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*
- *rendere tale espressione parametrica*

- Una *funzione* è un *componente software* che cattura l'idea matematica di funzione
  - molti possibili ingressi (parametri)
  - *una sola uscita* (il risultato)

Funzione come componente sw

- Una funzione
  - riceve dati di ingresso in corrispondenza dei parametri
  - ha come corpo una espressione, la cui valutazione fornisce un risultato
  - denota un valore tramite il suo nome

- **Esempio**

  se x vale 1

  e f è la funzione f : R –> R

  f = 3 * x2 + x - 3

  allora f(x) denota il valore 1

Invocazione di una funzione

- Una funzione è invocata attraverso il proprio *nome,* che racchiude le istruzioni che realizzano una certa *funzionalità*

- Il chiamante chiede alla funzione di svolgere il servizio fornito
  - invocando la funzione (per nome)
  - fornendole le necessarie informazioni

- La chiamata della funzione, avviente mediante *l'interfaccia* della funzione stessa

Interfaccia della 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***

La funzione comunica con il chiamante attraverso

- i *parametri* trasmessi dal chiamante all'atto
- della chiamata
- il *valore restituito* dalla funzione

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 di tipo intero **int**

Paramentri

Parametri *formali*:

- sono specificati nella ***dichiarazione*** della funzione
- indicano ***che cosa si aspetta la funzione*** *dal chiamante*

Parametri *attuali*:

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

Il chiamante passa informazioni al servitore mediante una serie di ***parametri attuali***

Esempio

Parametri FORMALI

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

Definizione della funzione

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

Chiamata della funzione

Parametri ATTUALI

Parametri formali ←→ attuali

Il legame tra parametri attuali e parametri formali si effettua *al momento della chiamata,* in modo dinamico

Tale legame:

– vale ***SOLO per l'invocazione corrente***

– vale ***SOLO per la durata della funzione***

Nell'esempio precedente:

All'atto della chiamata della funzione si effettua il legame tra

x e z

y e 4

Definizione di Funzione

<tipoValore> *<nome>*(<parametri-formali>)
{
  <corpo>
}

La forma base è:
return <espressione>

<parametri-formali>

o una **lista vuota**

o una **lista di variabili** (separate da virgole) *visibili solo entro il corpo della funzione (ambiente)*

<tipoValore>

deve coincidere con il tipo del valore restituito dalla funzione

Definizione di Funzione

- 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

"VITA" di una Funzione

- All'atto della chiamata, ***l'esecuzione del chiamante viene sospesa*** e ***il controllo passa alla funzione***
- La funzione "vive" solo per il tempo necessario a svolgere il servizio
- Al termine, la funzione "muore", e *l'esecuzione torna al chiamante*

Chiamata di funzione

La chiamata di funzione è un'espressione:

<nomefunzione> (<parametri-attuali>)

<parametri-attuali>

Elenco delle espressioni da passare alla funzione, separati da virgola

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

Risultato di una funzione

L'istruzione **return** provoca la *restituzione del controllo* al chiamante, 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 di esecuzione (1)

Funzione

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

Chiamante

```
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 ALLA FUNZIONE

Esempio di esecuzione (2)

Funzione

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

Chiamante

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

Viene valutata l'istruzione condizionale (16 > 13) che nell'ambiente corrente è vera. Pertanto si sceglie la strada return x

Esempio di esecuzione (3)

Funzione

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

Chiamante

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

Il valore 16 viene restituito al chiamante
IL SERVITORE TERMINA E IL CONTROLLO PASSA AL CHIAMANTE

NOTA: x e y vengono distrutti

Riassumendo …

All’atto dell’invocazione di una funzione:

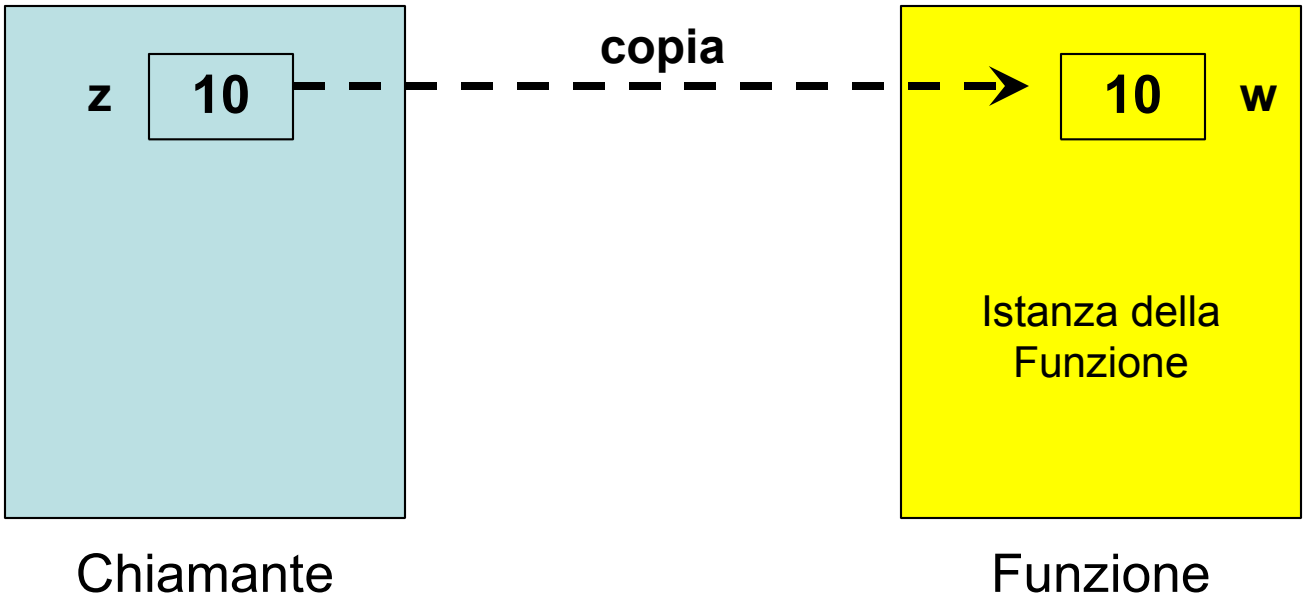
- si crea una ***nuova attivazione (istanza)*** *della funzione*
- si alloca la ***memoria per i parametri*** (e le eventuali variabili locali)
- si trasferiscono i parametri alla funzione
- si trasferisce il controllo alla funzione
- si esegue il codice della funzione

Passaggio dei parametri

In generale, un parametro è trasferito dal chiamante alla funzione

per valore o copia (*by value*)

cioè, si trasferisce *il valore* del parametro attuale



Ogni modifica effettuata su w è locale alla funzione (z non si modifica)

Passaggio per valore

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 quella istanza della funzione*
- la funzione potrebbe quindi alterare il valore ricevuto, *senza che ciò abbia alcun impatto sul chiamante*

Conseguenza:

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

Esempio: valore assoluto

Funzione

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

Chiamante

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

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

valAss restituisce il valore 87 che viene assegnato a absz

NOTA: IL VALORE DI z NON VIENE MODIFICATO

Passaggio per riferimento

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 alla funzione ***accesso diretto*** al parametro in possesso del chiamante
- la funzione ***accede e modifica*** *direttamente* il dato del chiamante

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

- il C lo fornisce indirettamente solo per alcuni tipi di dati, quindi, occorre costruirselo quando serve.

C++ e Java invece lo forniscono