Programmazione I

Il Linguaggio C

I puntatori

Daniel Riccio

Università di Napoli, Federico II

12 novembre 2021

Sommario

- Argomenti
  - I puntatori
  - Aritmetica dei puntatori
  - Assegnamento dei puntatori
  - Allocazione e deallocazione di memoria

Utilizzo dei puntatori

Sia `p`, che `*p` sono L-value modificabili, ossia sono un "qualcosa"
(variabile, elemento di vettore, ...) a cui si può assegnare un valore

L'operatore '`*`' ha priorità superiore a quella degli operatori matematici

```
x = 6 * *p;
```

equivale a:

```
x = 6 * (*p);
```

Per visualizzare il valore di un puntatore si può utilizzare la direttiva `%p` in una `printf`

Tipi e puntatori

L'informazione relativa al tipo è necessaria per permettere ai puntatori di conoscere la dimensione dell'oggetto puntato (usata nell'aritmetica dei puntatori)

Poiché oggetti di tipo diverso possono avere dimensione diversa, l'assegnazione tra puntatori di tipo diverso è in genere errata e il compilatore genera un warning

```
int *p, x=12;
long *q, y=26;
```

```
p = &x;      OK!
q = &y;      OK!
q = p;       NO! Warning
q = &x;      NO! Warning
```

Puntatori a void

Sono puntatori generici e non possono essere dereferenziati (non si può scrivere `*p`), possono essere utilizzati solo come contenitori temporanei di valori di tipo puntatore (a qualsiasi tipo)

```
void *h;
```

Non serve il cast `(void *)` per copiare un puntatore non-`void` in un puntatore `void`

`h = p;` (supponendo ad esempio `int *p`)

Qualsiasi tipo di puntatore può essere confrontato con un puntatore a `void`

`NULL` è definito come: `(void *)0`

Puntatori a void

Per dereferenziare il valore di un puntatore a `void` è necessario assegnarlo ad un puntatore al tipo appropriato (non `void`) per poter conoscere la dimensione dell'oggetto puntato

Può essere necessario il cast `(tipo *)` per copiare un puntatore `void` in un puntatore non-`void` (i compilatori C non lo richiedono, i compilatori C++ sì).

In riferimento all'esempio precedente:

```
int *q;
```

`q = h;`	OK, compilatore C
`q = (int *)h;`	OK, compilatore C++
`*q = 23;`	`x` ora contiene **23**

Nota: precedentemente sono state eseguite le assegnazioni

```
p = &x;
h = p;
```

Puntatori e vettori

Il nome (senza parentesi) di un vettore-di-**T** è un valore costante di tipo puntatore-a-**T**, corrisponde all'indirizzo di memoria del primo elemento di vettore

```
int vett[100];
int *p;
p = vett;
```

l'indirizzo di memoria di **`vett`** viene messo in **`p`**, equivale a scrivere:

```
p = &vett[0]
```

(le parentesi hanno priorità maggiore di **&**)

Puntatori e vettori

Attenzione:

```
vett = p;
```
NO!

Non si può assegnare un valore a **`vett`** in quanto NON è una variabile puntatore, ma un “sinonimo” di un indirizzo di memoria

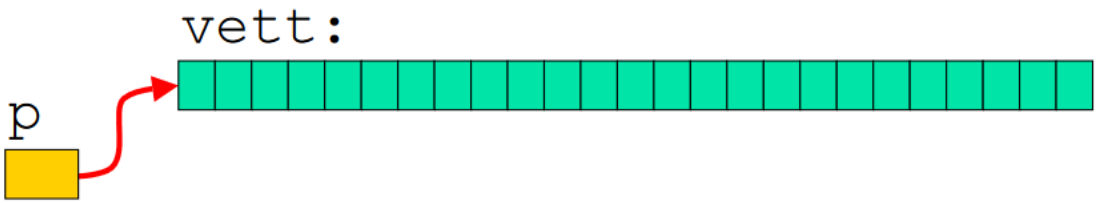
Gli indirizzi di memoria sono valori costanti stabiliti dal compilatore, non sono variabili e quindi non hanno uno spazio in memoria modificabile per contenere un valore

Il termine “puntatore” viene comunemente (e impropriamente) usato al posto di “indirizzo di memoria” (es. “**`&a`** dà il puntatore ad **`a`**”)

Puntatori e vettori

Una variabile di tipo puntatore-a-T, assegnata in modo che punti a (cioè contenga l'indirizzo di) un oggetto di tipo vettore-di-T, può essere utilizzata come se fosse un vettore-di-T

```
int vett[25];
int *p = vett;
```



Ad esempio, qui **`p`**`[3]` equivale a **`vett`**`[3]`

Il compilatore internamente trasforma le espressioni con notazione vettoriale `[]` in espressioni con i puntatori

Puntatori e vettori

Una variabile di tipo puntatore-a-T non “sa” se il valore (scalare) a cui punta è singolo o è l’elemento di un vettore, lo sa solo il programmatore e sta a questi utilizzarlo in modo coerente

```
int x = 10, vett[10], *p, *q;
p = &x;
p++;
q = p+1;
p = vett;
p++;
q = p+1;
```

NO! Non esiste l’oggetto puntato da `p+1`

NO! Idem

SI’! Ora `p` punta a `vett[1]`

SI’! Ora `q` punta a `vett[2]`

Priorità dell'operatore *

Dalla tabella delle priorità si vede che l'operatore di deriferimento ***** ha priorità quasi massima, inferiore solo alle parentesi (e a '`->`' e a '`.`'), e associatività da destra a sinistra

Quindi, considerando che gli operatori ***** e **++** hanno stessa priorità e associatività da D a S:

`*p++`	equivale a	`*(p++)`	incrementa `p`
`*++p`	equivale a	`*(++p)`	incrementa `p`
`++*p`	equivale a	`++(*p)`	incrementa `*p`

inoltre:

`(*p)++` incrementa `*p`

`*p+1` equivale a `(*p)+1` e non a `*(p+1)`

Copia di stringhe - 1

La stringa `y` viene copiata in `x`

```
char x[30], y[30], *t=x, *s=y;
int i=0;

gets(y);

while (s[i] != '\0'){
    t[i] = s[i];
    i++;
}

t[i] = '\0';
printf("%s\n", x);
```

Il `'\0'` viene copiato fuori dal ciclo

Qui **`s`** e **`t`** vengono inutilmente usati come semplici sinonimi di `x` e `y`, non come puntatori

Copia di stringhe - 2

La stringa `y` viene copiata in `x`

```
char x[30], y[30], *t=x, *s=y;
int i=0;

gets(y);

while ((t[i] = s[i]) != '\0')
    i++;

printf("%s\n", x);
```

Il '\0' viene copiato nel ciclo stesso

Qui **s** e **t** vengono inutilmente usati come semplici sinonimi di x e y, non come puntatori

Copia di stringhe - 3

La stringa `y` viene copiata in `x`

```
char x[30], y[30], *t=x, *s=y;

gets(y);

while ( (*t = *s) != '\0'){
    t++;
    s++;
}

printf("%s\n", x);
```

Il '\0' viene copiato nel ciclo stesso

Nota: **!=** '\0' può essere omesso

Copia di stringhe - 4

La stringa `y` viene copiata in `x`

```
char x[30], y[30], *t=x, *s=y;

gets(y);

while (*t++ = *s++)
;

printf("%s\n", x);
```

Il '\0' viene copiato nel ciclo stesso

Nota: 'istruzione nulla è più chiara se scritta in una riga a sé stante

Puntatori e stringhe

Si noti la differenza tra le seguenti definizioni:

`char str[100];` RISERVA spazio per contenere i caratteri, è una variabile e il suo contenuto può essere modificato

`char *s;` NON RISERVA spazio per contenere i caratteri, quindi per essere utilizzata come stringa le si deve assegnare una stringa “vera”:

Assegnazione di una stringa variabile:

```
s = str;
scanf("%s", s);
```

SI’

Assegnazione di una stringa costante:

```
s = "salve";
scanf("%s", s);
```

NO

Puntatori e stringhe

Si considerino i seguenti esempi:

`char str[] = "ciao";` È l'inizializzazione di una variabile stringa:

il compilatore riserva memoria per `str` e vi copia i caratteri di `"ciao"`;
la stringa costante `"ciao"` non esiste in memoria: è stata usata dal compilatore per inizializzare la stringa `str`, ma esiste in memoria la stringa variabile `"ciao"`

`str[0]='m';` **SI'**

`char *s = "hello";` È l'inizializzazione di una variabile puntatore:

il compilatore determina l'indirizzo della stringa costante `"hello"` (che esiste in memoria) e lo assegna alla variabile puntatore

`s[0]='b';` **NO!**

`"hello"` è costante!

Puntatori e stringhe

Si considerino i seguenti esempi (cont.):

`s = "salve";` È l'assegnazione ad una variabile puntatore:
il compilatore determina l'indirizzo della stringa costante `"salve"` (che esiste in memoria) e lo assegna alla variabile puntatore

`s[4]='o';` **NO!**
`"salve"` è costante!

`s = str;` È l'assegnazione ad una variabile puntatore:
il compilatore determina l'indirizzo della stringa variabile `str` (che esiste in memoria) e lo assegna alla variabile puntatore

`s[0]='m';` **SI'**

Puntatore variabile a valore costante

```
int const *p;
const int *p;
```

Sono equivalenti

`p` è una variabile di tipo puntatore-a-costante
(a un oggetto costante di tipo `int`)

```
const int x, y;
const int *p;
```

puntatore-a-costante

```
p = &x;
```
SI', `p` è una variabile

```
p = &y;
```
SI', `p` è una variabile

```
*p = 13;
```
NO, `*p` è costante

Puntatore variabile a valore costante

L'assegnazione di un valore di tipo puntatore-a-costante (l'indirizzo di un valore costante) ad una variabile di tipo puntatore-a-variabile genera un Warning del compilatore perché permette di by-passare la restrizione (`const`)

```
const int x = 12;
int y = 10;
int *p;
const int *q;
p = &x;
*p = 5;
q = &x;
*q = 5;
```

→ puntatore-a-variabile
→ puntatore-a-costante
→ **Warning**
→ non dà errore
→ **OK**
→ dà errore

Puntatore costante a valore variabile

```
int * const p;
```

`p` è una costante di tipo puntatore-a-variabile
(a un oggetto variabile di tipo `int`)

Le costanti possono essere solo inizializzate

```
int x, y;
int * const p = &x;
*p = 13;
*p
p = &y;
p
```

inizializzazione
SI',
è una variabile
NO,
è una costante