Lezione 11: I Puntatori

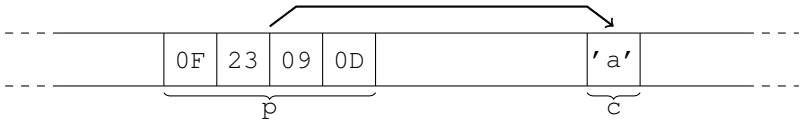
Laboratorio di Elementi di Architettura e Sistemi Operativi

16 Aprile 2013

Puntatori ed indirizzi

- Semplificando, la memoria di un computer può essere vista come un *vettore di celle* numerate in modo consecutivo
  - tipicamente, le celle hanno la dimensione di un byte
- ogni tipo di dato occupa un certo numero di celle di memoria:
  - `char`: un byte
  - `short int`: due byte
  - `long int`: quattro o otto byte
  - ...
- un *puntatore* è un tipo di variabile che contiene un *indirizzo di memoria*

Esempio:
`c` è un `char` e `p` un puntatore che *punta* a `c`, cioè *contiene l'indirizzo di memoria di* `c`



- Dichiarazione di un puntatore: `tipo *nome`
  - ogni puntatore è vincolato a puntare un certo tipo di dato
  - *eccezione:* un puntatore a `void` punta ad un dato generico
  - un puntatore a `void` *non può essere dereferenziato*
- L'operatore unario `&` fornisce l'indirizzo di un oggetto in memoria:
  - `p = &c;`
- L'operatore unario `*` *dereferenzia* un puntatore, ossia permette di *accedere al contenuto* dell'oggetto puntato:
  - `*p = 'a';  c = *p;`

Esempio di uso degli operatori `&` e `*`:

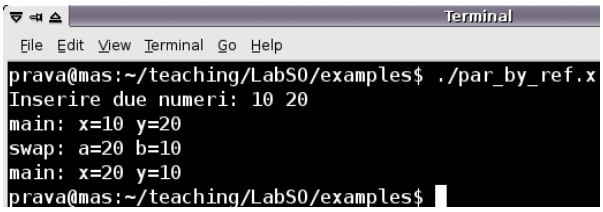
```
int x = 1, y = 2, z[10];
int *ip, *iq;  /* ip e iq sono puntatori ad int */

ip = &x;       /* ora ip punta a x */
y = *ip;       /* ora y vale 1 */
*ip = 0;       /* ora x vale 0 */
ip = &z[0];    /* ora ip punta a z[0] */
iq = ip;       /* ora iq punta a z[0] */
```

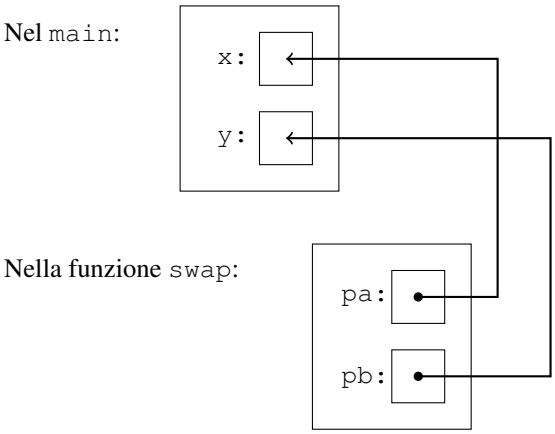
Puntatori e parametri di funzione

- In C, il passaggio dei parametri avviene per *valore*:
  - come abbiamo già visto, questo impedisce alle funzioni di modificarne il valore
- Usando i *puntatori* posso passare i parametri *per indirizzo* (by reference):
  - Parametri formali = puntatori al tipo corrispondente dei parametri
- Concetto
  - Passando gli indirizzi dei parametri attuali posso modificarne il valore

```
#include<stdio.h>
#include<stdlib.h>

void swap(int *pa, int *pb) {
  int tmp;
  tmp = *pa;
  *pa = *pb;
  *pb = tmp;
  printf("swap: a=%d
          b=%d\n",*pa,*pb);}
```

```
int main()
{
  int x,y;
  printf("Inserire due
          numeri: ");
  scanf("%d %d",&x,&y);
  printf("main: x=%d
          y=%d\n",x,y);
  swap(&x,&y);
  /* ORA x e y VENGONO
     MODIFICATI */
  printf("main: x=%d
          y=%d\n",x,y);}
```



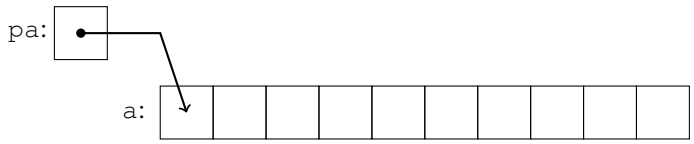


Puntatori e vettori

- In C c'è un legame molto stretto tra puntatori e vettori:
  - *nome del vettore = puntatore al primo elemento*

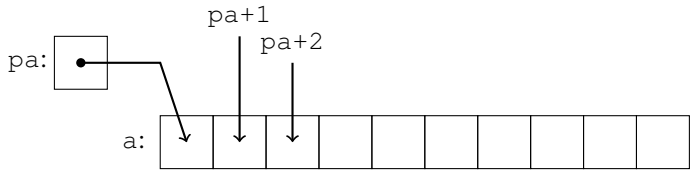
Esempio:

```
int a[10], *pa;

pa = a;
```



- Se `pa` punta ad un elemento di un vettore, allora:
  - `pa + 1` punta all'elemento successivo
  - `pa + i` punta ad `i` elementi dopo `pa`
  - `pa - i` punta ad `i` elementi prima di `pa`

```
int a[10], *pa;

pa = a;
```



- Indicizzare un vettore corrisponde ad incrementare il puntatore corrispondente:

`int a[10]`		`int *pa`		
`a`	`&a[0]`	`pa`	`&pa[0]`	puntatore al primo elemento
`&a[i]`	`(a+i)`	`&pa[i]`	`(pa+i)`	puntatore all'`i`-esimo elemento
`a[i]`	`*(a+i)`	`pa[i]`	`*(pa+i)`	valore dell'`i`-esimo elemento

- Ci sono però delle differenze:
  - un puntatore è una *variabile* che può essere modificata:
    - sia `pa = a` che `pa++` sono operazioni permesse
  - il nome di un vettore non è una variabile, e non si può modificare:
    - le operazioni `a = pa` e `a++` *non sono permesse!*
  - la definizione di un puntatore *non riserva spazio di memoria* per il contenuto
- Passare un vettore ad una funzione equivale a passare il *puntatore al primo elemento*

  Esempio: la funzione `strlen`

```
int strlen(char *s)
{
  int n;

  for (n = 0; *s != '\0'; s++)
  {
    n++;
  }
  return n;
}
```

- all'interno della funzione `s` è una variabile locale: l'operazione `s++` non ha nessun effetto sulla stringa passata alla funzione.