

Fondamenti di Informatica
Ingegneria Clinica
Lezione 10/12/2010



Raffaele Nicolussi
FUB - Fondazione Ugo Bordoni
Via del Policlinico, 147 – 00161
Roma

Esercizio

- **Leggere da tastiera una sequenza di interi e calcola la lunghezza della sequenza. L'immissione è terminata con uno 0 (che non deve essere conteggiato).**

Algoritmo

- **Inizializza la lunghezza della stringa a **-1****
 - **Leggi il numero**
 - **Incrementa di uno la lunghezza**
- **Mentre il numero letto è diverso da zero, torna ad eseguire dalla lettura del numero**
- **Stampa la lunghezza**

```
/* Legge da tastiera una sequenza di interi, terminata da 0.  
   Calcola la lunghezza della sequenza (senza lo 0) e la  
   stampa.*/
```

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

```
int main(void)
```

```
{
```

```
    int lunghezza;
```

```
    int dato;
```

```
    printf("Inserisci una sequenza di interi,  
          terminata da 0\n");
```

```
    lunghezza = -1;
```

```
        /* all'inizio la lunghezza e' nulla */
```

```
    do {
```

```
        scanf("%d", &dato);
```

```
        lunghezza = lunghezza + 1;
```

```
    } while (dato != 0);
```

```
    printf("La sequenza (senza lo 0 finale) e'  
          lunga %d\n", lunghezza);
```

```
    return 0;
```

```
}
```

```
seq_int.c
```

Esercizio

- **Contare il numero di spazi in una frase immessa da tastiera (fino all'immissione di un ritorno a capo)**

Algoritmo

- **Inserire una frase**
- **Inizializza un contatore**
 - **Leggere i caratteri uno per uno (usa `getchar()`)**
 - **Se il carattere è uguale allo spazio, incrementa il contatore**
 - **Fai questo fino a che il carattere non è uguale al carattere di ritorno a capo**

```
#include <stdio.h>
#include <stdlib.h>
int main( void )
{
    int ch, num_spazi = 0;

    printf( "Inserisci una frase:\n" );
    do
    {
        ch = getchar();
        if (ch == ' ')
            num_spazi++;
    } while (ch != '\n');

    printf( "Il numero degli spazi e'
            %d.\n",num_spazi);

    return 0;
```

spazi.c

Istruzioni break

- **break:** permette di uscire “forzatamente” da un blocco (while, for, do/while, switch)
 - da usare con cautela perché de-struttura il programma
 - nel caso dello switch è necessario
 - negli altri casi si possono trovare soluzioni alternative

Istruzioni return, exit

- **return:** consente ad una funzione di tornare il controllo alla funzione che l'ha chiamata
 - può restituire un valore
 - nella funzione **main** viene in genere usata con valore 0 per segnalare il termine normale del programma al sistema operativo
 - return 0
- **exit:** consente la terminazione di una funzione, con possibilità di gestione dell'errore
 - exit (<int>)

Esercizio

Esempio uso break

- Contare le "a/A" su una stringa di 50 caratteri. Fermarsi se arrivati a fine riga

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

```
int main (void) {  
    int ch, cont, quantea=0, lungh=50;  
    printf("Inserire una stringa \n");  
    for (cont=1; cont<=lungh; cont++)  
    {  
        ch=getchar( );  
        if (ch == '\n')  
            break;  
        else  
            if ((ch=='a') || (ch=='A'))  
                quantea++;  
    }  
    printf(" a o A nella stringa = %d \n",  
        quantea);  
    return 0;  
}
```


**/*Contare quanti voti di tipo A, B, C, D ha
riportato uno studente*/**

#include <stdio.h>

int main (void) {

**int grade, aCnt=0, bCnt=0, cCnt=0,
dCnt=0;**

printf("Inserisci i (A-D)\n");

printf("Termina con F\n");

while ((grade=getchar()) != 'F') {

switch (grade) {

**case 'A': case 'a': ++aCnt;
break;**

**case 'B': case 'b': ++bCnt;
break;**

**case 'C': case 'c': ++cCnt;
break;**

**case 'D': case 'd': ++dCnt;
break;**

case '\n' : case ' ': break;

**default: printf("Errore.
Inserisci un altro voto\n");**

}

}

```
printf("Totale per ogni voto\n");  
printf("A: %d\n", aCnt);  
printf("B: %d\n", bCnt);  
printf("C: %d\n", cCnt);  
printf("D: %d\n", dCnt);  
return 0;  
}
```

conta_voti

Soluzione per raffinamenti successivi

- La risoluzione dei problemi va affrontata per livelli di raffinamento successivi
 - Definire prima le operazioni da eseguire a grandi linee
 - Dettagliare ogni singola operazione
- Al primo livello ci concentriamo su operazioni di una certa complessità (macroscopiche), **tralasciando i dettagli realizzativi di ogni singola attività**
 - A successivi livelli di raffinamento ci preoccuperemo di dettagliare sempre più le attività complesse
 - **Analisi top-down**

Soluzione per raffinamenti successivi

- **Aspetti importanti:**
 - **Alcune funzionalità possono essere riutilizzate in altri programmi o più volte all'interno dello stesso programma**
 - **L'identificazione delle funzionalità è più semplice**
 - **La soluzione di problemi “piccoli” è più facile**
 - **Il programma è più facile da correggere**

Una funzionalità può essere implementata come funzione

Struttura di un programma C

- Un programma C è composto da blocchi, dette **funzioni**.
 - Ogni funzione
 - è un blocco **di istruzioni che astraggono una specifica operazione**
 - è individuata da
 - **un tipo, un nome, una lista di parametri**
- L'astrazione consiste
 - Nel creare unità di programma (funzioni) dando un **nome** ad un gruppo di istruzioni
 - Stabilire delle modalità di comunicazione tra la funzione così creata ed il resto del programma in cui si inserisce

Passaggio di parametri

- **Il passaggio di parametri ad una funzione può avvenire**
 - **Per valore**
 - **Per indirizzo**
- **Ma prima di entrare nel dettaglio un breve ripasso sulle funzioni ...**

Funzione

- Una funzione è identificata da un **nome**
- Ogni volta che si vuole eseguire questa funzione, **verrà usato il nome con cui è stata creata**
- L'effetto sarà l'esecuzione del blocco di istruzioni relativo alla funzione
 - **richiesta di attivazione** (o **chiamata o invocazione**) della **funzione**.

**Funzione chiamante
(della funzione **prova**)**

```
int main (void)  
{ .....  
  prova( ..);
```

**Richiesta di
attivazione
(chiamata)**

```
.....  
}  
  
void prova (...)  
{  
.....
```

**All'atto dell'attivazione
di una funzione **viene**
sospesa l'esecuzione del
blocco che contiene la
richiesta di
attivazione.**

**Il controllo passa
alla funzione attivata**

**Dopo che è stata completata l'esecuzione
di prova, l'attivazione termina e il
controllo torna al programma
chiamante**

- **Distinguiamo tra:**
 - **unità chiamante:** “chiama” un’altra unità per alcune operazioni; sospende l’esecuzione proprie istruzioni (nell’esempio **main**)
 - **unità chiamata:** il controllo passa alle istruzioni della unità chiamata e, al termine della esecuzione, torna all’unità chiamante (nell’esempio **prova**)
- **La comunicazione tra unità chiamanti e chiamate è gestita tramite il passaggio di parametri**
 - **meccanismo di comunicazione che permette l’attivazione dell’unità chiamata su istanze diverse delle stesse variabili**

- Nella **funzione chiamata** (es. prova) sono dichiarati i
 - **parametri formali**: all'atto dell'attivazione della funzione sono legate ai parametri dell'unità chiamante
 - Nella **funzione chiamante** (es. main) sono usati i
 - **parametri attuali o effettivi**: i valori che sono usati nella funzione chiamata
 - **Istanziamento dei parametri formali**
 - **Deve esserci compatibilità di tipo tra parametri formali e parametri attuali corrispondenti**
 - **L'attivazione di una funzione stabilisce**
 - **un collegamento tra i parametri attuali (valori) della chiamante e parametri formali della chiamata**

```
int main (void)
{
  int a=3, b=5, c;
  c=prova( a, b);
  return 0;
}
```

**a e b sono
parametri attuali**

3

5

```
int prova (int x, int y)
{
  int pro;
  pro=(x+y)-x*y+sqrt(x/y)
  return pro;
}
```

**x e y sono
parametri formali**

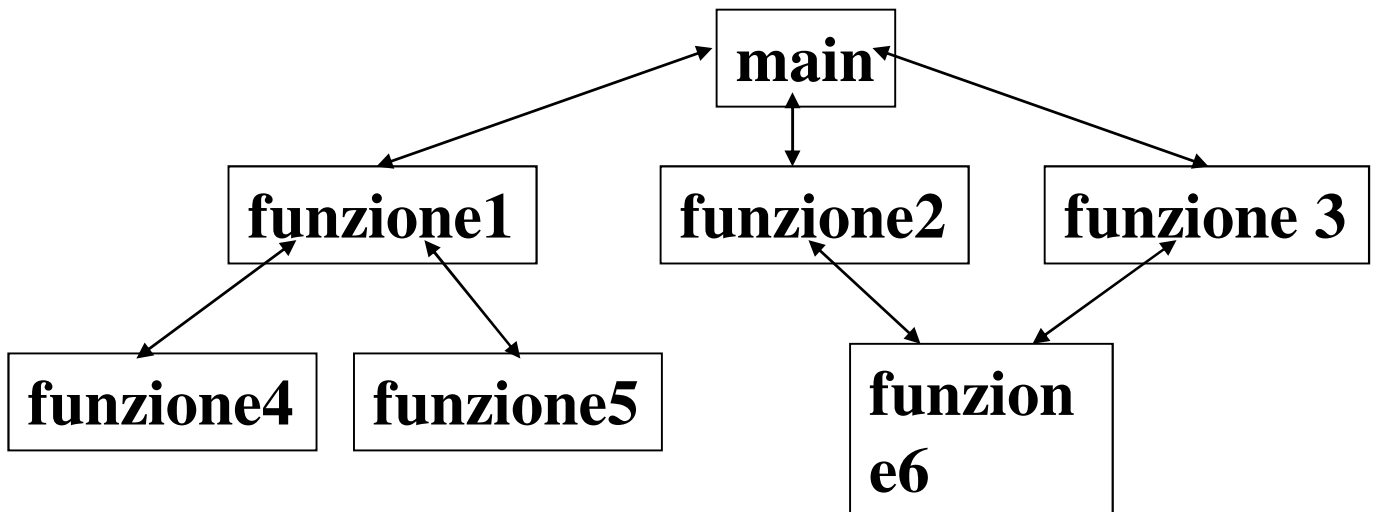
- **legame dei parametri:**
 - **associazione tra parametri formali e parametri attuali che avviene al momento dell'attivazione della funzione**
- **I parametri formali definiscono una lista di oggetti fittizi che hanno bisogno di avere assegnato un valore**
- **Quando la funzione viene chiamata sono sostituiti dai parametri attuali, che sono collegati al valore (ai valori) che dobbiamo elaborare**

```
void scambia (int x, int y) { ... }  
scambia (x1, y1)  
scambia (w, z)  
scambia (a, b) .....
```

- **scambia(x1, y1):** alla sua chiamata viene fatta un'associazione tra i **parametri formali (x, y)** e **attuali (x1, y1)**
 - **parametro formale x ->**
parametro attuale x1
 - **parametro formale y ->**
parametro attuale y1
- **scambia (w, z):**
 - **parametro formale x ->**
parametro attuale w
 - **parametro formale y ->**
parametro attuale z

Funzioni

- Un programma C è composto da insiemi di **funzioni** che possono usare al loro interno altre funzioni e/o le funzioni della libreria standard del C



Funzioni

- Le funzioni possono apparire come:
 - **prototipo**: dichiarazione del tipo, nome e argomenti
 - **definizione**: intestazione, dichiarazione variabili locali, istruzioni
 - **chiamata**: uso di una funzione all'interno di un'altra funzioni che ne utilizza le funzionalità o per eseguire istruzioni o per calcolare un valore

Definizione di una funzione

- Definisce il comportamento della funzione, il numero e il tipo degli argomenti

```
<tipo di ritorno> <nome>
(<lista_parametri>)
{
    dichiarazioni_variabili locali;
    istruzioni;
}
```

- Tipo di ritorno: è il tipo di dato restituito dalla funzione.
Default: *int*. Se non torna alcun valore si usa *void* (va specificato esplicitamente).
 - deve essere uno dei tipi del C


```
<tipo di ritorno> <nome>
(<lista_parametri>)
{
    dichiarazioni_variabili locali;
    istruzioni;
}
```

- **Lista parametri:** contiene la lista degli argomenti della funzione identificati secondo la coppia **<tipo nome>**. Se il tipo è omesso, viene considerato **int**. In caso di assenza, si usa **void**.
 - La funzione deve essere chiamata con lo stesso numero di argomenti e tipi compatibili
 - Sono detti **parametri formali**
 - Mezzo di comunicazione tra funzione ed esterno

```
<tipo di ritorno> <nome>  
(<lista_parametri>)  
{  
    dichiarazioni_variabili locali;  
    istruzioni;  
}
```

- **Dichiarazioni variabili locali:** sono variabili usate all'interno della funzioni. Non sono visibili all'esterno.
- **Istruzioni:** parte esecutiva. Se la funzione torna un valore, deve contenere un'istruzione adatta allo scopo (per esempio, return).

Prototipo di una funzione

- **Indica al compilatore:**
 - **il tipo di dato restituito dalla funzione**
 - **il numero dei parametri, il loro tipo e numero**
- **Il compilatore può controllare che i parametri attuali nelle chiamate delle funzioni siano compatibili con il tipo dei parametri formali**

```
int maximun (int, int, int);
```

```
int maximum (int x, int y, int z) {...}
```

- **Ma i nomi degli argomenti sono ignorati**
- **Una chiamata che non corrisponda al prototipo genera un errore di sintassi**

Coercizione degli argomenti

- **Importante caratteristica legata ai prototipi: conversione forzata degli argomenti**
 - **viene effettuata una conversione dell'argomento nel tipo specificato dal prototipo**
- **Per esempio, se nel prototipo la funzione richiede un argomento di tipo **float** e viene invece chiamata con un **int**, l'argomento intero (es. 5) viene convertito nel tipo dichiarato nel prototipo (float, es. 5.0) **PRIMA CHE LA FUNZIONE SIA CHIAMATA****

- **I valori degli argomenti che non corrispondono al tipo definito nel prototipo **sono convertiti** in modo appropriato (regole di promozione)**
- **Ci possono essere dei problemi nel risultato, se avviene un passaggio da un tipo più grande ad uno più piccolo (violazione delle regole di promozione)**
- **Convertire i dati dei tipi più alti provoca valori scorretti.**

- **L'assenza del prototipo elimina questi vantaggi**
- **In caso di mancanza, il programma costruisce un suo prototipo utilizzando la prima occorrenza della funzione, o la sua definizione o la sua chiamata.**
- **Per default, il compilatore assume `int` come `tipo tornato` e nulla circa gli argomenti.**

Chiamata (invocazione) di una funzione

- Trasferisce il controllo del programma alla funzione specificata.

<nome funzione> (<arg1>, ..., <argN>)

- Gli argomenti sono i *parametri attuali*.
- Gli argomenti devono essere in corrispondenza con quelli della definizione (prototipo) della funzione argomenti
 - il tipo dei valori deve essere compatibile
 - devono essere lo stesso numero
- La funzione può tornare o meno un valore

- **Tre modi per tornare il controllo:**
 - **esecuzione dell'ultima istruzione**
 - **Non torna valori**
 - **return;**
 - **Viene restituito il controllo, ma nessun valore**
 - **return <espressione>;**
 - **Viene restituito il controllo, e il valore di <espressione>**

Modalità di passaggio parametri

- Cosa accade ai valori dei parametri attuali passati alla funzione?
 - La modalità che abbiamo usato finora è detta
 - **Chiamata per valore**
 - **Si dice che i parametri sono legati per valore**
 - **Scenario:**
 - **funzione identificata con P dichiarata con un parametro formale pf**
 - **.... P (pf); (dichiarazione)**
 - **La funzione viene attivata con un parametro attuale pa**
 - **P(pa) (chiamata)**
- all'interno di un'altra funzione (compreso il main)**