Linguaggio C: PUNTATORI

- I puntatori sono una delle più importanti caratteristiche del linguaggio C.
- Permettono di lavorare *a basso livello*, mantenendo **flessibilità e praticità**.

- Il C utilizza molto i puntatori in maniera esplicita con: array, funzioni, strutture.

Definizione

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

L'indirizzo associato ad una variabile corrisponde a quello del primo byte; l'indirizzo associato ad un vettore corrisponde a quello del suo primo elemento.

- **p** è una variabile di tipo puntatore
- **p=&x** al puntatore p viene assegnato l'indirizzo della variabile x
- ***p** denota la variabile puntata da p (cioè x)

p = = &x

*p = = x

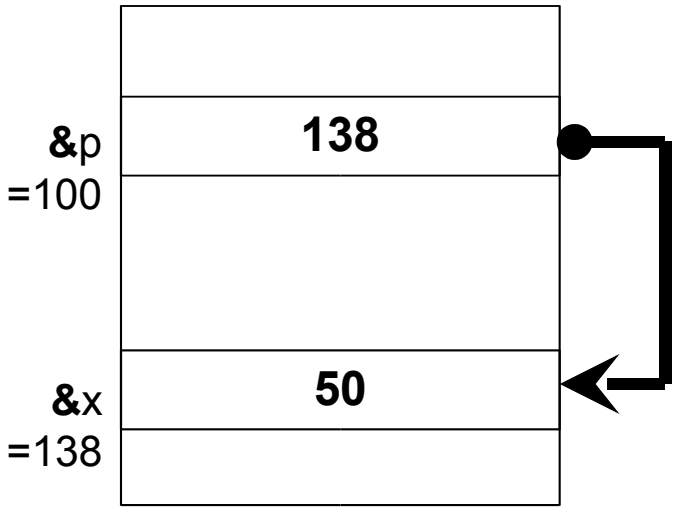
Operatori

- L'asterisco (*) viene chiamato operatore di indirezione o **deferenziazione** e restituisce il contenuto dell'oggetto puntato dal puntatore;
- l'operatore indirizzo (**&)** restituisce l'indirizzo della variabile da cui è seguito: puntatore = &variabile;
- i 2 operatori sono complementari: &*p coincide con p

Esempio:

• x è una variabile memorizzata all'indirizzo 138 e con valore 50;

• p è un puntatore alla variabile x e quindi contiene il suo indirizzo di memoria (138);



Dichiarazione

- **La dichiarazione di un *puntatore* è la seguente:**

  <tipo_base> *<identificatore_var_puntatore>

- **Il tipo base del puntatore serve per definire il tipo delle variabili a cui punterà il puntatore.**

```
int x=50;
int *p;
p=&x;  // => *p=x=50
```

Qualche esempio

int *p; /* dichiara p come un puntatore a int */
int x=1,y=2;

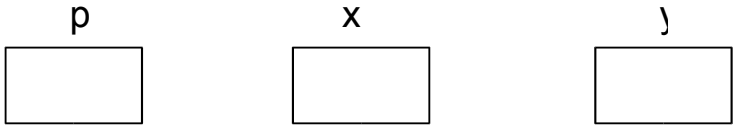
(1) p= &x; /* assegna a p l'indirizzo di x */

(2) y=*p; /* assegna a y il contenuto di p */

(3) x=p /* assegna ad x il valore di p, cioè l'indirizzo del puntato da p */

(4) *p=3; /* assegna al contenuto di pointer il valore 3 */

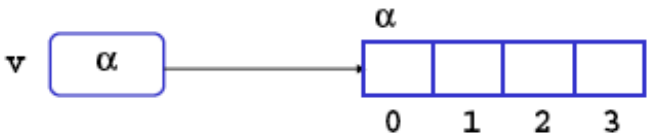
la variabile x si trova nella locazione di memoria 100, y nella 200 e p nella 1000



(1) fa si che pointer contenga il valore 100 (cioè l'indirizzo di memoria di x).
(2) fa si che y assuma valore 1 (il valore del puntato da p, cioè x).
(3) fa si che x assuma valore 100 (cioè il valore di p).
(4) assegna alla variabile puntata da p il valore 3 (quindi x=3).

Array

- **L'array è un puntatore** ad un'area di memoria pre-allocata, che **contiene l'indirizzo del primo elemento** del vettore.



- Indicano tutti la stessa cosa

  v &v[0] α

  v+1 &v[1] α+1

Puntatori: 3 valori

Quindi in merito ai puntatori possiamo avere tre possibili valori:

int *pointer;

- ***pointer*** contenuto o valore della variabile puntatore (indirizzo della locazione di memoria a cui punta)
- **&pointer** indirizzo fisico della locazione di memoria del puntatore
- ***pointer** contenuto della locazione di memoria a cui punta

NB Quando un puntatore viene dichiarato non punta a nulla: per poterlo utilizzare deve puntare a qualcosa!
E' un errore comune non assegnare un indirizzo di memoria a un puntatore prima di usarlo e nessun compilatore lo segnala:

Genera un errore

```
int *ip;
*ip=100;
```
← NO

Utilizzo corretto:

```
int *ip;
int x;
ip=&x;
*ip=100;
```

Funzioni: passaggio per indirizzo

- Generalmente gli argomenti vengono passati alle funzioni per **valore**.
- Il passaggio per indirizzo si usa quando si devono mantenere le modifiche apportate alle variabili all'interno delle funzioni.

esempio: la funzione **scambia (alfa, beta)** che trasferisce il valore di alfa in beta e viceversa; in questo caso al termine della funzione (con return) i valori delle variabili alfa e beta non sono stati modificati. Come ottenere lo scambio effettivo del valore delle due variabili?.

Si devono passare alla funzione, non i valori delle variabili, ma il loro indirizzo
scambia (&alfa, &beta)

```
#include <stdio.h>

void scambia(int *apt, int *bpt);

int main()
{
   int alfa = 5;
   int beta = 13;

   printf("alfa = %d, beta = %d\n", alfa, beta);

   scambia(&alfa, &beta);

   printf("alfa = %d, beta = %d\n", alfa, beta);
}
```

```
void scambia(int *apt, int *bpt)
{
   int temp;
   temp = *apt;
   *apt = *bpt;
   *bpt = temp;
}
```

Strutture

```
struct persona{
    char nome[20];
    int eta; } P1;
P1.eta=30;
```

Il puntatore a struttura accede ai campi dell'elemento a cui punta attraverso il simbolo freccia (->) invece del punto (.)

```
struct punto { int x, y; } elemento;
struct punto *puntatore;

    puntatore = &elemento;

    puntatore->x = 6; // equivale a elemento.x=6
    puntatore->y = 8;
```

Liste

I puntatori si usano anche per creare liste semplici: si collegano due elementi, attraverso l'inserimento di un campo aggiuntivo che rappresenta un puntatore ad una struttura delle stesso tipo (next):

```
typedef struct {
    int valore;  //informazione da memorizzare in ogni elemento
    ELEMENTO *next; //campo aggiuntivo di collegamento
} ELEMENTO;
ELEMENTO el1, el2;

// in questo modo l'elemento successivo di el1 diventa el2
el1.next = &el2;
```