Funzioni in C

Emilio Di Giacomo

Modularizzazione di programmi

- Quando si devono scrivere programmi complessi può essere utile suddividere il programma in più moduli
- Modularizzare un programma semplifica lo sviluppo e la manutenzione:
  - ogni modulo si occupa di uno specifico aspetto del problema
  - ogni modulo risulta più semplice dell'intero programma

Moduli in C

- Lo strumento che abbiamo a disposizione in C per modularizzare il programma sono le funzioni
- Un programma C risulta quindi una combinazione di funzioni:
  - alcune scritte da noi
  - alcune disponibili in librerie scritte da altri
- Esiste una ricca raccolta di funzioni che fa parte del linguaggio C
  - queste funzioni sono chiamate *libreria standard* del C
- Abbiamo già visto alcune funzione della libreria standard del C:
  - *printf*, *scanf*,...

Funzioni

- Vedremo due aspetti diversi delle funzioni
- La definizione della funzione
  - scrittura del codice che realizza la funzione desiderata
- L'invocazione della funzione
  - utilizzo della funzione

Definizione delle funzioni

- Una funzione è un blocco di istruzioni che ha parametri in ingresso (parametri formali) e restituisce un risultato

- Sintassi della definizione di funzione

<tipo di ritorno> <nome funzione> **(***<parametri formali>***){**

<corpo della funzione>

}

Definizione delle funzioni

- *<tipo di ritorno>*
  - specifica il tipo del risultato calcolato dalla funzione
- *<nome funzione>*
  - nome della funzione: è un qualsiasi identificatore C valido
- *<parametri formali>*
  - è una sequenza di dichiarazioni di parametro separate da virgola
  - ogni parametro è dichiarato con tipo e nome
  - se la funzione non ha bisogno di parametri, la lista è vuota (o può essere *void*)

Definizione delle funzioni

- *<corpo della funzione>*
  - insieme delle istruzioni che realizzano il comportamento della funzione
- Nel corpo possono essere presenti dichiarazioni di variabili
- Le variabili definite all'interno del corpo sono variabili locali
  - sono visibili soltanto all'interno della funzione stessa
- Anche i parametri formali sono variabili locali

Esempio: la funzione max

- Supponiamo di voler scrivere una funzione che dati due numeri interi ci dice qual è il maggiore dei due
- L'intestazione della funzione potrebbe essere

```
int max(int a, int b)
```

- Il nome della funzione è *max*
- La funzione riceve in ingresso i due numeri interi dei quali si vuole calcolare il massimo
  - ha quindi due parametri formali di tipo *int*
- La funzione dovrà restituire il massimo tra *a* e *b*
  - il tipo di ritorno è dunque un *int*

Parametri formali

- I parametri formali di una funzione sono variabili usate per contenere i valori che si passeranno alla funzione all'atto della sua invocazione
  - Vengono allocate in memoria all'atto dell'invocazione della funzione
  - Vengono rimosse quando l'esecuzione della funzione termina
  - Sono visibili soltanto all'interno della funzione
  - Variabili di questo tipo sono dette variabili locali

Parametri attuali

- Quando si invocherà la funzione, per esempio scrivendo

  int m=max(10, 20)

- i valori*10* e *20* verranno copiati rispettivamente nelle variabili *a* e *b* della funzione *max*, e poi verranno eseguite le sue istruzioni

- I valori copiati nei parametri formali all'atto dell'invocazione sono detti parametri attuali

Passaggio di parametri

- Ad ogni invocazione di una funzione i parametri attuali vengono copiati nei parametri formali della funzione
  - questo meccanismo si chiama passaggio di parametri per valore

Passaggio di parametri

- Nei vari linguaggi di programmazione esistono due modalità di passaggio dei parametri
  - passaggio per valore
  - passaggio per riferimento
- Immaginiamo che una variabile *a* venga passata come parametro attuale ad una funzione

```
int a=5
int b=funzione(a);
```

Passaggio di parametri

- Con il passaggio per valore, il valore di *a* viene copiato nel parametro formale della funzione
  - la variabile *a* non verrà modificata dalla funzione
    - eventuali modifiche opereranno sulla copia
- Con il passaggio per riferimento, un riferimento alla variabile *a* (cioè il suo indirizzo di memoria) viene passato alla funzione
  - in questo caso la variabile *a* potrebbe essere modificata dalla funzione cui è stata passata
    - ciò potrebbe causare errori

Passaggio di parametri

- In C il passaggio dei parametri avviene sempre per valore
  - con l'eccezione degli array (vedremo più avanti)
- Il passaggio per riferimento può essere realizzato utilizzando gli operatori di indirizzo e di indirezione (vedremo più avanti)
  - in realtà abbiamo utilizzato già l'operatore di indirizzo per realizzare un passaggio per riferimento quando abbiamo utilizzato la *scanf*
  - in quel caso infatti volevamo che la funzione modificasse il contenuto della variabile passata come parametro

La funzione max: corpo

- Il corpo va scritto assumendo che nei parametri formali siano memorizzati i valori passati alla funzione dall'esterno (cioè da chi lo invocherà)

```
int max (int a, int b){
      int m;
      if (a>b)
            m = a;
      else
            m = b;
      return m;
}
```

La funzione max: commenti

- Viene definita una variabile *m* per memorizzare il risultato
  - *m* è una variabile locale alla funzione *max*
- I due valori forniti in ingresso (*a* e *b*) vengono confrontati e il più grande dei due viene memorizzato in *m*
- L'istruzione finale *return m* ha un duplice effetto:
  - restituisce all'esterno il valore di *m*
  - causa la terminazione della funzione

La funzione max: commenti

- Le variabili definite nel corpo di una funzione sono variabili locali
  - *m* è una variabile locale
- Le variabili locali
  - vengono allocate in memoria all'atto dell'invocazione della funzione
  - vengono rimosse quando l'esecuzione della funzione termina
  - sono visibili soltanto all'interno della funzione

L’istruzione return

- L’istruzione *return* deve essere usata in ogni funzione che dichiara di restituire un valore
- La sintassi è la seguente:

  return <espressione>

- L'espressione viene valutata e il suo risultato viene restituito
- Il tipo di tale espressione deve essere coerente con il tipo di dato di ritorno
- l’istruzione *return* causa la terminazione immediata della funzione

max: versione alternativa

```
int max (int a, int b){
      if (a>b)
            return a;
      else
            return b;
}
```

- In questo codice non appena si stabilisce il valore da restituire esso viene restituito senza memorizzarlo in una variabile locale

Ancora sull'istruzione return

- Sebbene sia possibile avere più istruzioni *return* in una funzione (come nell'esempio precedente), è preferibile avere un'unica istruzione *return* come ultima istruzione della funzione
  - avere più punti di uscita da una funzione può essere fonte di errori

Ancora sull'istruzione return

- Se una funzione non deve restituire nessun valore (cioè se il suo tipo di ritorno è *void*) l'istruzione *return* non è necessaria
  - si può comunque utilizzarla (senza valore) per interrompere l'esecuzione della funzione

```
void stampaEta(int nascita,int anno){
     if (nascita>anno)
          return;
     printf("Età: %d", anno-nascita);
}
```

Invocazione di funzione: sintassi

- L'invocazione (o chiamata) di funzione ha la seguente sintassi

<nome funzione> (<parametri attuali>)

- *<nome funzione>* è il nome della funzione
- *<parametri attuali>* è una lista di espressioni separate da virgola
- i parametri attuali devono corrispondere in numero e tipo ai parametri formali

Invocazione di funzione

- Un esempio di invocazione della funzione *max* che abbiamo scritto in precedenza potrebbe essere il seguente

```
int main(){
     int a,b;
     printf("Inserisci due numeri\n");
     scanf("%d%d",&a,&b);
     int c=max(a,b);
     printf("Il massimo è: %d",c);
}
```

Invocazione di funz.: semantica

- Una chiamata di funzione è un'espressione
  - pertanto possiamo invocare una funzione in ogni punto in cui ci si aspetti un'espressione
- Se una funzione B viene chiamata all'interno di una funzione A:
  - l'esecuzione di A viene sospesa e il controllo passa a B dopo aver copiato i parametri attuali nei parametri formali
  - vengono eseguite le istruzioni del corpo di B
  - l'esecuzione di B termina con l'esecuzione dell'istruzione *return*
  - il valore restituito dalla *return* è il valore dell'espressione corrispondente alla chiamata
  - il controllo ritorna ad A, che prosegue l'esecuzione a partire dal punto in cui B era stata attivata

Regole di visibilità

- Dove devono essere scritte le funzioni?
- Le funzioni possono essere scritte nello stesso file che contiene la funzione *main*
  - è possibile scrivere le funzioni in file diversi (vedremo dopo)
- la dichiarazione di una funzione deve precedere il suo uso
  - altrimenti si ha errore di compilazione

Esempio

```
#include <stdio.h>

int max (int a, int b){
	int m;
	if (a>b)
		m = a;
	else
		m = b;
	return m;
}

int main(){
	int a,b;
	printf("Inserisci due numeri\n");
	scanf("%d%d",&a,&b);
	int c=max(a,b);
	printf("Il massimo è: %d",c);
}
```

OK!

Esempio

```
#include <stdio.h>

int main(){
	int a,b;
	printf("Inserisci due numeri\n");
	scanf("%d%d",&a,&b);
	int c=max(a,b);
	printf("Il massimo è: %d",c);
}

int max (int a, int b){
	int m;
	if (a>b)
		m = a;
	else
		m = b;
	return m;
}
```

NO!

Errore in fase di compilazione

Regole di visibilità

- In realtà il C distingue tra la dichiarazione di una funzione e la sua definizione
- Nella dichiarazione si specifica la struttura della funzione: tipo di ritorno, nome, parametri formali
- La dichiarazione di una funzione è detta in C prototipo o header
- Nella definizione si specifica completamente la funzione: tipo di ritorno, nome, parametri formali, corpo
- Per poter usare una funzione è sufficiente che la sua dichiarazione preceda l'uso

Esempio

```
#include <stdio.h>

int max(int a, int b);
```

<---------- Dichiarazione della funzione *max*

```
int main(){
        int a,b;
        printf("Inserisci due numeri\n");
        scanf("%d%d",&a,&b);
        int c=max(a,b);
        printf("Il massimo è: %d",c);
}

int max (int a, int b){
        int m;
        if (a>b)
                m = a;
        else
                m = b;
        return m;
}
```

OK!

File header

- Quando si sviluppano progetti complessi, può essere scomodo avere tutte le funzioni in un unico file
- Le funzioni possono essere distribuite in più file
- Vengono poi definiti uno o più file header con estensione *.h* che contiene solo la dichiarazione delle funzioni

File header

- Per utilizzare poi le funzioni definite nei diversi file è sufficiente includere i file header
  - è ciò che facciamo quando scriviamo

    #include <stdio.h>
- Non vedremo come definire file header e strutturare un programma in più file

Libreria standard del C

- Il linguaggio C offre una libreria di funzioni che sono considerate parte dello standard del linguaggio
- A tale libreria si dà il nome di libreria standard del C
- Le funzioni della libreria standard possono essere utilizzate includendo opportuni file header nel proprio codice:
  - abbiamo già incontrato *stdio.h*, *limits.h*, *float.h*,...

Libreria standard del C

- Uno studio esaustivo della libreria standard del C esula dagli scopi di questo corso
- Vedremo però alcune funzioni che ci potranno essere utili

La libreria math.h

- Un esempio di funzioni che fanno parte della libreria standard del C sono le funzioni della libreria *math.h*
- Ogni funzione di tale libreria realizza una specifica funzione matematica

Alcune funzioni di math.h

Funzione	Descrizione	Esempio
double sqrt(double x)	Calcola la radice quadrata di x	sqrt(25.0) è uguale a 5.0
double cbrt(double x)	Calcola la radice cubica di x	cbrt(27.0) è uguale a 3.0
double exp(double x)	Calcola la funzione esponenziale e^x	exp(1.0) è uguale a 2.718282
double log(double x)	Calcola il logaritmo naturale di x	log(2.718282) è uguale a 1.0
double log10(double x)	Calcola il logaritmo in base 10 di x	log10(100.0) è uguale a 2.0
double fabs(double x)	Calcola il valore assoluto di x	fabs(-12.0) è uguale a 12.0
double ceil(double x)	Arrotonda al più piccolo intero non minore di x	ceil(11.3) è uguale a 12.0

Alcune funzioni di math.h

Funzione	Descrizione	Esempio
double floor(double x)	Arrotonda al più grande intero non maggiore di x	floor(11.6) è uguale a 11.0
double pow(double x, double y)	Calcola x^y	pow(2.0, 3.0) è uguale a 8.0
double fmod(double x, double y)	Calcola il resto di x/y	fmod(9.0,5.0) è uguale a 4.0
double sin(double x)	Calcola il seno di x (in radianti)	sin(0.0) è uguale a 0.0
double cos(double x)	Calcola il coseno di x (in radianti)	cos(0.0) è uguale a 1.0
double tan(double x)	Calcola la tangente di x (in radianti)	tan(0.0) è uguale a 0.0

Esempio

- Scriviamo un programma C che chiede all'utente le coordinate cartesiane di due punti nel piano e calcola la loro distanza euclidea.

Esempio

```
#include <stdio.h>
#include <math.h>
int main(){
        double x1, x2, y1, y2;
        printf("Inserisci le coordinate di un punto\n");
        scanf("%lf%lf",&x1,&y1);
        printf("Inserisci le coordinate di un secondo punto\n");
        scanf("%lf%lf",&x2,&y2);
        double dx=x1-x2;
        double dy=y1-y2;
        double dx2=pow(dx,2);
        double dy2=pow(dy,2);
        double distanza=sqrt(dx2+dy2);
        printf("la distanza fra i due punti è %g\n", distanza);
}
```

Esempio: versione più compatta

```
#include <stdio.h>
#include <math.h>
int main(){
        double x1, x2, y1, y2;
        printf("Inserisci le coordinate di un punto\n");
        scanf("%lf%lf",&x1,&y1);
        printf("Inserisci le coordinate di un secondo punto\n");
        scanf("%lf%lf",&x2,&y2);
        double distanza=sqrt(pow(x1-x2,2)+pow(y1-y2,2));
        printf("la distanza fra i due punti è %g\n", distanza);
}
```

La libreria stdlib.h

- Un'altra libreria che useremo è la libreria *stdlib.h*
- Questa libreria contiene varie funzioni di uso generale
  - funzioni per la conversione da stringhe a tipi numerici
  - funzioni per l'allocazione dinamica della memoria
  - generazione di numeri casuali
  - ...
- Per adesso limitiamoci a vedere la funzione *rand()* per la generazione di numeri casuali

La funzione rand()

- In molti casi si ha la necessità di generare numeri casuali per introdurre un elemento di casualità nell'esecuzione di un programma
  - ciò accade soprattutto nei giochi
- La funzione *rand()* della libreria *stdlib.h* genera un numero casuale compreso tra *0* e *RAND_MAX* (valore che dipende dal compilatore)

Esempio: lancio di un dado

- Scriviamo un programma che simula il lancio di un dado (a 6 facce) per 20 volte
- Per simulare il lancio del dado usiamo la funzione *rand()*
  - estraiamo un numero *n* con *rand()*
    - *n* sarà compreso tra *0* e *MAX_RAND*
  - calcoliamo il resto della divisione di *n* per *6*
    - il risultato è compreso tra *0* e *5*
  - sommiamo *1*
    - il risultato è compreso tra *1* e *6*
- Più in generale, per generare un numero compreso tra *a* e *b*, usiamo *(rand()%(b-a+1))+a*

Esempio: lancio di un dado

```
#include<stdio.h>
#include<stdlib.h>
int main(){
     for(int i=1;i<=20;i++){
          int a=(rand()%6)+1;
          printf("%d\t", a);
          if(i%5==0)
               printf("\n");
     }
}
```

Esempio: lancio di un dado

- Se eseguiamo più volte il codice precedente la sequenza di numeri estratti è sempre la stessa. Perché?
- La sequenza generata non è veramente casuale
- Essa in realtà è una sequenza pseudo-casuale generata con un algoritmo deterministico
  - tale sequenza viene generata a partire da un numero iniziale detto seme
  - il seme può essere cambiato usando la funzione *srand()*

Esempio: lancio di un dado

```
#include<stdio.h>
#include<stdlib.h>
int main(){
      int seme;
      printf("Inserisci un intero\n");
      scanf("%d",&seme);
      srand(seme);
      for(int i=1;i<=20;i++){
            int a=(rand()%6)+1;
            printf("%d\t", a);
            if(i%5==0)
                  printf("\n");
      }
}
```

Esempio: lancio di un dado

- Con questo nuovo codice, se si inseriscono valori diversi per il seme, si ottengono sequenza diverse
- Ci piacerebbe poter cambiare il seme ogni volta senza doverlo inserire esplicitamente
- Possiamo usare la funzione *time()* della libreria *time.h*
  - tale funzione restituisce il valore dell'orologio di sistema
  - essa richiede un parametro che per i nostri scopi può essere impostato a *0*

Esempio: lancio di un dado

```
#include<stdio.h>
#include<stdlib.h>
#include<time.h>

int main(){
      srand(time(0));
      for(int i=1;i<=20;i++){
            int a=(rand()%6)+1;
            printf("%d\t", a);
            if(i%5==0)
                  printf("\n");
      }
}
```