Introduzione al linguaggio C
Funzioni

Violetta Lonati

Università degli studi di Milano
Dipartimento di Informatica

Laboratorio di algoritmi e strutture dati
Corso di laurea in Informatica

Argomenti

Funzioni

- Definizione di funzioni
- Chiamata di funzioni
- Dichiarazione di funzioni
- Parametri, argomenti e variabili locali
- L'istruzione return
- Organizzazione dei programmi
- Alcune osservazioni sulle funzioni
- Funzioni ricorsive

Funzioni

- Le funzioni sono costituite da una sequenze di istruzioni raccolte sotto un solo nome.
- Le funzioni possono avere argomenti e possono calcolare e restituire dei valori, ma anche no! (a differenza delle funzioni matematiche)
- I programmi visti sin qui erano costituiti da una sola funzione, il main.
- A cosa servono le funzioni?
  - permettono di evitare la duplicazioni di codice;
  - sono riutilizzabili.

Esempio: calcolare la media

```
#include <stdio.h>

float media (float x, float y ) {
  return (x + y) / 2;
}

int main( void ) {
  float a, b, c, m;
  scanf( "%f%f%f", &a, &b, &c );

  m = media(a, b);
  printf( "La media tra %f e %f e' %f\n", a, b, m );

  printf( "La media tra %f e %f e' %f\n",
                        a, c, media(a, c) );
  return 0;
}
```

Esempio: conto alla rovescia

```
#include <stdio.h>

void stampa_conteggio ( int n ) {
  printf( "meno␣%d...\n", n );
}

int main( void ) {
  int i;

  for ( i = 10; i > 0; --i )
    stampa_conteggio( i );
  return 0;
}
```

Definizione di funzioni

La forma generale di una funzione è la seguente:

```
TIPO_RESTITUITO NOME_FUNZIONE ( PARAMETRI ) {
  DICHIARAZIONI
  ISTRUZIONI
}
```

TIPO_RESTITUITO

E' il tipo del valore che la funzione restituisce attraverso l'uso di return:

- le funzioni non possono restituire array, ma non ci sono restrizioni sugli altri tipi;
- se `TIPO_RESTITUITO` è `void`, la funzione non restituisce alcun valore;
- se il `TIPO_RESTITUITO` è omesso, per default è `int`; è comunque meglio indicarlo sempre!

NOME_FUNZIONE

E' il nome della funzione, che si usa per chiamare la funzione stessa.

Definizione di funzioni - continua

PARAMETRI

E' una lista di parametri separati da virgole.

- ciascun parametro deve essere preceduto dal suo tipo (il tipo va ripetuto anche se ci sono più parametri dello stesso tipo)

```
float media( float x, y ) /* SBAGLIATO! */
```

Corpo della funzione

Il corpo della funzione può contenere dichiarazioni e istruzioni; le variabili dichiarate nel corpo di una funzione si chiamano locali.

```
float media (float x, float y ) {
  float sum;          /* dichiarazione */

  sum = x + y;        /* istruzione */
  return sum / 2;     /* istruzione */
}
```

Chiamata di funzioni

- Una chiamata di funzione consiste nel nome della funzione seguito da una lista di argomenti tra parentesi.

```
media( 3, 6 )
media( a, b * b )
stampa_conteggio( 9 )
getchar( )
```

- La chiamata di una funzione coinvolge sempre due funzioni (che possono anche coincidere, nella ricorsione):
  - una funzione chiamante
  - e una funzione chiamata.

  Nei primi esempi, la funzione chiamante è il main.
- Quando una funzione viene chiamata, il controllo passa dalla funzione chiamante alla funzione chiamata. Quando questa termina (alla fine delle istruzioni oppure in presenza dell'istruzione return), restituisce il controllo alla funzione chiamante.

Chiamata di funzioni - continua

- Se mancano le parentesi, la funzione non verrà chiamata. Se una funzione è di tipo void, bisognerà chiamarla così:

```
getchar( );
```

- Una chiamata di funzione di tipo void è una istruzione, quindi deve concludersi con punto-e-virgola;

```
stampa_conteggio( 9 );
```

- Una chiamata di funzione di tipo non void è un'espressione e produce un valore che può essere assegnato ad una variabile, testato, stampato, ecc

```
m = media( 3, 6 );
if ( media( a, b * b ) > c ) printf( "..." );
```

- Il valore restituito da una variabile può essere scartato:

```
printf( "Ciao␣mondo!\n" );
chars = printf( "Ciao␣mondo!\n" );
```

Dichiarazione di funzioni

- La definizione di una funzione non deve necessariamente precedere il punto in cui viene chiamata. Ad esempio, le definizioni di funzioni possono seguire il main.
- Se il compilatore trova una chiamata prima della definizione, non sa quanti e di che tipo siano i parametri, quindi fa delle assunzioni e effettua dei casting. Non è detto che le queste assunzioni siano corrette!
- Per evitare questo problema, è possibile dichiarare le funzioni prima che vengano chiamate.

```
TIPO_RESTITUITO NOME_FUNZIONE ( PARAMETRI );
```

  Una dichiarazione di questo tipo si chiama prototipo o segnatura.
- Attenzione: ci deve essere coerenza tra dichiarazione e definizione!!

Esempio: media rivisitata

```c
#include <stdio.h>

/* prototipo della funzione media */
float media (float x, float y );

/* main */
int main( void ) {
  float a, b, c, m;
  scanf( "%f%f%f", &a, &b, &c );

  printf( "La media tra %f e %f e' %f\n",
                              a, b, media( a, b ) );
  return 0;
}

/* definizione della funzione media */
float media (float x, float y ) {
  return (x + y) / 2;
}
```

Parametri e argomenti

- I parametri di una funzione compaiono nella sua definizione: sono nomi che rappresentano i valori da fornire alla funzione al momento della chiamata.
- Gli argomenti sono invece le espressioni che compaiono nella chiamata di funzione tra parentesi.
- A volte i parametri sono chiamati parametri formali mentre gli argomenti sono chiamati parametri attuali.
- Se il tipo di un argomento non corrisponde al tipo del parametro, viene eseguita una conversione coerente con il prototipo della funzione oppure una conversione di default.
- Gli argomenti sono passati per valore: al momento della chiamata, ogni argomento è valutato e il valore è assegnato al corrispondente parametro; eventuali cambiamenti di valore dei parametri durante l'esecuzione della funzione non influenzano il valore dell'argomento!

Parametri e variabili locali

Le variabili locali hanno le seguenti proprietà:

- sono utilizzabili solo dal punto in cui sono dichiarate fino alla fine della stessa funzione (block scope);
- non possono essere usate o modificate da altre funzioni;
- sono automaticaticamente allocate al momento della chiamata della funzione e deallocate al return (automatic storage duration);
- alla fine dell'esecuzione della funzione non conservano il loro valore.

I parametri hanno le stesse proprietà (block scope e automatico storage duration) delle variabili locali. Di fatto, la sola differenza tra variabili locali e parametri è che i parametri vengono inizializzati al momento della chiamata (con il valore degli argomenti della chiamata).

Conversione di tipi

- In C sono consentiti assegnamenti ed espressioni che mescolano i tipi, ma alcuni operandi verranno convertiti automaticamente (implicitamente) in modo da rendere possibile la valutazione dell'espressione.
- Nelle chiamate di funzione è consentito passare argomenti di tipo diverso da quelli dei parametri; anche in questo caso si avranno delle conversioni implicite.
- Le regole di conversione implicita sono complicate! In sintesi:
  - nelle espressioni aritmetiche gli operandi vengono promossi;
  - negli assegnamenti il lato destro viene convertito al tipo del lato sinistro;
  - se la funzione è dichiarata/definita prima della chiamata, ogni argomento è convertito implicitamente al tipo del parametro corrispondente;
  - se la funzione non è dichiarata/definita prima della chiamata, il compliatore esegue delle promozioni di default
- E' possibile anche effettuare conversioni esplicite con il cast.

Array come argomenti

- Se il vettore è unidimensionale, la dimensione può essere omessa.

```
int f( int a[] ) { ... }
```

- Se il vettore è multidimensionale, solo la prima dimensione può essere omessa.

```
int f( int a[][LUN] ) { ... }
```

- la funzione non ha modo di sapere quanto è lungo il vettore (l'operatore `sizeof` non può essere usato...) quindi può essere utile passare la lunghezza come parametro aggiuntivo:

```
int somma_array( int a[], int n ) {
  int i, sum = 0;
  for ( i = 0; i < n; i++ )
    sum += a[i];
  return sum;
}
```

L'istruzione return

- Ogni funzione non void contiene l'istruzione return per specificare quale valore deve essere restituito alla funzione chiamante.

```
return ESPRESSIONE;
```

- L'espressione può essere una semplice costante o variabile, oppure un'espressione più complicata (del tipo giusto!)

```
return i > j ? i : j;
```

- L'esecuzione dell'istruzione return ha per effetto la restituzione del controllo alla funzione chiamante. La restituzione del controllo alla funzione chiamante avviene comunque al termine dell'esecuzione della funzione.
- Il valore restituito dal main è un codice di stato: il codice 0 indica la terminazione senza errori.

Organizzazione dei programmi

Quando un programma è costituito da più di una funzione, è importante organizzarlo bene!

- Ricordare che variabili locali e parametri hanno block scope e automatic storage duration;
- Le funzioni comunicano fra loro attraverso i parametri, oppure attraverso variabili esterne o globali, che hanno queste proprietà:
  - sono definite fuori dal corpo di ogni funzione;
  - hanno static storage duration, cioè permangono e mantengono il loro valore indefinitamente;
  - sono visibili dal punto in cui sono definite per tutto il file.

Organizzazione dei programmi - continua

In generale, un programma può essere organizzato come segue:

- direttive `#include`
- direttive `#define`
- definizione di tipi con `typedef`
- dichiarazione di variabili esterne
- prototipi delle funzioni diverse dal `main`
- definizione del `main`
- definizione delle altre funzioni

Esercizi

- Scrivete una funzione avente due parametri interi b ed e che calcoli la potenza b^e.
- Scrivete una funzione con parametro un intero n che stabilisca se n è un numero primo.
- Scrivete un programma che generi a caso un array di interi e calcoli la somma dei suoi elementi: strutturate il programma usando una funzione per generare l'array e una per sommare i suoi elementi.
- Scrivete una variante dell'esercizio precedente per un array bidimensionale.

Esercizi - continua

- Scrivete un programma che legga un carattere, uno spazio e quindi una sequenza di caratteri minuscoli terminati da `'.'` e che stampi quanto ha letto dopo il primo spazio, ma sostituendo tutte le vocali con il primo carattere letto. Per farlo, usate una funzione che, dati due caratteri, restituisca il primo carattere se il secondo è una vocale minuscola, altrimenti restituisca il secondo carattere.
- Rivedete gli esercizi svolti nelle esercitazioni precedenti e provate a strutturarli usando delle funzioni. Ad esempio, modificate l'esercizio sulla rubrica usando una funzione per stampare la rubrica, una per leggere una nuova voce, una per inserire la nuova voce nell'ordine corretto.

Overloading

A differenza di Java, in C non è possibile fare overloading, ovvero, non si possono definire due funzioni con lo stesso nome e propotipo diverso!

```
int media_int( int x, int y ) {
  return ( x + y ) / 2
}
```

```
float media_float( float x, float y ) {
  return ( x + y ) / 2.0
}
```

Lista di parametri a lunghezza variabile

- Esistono funzioni con numero di parametri variabile (es: `printf`);
- Il file di intestazione stdarg.h fornisce gli strumenti per poterle gestire (qui non le vediamo, si veda il capitolo 26 del libro).
- Attenzione: chiamare una funzione di questo tipo è un'operazione sempre rischiosa!

Funzioni ricorsive

Una funzione si dice ricorsiva quando chiama se stessa.

```
int fact( int n ) {
  if ( n <= 1 )
    return 1;
  else
    return n * fact( n - 1 );
}
```

Le chiamate ricorsive si impilano.

Ad esempio, tracciamo l'esecuzione della chiamata `fact(3)`:

`fact(3)` chiama
  `fact(2)` che chiama
    `fact(1)` che restituisce 1, quindi
  `fact(2)` restituisce $2 \times 1 = 2$, quindi
`fact(3)` restituisce $3 \times 2 = 6$.

Esempio

La potenza può essere definita ricorsivamente, osservando che $b^0 = 1$ e $b^e = b * b^{e-1}$ per ogni $e > 0$.

```
int power( int b, int e ) {
  if ( e = = 0 )
    return 1;
  else
    return b * power( b, e - 1 );
}
```

La chiamata `power(5, 3)` sarà eseguita come segue:

`power(5, 3)` chiama
  `power(5, 2)` che chiama
    `power(5, 1)` che chiama
      `power(5, 0)` che restituisce 1, quindi
    `power(5, 1)` restituisce $5 \times 1 = 5$, quindi
  `power(5, 2)` restituisce $5 \times 5 = 25$, quindi
`power(5, 3)` restituisce $5 \times 25 = 125$.