Programmazione I

Il Linguaggio C

I puntatori

Daniel Riccio

Università di Napoli, Federico II

10 novembre 2021

Sommario

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

Puntatori - intro

Un tipo variabile che contiene un indirizzo di una locazione di memoria: l'indirizzo di un'altra variabile!

```
int* pippo;
```

pippo è una variabile di tipo `int*`,
cioè di tipo **puntatore ad intero**

o anche:

```
int *pippo;
```

***pippo**
(cioè il valore puntato da pippo)
è una **espressione di tipo intero**

Puntatori - sintassi

Entrambi possono essere sia letti che assegnati
(possono comparire da entrambi i lati di un assegnamento)

`pippo` il valore del **puntatore**.

`*pippo` il valore dell'**oggetto puntato**.

Puntatori - preambolo

Cosa succede normalmente…

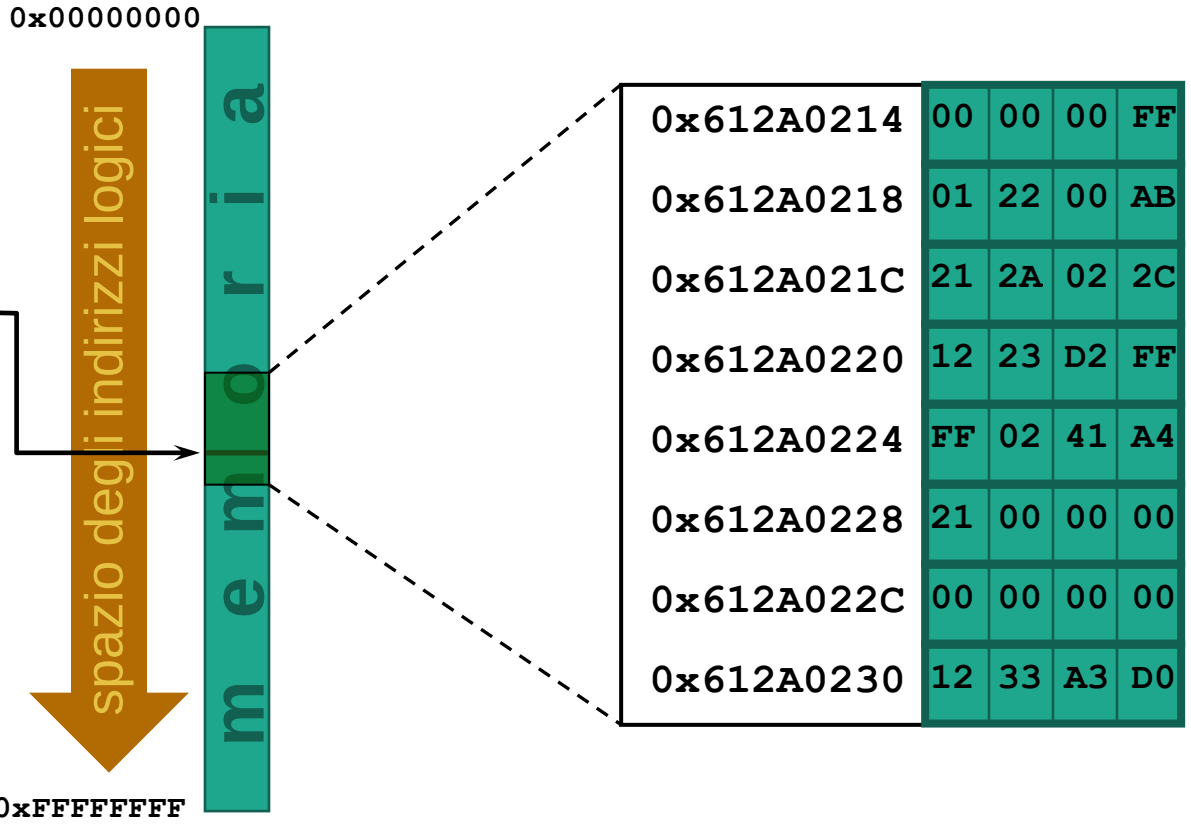
```
int pippo;
```

il compilatore assegna alla variabile pippo una locazione di memoria.

Ad esempio, la locazione **0x612A22C**

Inoltre, riserva quei quattro byte per la variabile pippo.

variabile	tipo	locazione
pippo	int	0x612A22C



Puntatori - significato

```
int* pippo;
```

variabile	tipo	locazione
pippo	int*	0x612A22C

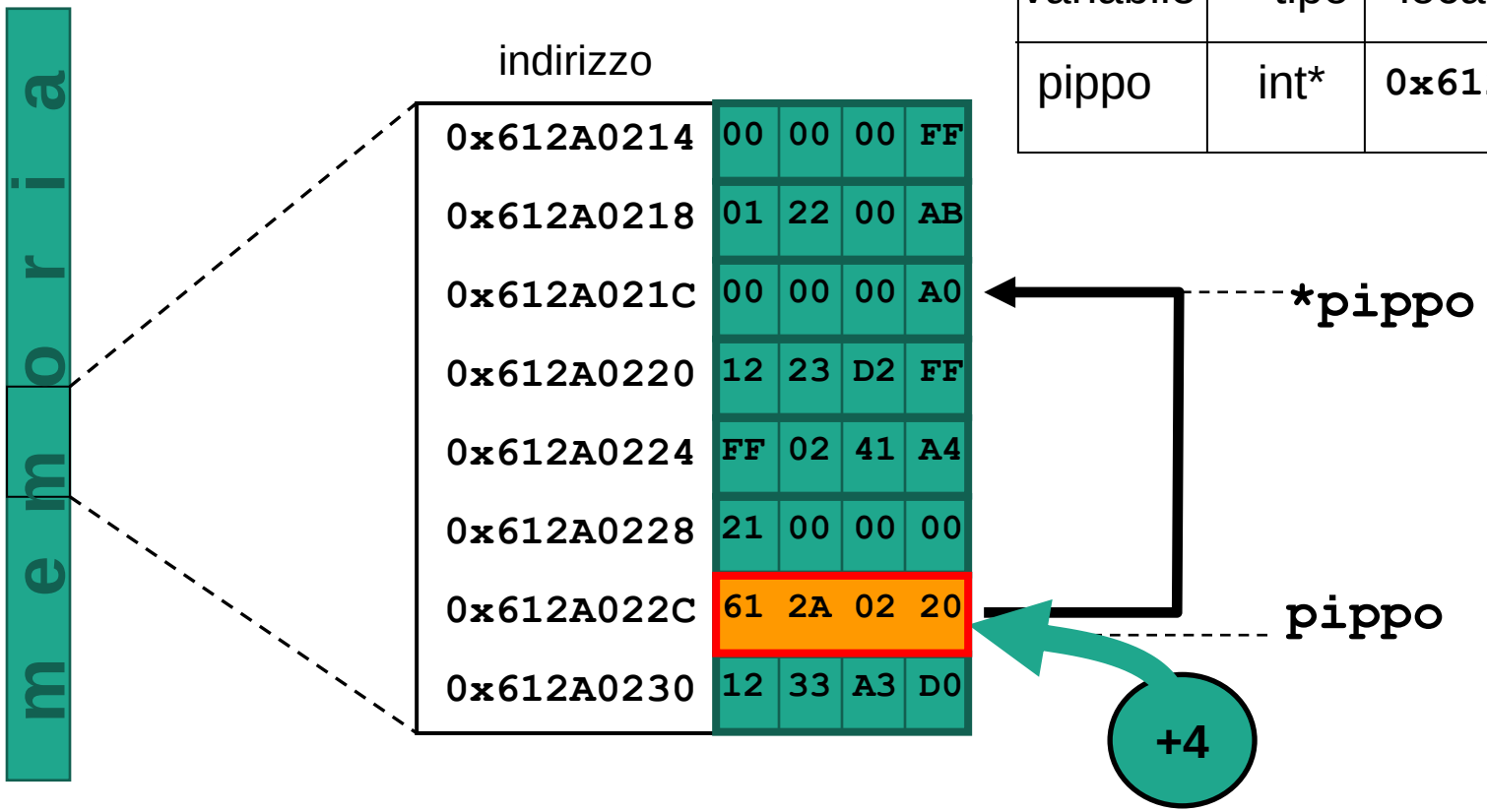
pippo (il **puntatore** stesso) vale... 0x612A021C

***pippo** (la variabile **puntata** da **pippo**) vale... 0x000000A0

Cambiare il valore del puntatore

```
pippo++;
```

variabile	tipo	locazione
pippo	int*	0x612A22C



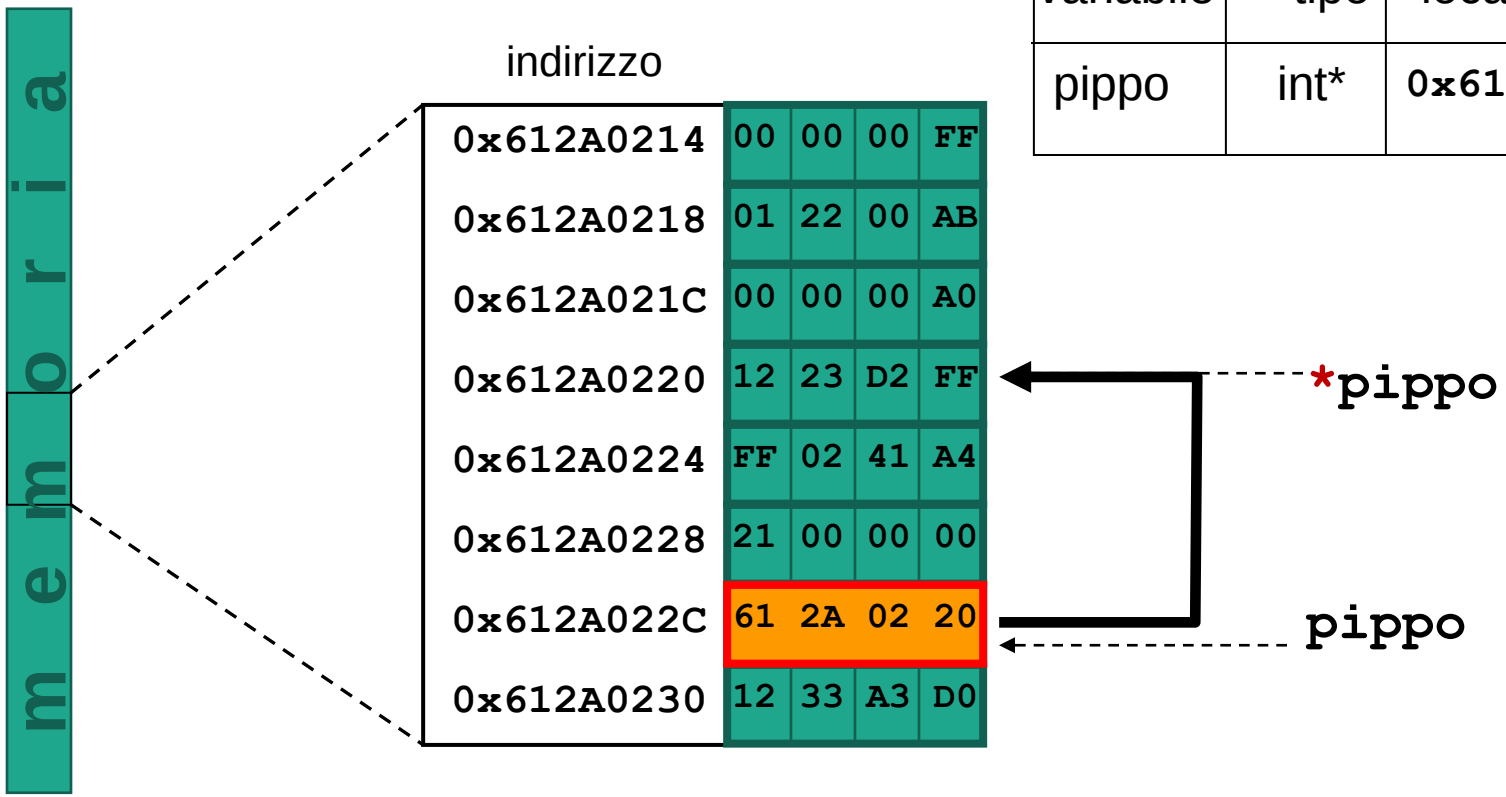
pippo (il **puntatore** stesso) vale... 0x612A021C

***pippo** (la **variabile puntata** da pippo) vale... 0x000000A0

Cambiare il valore del puntatore

```
pippo++;
```

variabile	tipo	locazione
pippo	int*	0x612A22C



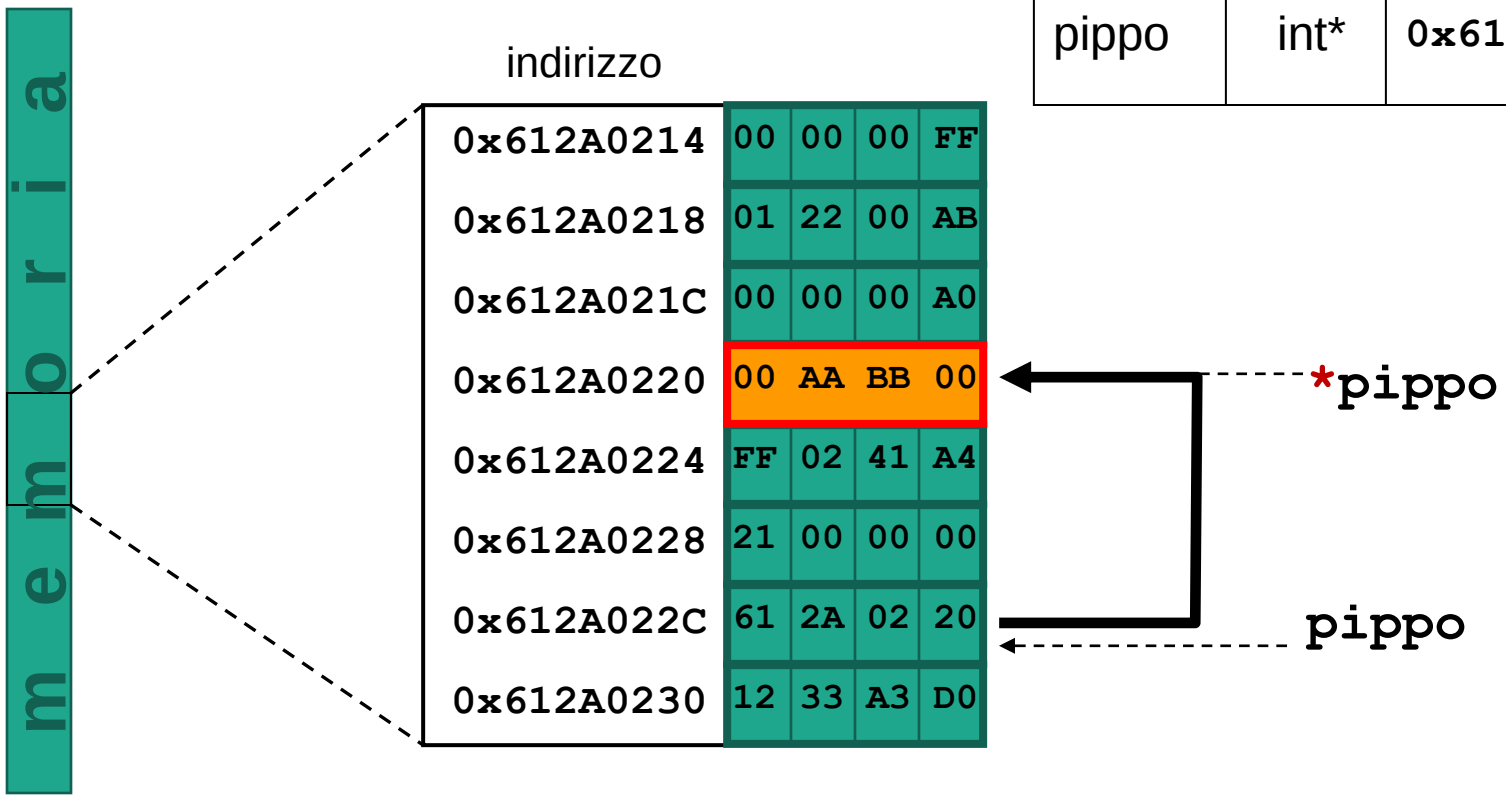
pippo (il **puntatore** stesso) vale... ~~0x612A021C~~ 0x612A0220

***pippo** (la **variabile puntata** da pippo) vale... ~~0x000000A0~~ 0x1223D2FF

Cambiare il valore puntato

```
*pippo = 0x00AABB00;
```

variabile	tipo	locazione
pippo	int*	0x612A22C



Considerazione sull'efficienza

```
const int I=10;
int i;
int* ip;

int x;
...
```

`I` è una costante intera (vale 10)

`i` è una variabile intera

`ip` è un puntatore ad un intero

tabella dei Simboli del compilatore

ide.	tipo	locazione o valore	
`I`	int	`---`	`10`
`i`	int	`0xAA000000`	`---`
`ip`	int*	`0xBB000000`	`---`
`x`	int	`0xCC000000`	`---`

`x = I;` → compilazione

```
STORE   10      0xCC000000
```

`x = i;` → compilazione

```
READ    TEMP    0xAA000000
STORE   TEMP    0xCC000000
```

`x = *ip;` → compilazione

```
READ    TEMP0   0xBB000000
READ    TEMP1   TEMP0
STORE   TEMP1   0xCC000000
```

Considerazioni sull'efficienza

```
const int I=10;
int i;
int* ip;

int x;
...
```

`I` è una costante intera (vale 10)

`i` è una variabile intera

`ip` è un puntatore ad un intero

tabella dei Simboli del compilatore

ide.	*tipo*	*locazione o valore*	
`I`	int	`---`	`10`
`i`	int	`0xAA000000`	`---`
`ip`	int*	`0xBB000000`	`---`
`x`	int	`0xCC000000`	`---`

(le costanti si assegnano solo durante l'inizializzazione)

```
I = 15;
```

```
i = 15;
```

compilazione

```
STORE   15   0xAA000000
```

```
*ip = 15;
```

compilazione

```
READ    TEMP   0xBB000000
STORE   15     TEMP
```

Considerazioni sull'efficienza

comando	accessi alla memoria in scrittura	accessi alla memoria in lettura
`a = 15;`	1	0
`a = b;`	1	1
`a = *p;`	1	2
`*p = 15;`	1	1
`*p = b;`	1	2
`*p = *p2;`	1	3

Aritmetica dei puntatori

L'operazione base sui puntatori:

somma con un intero

```
<puntatore ad un tipo T> + <intero>
```

espressione di tipo **puntatore** ad un tipo **T** (**T***)

Semantica:

```
p + i
```

- è il puntatore che punta ad una locazione **i** elementi (di tipo **T**) dopo **p**;
- come indirizzo di memoria, è l'indirizzo (**p + i**) **× (dimensione di T)**

Aritmetica dei puntatori

Esempi:

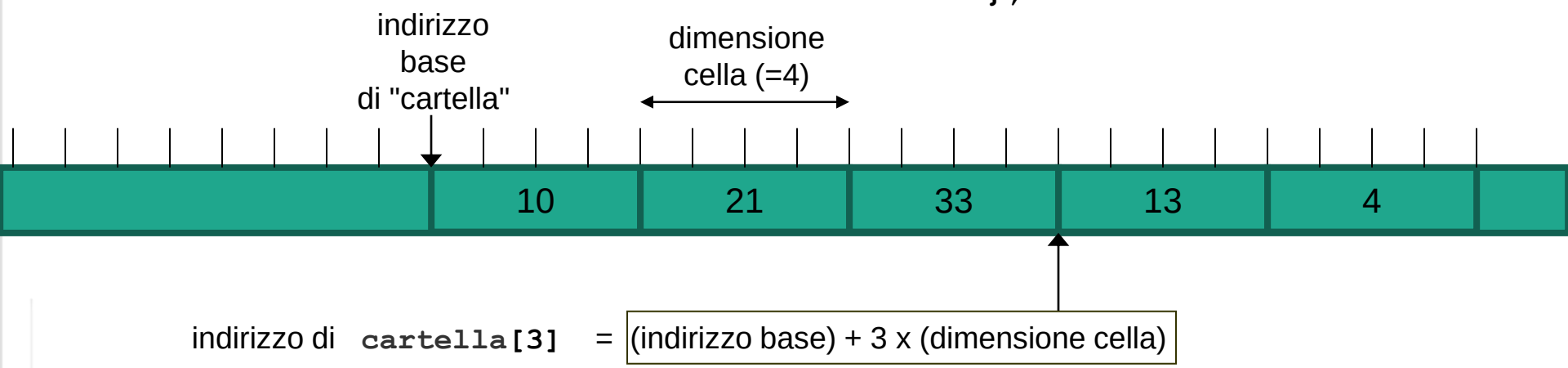
```
double *p, *q;
...
q = p + 3;
*(p + 3)=2.0;
q++;
q--;
q+=2;
```

Aritmetica dei puntatori

Richiamo sui vettori

- in memoria, gli elementi di un array sono memorizzati in una serie di celle contigue
- ogni cella ha la stessa grandezza
- per questo gli array sono strutture ad accesso casuale

```
int cartella[5]={
 10,21,33,13,4,
};
```



indirizzo di `cartella[3]` = (indirizzo base) + 3 x (dimensione cella)

Aritmetica dei puntatori

```
double *p;
...
```

L'accesso

```
p[ 5 ]
```

equivalente a

```
*(p + 5)
```

Puntatori e record

```
typedef struct {
  char nome[24];
  char cognome[24];
  int peso;
} Persona;

Persona *p;

...
```

come accedere al campo **peso** della Persona puntata da **p**?

~~`*p.peso`~~ → interpretato come → `*(p.peso)`

```
(*p).peso
```

o, equivalentemente, con l'apposito operatore "freccia":

```
p->peso
```

Puntatori e record

```
typedef struct  {
  char nome[24];
  char cognome[24];
  int peso;
} Persona;
```

```
Persona p;
  ...
  if (p.peso == ... )
  ...
```

```
typedef struct  {
  char nome[24];
  char cognome[24];
  int peso;
} Persona;
```

```
Persona* p
  ...
  if (p->peso == ... )
  ...
```

Assegnare i puntatori

In memoria, un puntatore è un indirizzo di memoria

(...di una variabile)

(...di cui è noto il tipo)

quale indirizzo?

- **Modo 1**: prendere l'indirizzo di una variabile esistente
  - il puntatore punterà a quella variabile
- Modo 2: allocare (riservare, prenotare) della memoria libera
  - il puntatore punterà ad una nuova variabile, memorizzata nella memoria così riservata
  - la nuova variabile è allocata dinamicamente

Assegnare i puntatori

Modo 1: prendere l'indirizzo di una variabile esistente
il puntatore punterà a quella variabile

Operatore **ampersand** (**&**)

In latino – le lettere **e** e **t** venivano occasionalmente scritte insieme, la sua invenzione è attribuita a **Marco Tullio Tirone**

Esempio:

```
double d = 9.0;
double *p;

p = &d;
printf("%f",*p);
*p = 21.5;
printf("%f",d);
```

il puntatore `p` punta all'indirizzo di memoria dove vive la variabile `d`

scrivi il valore di `*p`

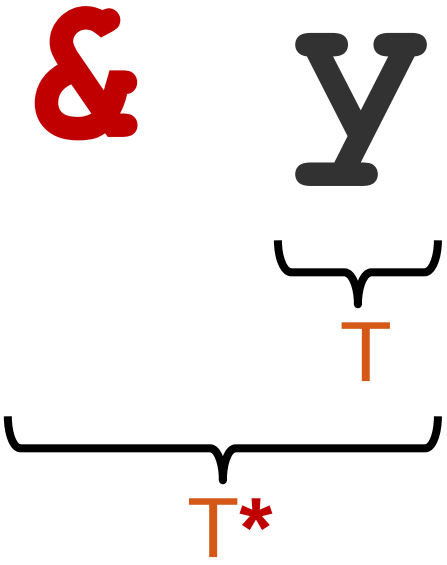
Cosa scrive?

scrivi il valore di `d`

Cosa scrive?

Operatore & e tipi

se **y** è una var di tipo **T**...



Operatore & e vettori

Errore `numeri` non è di tipo int

(e quindi `&numeri` non è di tipo int*)

```
int numeri[]={10,20,30,40};
int *punt;

punt = &numeri;
```

scrivere invece:

```
punt = &(numeri[0]);
```

oppure (un'altra scorciatoia sintattica):

```
punt = numeri;
```

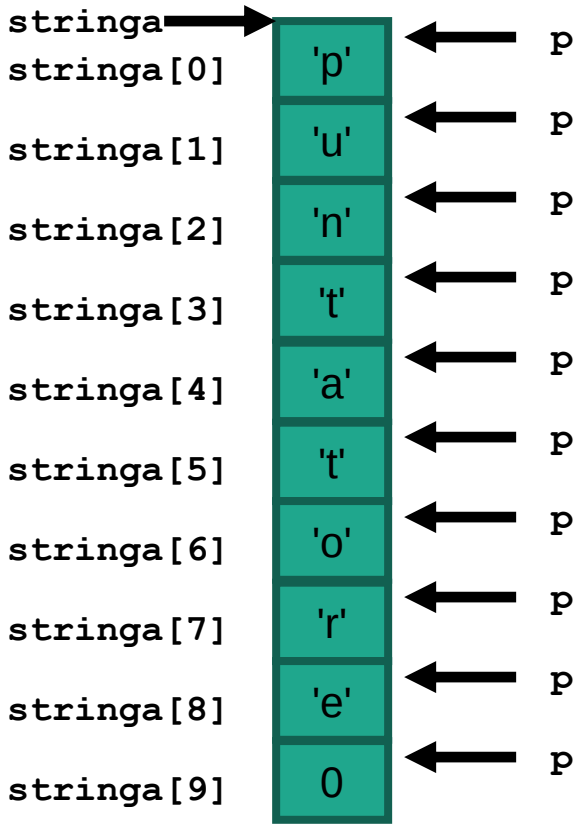
scriviamo tutti i 4 numeri:

```
int i;
for(i=0; i<4; i++) {
  printf("%d ", numeri[i]);
}
```

usando i puntatori:

```
int i;
for(i=0; i<4; i++) {
  printf("%d ", *(punt++));
}
```

Esempio




```
char stringa[]="puntatore\0";
int i;

while (stringa[i]) {
  stringa[i] = maiuscolo(stringa[i]);
  i++;
}
```

```
char stringa[]="Puntatore\0";
char *p = stringa;

while (*p) {
  *p = maiuscolo(*p);
  p++;
}
```