

FUNZIONI

Spesso può essere utile avere la possibilità di costruire nuove istruzioni che risolvano parti specifiche di un problema

Una *funzione* permette di

- *dare un nome a una espressione*
- *rendendola parametrica*

Esempi (pseudo-C):

```
float f(){ 2 + 3 * sin(0.75); }
float f1(int x) {
    2 + x * sin(0.75); }
```

FUNZIONI COME COMPONENTI SW

Una *funzione* è un *componente software* che cattura l'idea matematica di funzione

- molti possibili ingressi
(che *non vengono modificati*)
- una sola uscita (il risultato)

Una funzione

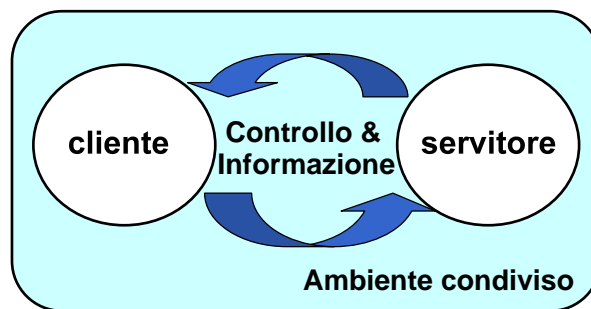
- riceve dati di ingresso in corrispondenza ai *parametri*
- ha come corpo una espressione, la cui valutazione fornisce un risultato
- denota un valore in corrispondenza al suo *nome*

FUNZIONI COME COMPONENTI SW

Esempio

- se **x vale 1**
- e **f** è la funzione $f : \mathbb{R} \rightarrow \mathbb{R}$
 $f = 3 * x^2 + x - 3$
- allora **f(x)** denota il valore **1**

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

MODELLO CLIENTE/SERVITORE

Un servitore può

- servire *molti clienti* oppure costituire la risorsa privata di uno *specifico cliente*
 - in particolare: può servire un cliente alla volta, *in sequenza*, oppure più clienti per volta, *in parallelo*
- *trasformarsi a sua volta in cliente*, invocando altri servitori o anche se stesso
- essere *passivo* o *attivo*

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

FUNZIONI COME SERVITORI

- Una funzione è un servitore
 - *passivo*
 - che serve *un cliente per volta*
 - che *può trasformarsi in cliente invocando altre funzioni o se stessa*
- Una funzione è un *servitore dotato di nome* che incapsula le istruzioni che realizzano un certo servizio
- Il cliente chiede al servitore di svolgere il servizio
 - chiamando tale servitore (per nome)
 - *forrendogli le necessarie informazioni*
- Nel caso di una funzione, cliente e servitore comunicano mediante *l'interfaccia* della funzione

INTERFACCIA DI UNA FUNZIONE

- L'*interfaccia* (o firma o *signature*) 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

ESEMPIO

```
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 attuali**

- **Parametri formali:**
 - sono specificati nella *dichiarazione* del servitore
 - esplicitano *il contratto* fra servitore e cliente
 - indicano *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

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

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 m;  
    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

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

|                                           |
|-------------------------------------------|
| La forma base è<br>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 risultato della funzione

DEFINIZIONE DI FUNZIONE

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

|                                           |
|-------------------------------------------|
| La forma base è<br>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 COME COMPONENTI SW: 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

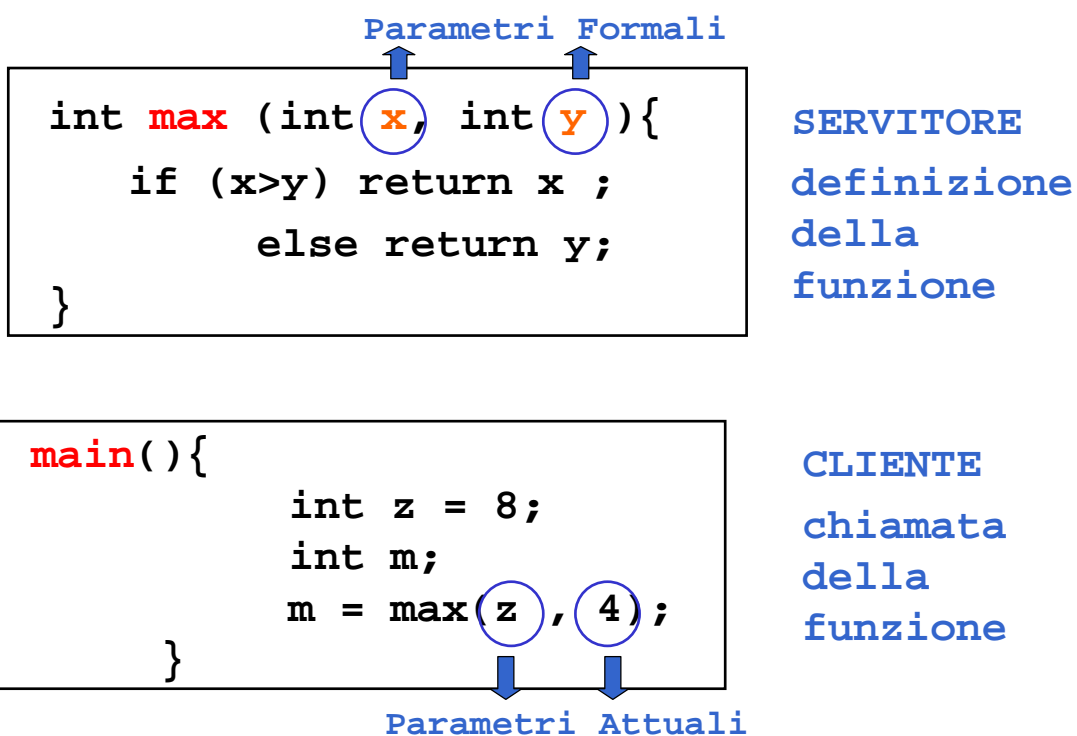
La chiamata di funzione è un'espressione della forma

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

dove:

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

ESEMPIO



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;
}
```

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

BINDING & ENVIRONMENT

`return x;` ➡ devo sapere cosa denota il simbolo x

La conoscenza di che cosa un simbolo denota viene espressa da un *legame* (*binding*) tra il simbolo e uno o più attributi

L'insieme dei *binding* validi in (un certo punto di) un programma si chiama *environment*

ESEMPIO

```
main(){
    int z = 8;
    int y, m;
    y = 5
    m = max(z, y);
}
```

In questo *environment* il simbolo **z** è legato al valore 8 tramite l'inizializzazione, mentre il simbolo **y** è legato al valore 5. Pertanto i parametri di cui la funzione **max** ha bisogno per calcolare il risultato sono noti all'atto dell'invocazione della funzione

ESEMPIO

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

In questo *environment* il simbolo **z** è legato al valore 8 tramite l'inizializzazione, mentre il simbolo **y** non è legato ad alcun valore. Pertanto i parametri di cui la funzione **max** ha bisogno per calcolare il risultato NON sono noti all'atto dell'invocazione della funzione e **la funzione non può essere valutata correttamente**

SCOPE E SCOPE-RULES

- Tutte le occorrenze di un nome nel testo di un programma a cui si applica un dato *binding* si dicono essere entro la stessa *portata* o *scope* del binding
- Le regole in base a cui si stabilisce la *portata* di un binding si dicono *regole di visibilità* o *scope rules*

ESEMPIO

- Il servitore...

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

- ... e un possibile cliente:

```
main(){  
    int z = 8;  
    int m;  
    m = max(2*z,13);  
}
```

*Valutazione del simbolo z
nell'environment corrente
z vale 8*

ESEMPIO

- Il servitore...

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

- ... e un possibile cliente:

```
main(){  
    int z = 8;  
    int m;  
    m = max(2*z,13);  
}
```

*Calcolo dell'espressione
2*z nell'environment
corrente*

*2*z vale 16*

ESEMPIO

- Il servitore...

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

- ... e un possibile cliente:

```
main() {  
    int z = 8;  
    int m;  
    m = max(2*z,13);  
}
```

Invocazione della chiamata a max con parametri attuali 16 e 13
IL CONTROLLO PASSA AL SERVITORE

ESEMPIO

- Il servitore...

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

- ... e un possibile cliente:

```
main() {  
    int z = 8;  
    int m;  
    m = max(2*z,13);  
}
```

Viene effettuato il legame dei parametri formali x e y con quelli attuali 16 e 13
INIZIA L'ESECUZIONE DEL SERVITORE

ESEMPIO

- Il servitore...

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

- ... e un possibile cliente:

```
main() {  
    int z = 8;  
    int m;  
    m = max(2*z,13);  
}
```

*Viene valutata
l'istruzione
condizionale (16 > 13)
che nell'environment
corrente è vera
Pertanto si sceglie la
strada*

`return x`

ESEMPIO

- Il servitore...

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

- ... e un possibile cliente:

```
main() {  
    int z = 8;  
    int m;  
    m = max(2*z,13);  
}
```

*Il valore 16 viene
restituito al cliente
IL SERVITORE
TERMINA E IL
CONTROLLO PASSA
AL CLIENTE*

*NOTA: i binding di x e y
vengono distrutti*

ESEMPIO

- Il servitore...

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

- ... e un possibile cliente:

```
main(){  
    int z = 8;  
    int m;  
    m = max(2*z,13);  
}
```

*Il valore restituito (16)
viene assegnato alla
variabile m
nell'environment del
cliente*

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

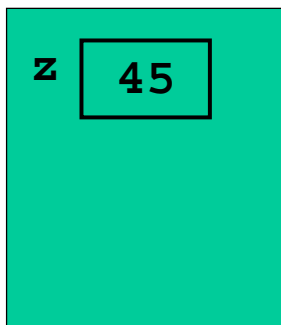
PASSAGGIO DEI PARAMETRI

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

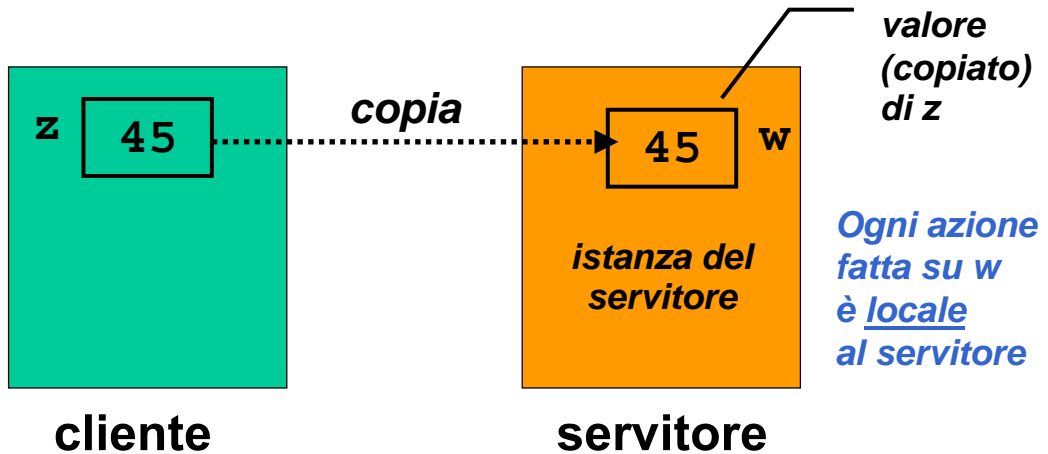
- si trasferisce una copia del valore del parametro attuale



cliente

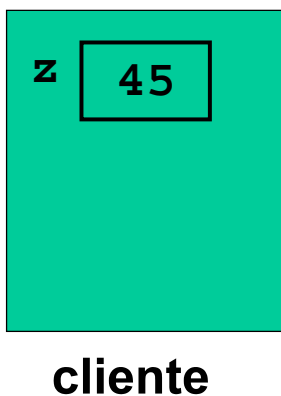
PASSAGGIO PER VALORE

- si trasferisce una copia del valore del parametro attuale



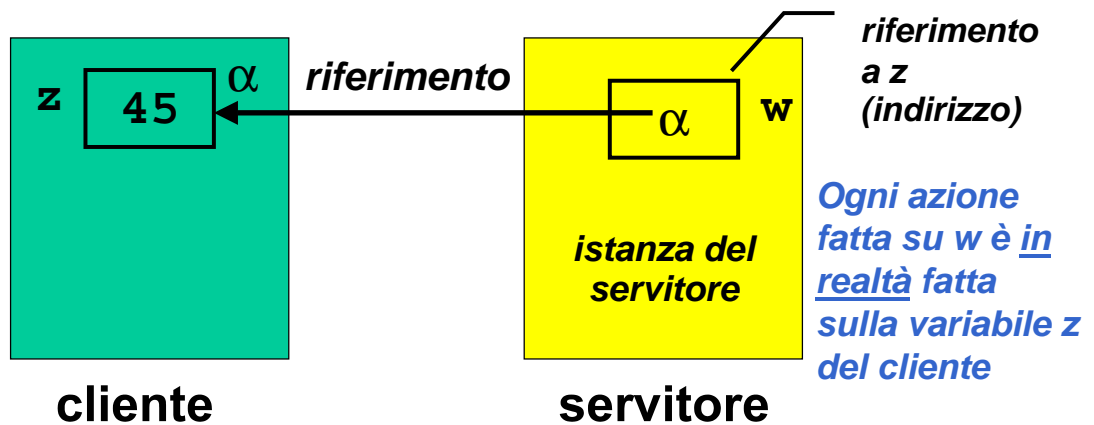
PASSAGGIO PER RIFERIMENTO

- 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: VALORE ASSOLUTO

Definizione formale:

$|x|: \mathbb{Z} \rightarrow \mathbb{N}$

$|x|$ vale x se $x \geq 0$

$|x|$ vale $-x$ se $x < 0$

Codifica sotto forma di funzione C:

```
int valAss(int x) {  
    if (x<0) return -x;  
    else return x;  
}
```

ESEMPIO: VALORE ASSOLUTO

- **Servitore**

```
int valAss(int x) {  
    if (x<0) return -x;  
    else return x;  
}
```

Quando valAss(z) viene chiamata, il valore attuale di z, valutato nell'environment corrente (-87), viene copiato e passato a valAss

- **Cliente**

```
main(){  
    int absz, z = -87;  
    absz = valAss(z);  
    printf("%d", z);  
}
```

ESEMPIO: VALORE ASSOLUTO

- **Servitore**

```
int valAss(int x) {  
    if (x<0) return -x;  
    else return x;  
}
```

valAss riceve quindi una copia del valore -87 e la lega al simbolo x. Poi si valuta l'istruzione condizionale, e si restituisce il valore 87

- **Cliente**

```
main(){  
    int absz, z = -87;  
    absz = valAss(z);  
    printf("%d", z);  
}
```

ESEMPIO: VALORE ASSOLUTO

- **Servitore: modifica**

```
int valAss(int x) {  
    if (x<0) x = -x;  
    return x;  
}
```

*Se x è negativo viene
MODIFICATO il suo valore
nella controparte positiva
Poi la funzione restituisce x*

- **Cliente**

```
main(){  
    int absz, z = -87;  
    absz = valAss(z);  
    printf("%d", z);  
}
```

ESEMPIO: VALORE ASSOLUTO

- **Servitore: modifica**

```
int valAss(int (x)) {  
    if (x<0) x = -x;  
    return x;  
}
```

x ~~-87~~ 87

*valAss restituisce il valore 87 che
viene assegnato a absz*

- **Cliente**

```
main(){  
    int absz, z = -87;  
    absz = valAss(z);  
    printf("%d", z);  
}
```

**NOTA: IL VALORE DI z NON
VIENE MODIFICATO**

PASSAGGIO DEI PARAMETRI IN C

Limiti:

- consente di restituire al cliente ***solo valori di tipo (relativamente) semplice***
- non consente di restituire ***collezioni di valori***
- **non consente di scrivere componenti software il cui scopo sia *diverso dal calcolo di una espressione***

PASSAGGIO DEI PARAMETRI

Molti linguaggi mettono a disposizione il passaggio per riferimento (*by reference*)

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

PASSAGGIO DEI PARAMETRI IN C

Il C *non* supporta *direttamente* il passaggio per riferimento

- è una grave mancanza
- il C lo fornisce indirettamente solo per alcuni tipi di dati
- quindi, occorre costruirselo quando serve.
(vedremo più avanti dei casi)

C++ e Java invece lo forniscono