Funzioni in C

Alessandra Giordani
agiordani@disi.unitn.it
Lunedì 26 aprile 2010
http://disi.unitn.it/~agiordani/

La scorsa lezione abbiamo visto

- Calcolo delle radici di equazioni di 2°grado
  - Utilizzo della libreria *math.h* per *sqrt()*

```
#include <math.h>
```

  - in compilazione dobbiamo aggiungere il flag -lm alla riga di comando

```
gcc radici.c –lm –o radici
./radici
```

Oggi faremo (ovvero farete)

→ Calcolo delle radici di equazioni di 2°grado utilizzando una funzione *delta(a,b,c)* dichiarata e definita separatamente

- Introdurremo le funzioni, i parametri attuali e formali e la visibilità delle variabili
- Se avremo tempo vedremo come vengono memorizzate le variabili e gli array

Breve ripasso su GCC

GCC - GNU C Compiler: esempi di invocazione

- `$ gcc miofile.c`
  - Compila il codice sorgente in `miofile.c`
  - Se il file non contiene errori, crea il file `a.out`
  - `a.out` contiene il codice eseguibile (macchina) del codice sorgente `miofile.c`
- `$ gcc -o fileout miofile.c`
  oppure
  `$ gcc miofile.c -o fileout`
  - Compila il codice sorgente in `miofile.c`
  - Se il file non contiene errori, crea il file `fileout`
  - `fileout` contiene il codice eseguibile (macchina) del codice sorgente `miofile.c`
- Path del file di input e di output assoluti o relativi
- Il nome del file di input deve terminare con `.c`

Fasi della compilazione

- Preprocessing (opzione "-E" di GCC):
  - Rimozione dei commenti
  - Interpretazione di speciali "direttive", ad es:
    - `#include <file.h>`
      include il contenuto di `file.h` nel codice sorgente
    - `#define x y`
      sostituisce nel codice seguente tutte le occorrenze di `x` con `y`
- Compilazione: traduce C in assembly (opzione "-S" di GCC)
- Assemblaggio: traduce assembly in codice oggetto (i.e. linguaggio macchina, opzione "-c" di GCC)
- Linking: combina codice oggetto da piú file per creare un eseguibile. Questo é necessario quando:
  - si usano funzioni di libreria;
  - il codice sorgente é suddiviso in piú file

Per saperne di più: `$ man gcc`

Le funzioni

Una funzione é un insieme di istruzioni che realizzano un algoritmo. È caratterizzata dalla sua signature:

- parametri formali: il tipo dei parametri dell'algoritmo (es: funzione che calcola il delta di un'eq di secondo grado, deve conoscere i coefficienti dell'equazione)
- nome: un nome arbitrario per identificarla (es: `delta`)
- ritorno: il tipo del risultato ritornato dalla funzione (es: `double`)

Una funzione senza argomenti é detta procedura

Ciclo di vita di una funzione: dichiarazione, implementazione, invocazione

Dichiarazione di una funzione

Informa il compilatore che esiste una funzione con una certa signature:

`<tipo di ritorno> <nome> ( <parametri formali> );`

Le regole per la validità dei nomi di funzioni sono le stesse che nel caso delle variabili. Alcuni esempi:

```
double delta(double a, double b, double c);
int max(int x, int y);
double radice_quadrata(double x);
```

- La dichiarazione di una funzione non dice nulla sulla sua semantica (cosa fa) né sulla sua implementazione (come lo fa)
- black-box reuse: se conosciamo la signature di una funzione e sappiamo cosa fa (documentazione), possiamo usarla anche ignorandone l'implementazione

Invocazione di una funzione

L'invocazione di una funzione avviene attraverso la sostituzione dei parametri formali con una tupla compatibile di parametri attuali. Esempi:

```
double delta(double a, double b, double c);

... /* da qualche parte l'implementazione della funzione
       delta deve essere definita */

double coeff_x2 = 8, coeff_x1 = 4, coeff_x0 = 2;
double d1 = delta(coeff_x2, coeff_x1, coeff_x0);
double d2 = delta(8, 4, 2); /* stesso risultato */
double d3 = delta(8/2, 5.6, 2.56);
double d4 = delta(coeff_x2*coeff_x1, coeff_x3, 2*coeff_x2)

char x = delta(8, 4, 2); /* errore: ritorno non compatibile */

/* errore, argomenti non compat. */
double y = delta("ciao", 3, 8);

/* se non serve, é ok ignorare il valore di ritorno */
delta(6, 10, 567.0);
```

Implementazione di funzione

Specifica le istruzioni associate ad un nome di funzione:

```
<tipo di ritorno> <nome> ( <parametri formali> )
{ // inizio delle istruzioni associate alla funzione
   <istruzione1>;
   <istruzione2>;
   ...
   return <valore compatibile con tipo di ritorno>;
} // fine delle istruzioni associate alla funzione
```

Le istruzioni possono essere: dichiarazioni di variabili, assegnamenti, espressioni, chiamate di funzione, etc.
Esempio:

```
/* Restituisce la somma dei primi n numeri interi */
int somma_primi_n_interi(int n)
{
   int result = n * (n+1) / 2;
   return result;
}
```

Esempio di una procedura

- Dichiarazione
- Utilizzo
- Definizione

```c
#include <stdio.h>

void stampamsg (int num) ;
int num = 3; //variabile globale

int main()
{
    int num = 6; //variabile locale
    stampamsg (12);
    printf("La variabile num vale: %d\n" , num);
    return 0;
}

void stampamsg (int num)
{
 printf("Il parametro num vale: %d\n" , num);
}
```

Esempio di una funzione (1/3)

- Abbiamo visto un algoritmo per il calcolo delle radici di un equazione di 2°grado.
- Questo programma è una possibile implementazione

```
#include <stdio.h>
#include <math.h>
int main()
{
float x1,x2,a=2,b=3,c=1,delta;

delta=b*b-4*a*c;
if (delta<0)
      printf("Non esistono radici reali\n");
      else
      {
      if (delta==0)
                x1=x2=-b/(2*a);
           else
                {
                x1=(-b+sqrt(delta))/(2*a);
                x2=(-b-sqrt(delta))/(2*a);
                }
      printf("x1=%f,x2=%f\n",x1,x2);
      }
return 0;
}
```

Esempio (2/3)

- Dichiarazione
- Utilizzo
- Definizione

```c
#include <stdio.h>
#include <math.h>
double delta (double a,double b,double c);

int main()
{
float x1,x2,a=2,b=3,c=1,d;
d=delta(a,b,c);
if (d<0)
      printf("Non esistono radici reali\n");
      else
      {
      if (d==0)
                x1=x2=-b/(2*a);
           else
                {
                x1=(-b+sqrt(d))/(2*a);
                x2=(-b-sqrt(d))/(2*a);
                }
      printf("x1=%f,x2=%f\n",x1,x2);
      }
return 0;
}

double delta (double a,double b,double c)
{
 double res;
 res=b*b-4*a*c;
 return res;
}
```

Esempio (3/3)

Definizione in delta.c

```c
#include "delta.h"

double delta (double a,double b,double c)
{
 double res;
 res=b*b-4*a*c;
 return res;
}
```

Compilazione sorgenti crea file oggetto:

```
gcc -c delta.c -o delta.o
gcc -c radici3.c -o radici3.o
```

Linker di file oggetto crea eseguibile

```
gcc -lm radici3.o delta.o -o
   radici3
./radici3
```

Dichiarazione in delta.h

```c
#include <stdio.h>
#include <math.h>

double delta (double a,double b,double c);
```

Utilizzo in radici3.c

```c
#include "delta.h"

int main()
{
float x1,x2,a=2,b=3,c=1,d;

d=delta(a,b,c);
if (d<0)
      printf("Non esistono radici reali\n");
      else
      {
      if (d==0)
                x1=x2=-b/(2*a);
           else
                {
                x1=(-b+sqrt(d))/(2*a);
                x2=(-b-sqrt(d))/(2*a);
                }
      printf("x1=%f,x2=%f\n",x1,x2);
      }
return 0;
}
```

Memorizzazione Variabili

- La dichiarazione di una variabile associa un tipo di dato ad un nome simbolico
- La dichiarazione “alloca” (cioè “riserva”) memoria per una variabile di un certo tipo
- L'allocazione della memoria in C è contigua
  - un dato *int* viene memorizzato in 4 byte di memoria adiacenti
  - un dato *double* in 8 byte di memoria adiacenti
  - un *array di 5 interi* in 20 byte di memoria adiacenti
  - …

L'operatore **sizeof()**

- NB: non è una funzione!

```c
#include <stdio.h>

int main()
{
    printf("Size of (in bytes):\n");
    printf("int: %d\n", sizeof(int));
    printf("short int: %d\n", sizeof(short int));
    printf("long int: %d\n", sizeof(long int));
    printf("char: %d\n", sizeof(char));
    printf("float: %d\n", sizeof(float));
    printf("double: %d\n", sizeof(double));
    printf("long double: %d\n", sizeof(long double));
    return 0;
}
```

Memorizzazione Array 1/2

- In C gli array sono intrinsecamente legati al concetto di **puntatore**
- Per capire veramente come funzionano gli array (e per sfruttarli a dovere) bisogna capire i puntatori
- L'utilizzo dei puntatori (e dell'allocazione dinamica della memoria) consentono un uso molto più flessibile degli array
- MA li vedremo prossimamente...

Memorizzazione Array 2/2

- La dichiarazione di un array di n elementi di tipo T causa l'allocazione contigua di **n x sizeof (T)** byte. Ad es:

```
int myarr[7];
```

- dichiara un array di 7 interi chiamato myarr, e alloca 7xsizeof(int) = 28 byte di memoria contigua
- Il nome dell'array (`myarr`) è un puntatore al 1°byte di memoria allocata
  - L'indirizzo del 2° elemento dell'array e dato da `myarr + 1`
  - L'indirizzo del 3° elemento dell'array e dato da `myarr + 2`
  - … e così via, ma lo vedremo meglio più avanti!

Esempi di esami

All’indirizzo

http://danielepighin.net/cms/teaching

Sez. Informatica Generale 2007-2008

troverete tracce e soluzioni degli appelli degli anni passati.