Corso di Laboratorio di Sistemi Operativi
A.A. 2016–2017

Lezione 8

Ivan Scagnetto
ivan.scagnetto@uniud.it
Nicola Gigante
gigante.nicola@spes.uniud.it

Dipartimento di Scienze Matematiche, Informatiche e Fisiche
Università degli Studi di Udine

A.A. 2016–2017 - Primo Semestre - 28/10/2016

Funzioni

Le funzioni in C

I programmi C sono costituiti da **dichiarazioni** di variabili, tipi, e **funzioni**, e da **definizioni** di variabili e funzioni.

La **dichiarazione** di una funzione (chiamato anche **prototipo**) ha la seguente sintassi:

```
tipo_ritornato nome_funzione(lista_parametri);
```

Esempio:

```
int factorial(int n);
```

Le funzioni in C

I programmi C sono costituiti da **dichiarazioni** di variabili, tipi, e **funzioni**, e da **definizioni** di variabili e funzioni.

È possibile scrivere funzioni che non ritornano nulla, dichiarando il tipo `void`. Esempio:

```
void putchar(int c);
```

Le funzioni in C

I programmi C sono costituiti da **dichiarazioni** di variabili, tipi, e **funzioni**, e da **definizioni** di variabili e funzioni.

La **definizione** di una funzione implementa effettivamente una funzione dichiarata precedentemente:

```
tipo_ritornato nome_funzione(lista_parametri)
{
  istruzione;
  istruzione;

  ...

  return valore;
}
```

Le funzioni in C

I programmi C sono costituiti da **dichiarazioni** di variabili, tipi, e **funzioni**, e da **definizioni** di variabili e funzioni.

La **definizione** di una funzione implementa effettivamente una funzione dichiarata precedentemente. Esempio:

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

Esempio

Il seguente programma è costituito da una funzione `int power(int m, int n)` che calcola la potenza m^n, e dalla funzione principale `main` che la utilizza.

```c
#include <stdio.h>

int power(int, int); // dichiarazione

int main()
{
   for(int i = 0; i < 10; ++i)
      printf("%d %d %d\n", i, power(2,i), power(-3,i));

   return 0;
}

int power(int m, int n) // definizione
{
   int p = 1;
   for (int i = 0; i < n; ++i)
      p = p * m;
   return p; // restituisce il valore di p al chiamante
}
```

Alcuni dettagli

Alcune cose importanti da tenere a mente:

- La dichiarazione, o la definizione, di una funzione deve **precedere** il punto in cui la funzione viene **chiamata**.
- La dichiarazione è necessaria se la funzione viene definita e chiamata in file diversi.
- In assenza della dichiarazione, il compilatore non può controllare che la funzione venga chiamata con tipi corretti.
- Se una funzione indica un tipo di ritorno, **deve** contenere tutte le istruzioni `return` necessarie.

Attenzione: Il compilatore **non** controlla questi obblighi.
In caso di violazioni il programma può comportarsi in modo errato durante l'**esecuzione**, in modo imprevedibile.

Passaggio per valore

- In C tutti gli argomenti delle funzioni sono passati **per valore**, ovvero le funzioni lavorano su copie dei parametri e non modificano i parametri passati dal chiamante.
- Il passaggio **per riferimento**, presente in altri linguaggi, non esiste in C. Si può simulare per mezzo dei **puntatori**, ovvero passando l'indirizzo in memoria dell'argomento.
  Sarà argomento della prossima lezione.

Passaggio per valore

Esempio

```c
#include <stdio.h>

void fake_swap(int x, int y) {
  int z = x;
  x = y;
  y = z;
}

int main()
{
  int x = 42, y = 0;

  fake_swap(x,y);

  printf("x = %d, y = %d\n", x, y);

  return 0;
}
```

Variabili e scope

Lo **scope** di una variabile è la parte di codice in cui essa è **visibile**.

- Le variabili dichiarate all'**interno** delle funzioni sono **locali** a queste ultime e sono dette **automatiche**, in quanto sono create al momento della chiamata della funzione e cessano di esistere quando questa termina.
- Lo stesso discorso vale per le variabli dichiarate in blocchi interni (es. in un ciclo `for`).
- È buona norma **inizializzare** ogni variabile al momento della dicharazione (es. `int x = 0;` invece di `int x;`), perchè altrimenti il valore iniziale della variabile non è definito a priori.
- È possibile definire variabili **globali** al di fuori delle funzioni, che saranno accessibili a tutto il programma. L'uso di variabili globali è sconsigliabile dal punto di vista del design del codice.

Array

Array

Un array è un tipo di dato **aggregato** formato da un numero **fissato** di elementi dello stesso tipo.

Per dichiarare un array:

```
tipo nome[N] = { valori };
```

Esempio:

```
int voti[12] = { 18, 30, 27, 20, 22, 28, 25, 25, 30, 27, 18, 26 };
```

Array

Un array è un tipo di dato **aggregato** formato da un numero **fissato** di elementi dello stesso tipo.

È possibile inizializzare tutti i valori a zero in un colpo:

```
int statistiche[300] = { };
```

Se la dimensione non viene specificata, si tratterà del numero di elementi elencati nell'inizializzazione:

```
int voti[] = { 18, 30, 27, 20, 22, 28, 25, 25, 30, 27, 18, 26 };
```

Esempio I

Media pesata dei valori contenuti in un array

```c
#include <stdio.h>

#define N_ESAMI 12

int main() {
  int voti[N_ESAMI] = { 18, 30, 27, 20, 22, 28, 25, 25, 30, 27, 18, 26 };
  int crediti[N_ESAMI] = { 12, 12, 12, 12, 6, 6, 6, 6, 6, 6, 6, 6 };

  int somma = 0, totale = 0;

  for(int i = 0; i < N_ESAMI; ++i) {
    somma += voti[i] * crediti[i];
    totale += crediti[i];
  }

  float media = (float) somma / totale; // Notare il cast esplicito a float

  printf("La mia media pesata e': %2.2f\n", media);

  return 0;
}
```

Esempio II

Tabella dei valori della la funzione $\sin x$

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

#define N_CAMPIONI 10

int main()
{
  float valori[N_CAMPIONI] = { 0.0 };

  float step = 2 * M_PI / N_CAMPIONI;

  for(int i = 0; i < N_CAMPIONI; ++i) {
    valori[i] = sin(i * step);
  }

  for(int i = 0; i < N_CAMPIONI; ++i) {
    printf("sin(%1.3f) = %1.3f\n", i * step, valori[i]);
  }

  return 0;
}
```

Array e problemi di sicurezza

In C gli array, come il resto, vengono gestiti a **basso livello**:

- Il programma fa esattamente ciò che scrivete, né più né meno.
- In particolare, il linguaggio non prevede controlli sulla **validità degli indici** usati per accedere agli array.
- Accedere ad un array con un indice troppo grande (*out of bounds*) comporta seri problemi.
  - Nel migliore dei casi, il sistema operativo uccide il processo.
  - Nella maggioranza dei casi, vengono scritte altre posizioni di memoria non prevedibili a priori.
  - In ogni caso, il comportamento del programma è **indefinito**.

Attenzione: Errori di programmazione che causano la scrittura oltre il termine di un array sono la causa del 90% dei bug e dei problemi di sicurezza del software.

Array e problemi di sicurezza

Esempio

```c
int main() {
  int valori[10] = { 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 };
  int risposta = 42;

  // Azzeriamo l'array
  for(int i = 0; i <= 10; ++i) {
    valori[i] = 0;
  }

  printf("La risposta e': %d\n", risposta);

  return 0;
}
```

Qual è l'output di questa programma?

Esercizi

Esercizi

Per venerdì 4 novembre

1. Ripetere l'esercizio 2 o 3 della Lezione 7, definendo però una funzione `is_whitespace()` per controllare se un carattere è di spazio bianco oppure no.
   - Nota: non esiste il tipo `bool`, a differenza di C++ o Java. Vedere le aggiunte a riguardo nelle slide della Lezione 7.
2. Definire una funzione `int lg(int n)` che trovi il massimo numero m tale che $10^m \leq n$ (ovvero la parte intera di $\log_{10} n$).
3. Scrivere un programma che ordini un array di numeri interi, utilizzando un algoritmo di ordinamento visto a lezione di Algoritmi e Strutture Dati (es. bubble sort o insertion sort).
   - Nota: In attesa di vedere i puntatori (necessari per passare array come argomenti a funzioni), ci si limiti ad ordinare un array dichiarato e inizializzato nel medesimo blocco.

Attenzione: Fornire sempre la dichiarazione di ogni funzione.