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

Lezione 9

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

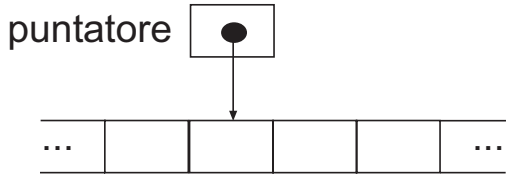
Dipartimento di Scienze Matematiche, Informatiche e Fisiche
Università degli Studi di Udine

A.A. 2016–2017 - Primo Semestre - 2/11/2016

Puntatori

I puntatori

Un **puntatore** è una variabile che contiene l'**indirizzo** di un'altra variabile.



L'uso e la manipolazione dei puntatori e ciò che distingue maggiormente il C dagli altri linguaggi.

I puntatori

La dichiarazione di un puntatore avviene in modo simile a quella di un'altra variabile:

```
int *p = valore iniziale;
```

Il tipo `int *` indica un puntatore a variabili di tipo `int`.

Il puntatore va inizializzato con l'**indirizzo** di una variabile di tipo `int`, oppure con il valore speciale `NULL`.

```
int x = 42;

int *p = &x;   // ora p punta ad x
int *q = NULL; // q non punta a nulla
```

Uso dei puntatori

L'uso dei puntatori gira intorno a due **operatori unari**.

- L'operatore *address of* (&) ottiene un puntatore ad una variabile:

```
int x = 42;
int *p = &x; // ora p punta ad x
```

- L'operatore di *dereferenziazione* (*), permette di accedere alla variabile puntata da un puntatore:

```
int x = 42;
int *p = &x;

printf("%d\n", *p); // stampa 42
```

Uso dei puntatori

Esempio: funzione swap()

Versione non funzionante:

```c
#include <stdio.h>

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

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

  fake_swap(x,y);

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

  return 0;
}
```

Versione funzionante:

```c
#include <stdio.h>

void swap(int *x, int *y) {
  int temp = *x;
  *x = *y;
  *y = temp;
}

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

  swap(&x, &y);

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

  return 0;
}
```

Uso dei puntatori

La funzione `scanf()`

La funzione `scanf()` è la controparte di `printf()` per **leggere** valori dallo standard input.

```c
#include <stdio.h>

int main()
{
  int x = 0;

  printf("Inserisci un numero intero: ");
  scanf("%d", &x);

  printf("Il doppio di %d e' %d", x, x * 2);

  return 0;
}
```

Uso dei puntatori

La funzione `scanf()`

L'uso di `scanf()` è simile a quello di `printf()`:

```
int x = 0;
scanf("%d", &x);
```

- Il primo argomento è una striga contenente la **specifica di formato** dei dati in input, con sintassi simile a `printf()`
  - `"%d"` per numeri interi, `"%f"` per numeri di tipo `float`, ecc...
- Gli argomenti seguenti sono invece **puntatori** alle variabili dove andranno scritti i valori letti in input.
- La funzione ritorna il numero di elementi letti con successo, oppure `EOF` se l'input è terminato prima di leggere qualcosa.
- Maggiori dettagli alla lezione sulla manipolazione di stringhe.

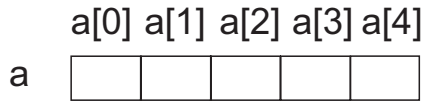
Puntatori e array

Puntatori e array

I puntatori sono lo strumento principale per la manipolazione degli array in C.

Supponiamo di dichiarare un array come segue:

```
int a[5] = { 0, 1, 2, 3, 4 };
```

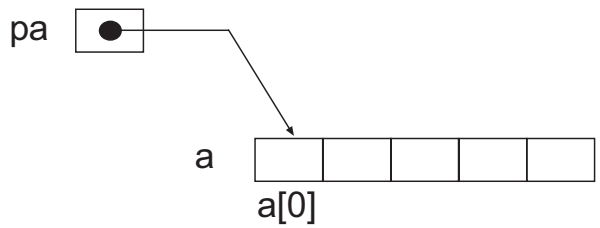


Puntatori e array

I puntatori sono lo strumento principale per la manipolazione degli array in C.

Ora, estraiamo un puntatore al suo primo elemento:

```
int *pa = &a[0];
printf("%d\n", *pa); // stampa zero
```



Per ottenere il puntatore al primo elemento di un array si può anche menzionare semplicemente il nome dell'array:

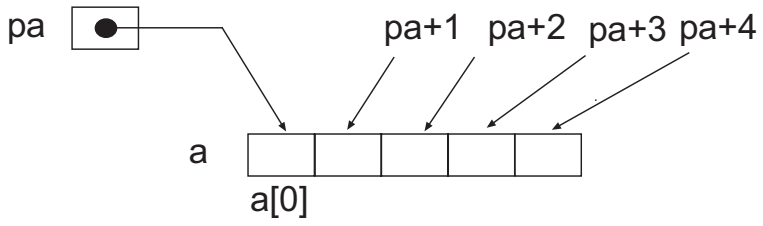
```
int *pa = a; // equivalente ad &a[0]
```

Puntatori e array

I puntatori sono lo strumento principale per la manipolazione degli array in C.

Incrementando il puntatore è possibile ottenere puntatori agli elementi successivi:

```
pa = pa + 1;
printf("%d\n", *pa); // stampa 1
```



Puntatori e array

Esempio

```c
#include <stdio.h>

#define SIZE 10

int main() {
  int array[SIZE] = { };
  int val = 0;

  printf("Inserisci un valore: ");
  scanf("%d", &val);

  for(int *p = array; p < array + SIZE; ++p) {
    *p = val;
  }

  for(int i = 0; i < SIZE; ++i) {
    printf("array[%d] = %d\n", i, array[i]);
  }

  return 0;
}
```

Puntatori e array

Esempio 2

```c
#include <stdio.h>
#define SIZE 10

void fill(int *, int, int);

int main() {
  int array[SIZE] = { }, val = 0;

  printf("Inserisci un valore: ");
  scanf("%d", &val);

  fill(array, SIZE, val);
  for(int i = 0; i < SIZE; ++i)
    printf("array[%d] = %d\n", i, array[i]);

  return 0;
}

void fill(int *begin, int size, int value) {
  for(int *p = begin; p < begin + size; ++p)
    *p = value;
}
```

Puntatori e array

Passaggio di array a funzioni

```
void fill(int *begin, int size, int value) {
  for(int *p = begin; p < begin + size; ++p)
    *p = value;
}
```

I puntatori sono l'unico modo per "passare" array alle funzioni:

- Passare direttamente l'array non è possibile.
- Invece, per scrivere una funzione che operi su un array di input è necessario passare:
  - Un puntatore al primo elemento
  - Un numero intero che indichi il numero di elementi dell'array

In ogni caso, è sempre necessario comunicare la lunghezza dell'array a chi ci opera.

Puntatori e array

Passaggio di array a funzioni

Attenzione: il C supporta anche la seguente sintassi nella dichiarazione delle funzioni:

```
int func(int array[42]);
```

che però è obsoleta e perfettamente equivalente a:

```
int func(int *array);
```

Quindi, l'uso di questa sintassi è **fuorviante**:

- Dà l'idea che la dimensione dell'array sia nota, mentre invece viene **ignorata**.
- L'array non viene passato per valore come ci si aspetterebbe per coerenza con variabili di altro tipo.

Puntatori e array

Aritmetica sui puntatori

```
void fill(int *begin, int size, int value) {
  for(int *p = begin; p < begin + size; ++p)
    *p = value;
}
```

L'aritmetica sui puntatori consente di **navigare** all'interno di un array usandoli come cursori da poter spostare a piacimento.

- Aggiungere un numero intero `i` ad un puntatore `p` fa avanzare il puntatore di `i * sizeof T`, dove:
  - `T` è il tipo puntato (es. `int`)
  - `sizeof T` è la **dimensione** in byte del tipo `T` (es. 4 byte)
- Ad esempio, con due puntatori `float *a;` e `double *b;`
  - `a + 1` punta 4 byte (32bit) più avanti rispetto ad `a`.
  - `b + 1` punta 8 byte (64bit) più avanti rispetto ad `b`.

Puntatori e array

Aritmetica sui puntatori

```
void fill(int *begin, int size, int value) {
  for(int *p = begin; p < begin + size; ++p)
    *p = value;
}
```

L'aritmetica sui puntatori consente di **navigare** all'interno di un array usandoli come cursori da poter spostare a piacimento.

- Non a caso, è quanto basta per avanzare di i posizioni all'interno di un array.
- Di conseguenza, `*(a + i)` equivale ad `a[i]`.
- In effetti, la sintassi a parentesi quadre si applica a puntatori:

```
int a[3] = { 42, 42, 42 };
int *pa = a;

printf("%d %d\n", pa[2], *(pa + 2)); // stampa "42 42"
```

Puntatori e array

Avvertenze

In C, il nome di un array può essere usato per nominare il puntatore al primo elemento dell'array stesso.

Attenzione: ciò genera spesso molta confusione, e soprattutto su internet si trovano molte affermazioni non corrette:

- Puntatori e array in C sono la stessa cosa. **Falso**.
- Un array **è** un puntatore. **Falso**.
- Dichiarare un puntatore ha qualcosa a che fare con l'allocazione di memoria. **Falso**.

Esercizi

Esercizi

Per mercoledì 9 novembre

1. Scrivere un programma che legga dallo standard input dei numeri e ne stampi la somma totale.
2. Scrivere le seguenti funzioni che manipolino un array passato come argomento (nei modi visti):
   - Una funzione `reverse()` per invertire l'ordine degli elementi in un array.
   - Una funzione `sort()` per ordinare un array di numeri interi (riutilizzare quanto scritto per l'esercizio 3 della Lezione 8).
   - Scrivere una funzione `qsort()` che implementi un algoritmo di ordinamento in maniera ricorsiva.
3. Scrivete tre programmi che utilizzino le funzioni scritte qui sopra per invertire/ordinare/ecc... una sequenza di numeri interi letti da standard input.