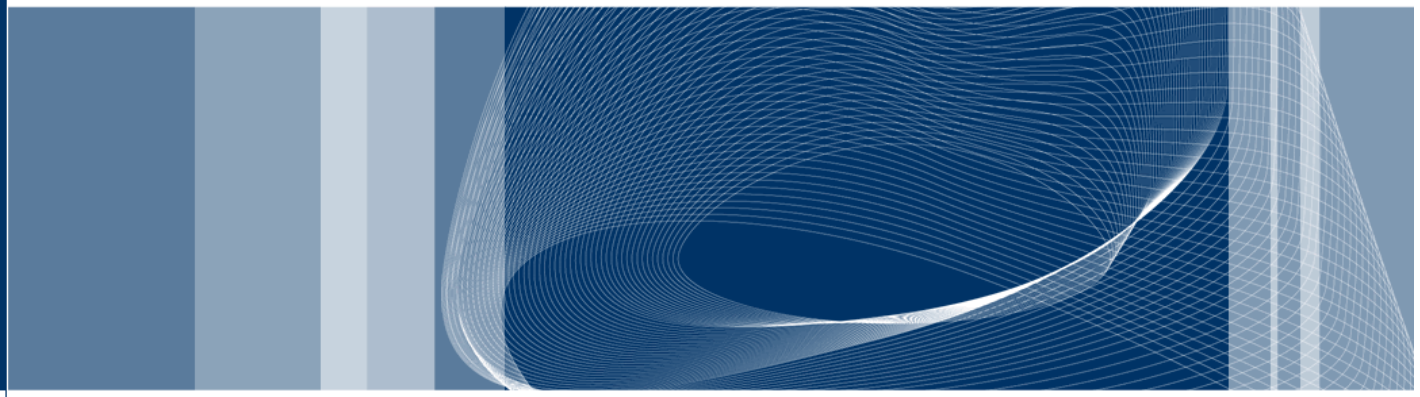
POLITECNICO DI MILANO



Puntatori in C

Fondamenti di Informatica

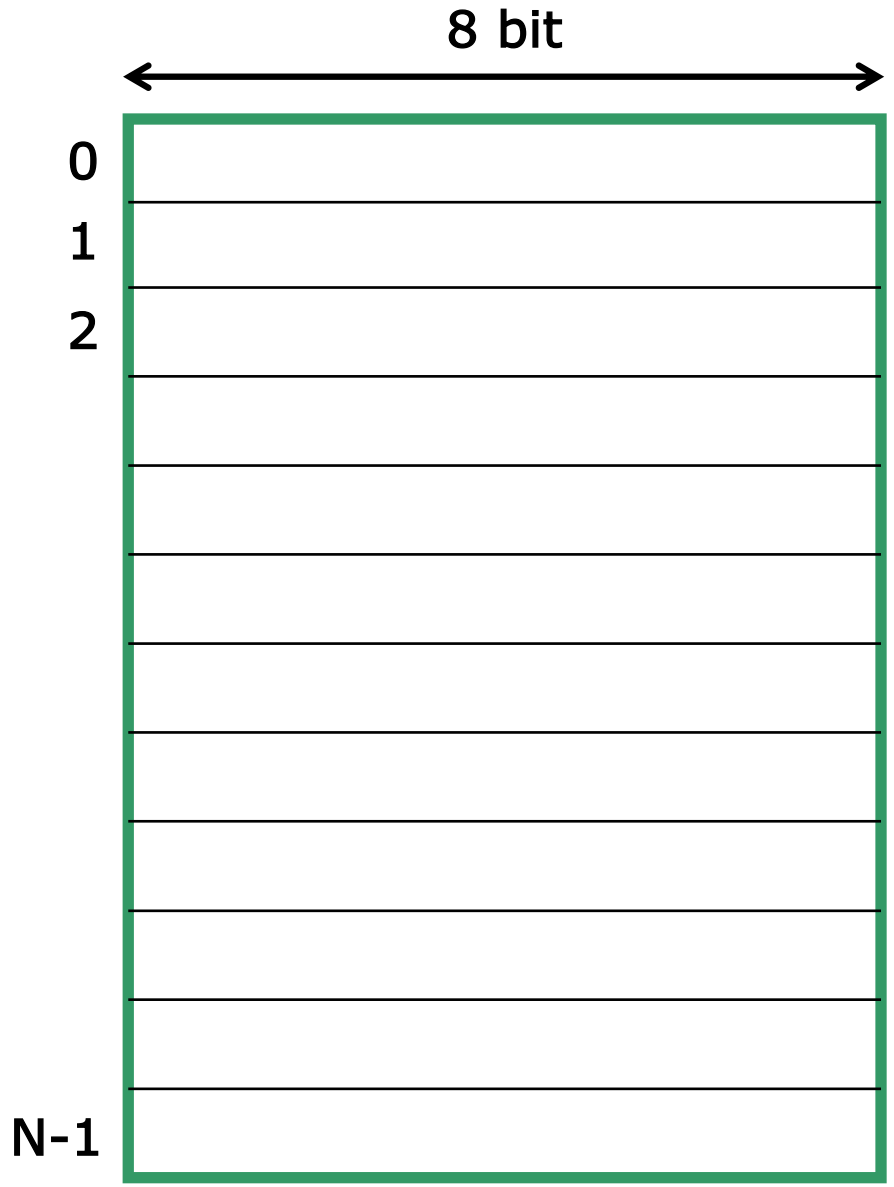
Daniele Loiacono

Puntatori

Come è fatta la memoria del calcolatore?

Un modello "concettuale" della memoria

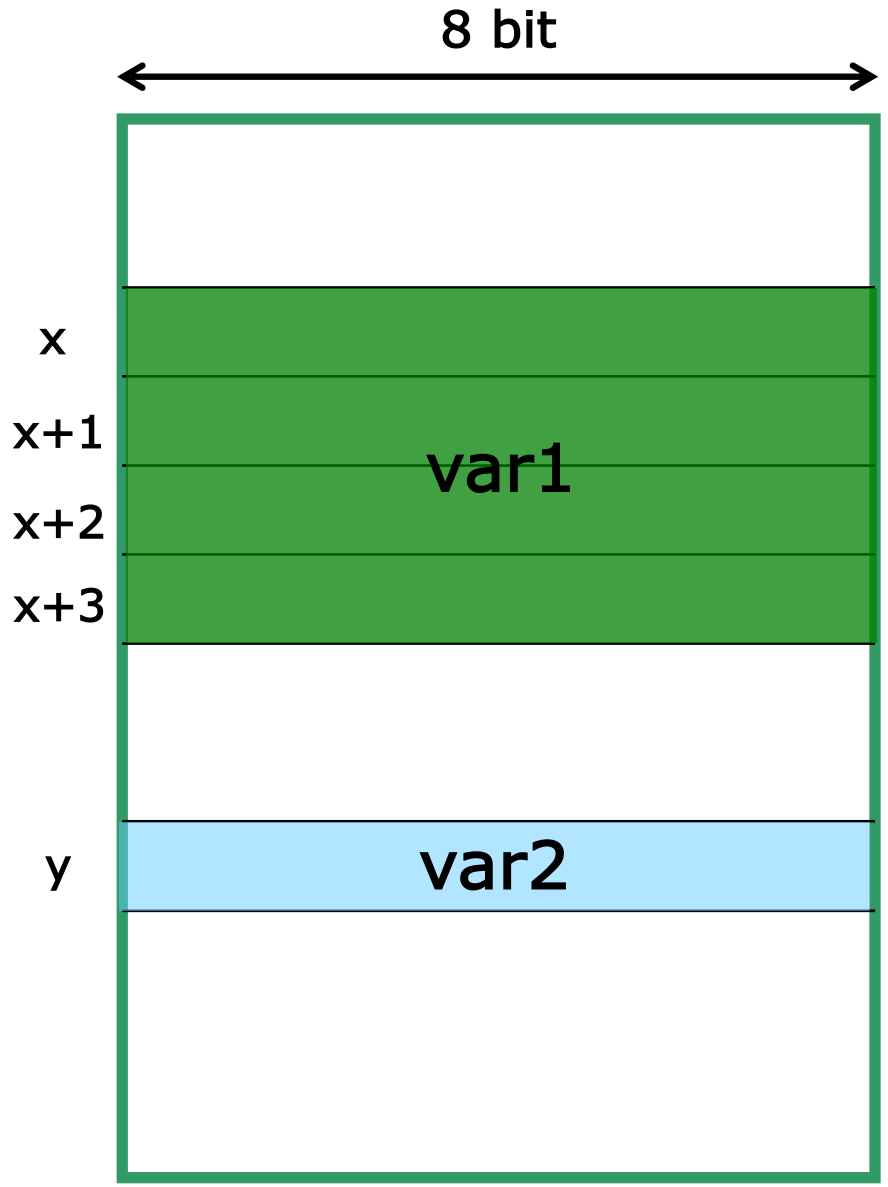
- La memoria centrale di un calcolatore è una **sequenza di celle numerate** (da 0 fino a un valore massimo N-1)
  - Il numero che identifica ogni cella è detto **indirizzo**
  - Le dimensione delle celle dipende dal tipo di calcolatore (nei calcolatori recenti è 8 bit)
- Anche gli indirizzi sono rappresentati da una sequenza di bit
  - La dimensione degli indirizzi dipende dal sistema operativo
  - Oggi, si usano 32/64 bit per gli indirizzi



Variabili e memoria

- Cosa succede quando dichiariamo una variabile in un programma C?
  - il compilatore *sceglie* un opportuno indirizzo di memoria per ospitarla
  - possono essere necessarie più celle per ospitare una variabile
  - il compilatore associa il nome della variabile all'indirizzo scelto
- Esempio

```
float var1;
char var2;
```



Possiamo accedere alle variabili
tramite il loro indirizzo?

In C è possibile grazie ai puntatori!

Cosa è un puntatore?

- Un puntatore è un **tipo di dato** che viene utilizzato in C per dichiarare una variabile che deve contenere un indirizzo di una cella di memoria.
- In gergo, si dice che la variabile puntatore "punta" alla cella di memoria, il cui indirizzo è contenuto nella variabile puntatore.
- Quando viene dichiarata una variabile puntatore, è necessario anche specificare il tipo di dato contenuto nelle celle di memoria che verranno "puntate" dalla variabile.
- La sintassi per dichiarare una variabile puntatore è:

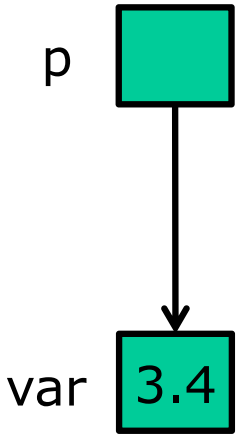
```
tipo *nome;
```

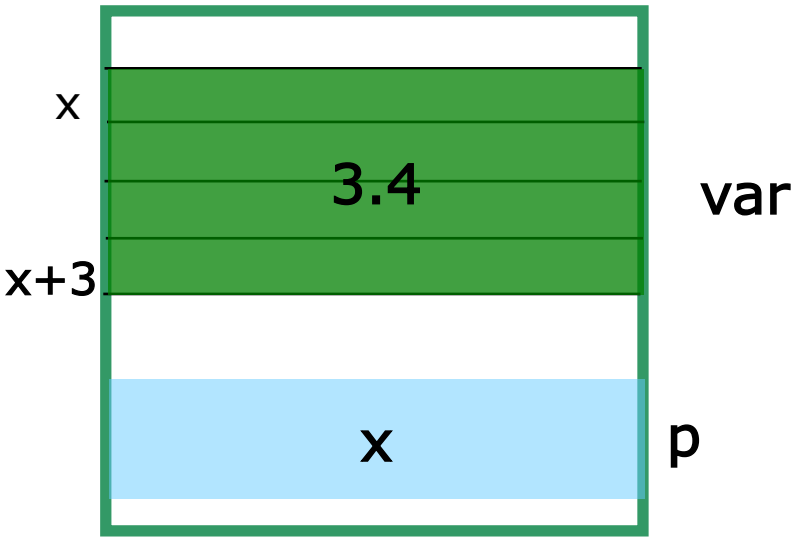
Operatore &

- Come si assegna un indirizzo di memoria di una variabile ad un puntatore?

```
float var;
float *p;

var=3.4;
p = &var;
```





- L'operatore & ("indirizzo di") è un operatore unario e restituisce l'indirizzo di memoria di una variabile qualunque:

```
&variabile
```

Abbiamo svelato il "mistero" della scanf

```
scanf("str ctrl",indirizzo)
```

- La funzione `scanf` ha come parametri la stringa di controllo e l'**indirizzo** delle variabile in cui deve essere memorizzato il valore letto da tastiera.

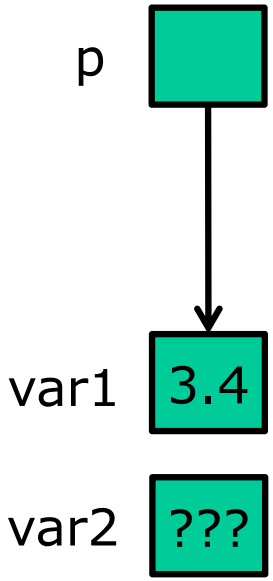
Dereferenziazione di un puntatore

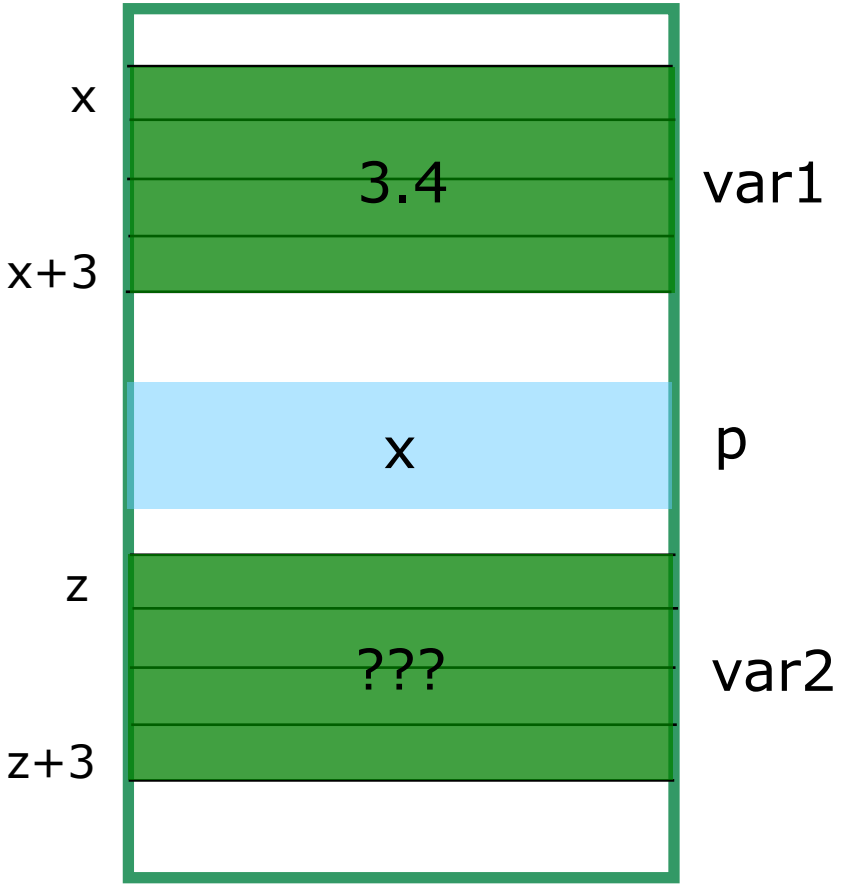
- Come si accede al **contenuto** della cella di memoria puntata dal puntatore

```
*puntatore
```

```
float var1,var2;
float *p;

var1 = 3.4;
p = &var1;
```





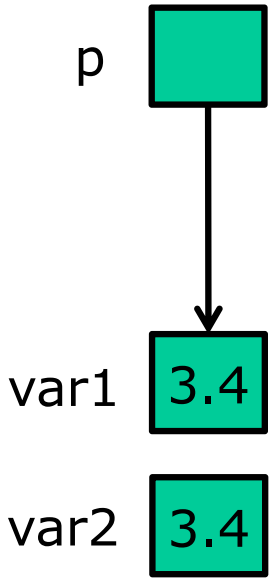
Dereferenziazione di un puntatore

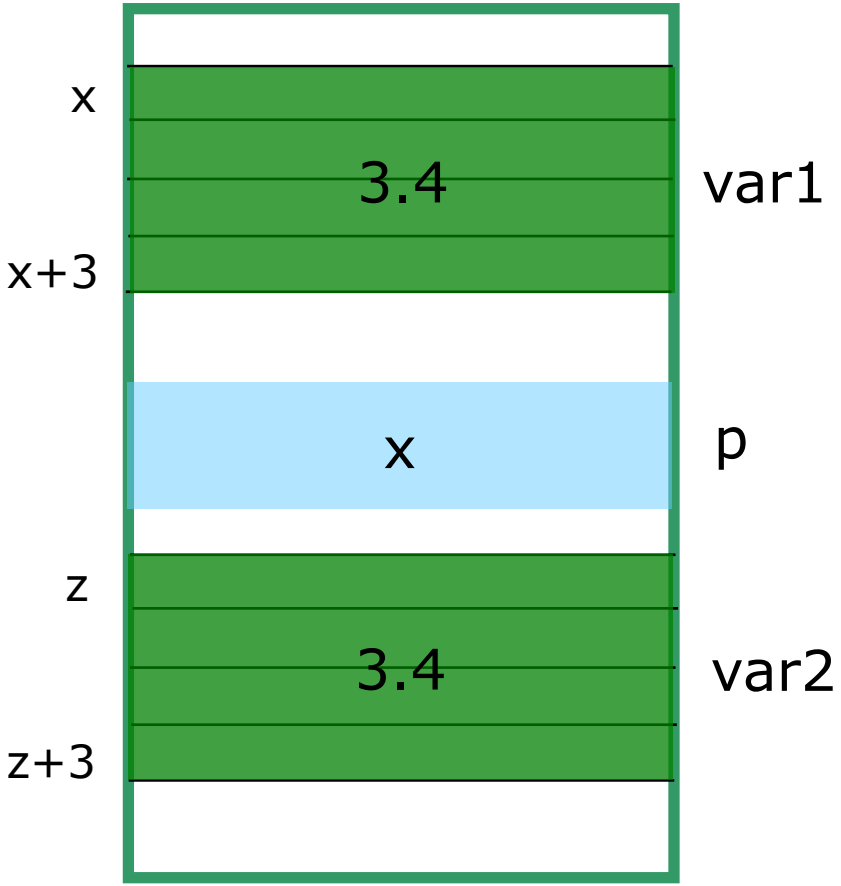
- Come si accede al **contenuto** della cella di memoria puntata dal puntatore

`*puntatore`

```
float var1,var2;
float *p;

var1 = 3.4;
p = &var1;
var2 = *p;
```





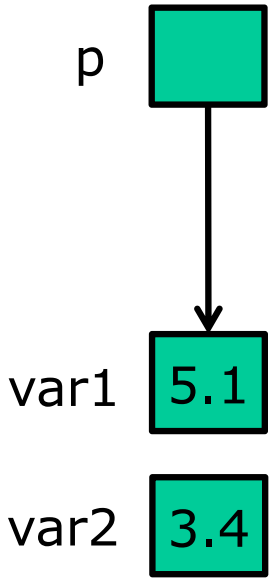
Dereferenziazione di un puntatore

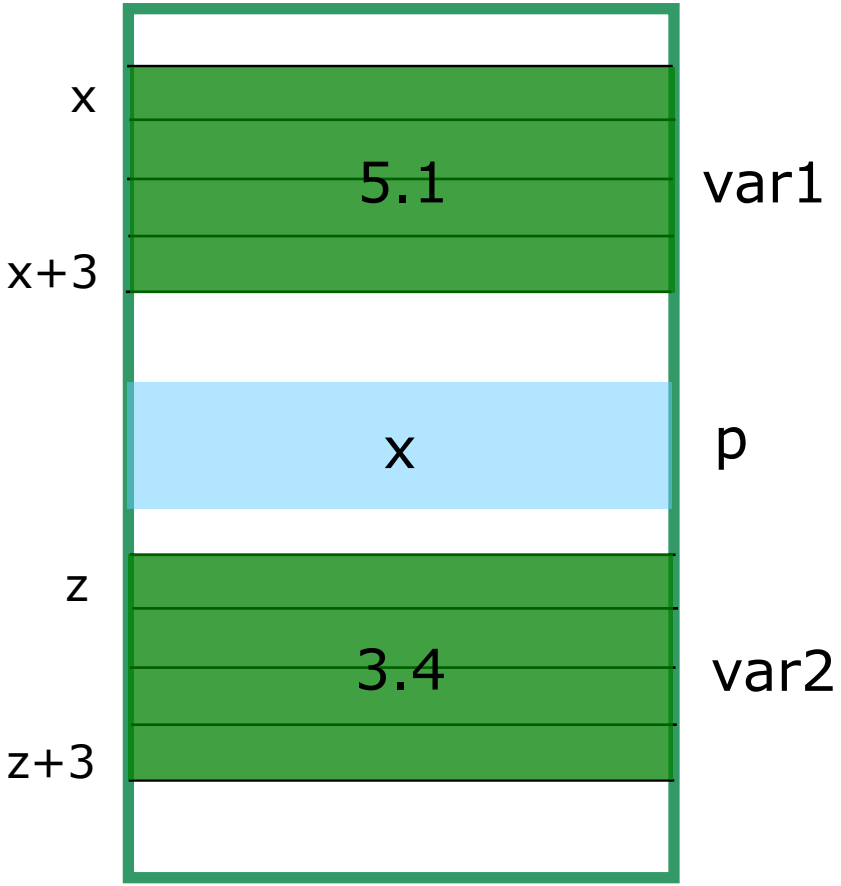
- Come si accede al **contenuto** della cella di memoria puntata dal puntatore

```
*puntatore
```

```
float var1,var2;
float *p;

var1 = 3.4;
p = &var1;
var2 = *p;
*p = 5.1;
```

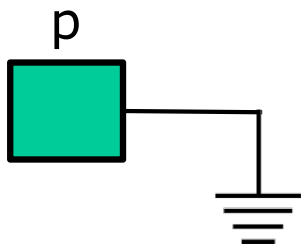




Operazioni sui puntatori

- Inizializzazione

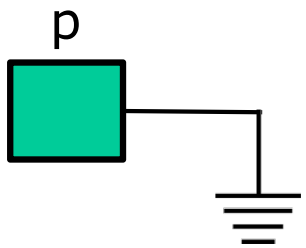
```
float *p = NULL;
```



Operazioni sui puntatori

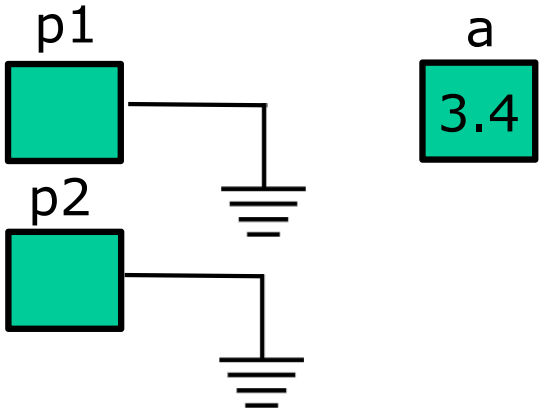
- Inizializzazione

```
float *p = NULL;
```



- Assegnamento fra puntatori

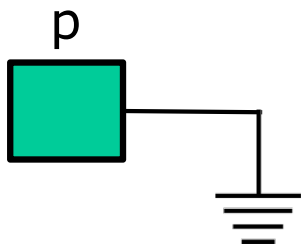
```
float a=3.4;
float *p1=NULL,*p2=NULL;
```



Operazioni sui puntatori

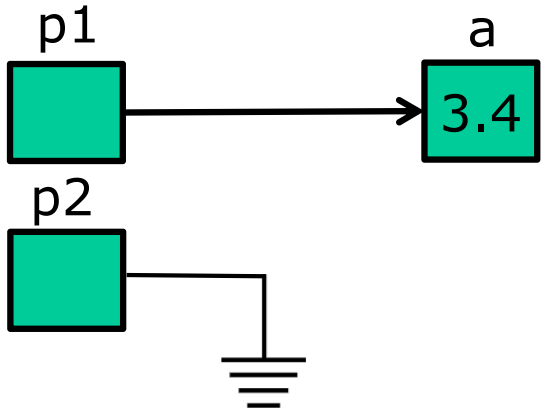
- Inizializzazione

```
float *p = NULL;
```



- Assegnamento fra puntatori

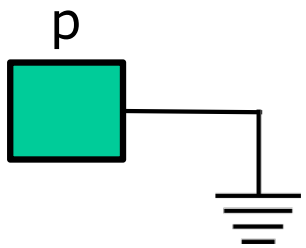
```
float a=3.4;
float *p1=NULL,*p2=NULL;
p1 = &a;
```



Operazioni sui puntatori

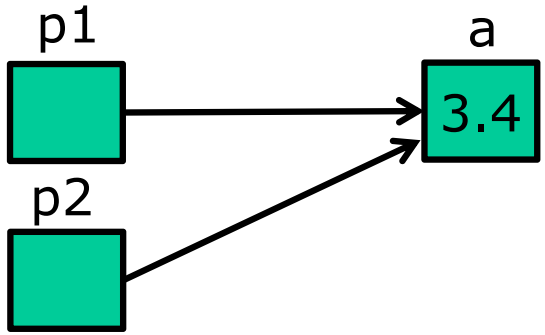
- Inizializzazione

```
float *p = NULL;
```



- Assegnamento fra puntatori

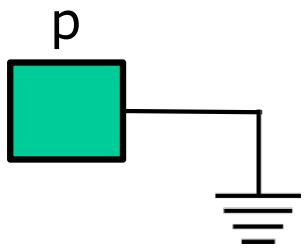
```
float a=3.4;
float *p1=NULL,*p2=NULL;
p1 = &a;
p2 = p1;
```



Operazioni sui puntatori

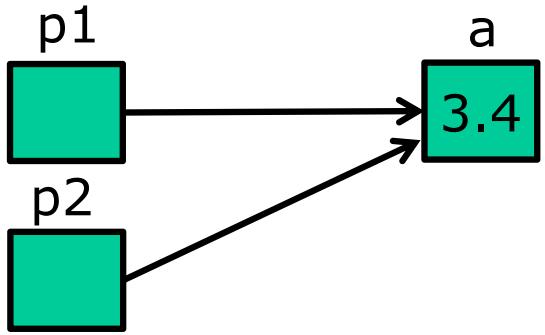
- Inizializzazione

```
float *p = NULL;
```



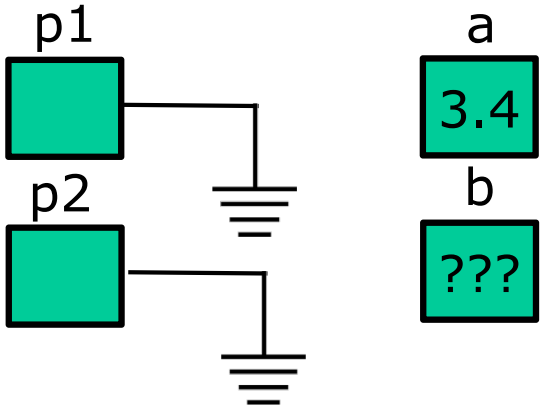
- Assegnamento fra puntatori

```
float a=3.4;
float *p1=NULL,*p2=NULL;
p1 = &a;
p2 = p1;
```



- Assegnamento con dereferenziazione

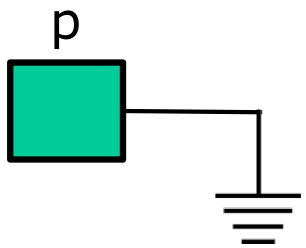
```
float a=3.4,b;
float *p1=NULL,*p2=NULL;
```



Operazioni sui puntatori

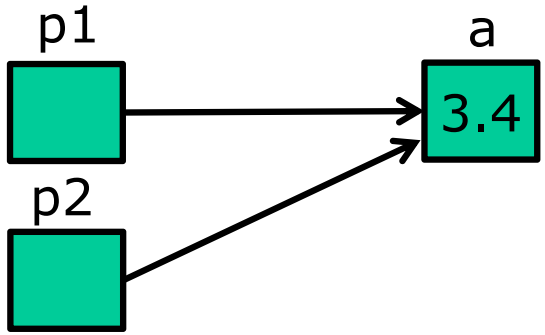
- Inizializzazione

```
float *p = NULL;
```



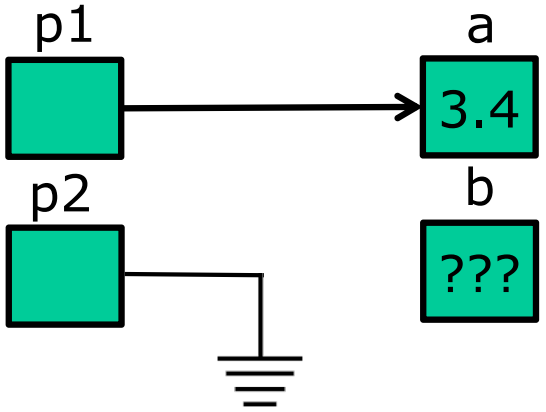
- Assegnamento fra puntatori

```
float a=3.4;
float *p1=NULL,*p2=NULL;
p1 = &a;
p2 = p1;
```



- Assegnamento con dereferenziazione

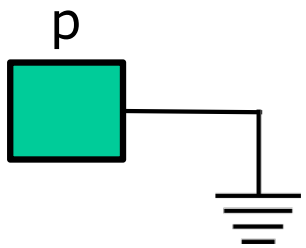
```
float a=3.4,b;
float *p1=NULL,*p2=NULL;
p1 = &a;
```



Operazioni sui puntatori

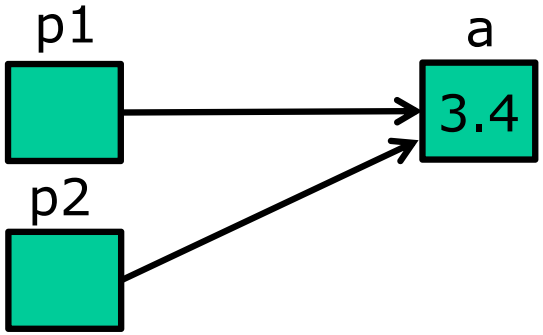
- Inizializzazione

```
float *p = NULL;
```



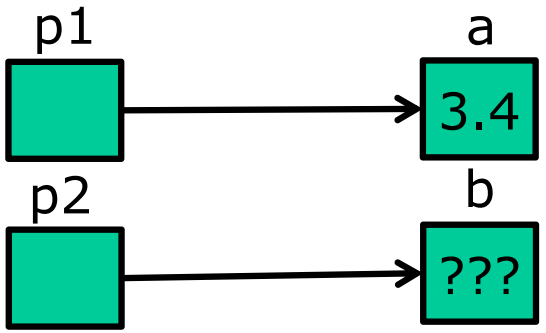
- Assegnamento fra puntatori

```
float a=3.4;
float *p1=NULL,*p2=NULL;
p1 = &a;
p2 = p1;
```



- Assegnamento con dereferenziazione

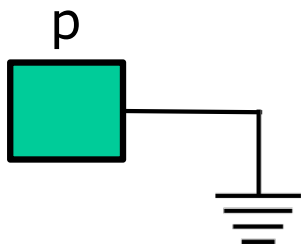
```
float a=3.4,b;
float *p1=NULL,*p2=NULL;
p1 = &a;
p2 = &b;
```



Operazioni sui puntatori

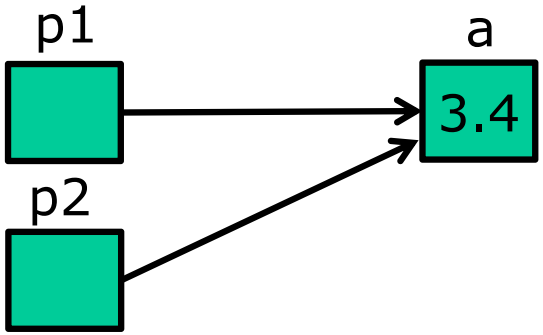
- Inizializzazione

```
float *p = NULL;
```



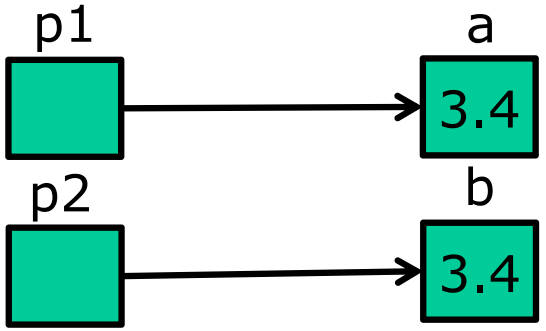
- Assegnamento fra puntatori

```
float a=3.4;
float *p1=NULL,*p2=NULL;
p1 = &a;
p2 = p1;
```



- Assegnamento con dereferenziazione

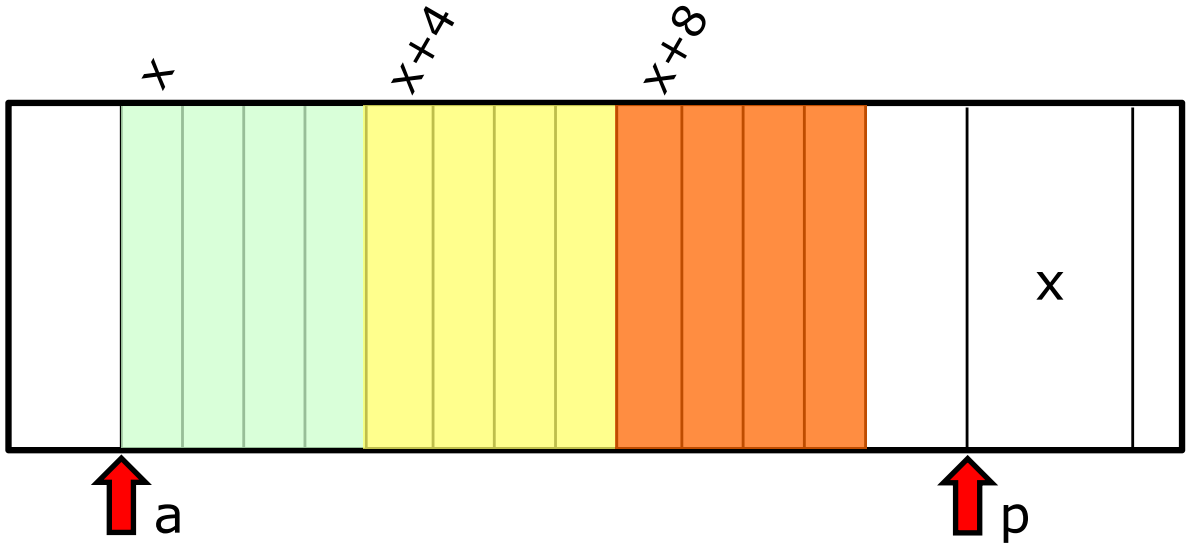
```
float a=3.4,b;
float *p1=NULL,*p2=NULL;
p1 = &a;
p2 = &b;
*p2=*p1;
```



Aritmetica dei puntatori

- Il C consente di effettuare somme e sottrazioni sui puntatori

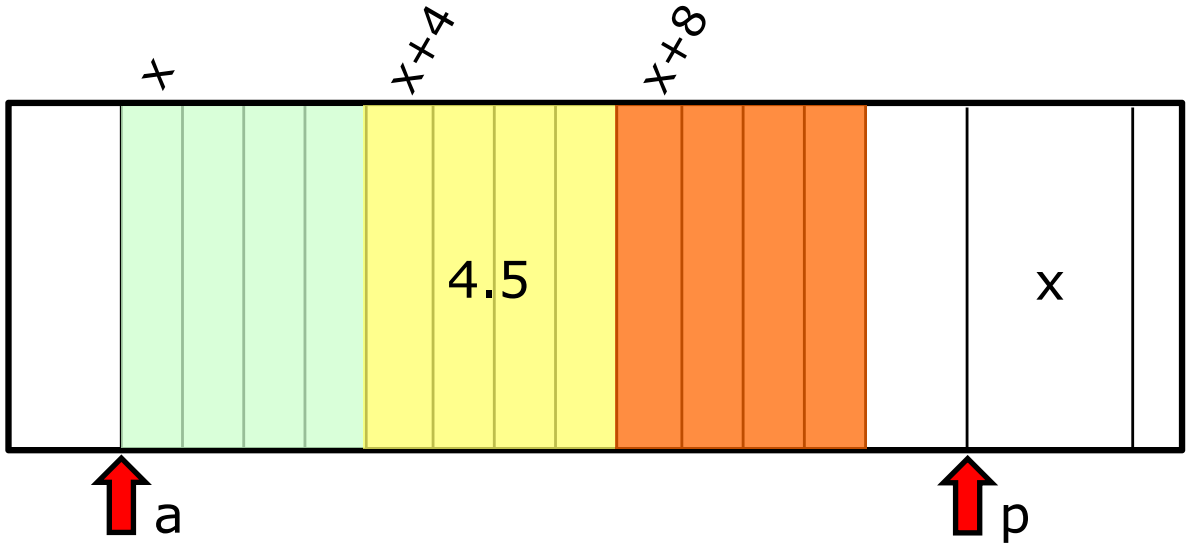
```
float *p;
float a;
p=&a;
```



Aritmetica dei puntatori

- Il C consente di effettuare somme e sottrazioni sui puntatori

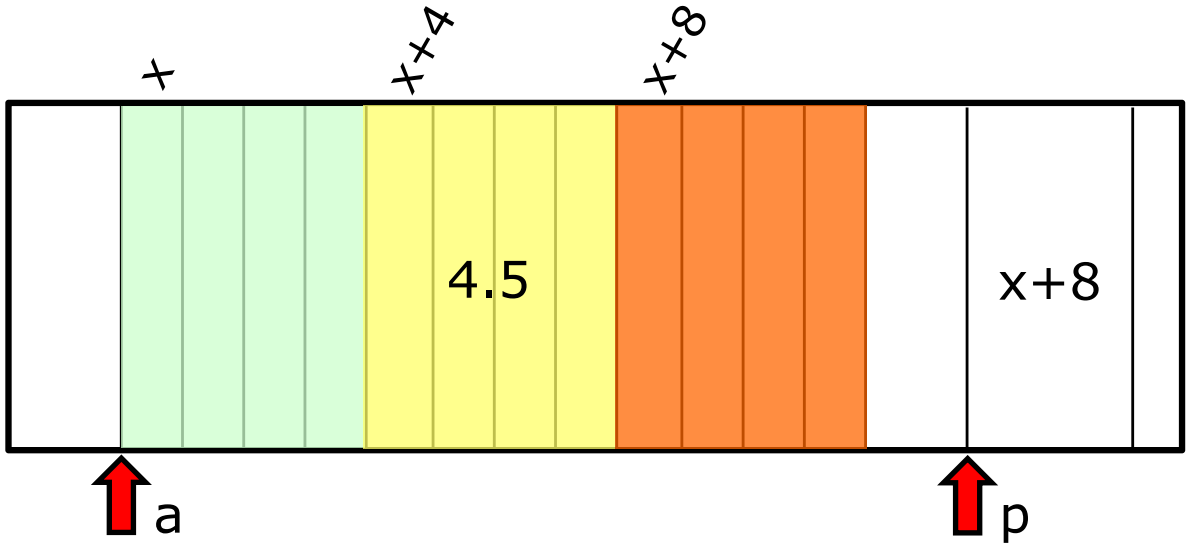
```
float *p;
float a;
p=&a;
*(p+1)=4.5;
```



Aritmetica dei puntatori

- Il C consente di effettuare somme e sottrazioni sui puntatori

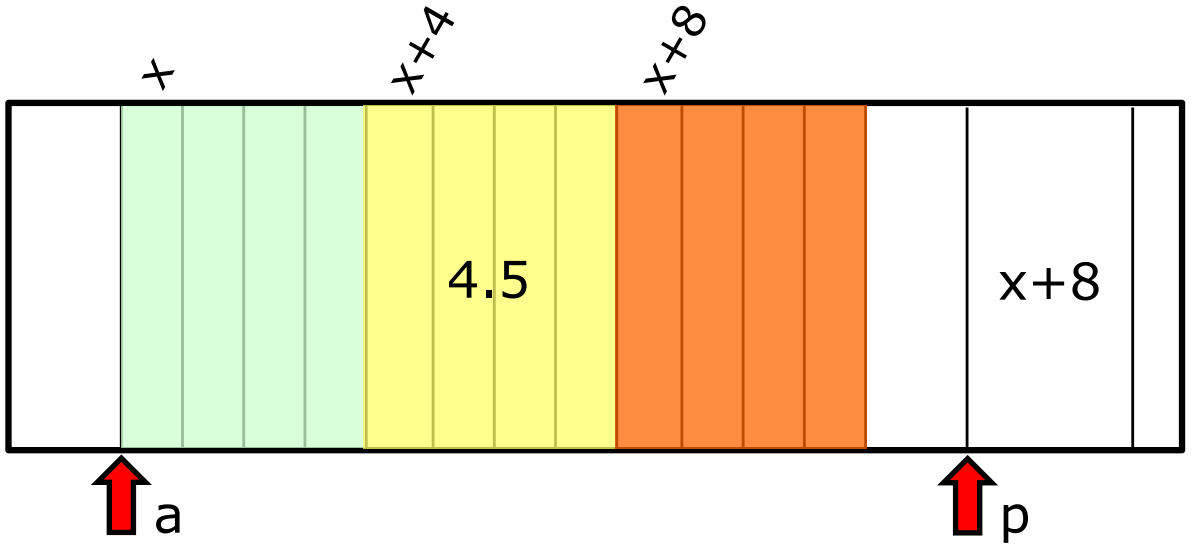
```
float *p;
float a;
p=&a;
*(p+1)=4.5;
p=p+2;
```



Aritmetica dei puntatori

- Il C consente di effettuare somme e sottrazioni sui puntatori

```
float *p;
float a;
p=&a;
*(p+1)=4.5;
p=p+2;
```



- Sintassi

```
puntatore = puntatore+x
```

- Semantica
  - incrementa/decrementa l'indirizzo contenuto nel puntatore di x **posizioni**
  - la dimensione di ogni posizione, dipende dal tipo del puntatore (es., 4 byte per un float, 1 byte per un char)

Ricordate la funzione `sizeof`?

```
sizeof(<arg>)
```

- Se <arg> è
  - un tipo di dato, ritorna la quantità di memoria (in byte) necessaria per rappresentare un valore di quel tipo
  - una variabile scalare, ritorna la quantità di memoria (in byte) occupata da quella variabile
  - un array, ritorna la quantità di memoria (in byte) occupata dall'intero array

La funzione `sizeof`: esempi

```
float f,v[5],*pf;
char c,*pc;

printf("%lu\n",sizeof(f)); //stampa “4”
printf("%lu\n",sizeof(v)); //stampa “20” (5*4 byte)
printf("%lu\n",sizeof(pf)); //stampa “8” (64 bit)
printf("%lu\n",sizeof(*pf)); //stampa “4”
printf("%lu\n",sizeof(c)); //stampa 1
printf("%lu\n",sizeof(pc)); //stampa “8” (64 bit)
printf("%lu\n",sizeof(char)); //stampa “1”
printf("%lu\n",sizeof(float)); //stampa “4”
```

Puntatori a struct

- Sia p il puntatore ad un tipo struct, allora:

`(*p).campo` = **`p->campo`**

- Esempio

```
//sia p un puntatore definito come segue
struct { float a;} *p, var;
P = &var;

//le seguenti notazioni sono equivalenti
(*p).a = 4.5;
p->a = 4.5;
```

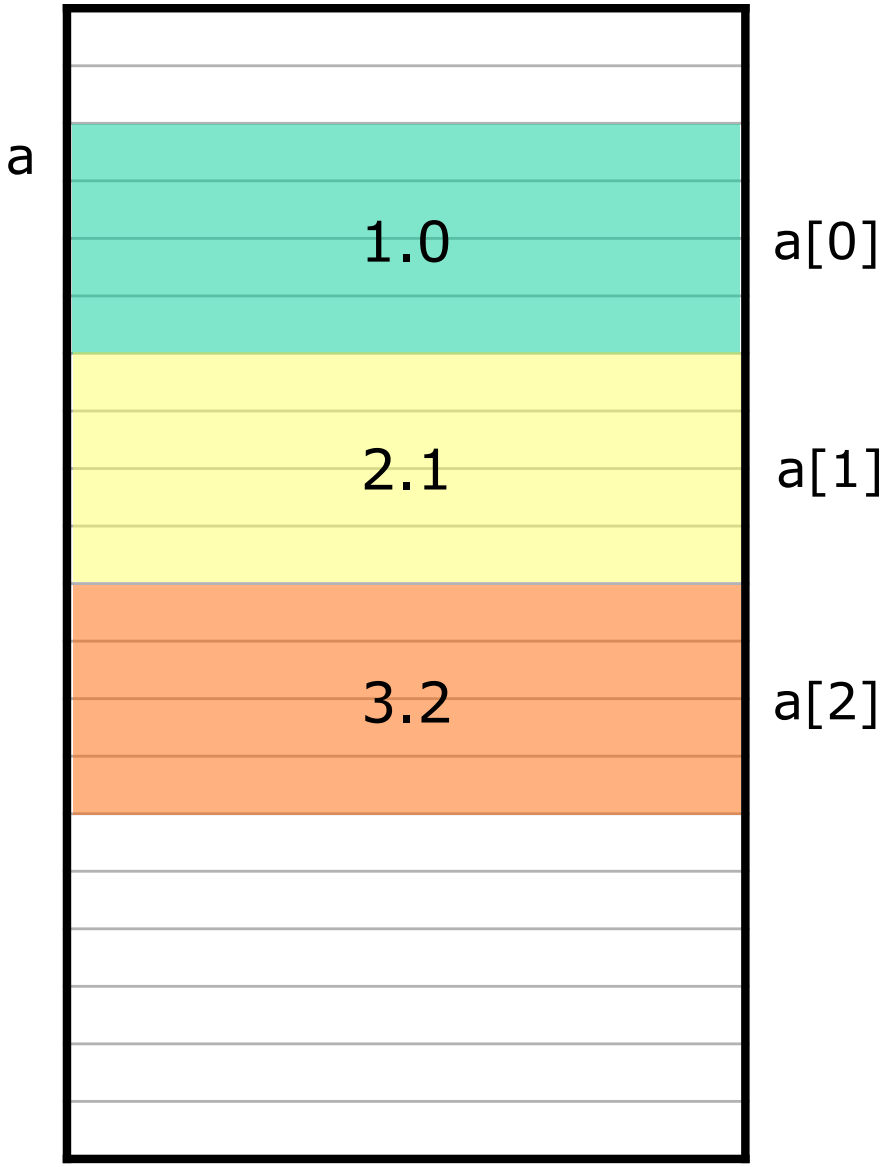
Array e puntatori

Gli array nella memoria centrale

- Un array viene memorizzato come un *blocco contiguo* a partire da un indirizzo di partenza (**indirizzo base**)
- Il nome della variabile array (senza specificare la posizione con []) equivale l'indirizzo di partenza (del primo elemento dell'array)

- Esempio

```
float a[3]={1.0,2.1,3.2};
```



Usare gli array come puntatori

- E' possibile accedere ad un array attraverso i puntatori

```
float a[5];
for (int i=0; i<5; i++)
        scanf("%f",a+i);
```

- Più in generale, se `a` è una variabile di tipo array

- Dimensione e ***offset*** di un array

```
float a[5];
float *p = &a[3];
printf("dimensione a: %lu", sizeof(a)/sizeof(float)); //stampa 5
printf("posizione p: %lu", p-a); //stampa 3
```

Ancora su array e puntatori…

```
int *p, a[5];
p = a;
```

- Abbiamo già visto scrivere `a` all'interno di un'istruzione, significa scrivere un indirizzo di memoria (l'indirizzo base dell'array `a`)
- Tuttavia `a` e `p` non sono proprio la stessa cosa: a è un valore puntatore (un indirizzo) costante (non modificabile)
  - a `p` è possibile assegnare un valore (un indirizzo di memoria) mentre non è possibile farlo con `a`
  - posso scrivere `a` al posto di `p` in tutte le istruzioni che **non modificano** il valore di `p`, ma che lo *utilizzano* soltanto

Ecco perché…

- … non è possibile usare l'operatore = per copiare il contenuto di un array o di una stringa

```
float a[5],b[5];
…
b=a; //errore: b ed a sono solo gli indirizzi base
```

- … non si usa & per leggere una stringa: il nome della variabile è già il suo indirizzo!
- … non si può usare l'operatore == per confrontare due array o due stringhe

```
char a[20],b[20];
…
if(a==b) //vero se a e b hanno lo stesso indirizzo
…
```

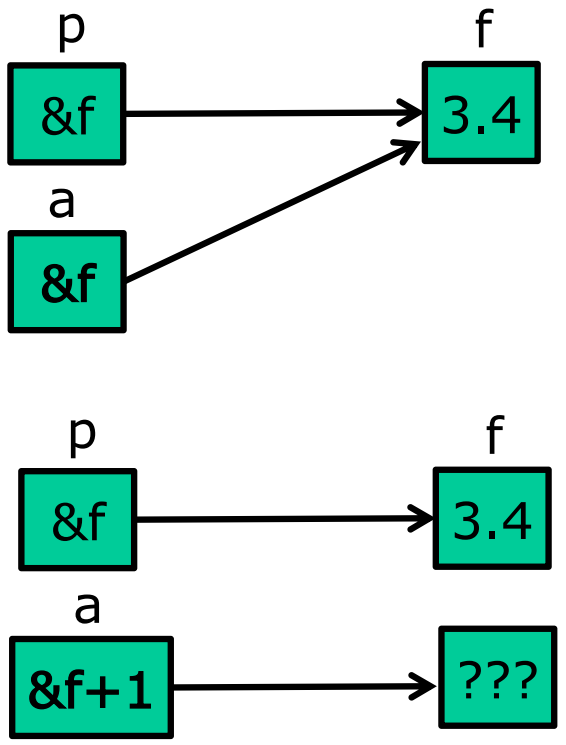
Funzioni e puntatori

Puntatori come parametri

- In C, i parametri di una funzione possono essere dei puntatori.
- Cosa succede? Niente di diverso dal solito…

```
void fun(float *a){
   a++;
}

int main() {
   float f=3.4,*p;
   p=&f;
   fun(p);
   return 0;
}
```

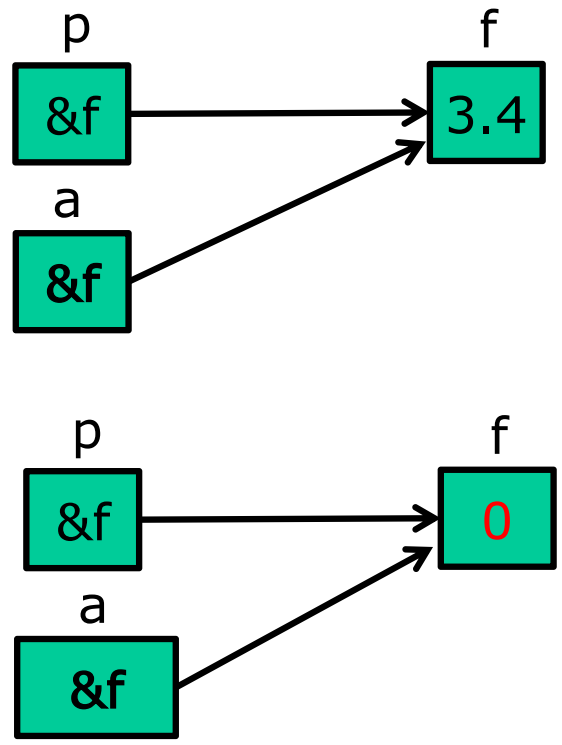


Puntatori come parametri (2)

- Attenzione però…

```
void fun(float *a){
   *a=0;
}

int main() {
   float f=3.4,*p;
   p=&f;
   fun(p);
   return 0;
}
```

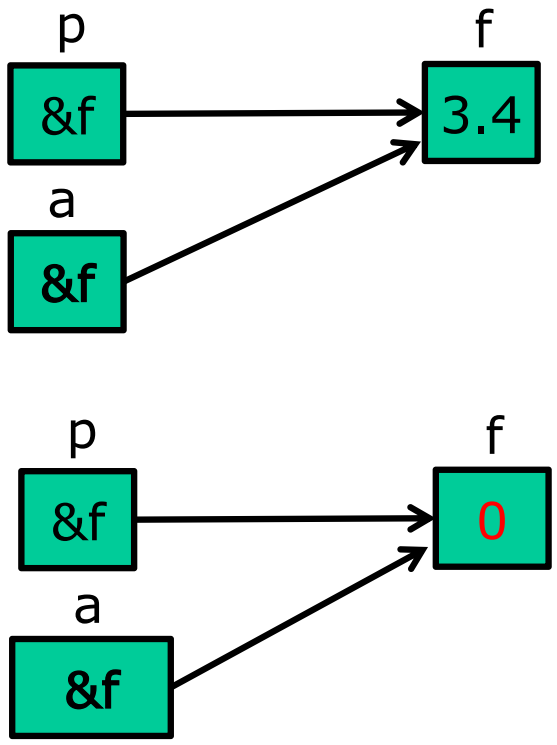


Puntatori come parametri (2)

- Ecco perché si parla (un po' impropriamente) di passaggio per indirizzo e per valore:

```
void fun(float *a){
   *a=0;
}

int main() {
   float f=3.4;
   fun(&f);
   return 0;
}
```



Array come parametri

- In C, ci sono diverse notazioni che si possono usare per dichiarare un parametro di tipo array:

```
void f(float a[DIM]);
void f(float a[]);
void f(float *a);
```

- La differenza è solo estetica: tutte e tre le varianti si comportano allo stesso modo.
- Quindi, passare un argomento array ad una funzione equivale a passare l'indirizzo base dell'array
  - Attenzione a non modificare l'array accidentalmente!
  - Attenzione all'uso di `sizeof()`!

Funzioni che ritornano un puntatore

- Può una funzione ritornare un puntatore?

```
int *array_vuoto(int n)
{
    int a[n];
    for (int i = 0; i < n; i++)
          a[i]=0;

    return a;
}
```

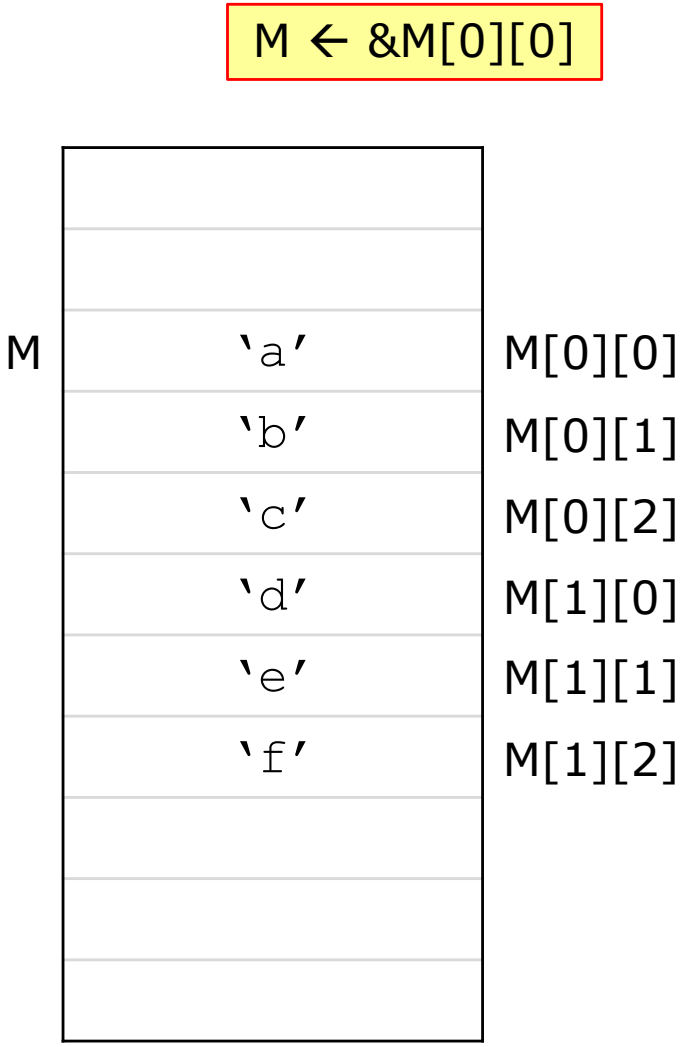
- Sintatticamente la funzione è corretta, ma cosa succede quando la funzione è conclusa?

Matrici e puntatori

Matrici e puntatori

- In maniera simile agli array, le matrici vengono memorizzate come un *blocco contiguo* a partire da un **indirizzo base, riga dopo riga.**
- Il nome della variabile matrice (usato senza specificare un indice di riga e colonna con []), è l'indirizzo base della matrice.
- Esempio

```
char M[2][3]={{'a','b','c'},{'d','e','f'}};
```



Matrici e puntatori (2)

```
float M[5][4];
```

- `M` **non** è un puntatore float: `float *`
  - `float *p = M; //genera un warning`
- `M` è un puntatore ad un array di 4 float: `float (*)[4]`
  - `float (*p)[4] = M; //p punta alla prima riga di M`
- E' comunque possibile puntare ad un singolo elemento di M
  - `float *p = &M[i][j]; //punta all'elemento M[i][j]`
- Le operazioni di aritmetica sul puntatore hanno un effetto diverso a seconda di come è dichiarato il puntatore

```
float *p1 = &M[0][0];
float (*p2)[4] = M;
```

  - `p1+i` è l'indirizzo dell'*i-esimo elemento* di M (letta per righe)
  - `p2+i` denota l'indirizzo di partenza dell'i-esima riga di M
  - `p1+i` equivale a `&M[0][0]+i`
  - `p2+i` equivale a `M+i`

Matrici e puntatori (3)

- E' possibile accedere ad una matrice attraverso gli indirizzi

```
float M[5][3];
for (int i=0; i<5; i++)
    for (int j=0; j<3; j++)
        scanf("%f",&M[0][0]+i*3+j);
```

- Più in generale, se `M` è una matrice di `r` righe ed `c` colonne

`M[i][j]` = `*(&M[0][0]+i*c+j)`

- Calcolare indice di riga e colonna di una

```
float M[r][c]; //r e c costanti
float *p = &M[rr][cc];  // puntatore ad un elemento di M
…
int i = (p - &M[0][0])/c; //calcola l'indice di riga (div intera!)
int j = (p - &M[0][0])%c; //calcola l'indice di colonna
```

Matrici e funzioni

- Come per gli array, ci sono tre notazioni equivalenti per dichiarare un parametro formale di tipo matrice:

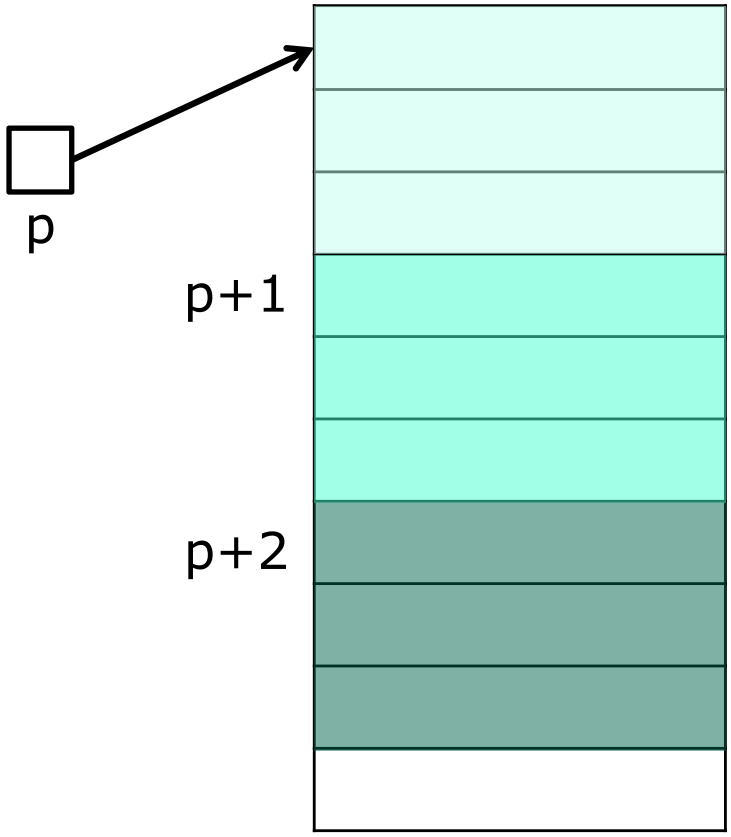
```
void f(float a[R][C]);
void f(float a[][C]);
void f(float (*a)[C]);
```

- La differenza è solo estetica! Tutte e tre le varianti si comportano allo stesso modo, perché il parametro formale è **un puntatore ad un array di dimensione fissa**.
- E' sempre necessario specificare la seconda dimensione della matrice (numero di colonne), senza la quale l'aritmetica degli indirizzi non potrebbe essere applicata correttamente.

Array di puntatori vs puntatore ad array

- Un puntatore ad un array è sostanzialmente una matrice

```
float (*p)[3];
```



- Un array di puntatori è più flessibile e permette di creare una matrice con righe di dimensione diverse

```
float *p[3];
```