tutti i blocchi

```
void dividi(int x, int y) {  
    resto = x % y; quoziente = x/y;  
}
```

```
main() {  
    dividi(33, 6);  
    printf("%d%d", quoziente, resto);  
}
```

Il risultato è disponibile per il cliente nelle variabili globali **quoziente** e **resto**

Esempio

- **Esempio:** Divisione intera x/y con calcolo di quoziente e resto. Con il passaggio dei parametri per indirizzo avremmo il seguente codice.

Non ho più variabili globali

```
void dividi(int x, int y, int *quoziente, int *resto) {  
    *resto = x % y; *quoziente = x/y;  
}
```

```
main() {  
    int k=33, h=6, quoz, rest;  
    int *pq= &quoz, *pr = &rest;  
    dividi(k, h, pq, pr);  
    printf("%d%d", quoz, rest);  
}
```

Il risultato è disponibile per il cliente nelle variabili **quoz** e **rest** di cui ho passato l'indirizzo

Progetto di una funzione

- *Scegliere un nome significativo per la funzione*
- *La funzione deve ricevere qualche dato dalla funzione chiamante?*
 - ➔ Se sì, elencare ed identificare tutti i tipi di dato da passare alla funzione (lista dei parametri)
 - ➔ Se no, la lista dei parametri è void
- *La funzione deve restituire un valore alla funzione chiamante?*
 - ➔ Se sì, identificare il tipo di dato
 - ➔ Se no, il tipo di ritorno della funzione è void
- *La funzione deve restituire più di un valore alla funzione chiamante?*
 - ➔ Se sì, bisogna inserire nella lista dei parametri dei parametri passati esplicitamente per riferimento, cioè dei puntatori

Progetto di una funzione: lista dei parametri

- **Parametri formali:**
argomenti dichiarati nella definizione di funzione
- Devono essere variabili:
 - ➔ Non appena l'ambiente della funzione chiamata **viene attivato**, i parametri formali vengono **dichiarati** (come variabili locali all'ambiente della funzione) ed **inizializzati** al valore del corrispondente parametro attuale
 - ➔ La **corrispondenza** tra parametri formali ed attuali è **sia posizionale sia di tipo**. Ovvero si presume che la lista dei parametri formali e la lista dei parametri attuali abbia lo stesso numero e tipo di elementi
 - ➔ I nomi dei parametri attuali e formali **non hanno importanza**. *Possono essere gli stessi o diversi*. L'importante è la posizione ed il valore che assume un parametro attuale al momento della chiamata

Progetto di una funzione: lista dei parametri

- **Parametri attuali:**

argomenti inseriti al momento della chiamata di funzione

- Possono essere **espressioni** (*costanti, variabili, espressioni aritmetiche, ...*) di qualunque tipo, purchè compatibile con il corrispondente tipo del parametro formale
- **Attenzione:** in corrispondenza di un parametro passato per riferimento (puntatore) bisogna inserire un indirizzo di variabile