Laboratorio di Programmazione

Lezione 7 – Puntatori I

Marco Anisetti (teoria)
Dipartimento di Informatica
marco.anisetti@unimi.it

Matteo Luperto (lab. turno A)
Dipartimento di Informatica
matteo.luperto@unimi.it

Nicola Bena (lab. turno B)
Dipartimento di Informatica
nicola.bena@unimi.it

Funzione `read_int`

```
int read_int(){
    int n;
    do {
       printf("Inserisci numero > 0: ");
       scanf("%d", &n);
    } while (n <= 0);
    return n;
}
```

Funzione `read_int`

```c
#include <stdio.h>

int read_int();

int main(){
	int n1, n2, sum;
	n1 = read_int();
	n2 = read_int();
	sum = n1 + n2;
	printf("La somma e': %d\n", sum);
	return 0;
}
```

```c
int read_int(){
	int n;
	do {
		printf("Inserisci numero > 0: ");
		scanf("%d", &n);
	} while (n <= 0);
	return n;
}
```

Funzione `read_int`

```c
#include <stdio.h>
int read_int();
int main(){
    int n1, n2, sum;
    n1 = read_int();
    n2 = read_int();
    sum = n1 + n2;
    printf("La somma e': %d\n", sum);
    return 0;
}
```

```c
int read_int(){
    int n;
    do {
        printf("Inserisci numero > 0: ");
        scanf("%d", &n);
    } while (n <= 0);
    return n;
}
```

- L'implementazione della funzione è *dopo* `main`
- Il prototipo è dichiarato *prima*
- In alternativa, possiamo definire la funzione prima di `main`, senza prototipo

Funzione `read_int`

```c
#include <stdio.h>

int read_int();

int main(){
	int n1, n2, sum;
	n1 = read_int();
	n2 = read_int();
	sum = n1 + n2;
	printf("La somma e': %d\n", sum);
	return 0;
}
```

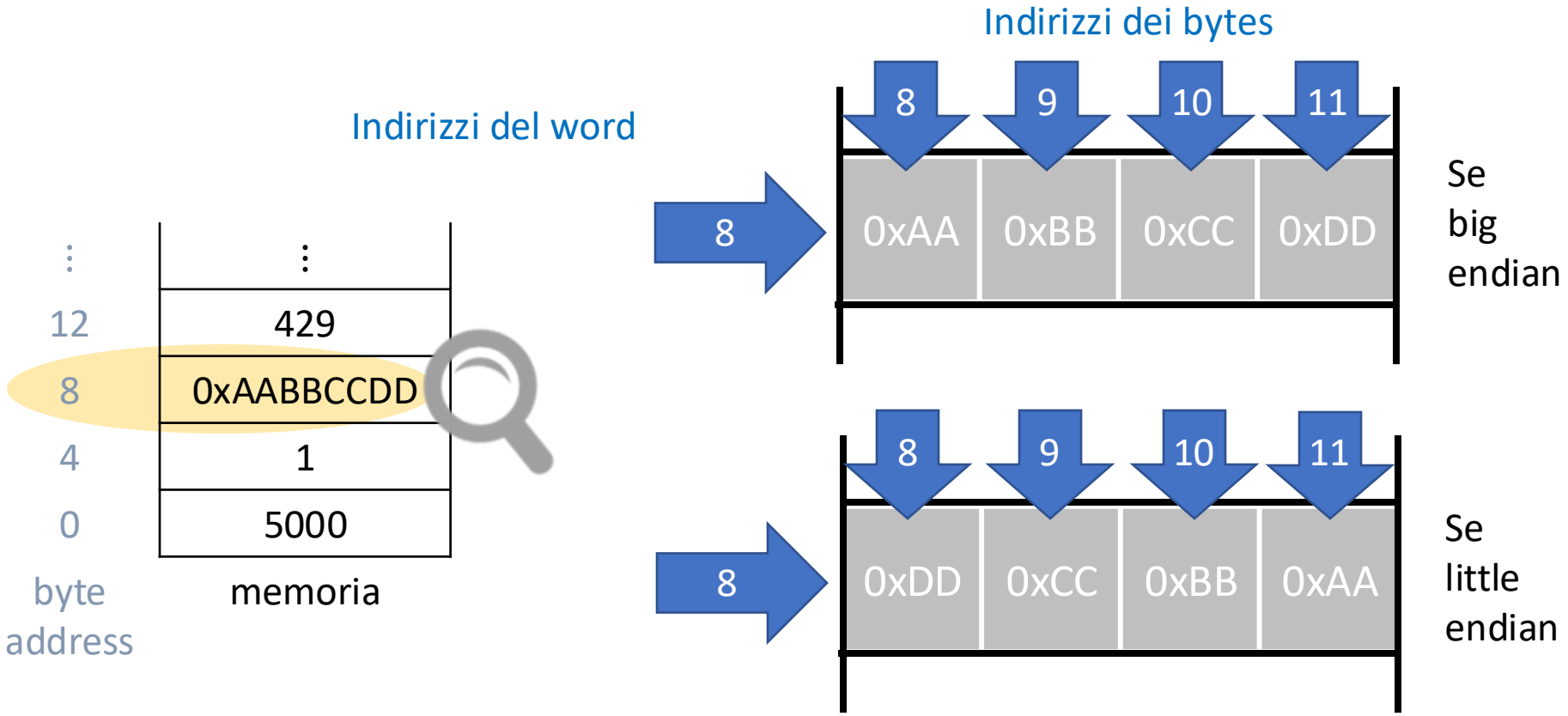
```c
int read_int(){
	int n;
	do {
		printf("Inserisci numero > 0: ");
		scanf("%d", &n);
	} while (n <= 0);
	return n;
}
```

Puntatori

- La memoria di ogni processo è organizzata come un insieme contiguo di celle (tipicamente di 1 byte = 8 bit)
- Più celle formano una *parola (word)*, che dipende dalla architettura della macchina (tipicamente 32 o 64 bit, quindi 4 o 8 celle da 1 byte)
- Ogni variabile occupa un certo numero di byte consecutive sulla base del tipo
  - Funzione `sizeof(<tipo>)` ritorna il numero di byte occupato da una variabile di tipo <tipo> (tipo di ritorno: `size_t`)
- Ogni cella è identificata da un indirizzo numerico (da `0` a `N-1`)
  - Per convenzione viene scritto in esadecimale, ad es., `0x7ffe93f13bb4`
- L'*indirizzo* di una variabile è l'indirizzo della prima cella che occupa

Endianness

L'indirizzo di una parola di memoria è anche l'indirizzo di uno dei 4 byte che compongono quella parola
In questo esempio abbiamo una architettura a 32 bit (4 byte).

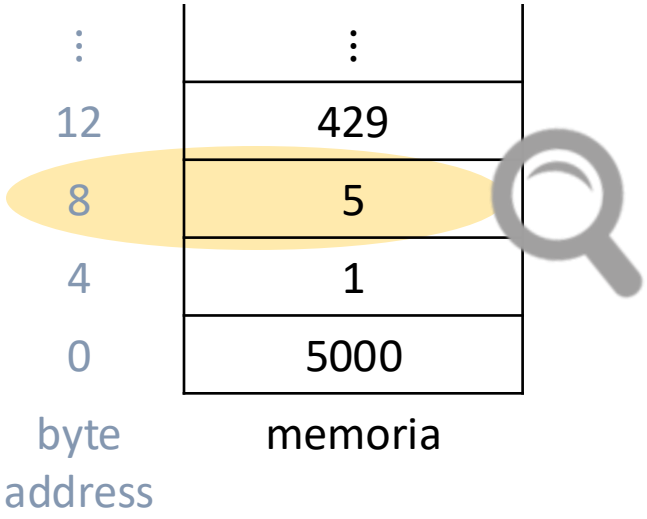


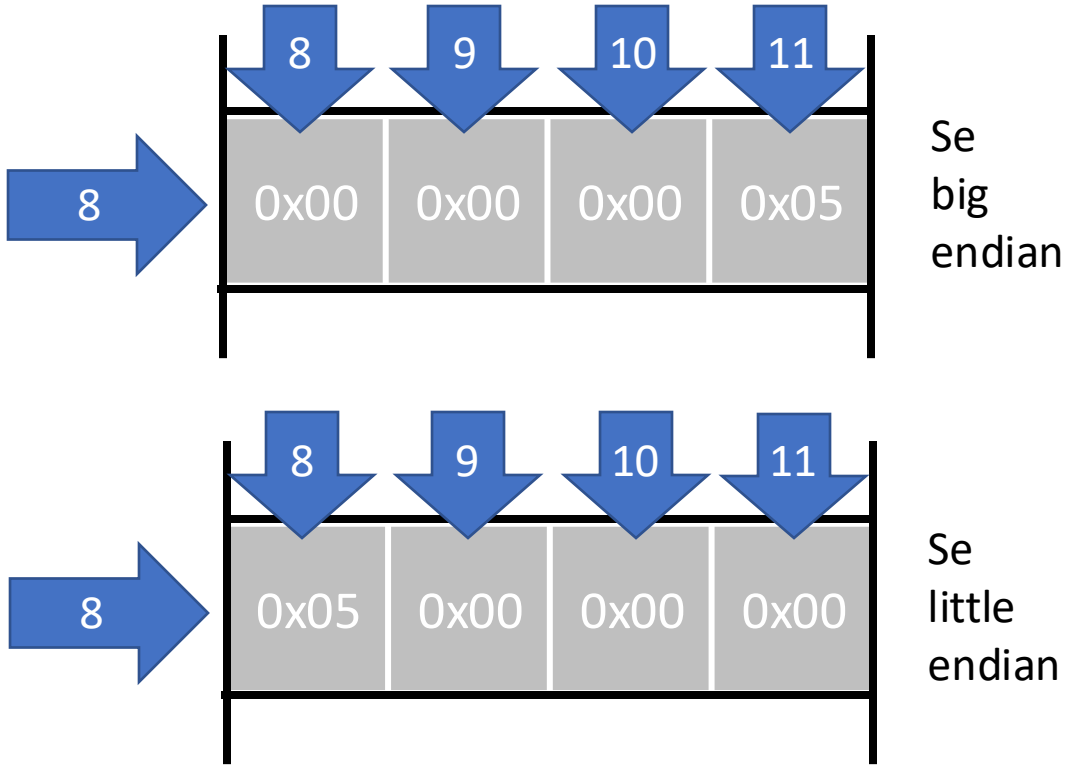
Ma, tra i 4, **quale?** Dipende dall'ordine dei byte: la «**endianness**» dell'architettura

Endianness

L'indirizzo di una parola di memoria è anche l'indirizzo di uno dei 4 byte che compongono quella parola
In questo esempio abbiamo una architettura a 32 bit (4 byte).

```
int x = 5;
sizeof(int) == 4
```





Ma, tra i 4, **quale?** Dipende dall'ordine dei byte: la «**endianness**» dell'architettura

Tipo Puntatore

- *Tipo puntatore*: tipo disponibile in C usato per memorizzare l'indirizzo di un oggetto (variabile, funzione)
- Il tipo puntatore include anche la specifica del tipo dell'oggetto puntato
- Sintassi:

```
<tipo_puntato> * <identificatore>
int * pointer;
```

Tipo Puntatore

- *Tipo puntatore*: tipo disponibile in `C` usato per memorizzare l'indirizzo di un oggetto (variabile, funzione)
- Il tipo puntatore include anche la specifica del tipo dell'oggetto puntato
- Sintassi:

```
<tipo_puntato> * <identificatore>
int * pointer;
```

- La dimensione (=numero di celle occupate) da una variabile di tipo puntatore è sempre lo stesso e indipendente dal tipo puntato
  - Contiene un indirizzo, cioè un numero
  - Dimensione sufficiente a rappresentare lo *spazio di indirizzamento* della macchina

Tipo puntatore

```
sizeof(char)  = 1
sizeof(short) = 2
sizeof(int)   = 4
sizeof(long)  = 8
sizeof(void*) = 8
```

```c
#include <stdio.h>

int main() {
printf("sizeof(char) = %zu\n", sizeof(char));
printf("sizeof(short) = %zu\n", sizeof(short));
printf("sizeof(int) = %zu\n", sizeof(int));
printf("sizeof(long) = %zu\n", sizeof(long));
// dimensione di un puntatore
printf("sizeof(void*) = %zu\n", sizeof(void*)); return 0;
}
```

Qui il puntatore è lungo 8 byte = 64 bit → word di 64 bit, architettura a 64 bit

Nota: posso mettere più variabili dentro la stessa parola (8 `char` in 1 word qui), perché la memoria è indirizzata al byte.

Operatore &

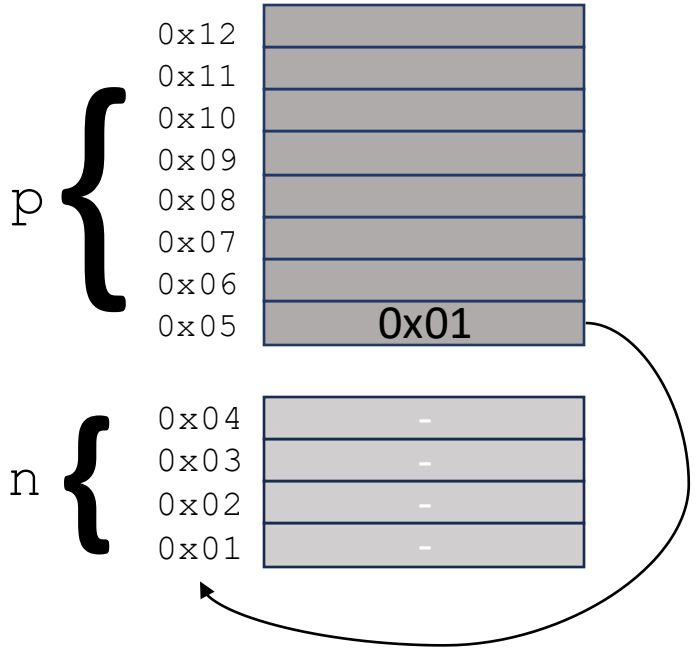
- L'operatore `&` fornisce l'indirizzo di un oggetto (referenziazione)

```
int n;
int *p;
p = &n; // p contiene l'indirizzo
della prima cella occupata da n
```

Operatore &

- L'operatore `&` fornisce l'indirizzo di un oggetto (referenziazione)

```
int n;
int *p;
p = &n;
```



Nota:

```
sizeof(int)   → 4 (32 bit)
sizeof(int*)  → 8 (64 bit)
```

Operatore *

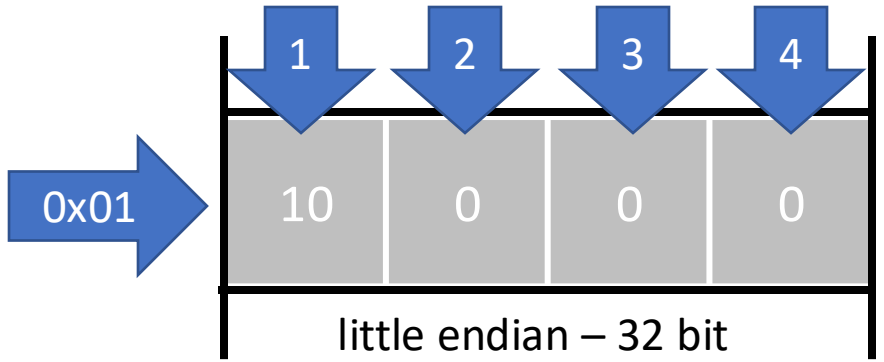
- L'operatore `*` fornisce l'oggetto puntato da un puntatore (deferenziazione)
  - Applicabile solo a variabili di tipo puntatore

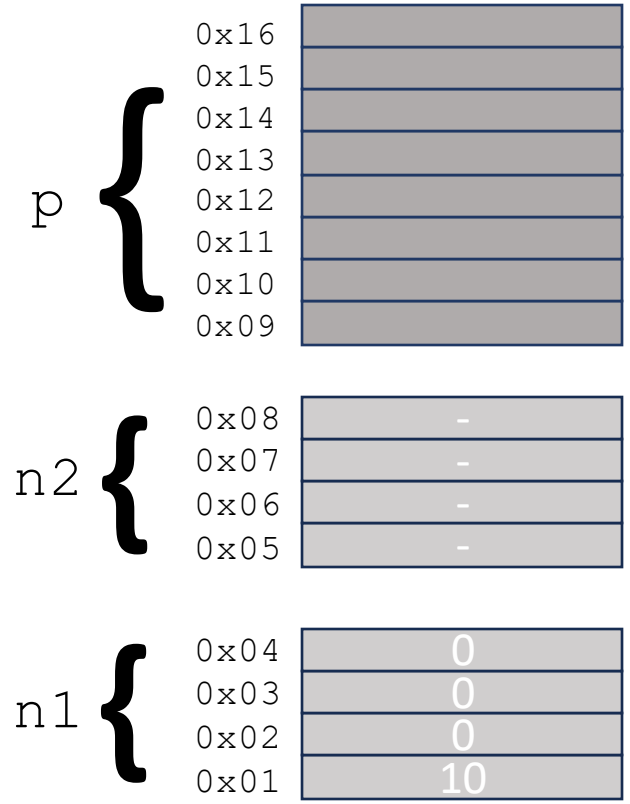
```
int n1 = 10, n2;
int *p;
p = &n1;
n2 = *p;
```

Operatore *

- L'operatore * fornisce l'oggetto puntato da un puntatore (deferenziazione)
  - Applicabile solo a variabili di tipo puntatore

```
int n1 = 10, n2;
int *p;
p = &n1;
n2 = *p;
```

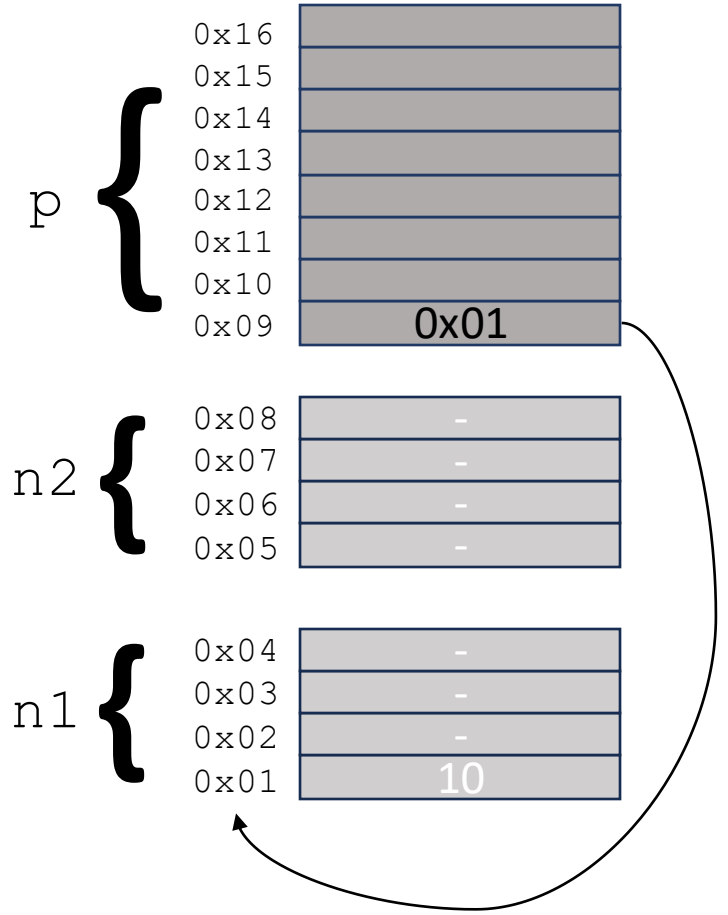




Operatore *

- L'operatore * fornisce l'oggetto puntato da un puntatore (deferenziazione)
  - Applicabile solo a variabili di tipo puntatore

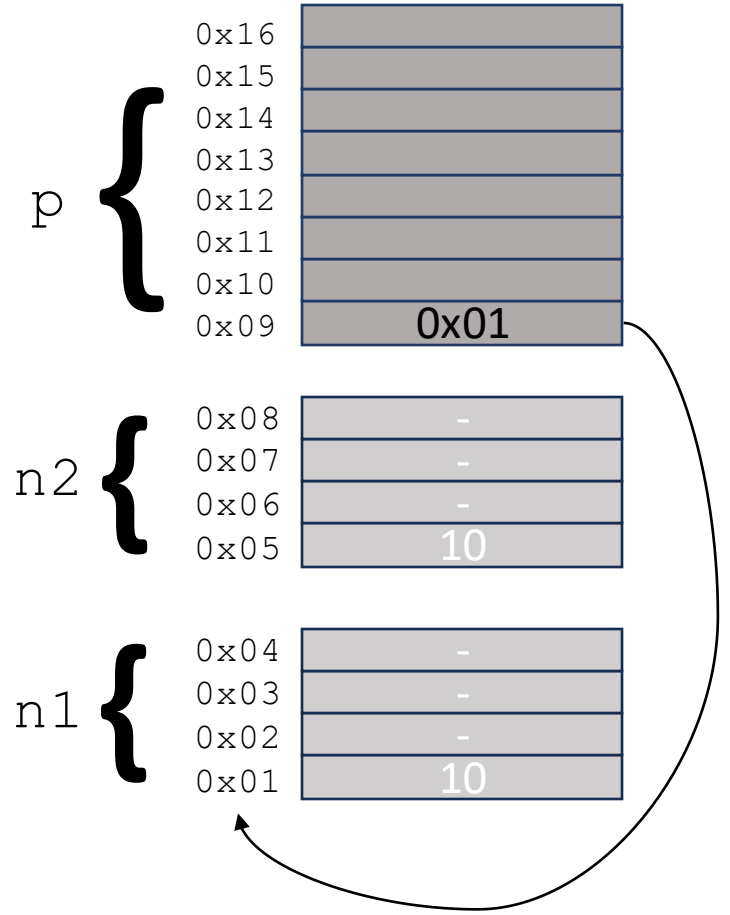
```
int n1 = 10, n2;
int *p;
p = &n1;
n2 = *p;
```



Operatore *

- L'operatore * fornisce l'oggetto puntato da un puntatore (deferenziazione)
  - Applicabile solo a variabili di tipo puntatore

```
int n1 = 10, n2;
int *p;
p = &n1;
n2 = *p;
```



Operatore *

- L'operatore `*` fornisce l'oggetto puntato da un puntatore (deferenziazione)
  - Applicabile solo a variabili di tipo puntatore
- Attenzione
  - Il puntatore dev'essere inizializzato

```
int n1, n2;
int *p;
n2 = *p;
```

Undefined behavior

Operatore *

- L'operatore `*` fornisce l'oggetto puntato da un puntatore (deferenziazione)
  - Applicabile solo a variabili di tipo puntatore
- Attenzione
  - L'indirizzo indicato dal puntatore dev'essere valido

```
int * p = (int *)10;
int n2 = *p;
```

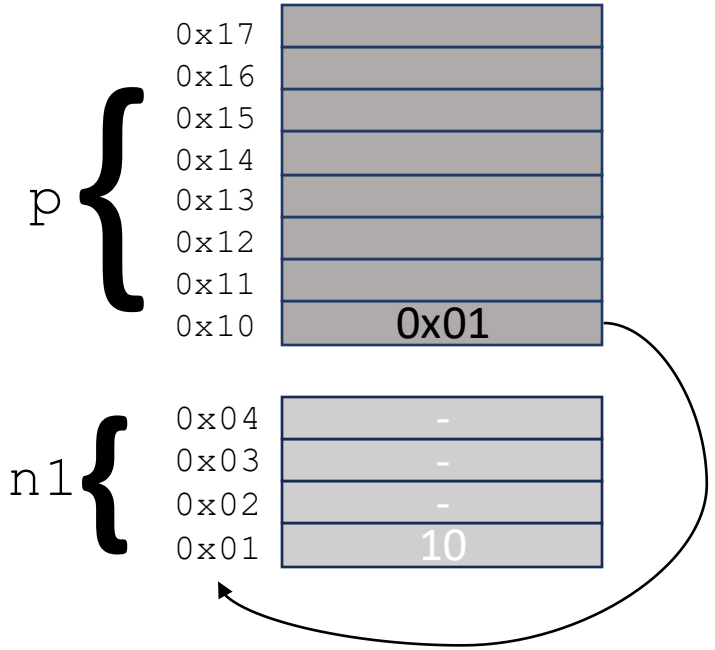
Errore

  - Nota: il contenuto di una variabile di tipo puntatore è un indirizzo, cioè un *numero*
    - Aritmetica dei puntatori (next week)

Puntatori

```
int n1 = 10;
int *p = &n1;
*p = *p + 1;
```

→ `*p` ed `n1` sono due modi diversi per riferirsi allo stesso dato
(*puntano* allo stessa cella di memoria)

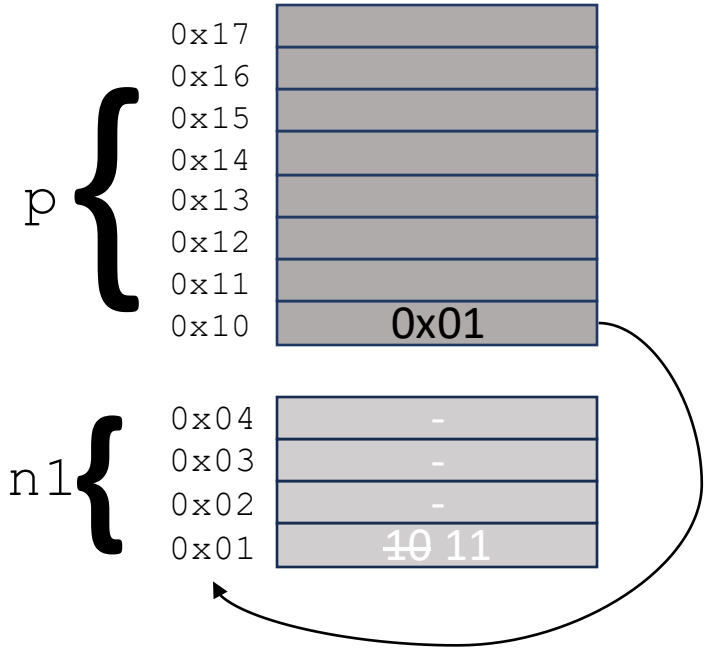


Puntatori

```
int n1 = 10;
int *p = &n1;
*p = *p + 1;
```

→ `*p` ed `n1` sono due modi diversi per riferirsi esattamente allo stesso dato

→ modificando `*p` modifico anche `n1`

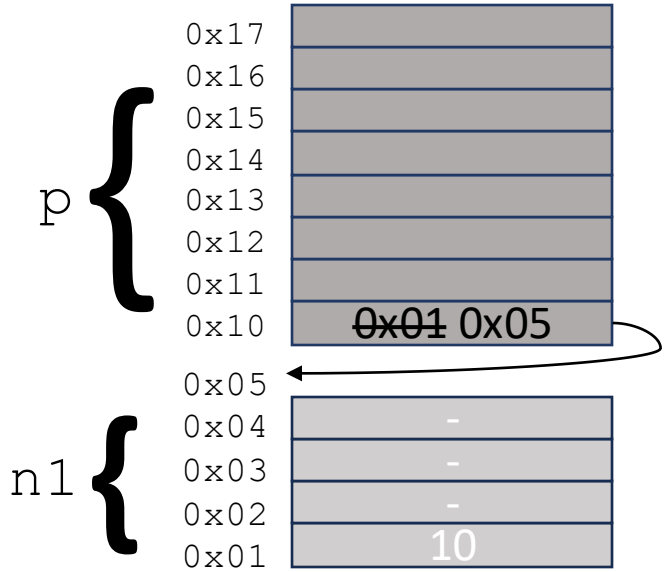


Puntatori

```
int n1 = 10;
int *p = &n1;
*p++;
```

- Attenzione alla priorità
- `*` e `&` hanno priorità intermedia tra `=` e `++`, `--`, `[]`, `.`
- → `*p++` dereferenzia `p`, poi incrementa `p` di `1`
  - `<espressione>++`
    1. Ha come valore il valore di `<espressione>`
    2. Incrementa di 1 il valore di `<espressione>`

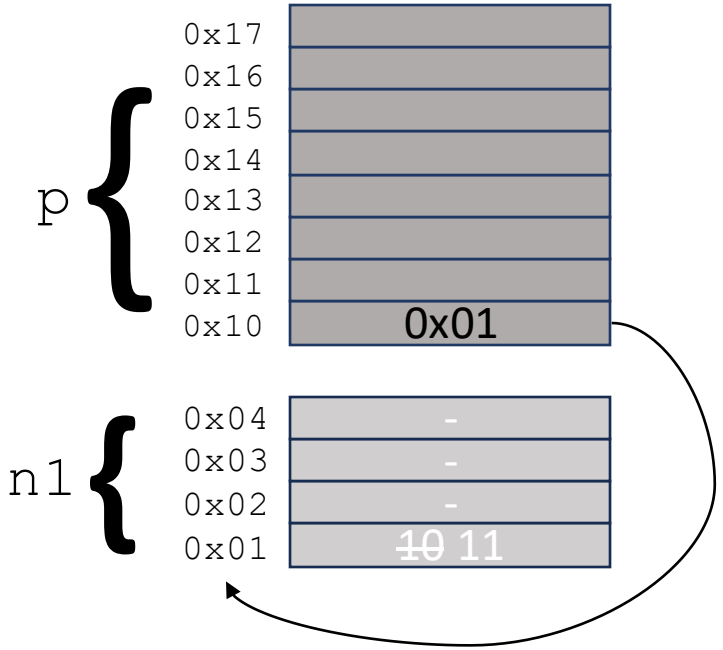
    - provate `int a = b++;`



L'incremento da `0x01` a `0x05` è dovuto all'*aritmetica dei puntatori*

Puntatori

```
int n1 = 10;
int *p = &n1;
(*p)++;
```

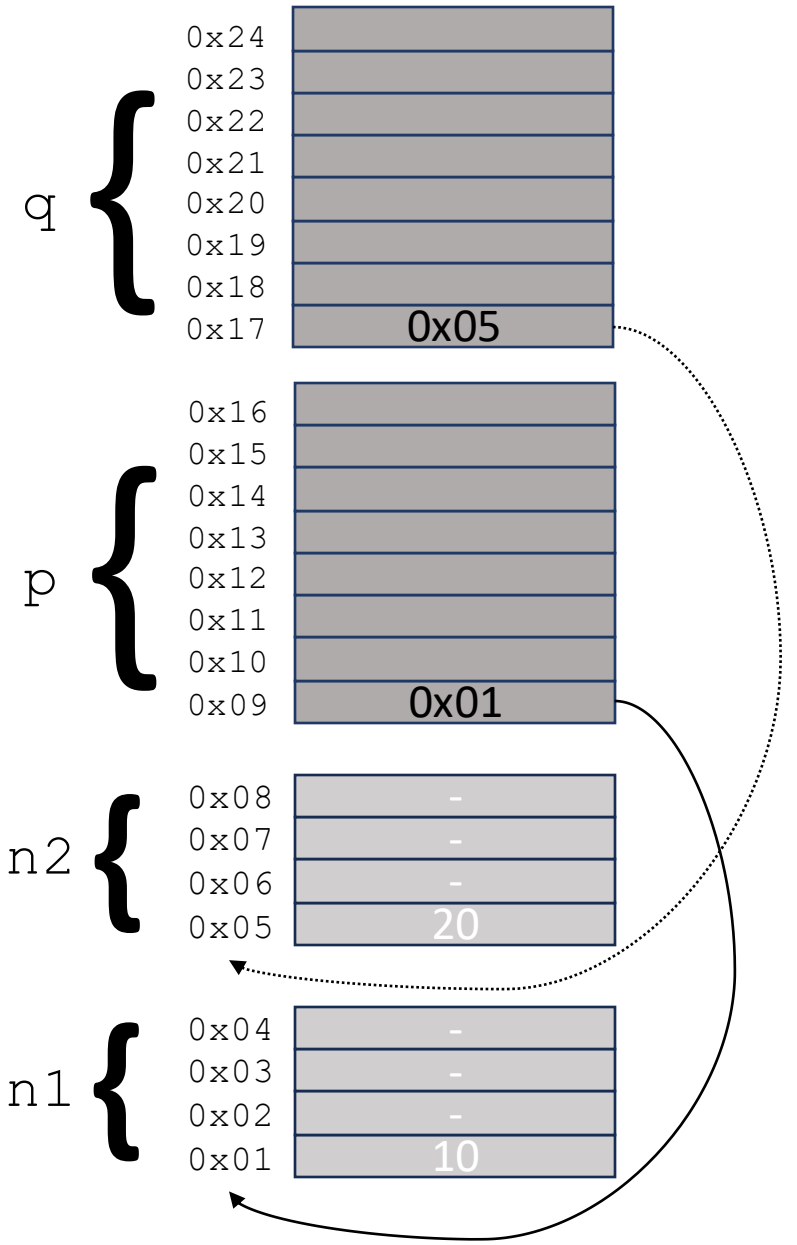


Assegnamento e Modifiche

```
int n1 = 10, n2 = 20;
int *p = &n1, *q = &n2;
```

Assegnamento e Modifiche

```
int n1 = 10, n2 = 20;
int *p = &n1, *q = &n2;
```

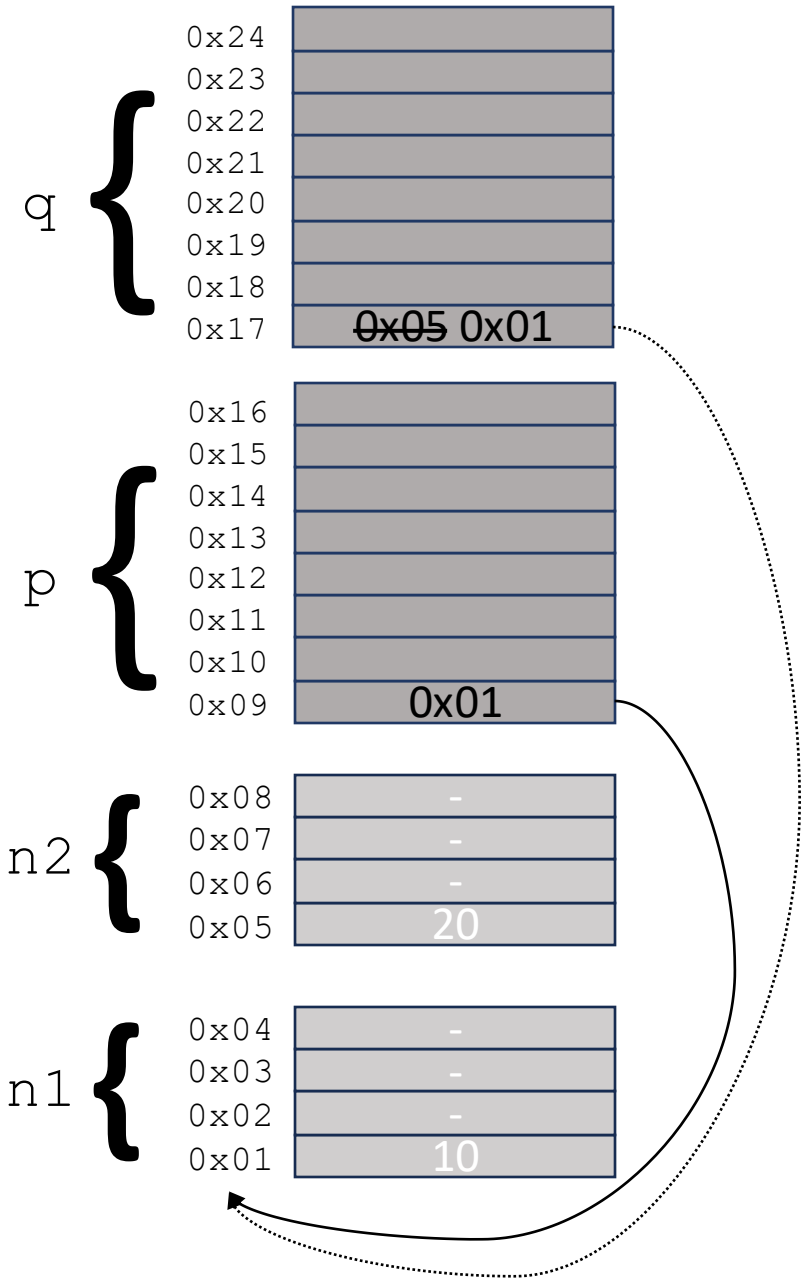


Assegnamento e Modifiche

```
int n1 = 10, n2 = 20;
int *p = &n1, *q = &n2;

q = p;
```

Ora `p` e `q` puntano allo stesso oggetto



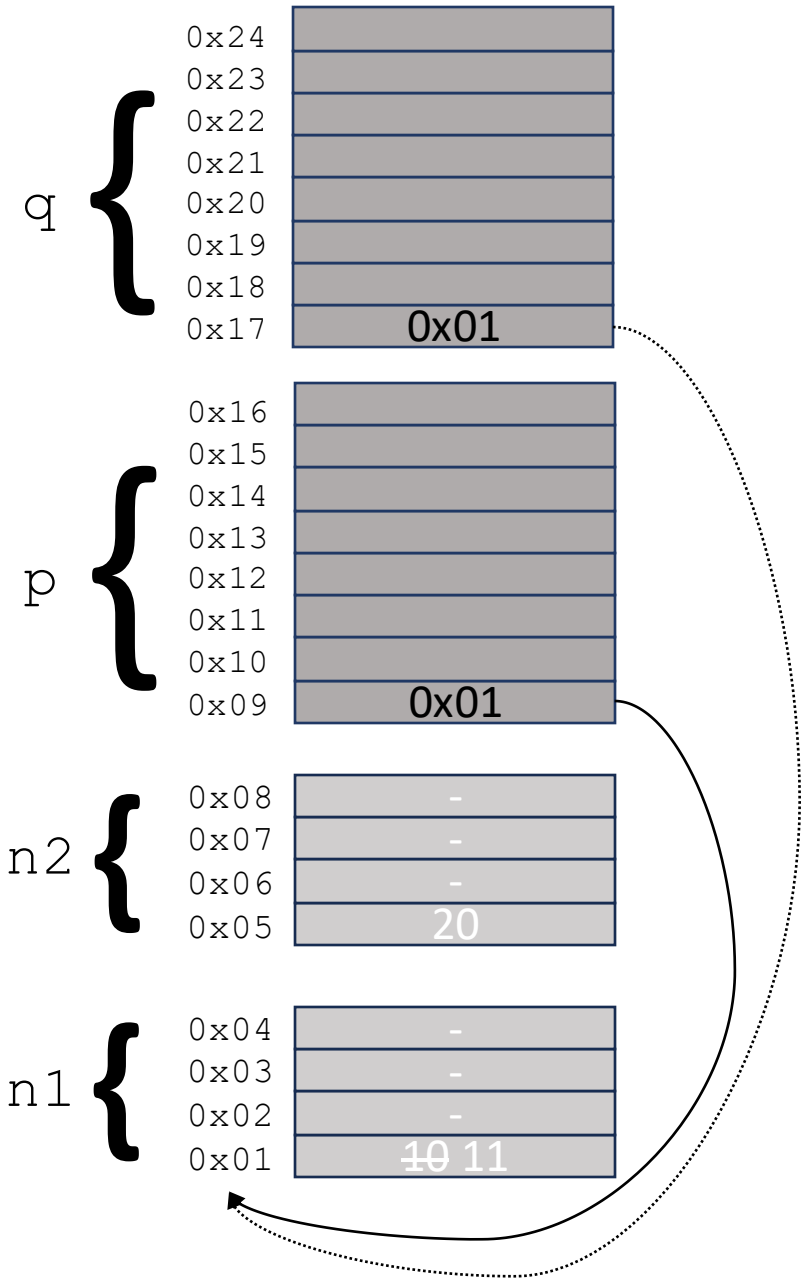
Assegnamento e Modifiche

```
int n1 = 10, n2 = 20;
int *p = &n1, *q = &n2;

q = p;
*p = *p + 1;
```

Ora `p` e `q` puntano allo stesso oggetto

Modificando `*p` modifico anche `*q` e `n1`, e viceversa

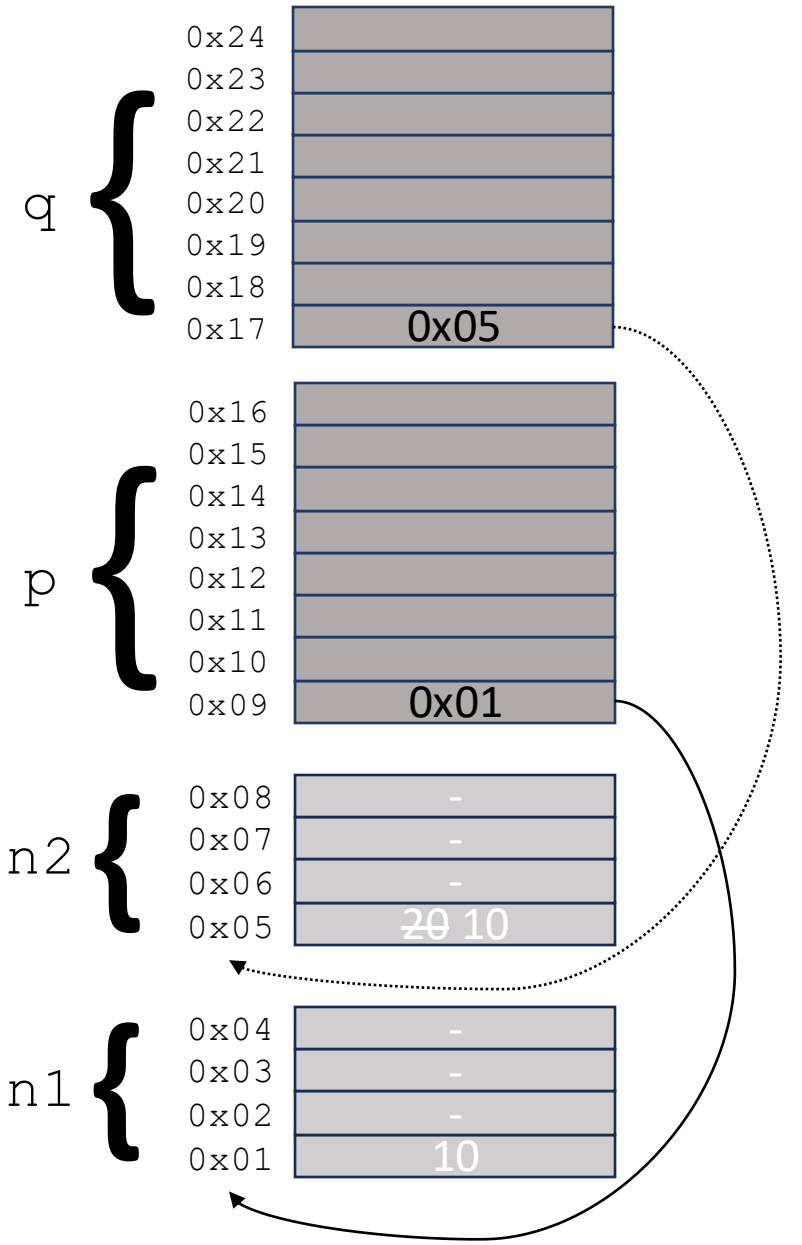


Assegnamento e Modifiche

```
int n1 = 10, n2 = 20;
int *p = &n1, *q = &n2;

*q = *p;
```

Ora `*p` a `*q` puntano a una copia dello stesso oggetto



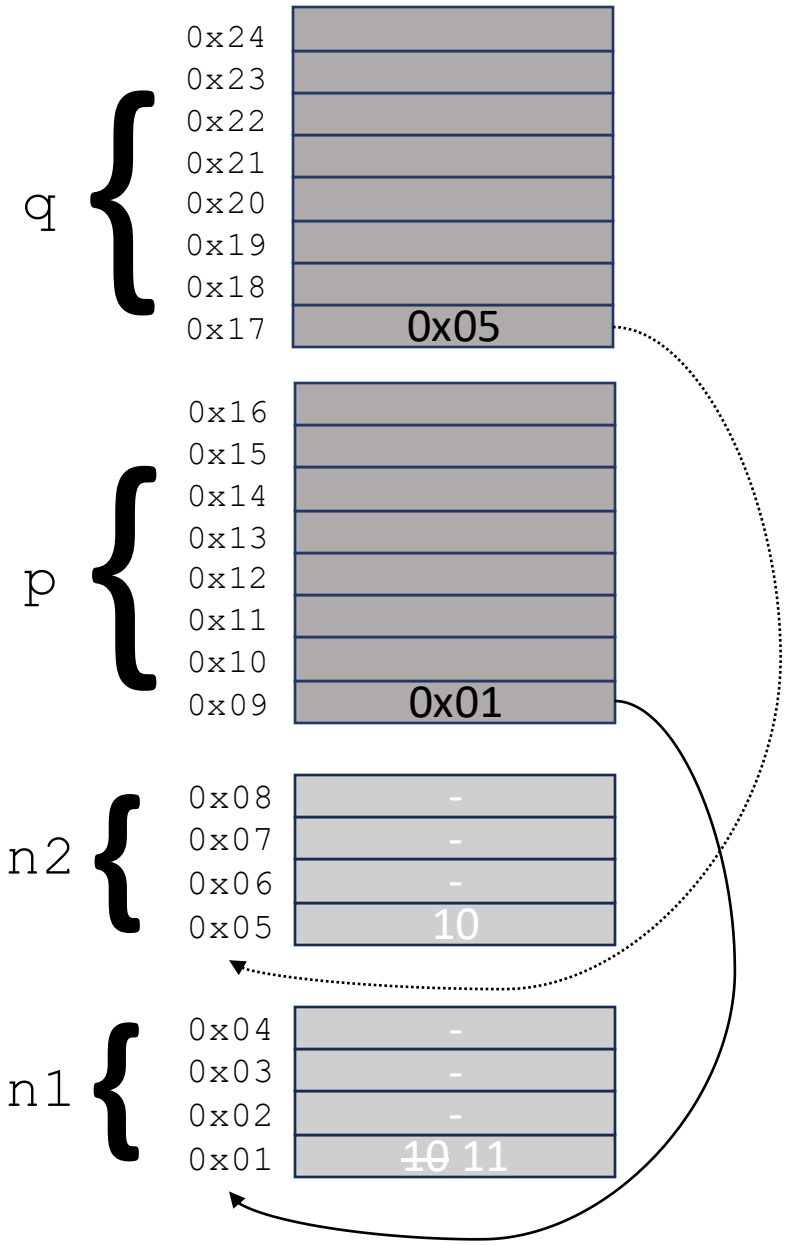
Assegnamento e Modifiche

```
int n1 = 10, n2 = 20;
int *p = &n1, *q = &n2;

*q = *p;
*p = *p + 1;
```

Ora `*p` a `*q` puntano a una copia dello stesso oggetto

Modificando `*p` modifico `n1` ma non `*q`



Restituire più di un valore con una funzione

La funzione termina quando il primo `return` viene eseguito, e viene restituito la variabile corrispondente.

Come posso fare se voglio restituire più di un parametro?

- Passare i valori per riferimento (puntatori)
- Restituire una `struct` (se i dati sono di tipo eterogeneo)
- Restituire un array (se i dati sono omogenei)

Esempio

Funzione che restituisce il min e max di un array lungo n elementi

```
void trovaMinMax(int arr[], int n, int *min, int *max) {
        *min = arr[0];
        *max = arr[0];

        for (int i = 1; i < n; i++) {
                if (arr[i] < *min)
                        *min = arr[i];
                if (arr[i] > *max)
                        *max = arr[i];
        }
}
```

Non uso il valore di ritorno, mi può essere utile?

Esempio

```
int trovaMinMax(int arr[], int n, int *min, int *max) {
        *min = arr[0];
        *max = arr[0];

        if (n<=0)
                return -1;

        for (int i = 1; i < n; i++) {
                if (arr[i] < *min)
                        *min = arr[i];
                if (arr[i] > *max)
                *max = arr[i];
        }

        return 1;
}
```

Buone Norme

Se non utilizzato, il valore di ritorno della funzione può essere usato per segnare eventuali errori della funzione (valori negativi) o che la funzione ha fatto il suo lavoro correttamente.

Ad esempio, la scanf si comporta in questo modo, dandovi come valore di ritorno il numero di elementi letti correttamente.