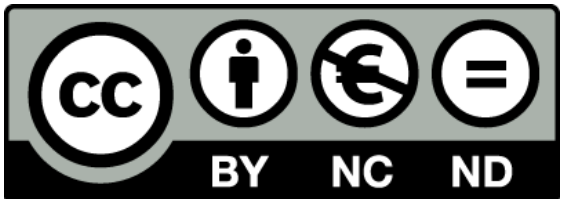
Laboratorio di Programmazione di Sistema

Puntatori in C

Luca Forlizzi

Versione 20.1



Puntatore Generico

- Come sappiamo, per ogni tipo di dato **T** di C Standard, è possibile costruire un puntatore ad oggetti di tipo **T**, che può ovviamente riferirsi a variabili di tipo **T**
- A volte, però, è utile poter memorizzare in una variabile un puntatore *generico*, che contiene un indirizzo di memoria ma non specifica il tipo degli oggetti puntati
- È possibile dichiarare un puntatore generico `p` con la dichiarazione `void *p;`
- Per via della forma della dichiarazione, i puntatori generici vengono, informalmente detti "puntatori a void"
- Applicare l'operatore di indirezione ad un puntatore generico, causa un undefined behavior
- In future presentazioni mostreremo alcune applicazioni dei puntatori generici

Uguaglianza tra Puntatori

- È possibile applicare gli operatori == e != ad una coppia di puntatori dello stesso tipo
- Se due puntatori puntano lo stesso oggetto, essi risultano uguali

```
int x = 0, *p, *q;
p = &x; q = &x;
if (p != q) x += 100;
if (p == q) x += 666;
printf("%d", x); /* stampa 666 */
```

- Di solito è vero anche il viceversa, ma c'è un'eccezione, che verrà discussa in una futura presentazione

Null Pointer

- Ogni tipo puntatore contiene un valore speciale, detto *null pointer*
- Il null pointer non va confuso con un puntatore a void: il primo è un valore, il secondo un tipo
- Il null pointer viene usato per indicare che una variabile di tipo puntatore non contiene un riferimento ad un oggetto
- Infatti, se confrontato con un puntatore ad un oggetto, il null pointer risulta diverso (mentre non è garantito che un puntatore non inizializzato risulti diverso da l'indirizzo di un oggetto)

Null Pointer

- Applicare l'operatore di indirezione al null pointer, causa un undefined behavior
- Il null pointer può essere rappresentato nel codice sorgente come
  - la costante `0`
  - l'espressione `(void *)0`
  - la macro `NULL` definita nell'header `<stddef.h>`

Puntatori e Tipi Aggregati

- Come mostrato nei capitoli 11 e 16 di **[Ki]** è possibile referenziare mediante un puntatore anche un elemento di un array o un membro di una struttura, purché abbiano il tipo corretto

```
float *p1, *p2, *p0, b[3] = { 4.1f, 1.2f };
struct { int m1; float m2 } s = { 1, 2.5f };

p1 = &b[1];    /* p1 punta b[1] */
p2 = &b[2];    /* p2 punta b[2] */
p0 = &s.m2;    /* p0 punta s.m2 */
*p2 = b[0] + *p1 + *p0;
printf("%f", b[2]); /* stampa 7.8 */
```

Puntatori e Strutture

- È possibile definire puntatori a tipi struttura

```
struct type1 { int m1; long m2 };
struct type1 s1 = { -1, 100000 }, s2, *ps;

ps = &s1;    /* ps punta s1 */
s2 = *ps;  /* copia *ps (ovvero s1) in s2 */
```

Puntatori e Strutture

- Attraverso un puntatore a una struttura `ps`, si può accedere ai membri della struttura utilizzando due operatori in sequenza
  - Applicando `*` a `ps` si accede alla struttura puntata
  - Applicando `.` a `*ps` si accede ai membri della struttura
- Ovviamente, è necessario applicare prima `*` e poi `.`

Puntatori e Strutture

- Si osservi che * è un operatore prefisso e . è un operatore postfisso
- In base alle regole sintattiche di C Standard, gli operatori postfissi hanno sempre precedenza maggiore rispetto ai prefissi
- Quindi per accedere ad un membro di una struttura attraverso un puntatore è necessario scrivere delle parentesi tonde attorno all'operatore * applicato al puntatore: ciò non è molto comodo e il codice ha un aspetto poco elegante

```
struct type1 { int m1; long m2 };
struct type1 s = { 0, -800 }, *ps = &s;
long v;
v = (*ps).m2;  /* copia s.m2 in v */
```

Puntatori e Strutture

- Per rendere più comodo da leggere e da scrivere il codice che usa un puntatore per accedere ai membri di una struttura, C Standard definisce un operatore apposito, indicato da una coppia di simboli `->` e talvolta chiamato informalmente "operatore freccia"
- Se `p` è un puntatore a una struttura che ha un membro `m`, l'operazione `p -> m` è semanticamente equivalente a `(*p).m`

```
struct type1 { int m1; long m2 };
struct type1 s = { 0, -800 }, *ps = &s;
long v;
v = ps -> m2;  /* copia s.m2 in v */
```

Puntatori e Strutture

- Lavorando su strutture attraverso puntatori, si ottiene una semantica simile a quella di Java

```
struct type1 { int m1; long m2 };
struct type1 s = { 1, 100000 }, *ps1, *ps2;

ps1 = &s;    /* ps1 punta s */
(*ps1).m1 = 2;
printf("%d", s.m1 ); /* stampa 2 */
ps2 = ps1;
ps2 -> m1 += 2;
printf("%d", ps1 -> m1 ); /* stampa 4 */
printf("%d", s.m1 ); /* stampa 4 */
```