

Puntatori

Obiettivi:

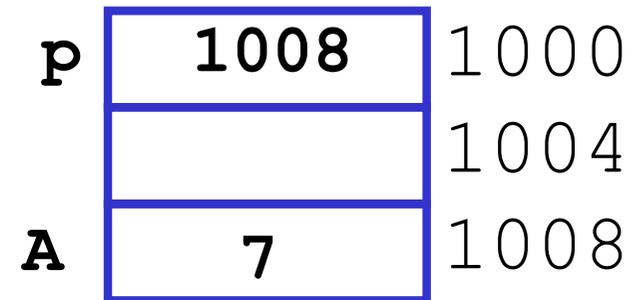
- Richiamare quanto noto sui puntatori dal modulo A
- Presentare l'analogia tra puntatori e vettori e l'aritmetica dei puntatori

Il puntatore

- Un tipo puntatore è un tipo scalare per memorizzare indirizzi in memoria
- Costruttore *

`<TipoElemPuntato> *<NomePuntatore>;`

```
int *p; ...  
int A=7;  
p=&A;  
*p=5;
```

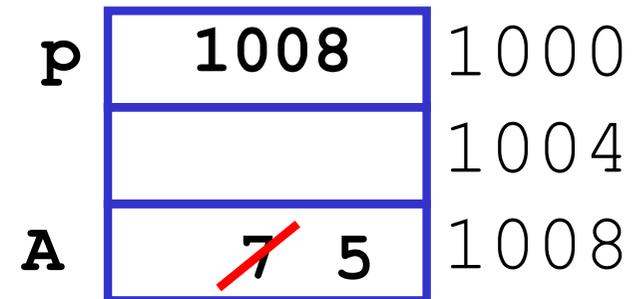


Il puntatore

- Un tipo puntatore è un tipo scalare per memorizzare indirizzi in memoria
- Costruttore *

`<TipoElemPuntato> *<NomePuntatore>;`

```
int *p; ...  
int A=7;  
p=&A;  
*p=5;
```



Operatori

- **=**, assegnamento (tra puntatori dello stesso tipo)
- **NULL** costante, per indicare l'indirizzo nullo
- ***** (**dereferencing**, unario), si applica a un puntatore e restituisce il valore contenuto nella cella puntata
- **&** (**indirizzo**), si applica ad una variabile e ne restituisce l'indirizzo
- **+**, **-**, operatori **aritmetici** (vedi vettori & puntatori)
- **>**, **<**, **==**, **!=** , operatori **relazionali**

Puntatori

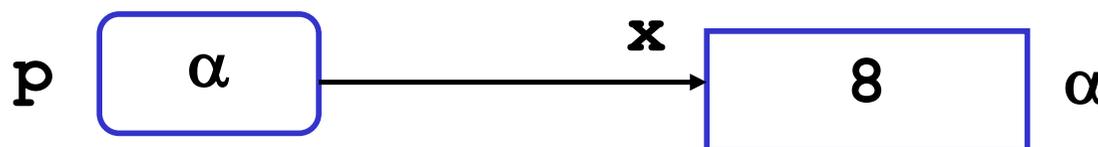
- Un *puntatore* è una variabile *destinata a contenere l'indirizzo di un'altra variabile*
- Vincolo di tipo: un puntatore a T può contenere solo l'indirizzo di variabili di tipo T.

- Esempio:

```
int x = 8;
```

```
int* p;
```

```
p = &x; →
```



Da questo momento, `*p` e `x` sono *due modi alternativi per denotare la stessa variabile*

Puntatori

Un puntatore non è legato per sempre alla stessa variabile: può puntare altrove.

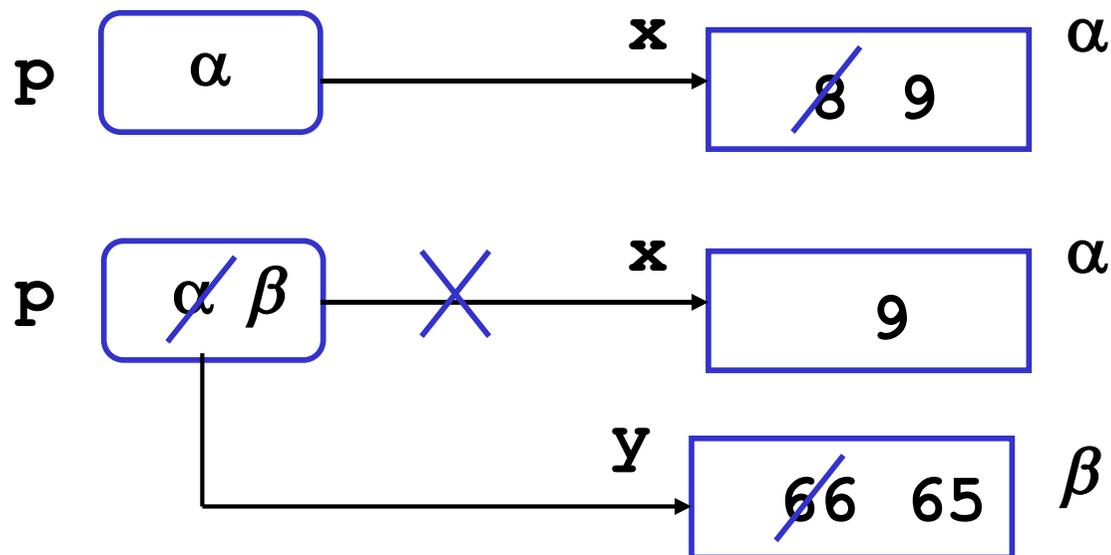
```
int x = 8, y = 66;
```

```
int *p = &x;
```

```
(*p)++;
```

```
p = &y;
```

```
(*p)--;
```



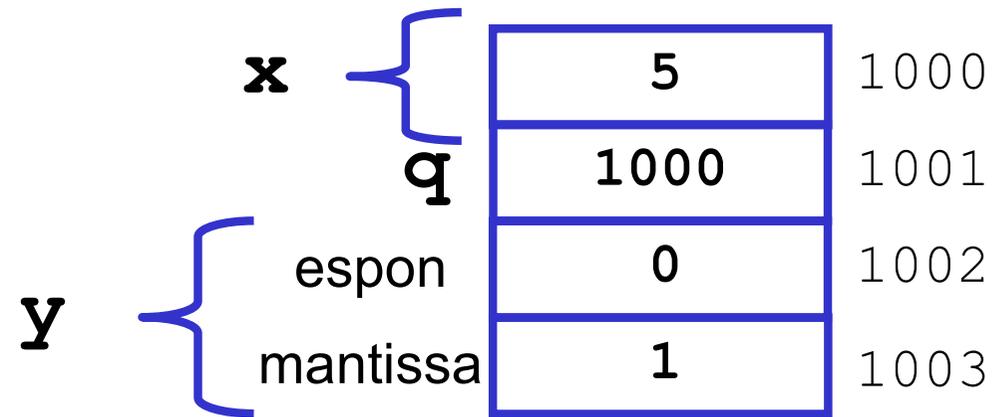
Le parentesi sono necessarie per riferirsi alla variabile puntata da p

Controlli di tipo

- Nella definizione di un puntatore è **necessario** indicare il tipo della variabile puntata
- Il compilatore effettua controlli statici sull'uso dei puntatori
 - un **puntatore a int** può contenere solo indirizzi di variabili di tipo **int**
 - un **puntatore a float** può contenere solo indirizzi di variabili di tipo **float**
 - un **puntatore a struct frazione** può contenere solo indirizzi di variabili di tipo **struct frazione**

se facessi ...

```
char x=5;  
float *q, y=1e0;  
q=&x;  
y = *q;  
printf("%f\n", y);
```



warning: incompatible types

-107372584.000000

- Casting?

Vettori e puntatori

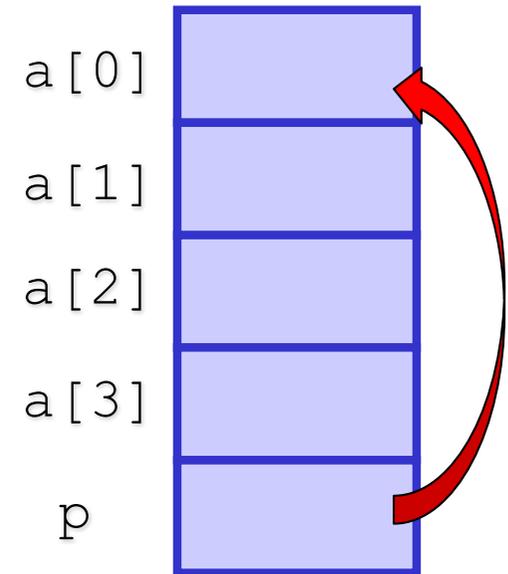
- Una volta dichiarato un array, `a`

```
int a[4]; int *p;
```
- I suoi elementi sono allocati in celle consecutive in memoria, con dimensione dipendente dal tipo di elementi dell'array
- Il nome dell'array è l'indirizzo (costante) del primo elemento di questa batteria di celle
- `a` è una costante ed equivale a `&a[0]` ed il suo tipo è puntatore a `int`
- Posso scrivere:

```
p=a; /* p punta ad a[0],  
      ovvero p è uguale a &a[0]*/
```


ma non:

```
a=p; /* ERRORE! */
```



Notazioni equivalenti

Lunghezza di una stringa, da così...

```
int lunghezza(char s[]) {  
    int lung=0;  
    for (lung=0; s[lung]!='\0'; lung++);  
    return lung;  
}
```

... a così:

```
int lunghezza(char *s) {  
    int lung=0;  
    for (lung=0; s[lung]!='\0'; lung++);  
    return lung;  
}
```

↑
Oppure: *(s+lung)

Operatori di *dereferencing*

- L'operatore *****, applicato a un puntatore, accede alla variabile da esso puntata
- L'operatore **[]**, applicato a un nome di array e a un intero *i* (più in generale ad una espressione con valore *i*) accede alla *i*-esima variabile dell'array

Sono entrambi operatori di dereferencing

$$\mathbf{*v} \equiv \mathbf{v[0]}$$

Aritmetica dei puntatori (1)

Vale anche $*v \equiv v[0] \equiv *(v+0)$, e anche:

$$*(v+1) \equiv v[1]$$

...

$$*(v+i) \equiv v[i]$$

Gli operatori $*$ e
 $[]$ sono
intercambiabili

Espressioni della forma $p+i$ vanno sotto il nome di *aritmetica dei puntatori*, e denotano l'indirizzo posto i celle *dopo l'indirizzo denotato da p* (**celle, non byte**)

Aritmetica dei puntatori (2)

Non solo sono consentite operazioni di somma fra puntatori e costanti intere ma anche:

- **Assegnamento e differenza** fra puntatori

```
int *p, *q;  p=q;  p-q;  p=p-q;
```

La differenza però produce un warning

- Altre operazioni aritmetiche fra puntatori **non sono consentite:**

```
int *p, *q;      p=p*2;  q=q+p;
```

Le operazioni sono **corrette** se i puntatori riferiscono lo **STESSO TIPO** (*non tipi compatibili*). Attenzione: comunque solo **warning** dal compilatore negli altri casi

Esempio (2)

```
void main (void)
{ char V[] = "0123456789";
      /* a vettore di caratteri */
  int i = 5;
  printf ("%c%c%c%c\n", V[i], V[5], i[V], 5[V] );
}      /* stampa 5 5 5 5 */
```

- Per il compilatore $V[i]$ e $i[V]$ sono lo stesso elemento, perché viene sempre eseguita la conversione:

$$V[i] \equiv *(V+i)$$

senza eseguire alcun controllo né su V né su i .

Vettori e puntatori

[] ha precedenza su *

Esempio: `char *a[10];`

È un vettore di puntatori a carattere

- Per dichiarare un puntatore `a` a un vettore di caratteri:

`char (*a)[10];` oppure:

`typedef char Vetch [10];`

`Vetch *a;`

Nelle
dichiarazioni,
l'ordine degli
operatori * e []
è significativo

Puntatori a strutture

- Esempio:

```
typedef struct {int campo1,  
                campo2} TipoDato;
```

```
TipoDato S, *P;
```

```
P=&S;
```

$(*P) . \text{campo1} \equiv P \rightarrow \text{campo1}$

- L'operatore \rightarrow consente di accedere ad un campo di una struttura referenziata da un puntatore in modo più sintetico