Programmazione I

Il Linguaggio C

Le Funzioni

Daniel Riccio

Università di Napoli, Federico II

19 novembre 2021

Sommario

- Argomenti
  - Le funzioni
  - Chiamata di una funzione e ritorno da una funzione
  - Parametri formali e parametri attuali
  - Passaggio diparametri
  - Passaggio di vettori e matrici

Struttura modulare

Per semplificare la struttura di un programma complesso è possibile suddividerlo in **moduli**

Un **modulo** è un blocco di codice che assolve ad un compito preciso (ad es. calcola la radice quadrata) e a cui è stato dato un nome

Un programma consta di un **modulo principale** (il **main**) ed eventuali altri moduli di supporto

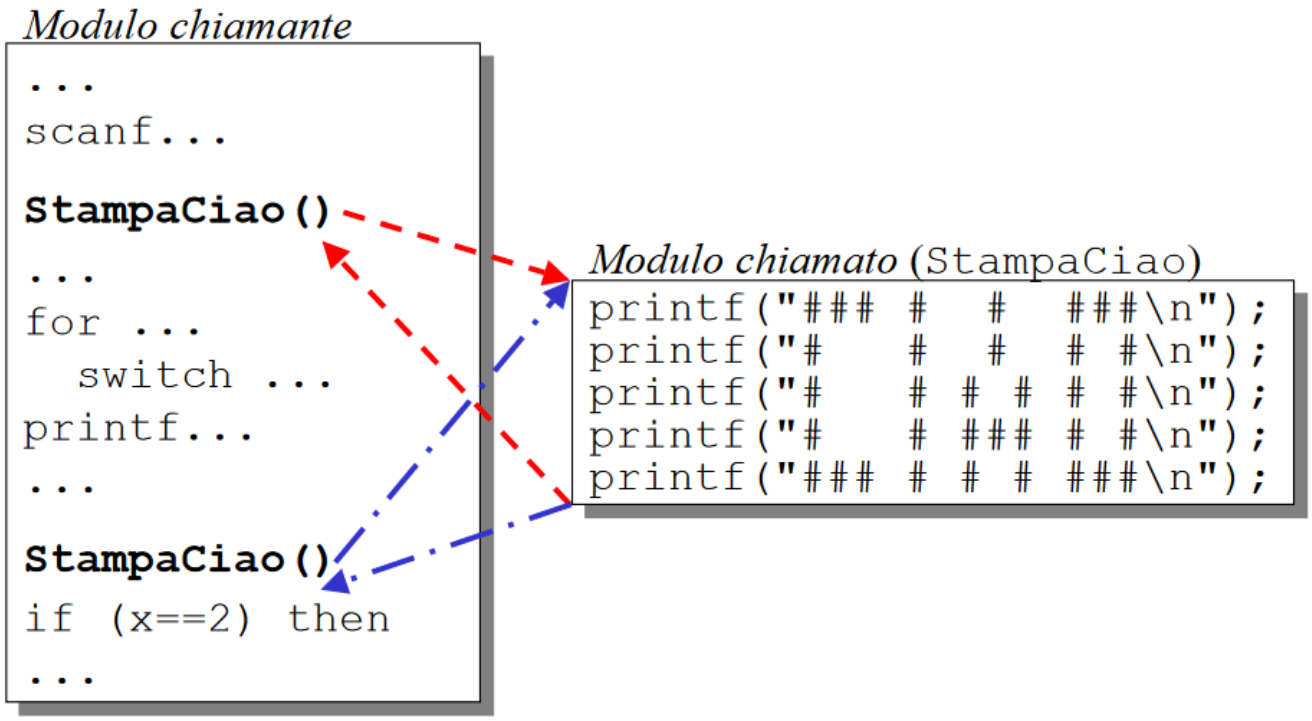
Quando un modulo richiama un altro modulo, il chiamante viene sospeso finché il chiamato non ha terminato la sua esecuzione

In **C** i moduli sono chiamati funzioni

Ogni funzione può richiamare (far eseguire) qualsiasi altra funzione (anche se stessa)

Struttura modulare

Le due chiamate del modulo StampaCiao fanno eseguire ogni volta le istruzioni che lo costituiscono (sospendendo il chiamante)



Struttura modulare

Vantaggi della programmazione modulare:

- Poiché i moduli “nascondono” al loro interno i dettagli di come una certa funzionalità venga realizzata, il programma complessivo ha un livello di astrazione maggiore: il modulo viene visto come un insieme di macro-istruzioni (“black-box”)
- Il codice per ottenere una certa funzionalità viene scritto una volta sola e viene richiamato ogni volta che è necessario (ma la chiamata richiede tempo)
- Il codice complessivo è più corto
- Essendo più piccoli, i moduli sono più semplici da realizzare e da verificare
- Il codice di un modulo correttamente funzionante può essere riutilizzato in altri programmi

Variabili locali

Ogni **funzione** è un piccolo programma a sé stante, isolato dalle altre funzioni

All'interno di una funzione possono essere definite delle **variabili locali** (cioè hanno scope locale): le altre funzioni non le "vedono"

Variabili con lo stesso nome in funzioni diverse sono quindi completamente scorrelate (possono dunque anche essere di tipo diverso)

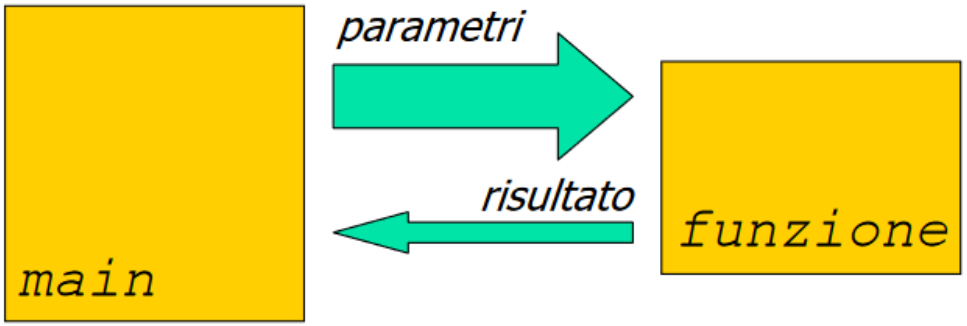
Vengono create ogni volta che si entra nella funzione e distrutte (perdendo il valore) quando si esce

Le inizializzazioni avvengono ad ogni chiamata, senza inizializzazione il contenuto è indefinito

Parametri e valore restituito

Essendo le variabili interne locali, private, per passare ad una funzione i dati da elaborare è necessario utilizzare variabili speciali dette **parametri**

La funzione comunica al modulo chiamante il risultato della sua elaborazione producendo un unico valore detto valore restituito o **valore di ritorno**



Definizione di funzione

```
tipo nomeFunzione (parametri)
{
  definizione_variabili_locali

  istruzioni

  eventuale return
}
```

Corpo della funzione

tipo indica il tipo del valore restituito (es. sqrt restituisce un double)

Se la funzione non restituisce valori (ad esempio StampaCiao visualizza soltanto) bisogna indicare il tipo **void**:

void StampaCiao(....)

Se non si mette nulla viene supposto **int**

Se la funzione non ha parametri (ad esempio StampaCiao), si indica void tra le parentesi:

void StampaCiao(void)

Chiamata di funzione

Si chiama una funzione indicandone il nome seguito da una coppia di parentesi contenenti i valori da elaborare (separati da virgole)

```
eleva(y,2);
```

Se la funzione non richiede parametri, le parentesi sono vuote, ma devono esserci:
StampaCiao();

Il valore restituito può essere assegnato ad una variabile o utilizzato in un'espressione, altrimenti viene semplicemente scartato:

```
x = eleva(y,2);
y = 3*eleva(2,k)-4*k;
eleva(3,5);
```

Ritorno da una funzione

La **funzione** termina (cioè l'esecuzione torna al modulo chiamante) quando viene eseguita l'istruzione **return risultato**;

Una funzione può avere più istruzioni **return**

Risultato è il valore restituito dalla funzione al chiamante, è un'espressione qualsiasi (es. **return** x*2;)

Se il tipo restituito dalla funzione è void, non si deve indicare risultato, inoltre la return che precede la graffa di chiusura (solo questa) può essere **omessa**

I valori delle variabili locali vengono persi

Ritorno da una funzione

Nel main la **return** termina il programma

Per terminare un programma dall'interno di una funzione (e passare lo status al Sistema Operativo) si utilizza la funzione

exit(**status**)

dichiarata in <**stdlib.h**>:

exit(**EXIT_SUCCESS**);

Esempio

```
int main(void)
{
  int x, y ;

/* leggi un numero
   tra 50 e 100 e
   memorizzalo
   in x */
/* leggi un numero
   tra 1 e 10 e
   memorizzalo
   in y */

  printf("%d %d\n",
    x, y ) ;
}
```

```
int main(void)
{
  int x, y ;

  x = leggi(50, 100) ;
  y = leggi(1, 10) ;

  printf("%d %d\n",
    x, y ) ;
}
```

```
int leggi(int min,
          int max)
{
  int v ;

  do {
    scanf("%d", &v) ;
  } while( v<min ||
           v>max) ;

  return v ;
}
```

Tipo di una funzione

Il tipo di una funzione è determinato dal tipo del valore restituito e dal tipo, numero e ordine dei suoi parametri

```
int eleva(int b, int e)
```

eleva è una funzione che ha un primo paramero **int**, un secondo parametro **int** e restituisce un **int**

Scope di una funzione

Lo scope di una funzione (nome e tipo) si estende dal punto in cui viene definita fino alla fine del file.

La funzione può essere utilizzata solo dalle funzioni che nello stesso file seguono la sua definizione (vale anche per il main)

```
f1()
{...}
f2()
{...}
main()
{...}
```

f1 non “vede” e non può quindi usare **f2**, **f2** vede **f1**, il **main** vede **f1** e **f2**

Scope di una funzione

Il compilatore verifica che le chiamate a funzione siano coerenti con le corrispondenti definizioni (cioè abbiano lo stesso tipo)

E' necessario che la chiamata a funzione sia nello scope della funzione stessa

Se il compilatore trova una funzione di cui non conosce il tipo (non è in scope, ad esempio **f1** che chiama **f2**), allora presuppone che essa sia definita altrove e quindi:

- non fa controlli sugli argomenti
- presuppone che restituisca un int
- segnala il possibile problema con un Warning

Prototipo di una funzione

Il **prototipo** di una funzione è una dichiarazione che estende lo scope della funzione (nome e tipo)

Il **corpo della funzione** (la sua definizione) può quindi essere collocato:

- in un punto successivo a dove viene chiamata (nell'esempio seguente, eleva è definita dopo il main dove viene utilizzata)
- in un altro file di codice sorgente C
- in una libreria (compilata)
- Lo scopo primario degli header file è quello di fornire al compilatore i prototipi delle funzioni delle librerie del C (ad es. stdio.h contiene i prototipi di scanf, printf, getchar, etc.)

Esempio di una funzione

```c
#include <stdio.h>

//prototipo
int eleva(int b, int e);

int main()
{
  int x, y;

  printf("Introduci numero: ");
  scanf("%d", &x);

  y = eleva(x, 2);
  printf("%d^%d = %d\n", x,2,y);

  return 0;
}
```

```c
int eleva(int b, int e)
{
  int k=1;

  while (e-- > 0)
    k *= b;

  return k;
}
```

Prototipo di una funzione

I **prototipi** possono essere collocati:

- prima del main (come nell'esempio eleva)
- tra una funzione e l'altra
- insieme alle definizioni delle variabili locali di una funzione

Il **prototipo** estende lo scope della funzione (nome e tipo) dal punto dove è indicato:

- fino alla fine del file se esso è collocato esternamente alle funzioni (prima del main o tra due funzioni)
- fino alla fine della funzione se è interno ad una funzione (collocato con le variabili locali di una funzione)

Prototipo di una funzione

Il **prototipo** di una funzione è simile alla definizione della funzione, salvo che:

- manca il corpo
- i nomi dei parametri possono essere omessi (ma i tipi devono essere presenti!)
- ha un punto e virgola alla fine **int eleva**(**int**, **int**);

I nomi dei **parametri** dei prototipi:

- se non sono omessi, possono essere diversi da quelli usati nella definizione della funzione
- sono scorrelati dagli altri identificatori (nomi uguali si riferiscono comunque a identificatori diversi)
- sono utili per descrivere il significato dei parametri:
  int eleva(**int** base, **int** esponente);

Parametri formali ed attuali

Parametri formali:
sono le variabili indicate tra le parentesi nella definizione della funzione
int eleva(**int** b, **int** e)

Parametri attuali (o **argomenti**):
sono i valori (variabili, costanti o espressioni) indicati tra le parentesi
alla chiamata di una funzione **eleva**(x,2)

Nella chiamata ad una funzione bisogna indicare un argomento per ciascuno dei parametri formali

I parametri attuali e quelli formali corrispondono in base alla posizione (il primo attuale al primo formale, etc.)

I nomi dei parametri formali sono scorrelati (e quindi tipicamente diversi) dai nomi di eventuali variabili usate come argomenti (inoltre gli argomenti possono essere valori costanti o il risultato di espressioni)

I parametri formali hanno lo stesso scope delle variabili locali della funzione

Parametri formali ed attuali

Le funzioni possono ricevere dei parametri dal proprio chiamante

Nella funzione:

- Parametri **formali**
- Nomi "interni" dei parametri

```
int leggi(int min,
          int max)
{
    ...
}
```

Nel chiamante:

- Parametri **attuali**
- Valori effettivi (costanti, variabili, espressioni)

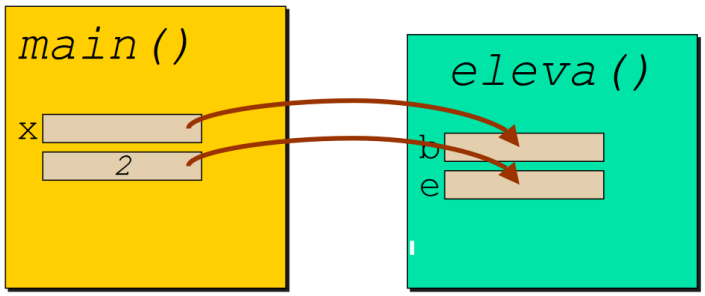
```
int main(void)
{
    ...
    y = leggi(1, 10) ;
    ...
}
```

```
int main(void)
{
  int x, y ;

  x = leggi(50, 100) ;
  y = leggi(1, 10) ;

  printf("%d %d\n",
    x, y ) ;
}
```

Passaggio dei parametri

I dati possono essere passati ad una funzione esclusivamente per valore (by value):

alla chiamata della funzione vengono create nuove variabili con i nomi di ciascuno dei parametri formali e in esse viene copiato il valore del corrispondente parametro attuale



Come per le assegnazioni, se il **parametro attuale** e il corrispondente **formale** sono di tipo diverso c'è una conversione automatica al tipo del **parametro formale** (se è di tipo meno capiente può essere generato un Warning)

Poiché in memoria i parametri formali e quelli attuali sono completamente **distinti** e **indipendenti**, cambiare il valore di un parametro formale non modifica il parametro attuale corrispondente, neppure se questo è una semplice variabile (è ovviamente impossibile modificare una costante o il risultato di un'espressione): nell'esempio visto **la modifica di b non si ripercuote su x**