Funzioni

Nicola Bicocchi

DIEF - UNIMORE

Le funzioni

- Un funzione è una sequenza di istruzioni che vengono attivate a seguito di una apposita chiamata
- Vantaggi:
  - favoriscono modularizzazione del codice
  - favoriscono il riuso del codice (librerie)
  - favoriscono lo sviluppo incrementale (creazione di interfacce che disaccoppiano parti di software)
  - favoriscono la leggibilità del codice
- Svantaggi:
  - Determinazione dell'indirizzo di rientro al codice chiamante
  - Scambio di informazioni fra sottoprogramma e codice chiamante (passaggio dei parametri)

Dichiarazione di funzioni

- Serve per segnalare al compilatore l'esistenza di una determinata funzione (e come invocarla) ma non specifica le istruzioni che compongono la funzione
- La dichiarazione di una funzione deve sempre precedere nel sorgente la prima invocazione della stessa. La definizione, invece, può essere presente in un qualunque punto del sorgente o in una libreria esterna
- La dichiarazione specifica il *prototipo* della funzione:
  - il tipo ritornato
  - il nome della funzione
  - l'elenco degli argomenti (o parametri)
- In fase di dichiarazione si può omettere il nome dei parametri

```
1 int secondi(int h, int m, int s);
2 // oppure
3 int secondi(int, int, int);
```

Definizione di funzioni

Una definizione è costituita da due parti:

- la dichiarazione della funzione
- il corpo della funzione, racchiuso tra parentesi graffe e comprendente zero o più di queste componenti:
  - dichiarazioni e definizioni di variabili
  - istruzioni
  - istruzione return

```
1 /* esempio di definizione */
2 int secondi(int h, int m, int s) {
3     return (3600 * h + 60 * m + s);
4 }
```

Definizione e invocazione

- L'invocazione di una funzione è l'operazione con la quale si richiama l'esecuzione della funzione
- Per richiamare una funzione si deve utilizzare il nome della funzione seguita dagli argomenti passati alla funzione racchiusi da parentesi tonde e separati da virgole
- Un'invocazione di funzione trasferisce il controllo alla prima istruzione della funzione stessa
- Una funzione termina quando (a) si incontra l'istruzione return, oppure (b) si esegue la sua ultima istruzione

```
1  int secondi(int h, int m, int s) {
2      return (3600 * h + 60 * m + s);
3  }
4
5  int main() {
6      int h=1, m=1, s=1, totale_secondi;
7      totale_secondi = secondi(h,m,s);
8      printf("Totale secondi: %d\n", totale_secondi);
9  }
```

Tipo void

- L'uso del tipo **void** nelle funzioni identifica *tipi nulli*
- Se usato come tipo di ritorno, la funziona non restituisce alcun valore
- Se usato come parametro di input, la funzione non accetta nessun parametro

```c
void say_hi(void) {
    printf("Hi!\n");
}

int main() {
    say_hi():
    return 0;
}
```

Parametri di input

- È obbligatorio indicare il tipo delle variabili. Se non ci sono variabili, si usa il tipo speciale void
- Non è possibile indicare parametri facoltativi
- È possibile indicare funzioni con numero di parametri variabile, in stile printf
- Il passaggio dei parametri avviene *per copia*

```c
void try_modification(int value) {
    // solo la copia ricevuta dalla funzione viene modificata
    value = 0;
}

int main(){
    int a=5;
    try_modification(a);
    printf("a = %d\n", a);
    return 0;
}
```

Visibilità

- Le variabili che abbiamo utilizzato fin'ora sono **variabili locali**, visibili solo all'interno della funzione
- Le funzioni invocate **non hanno accesso** alle variabili di livello superiore!
- Il C supporta anche **variabili globali**, visibili da tutte le funzioni
- *Preferibile limitare utilizzo di variabili globali, oppure utilizzarle come costanti*

Variabili globali

```c
#include<stdio.h>

int a;
// const int a produce invece un errore!

void test_visibilita(void){
    printf("a=%d\n", a);
    a = 10;
}

int main(){
    a=5;
    test_visibilita();
    printf("a=%d\n", a);
    return 0;
}
```

Tempo di vita delle variabili locali

- La gestione della memoria delle variabili locali è **automatica**
  - Vengono allocate al momento dell'invocazione della funzione
  - Vengono de-allocate al momento del ritorno della funzione
- Ad ogni invocazione le variabili e i parametri della funzione non dipendono dalle esecuzioni precedenti!

Variabili locali statiche

- Una variabile locale è detta **statica** se il suo tempo di vita corrisponde a quelllo del processo
- E' possibile utilizzare variabili statiche per avere funzioni che mantengono uno stato fra diverse invocazioni

```c
void counter() {
    static int count=0;
    count++;
    printf("Il valore di count é %d\n", count);
}

int main(){
    counter();
    counter();
    return 0;
}
```

Funzioni matematiche

Funzione	Descrizione
abs(x)	Restituisce il valore assoluto
fabs(x)	Restituisce il valore assoluto
ceil(x)	Restituisce l'intero più piccolo maggiore o uguale a x
floor(x)	Restituisce l'intero più grande minore o uguale a x
round(x)	Restituisce l'intero più vicino a x
sqrt(x)	Restituisce la radice quadrata
pow(a, b)	Operazione di elevamento a potenza a^b
exp(x)	Funzione esponenziale
log(x)	Restituisce il logaritmo naturale (in base e)

Passaggio per valore (copia)

- Secondo la modalità del passaggio per valore **ogni funzione ha una propria zona di memoria per memorizzare i dati** (messa a disposizione solo al momento dell'effettivo utilizzo e rilasciata quando non è più necessaria)
- Al momento dell'uso della funzione **i parametri sono copiati**, quindi non vi è un accesso diretto ai valori del codice chiamante

```c
void scambia(int a, int b) {
    int tmp = a;
    a = b;
    b = tmp;
}

int main() {
    int a = 2, b = 3;
    scambia(b, a);
    printf("a=%d b=%d\n", a, b);
}
```

Passaggio per valore (copia)

Figura 6.2 rifatta

Passaggio per riferimento (copia del riferimento)

- Permette alla funzione chiamata di modificare il valore della variabile passata dal chiamante
- Evita la copia di variabili voluminose
- Contente alla funzione chiamata di ritornare più di un valore di ritorno
- Il passaggio per riferimento **implica il passaggio per valore di un puntatore alla variabile**

Passaggio per riferimento (copia del riferimento)

```c
void scambia(int *a, int *b) {
    int tmp = *a;
    *a = *b;
    *b = tmp;
}

int main() {
    int a = 2, b = 3;
    scambia(&b, &a);
    printf("a=%d b=%d\n", a, b);
}
```

Passaggio per riferimento (copia del riferimento)

Figura 6.2 rifatta

Passaggio di puntatori const

Passaggio di parametri al programma principale

Ricorsione