Introduzione al C
Lez. 3

Puntatori

Variabili e memoria

Variabile tradizionale

Es: `int a = 10;`
Proprietà:
- nome: a
- tipo: int
- valore: 10
- dimensione in byte: 4 (usare sizeof(tipo))
- indirizzo: 104

&a → operatore di redirezione “&”. Denota l'indirizzo di memoria della var a

Una variabile (puntatore) può contenere un indirizzo

Es:

```
int *p;
double *p;
char *p;
int **p;        ?
```

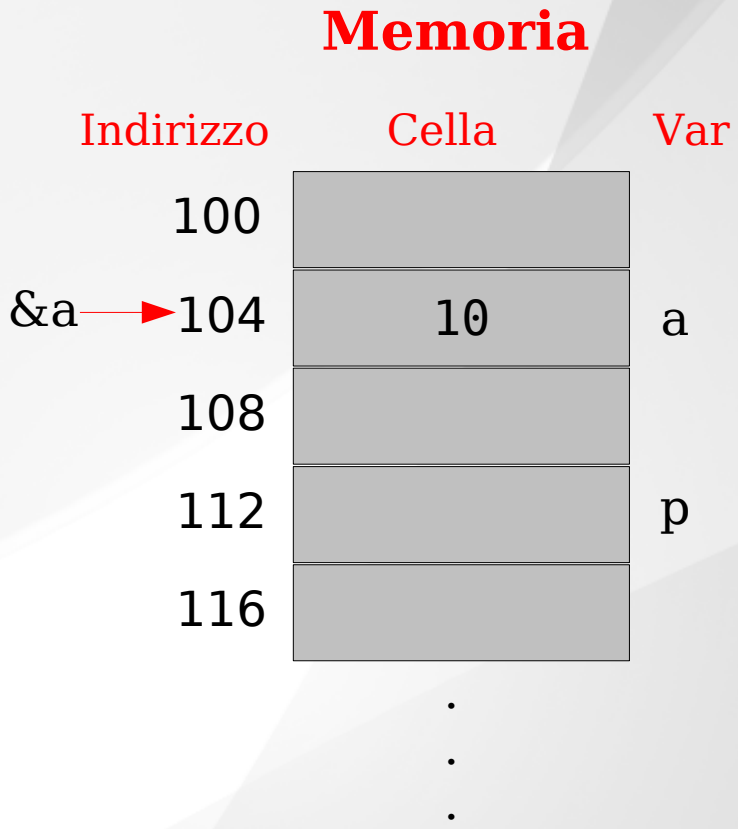
Operazioni con i puntatori

Referenziazione: &a

Denota l'indirizzo di memoria di a

Dereferenziazione: *p

Denota la cella puntata da p



Operazioni con i puntatori

Referenziazione: &a

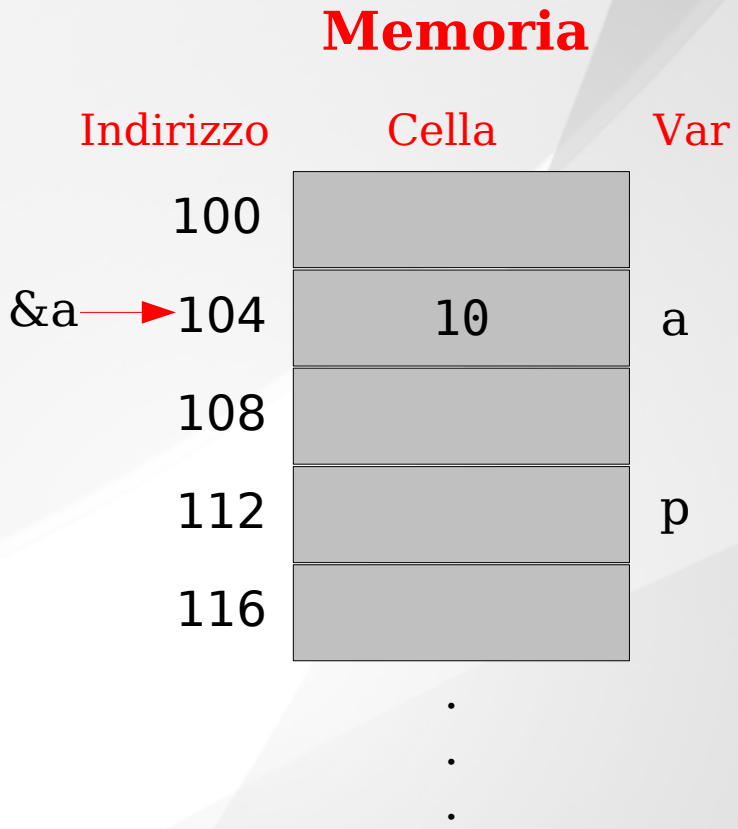
Denota l'indirizzo di memoria di a

Dereferenziazione: *p

Denota la cella puntata da p

Esempio

```
int a = 10;
int *p;

p = &a;          ?
```



Operazioni con i puntatori

Referenziazione: &a

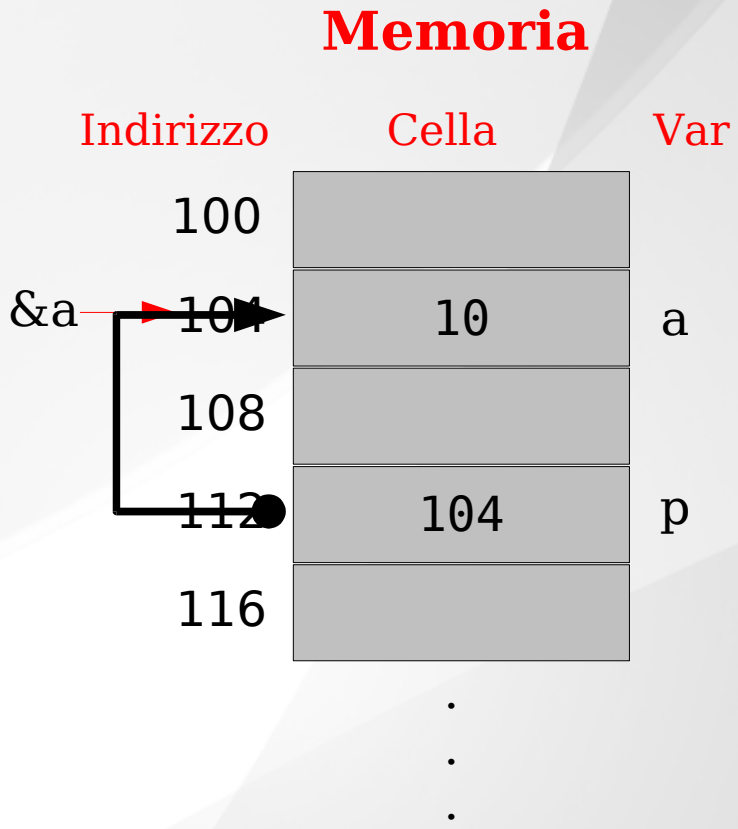
Denota l'indirizzo di memoria di a

Dereferenziazione: *p

Denota la cella puntata da p

Esempio

```
int a = 10;
int *p;

p = &a;
```



Operazioni con i puntatori

Referenziazione: &a

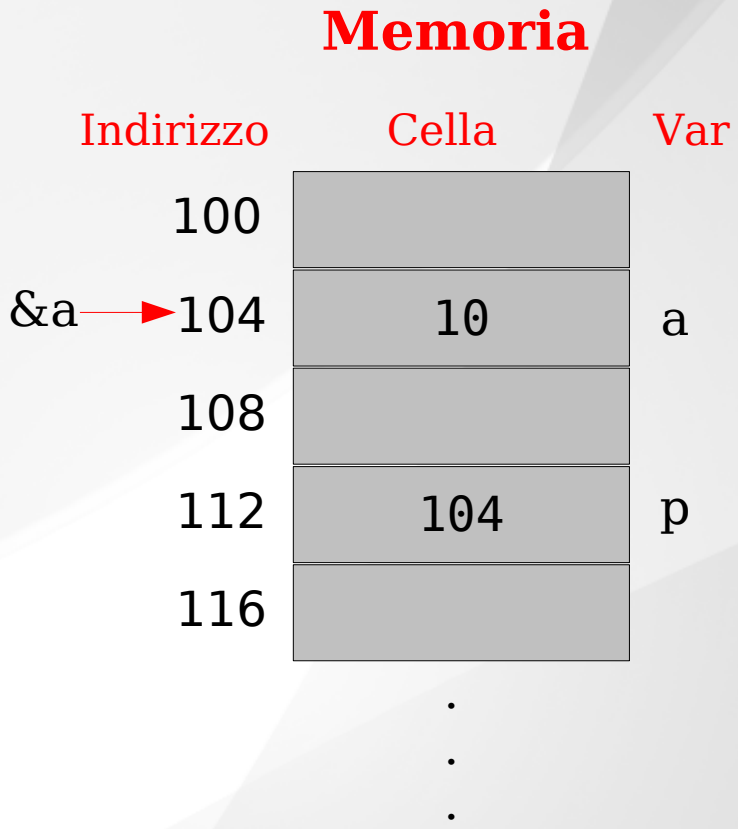
Denota l'indirizzo di memoria di a

Dereferenziazione: *p

Denota la cella puntata da p

Esempio

```
int a = 10;
int *p;

p = &a;
&p                  ?
```



Operazioni con i puntatori

Referenziazione: &a

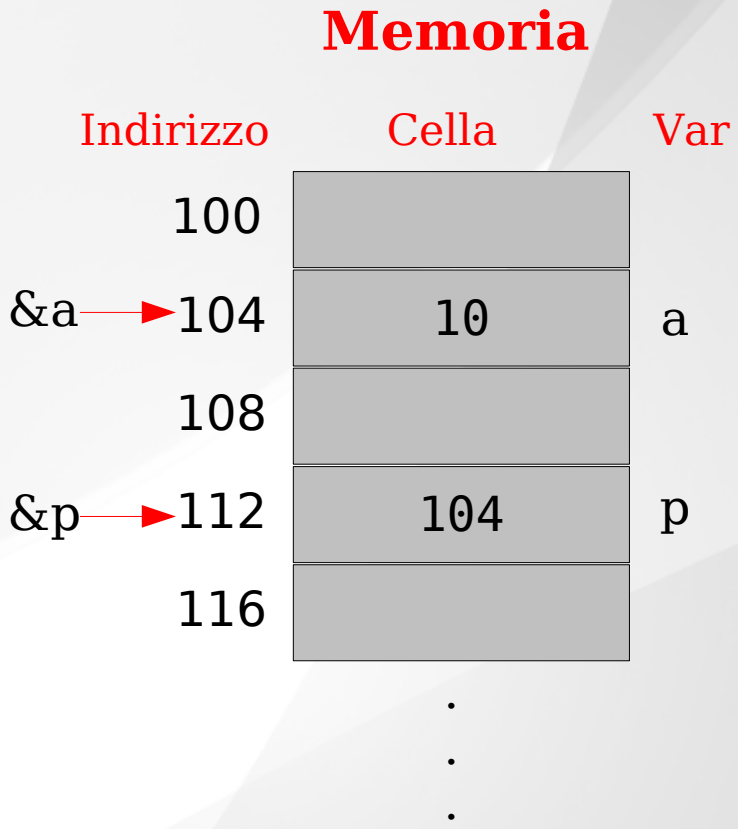
Denota l'indirizzo di memoria di a

Dereferenziazione: *p

Denota la cella puntata da p

Esempio

```
int a = 10;
int *p;

p = &a;
&p
```



Operazioni con i puntatori

Referenziazione: &a

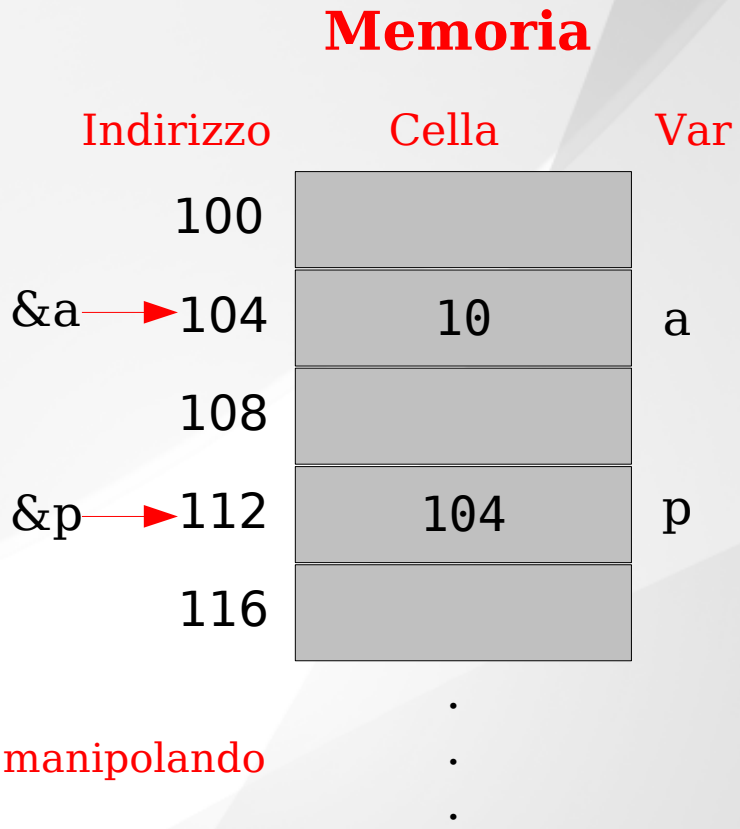
Denota l'indirizzo di memoria di a

Dereferenziazione: *p

Denota la cella puntata da p

Esempio

```
int a = 10;
int *p;

p = &a;
&p
```



Accedere e/o modificare il contenuto di una variabile manipolando direttamente il suo puntatore

```
*p                  ?
*p = 20;            ?
*p = *p + 4;        ?
```

Aritmetica dei puntatori

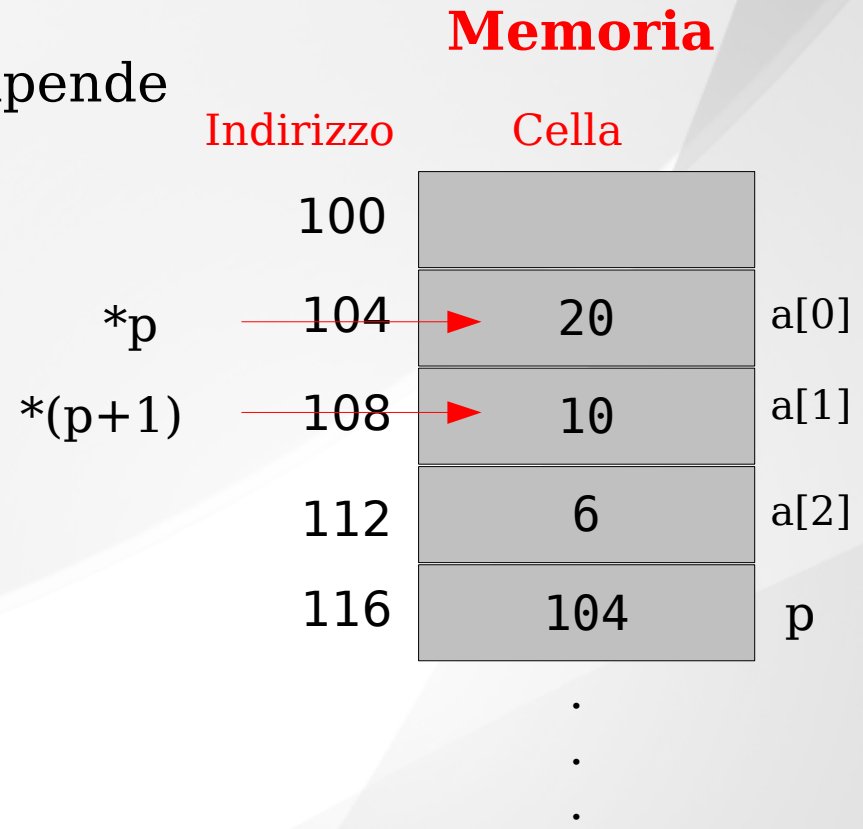
Si possono utilizzare espressioni puntatore che includono gli usuali operatori aritmetici (+, -, ++, –, ecc.)

Aritmetica dei puntatori

Si possono utilizzare espressioni puntatore che includono gli usuali operatori aritmetici (+, -, ++, –, ecc.)

L'incremento, in termini di indirizzo, dipende dal tipo puntato.

```
int a[3], *p = &a[0];
*(p+i) <===> a[i]
```



Aritmetica dei puntatori

Si possono utilizzare espressioni puntatore che includono gli usuali operatori aritmetici (+, -, ++, –, ecc.)

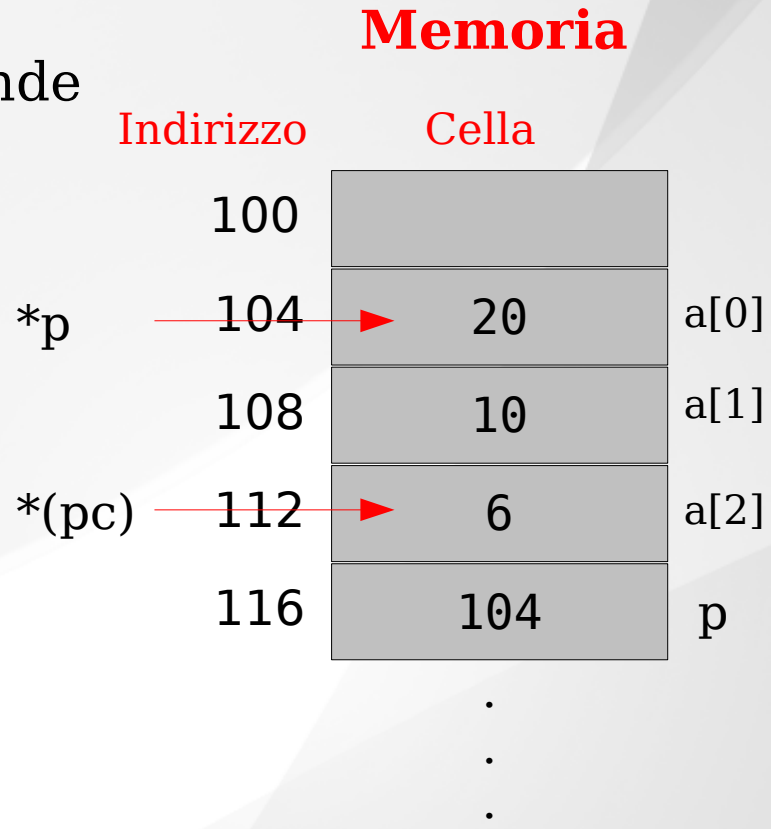
L'incremento, in termini di indirizzo, dipende dal tipo puntato.

```
int a[3], *p = &a[0];
*(p+i) <===> a[i]
```

Altro esempio

```
int *pc, a[3], *p = &a[0];
...
pc = p + 2; // ind 112! 104 + 2*4
```

Un int occupa 4 byte.
pc punta al terzo elemento di a



Aritmetica dei puntatori

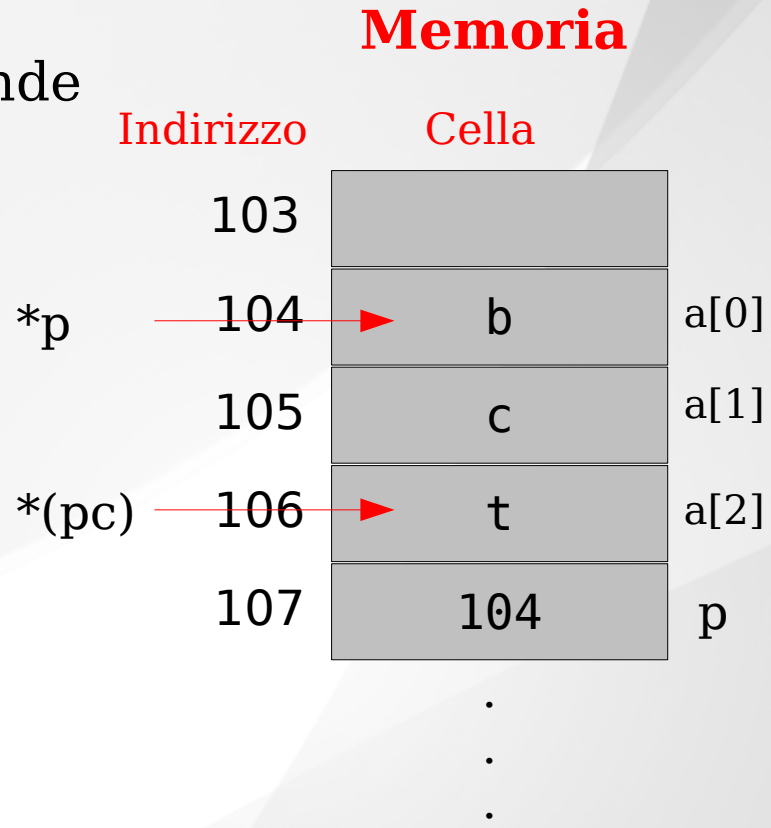
Si possono utilizzare espressioni puntatore che includono gli usuali operatori aritmetici (+, -, ++, –, ecc.)

L'incremento, in termini di indirizzo, dipende dal tipo puntato.

```
char a[3], *p = &a[0];
*(p+i) <===> a[i]
```

Altro esempio

```
char *pc, a[3], *p = &a[0];
...
pc = p + 2; // ind 106! 104+1*2
```

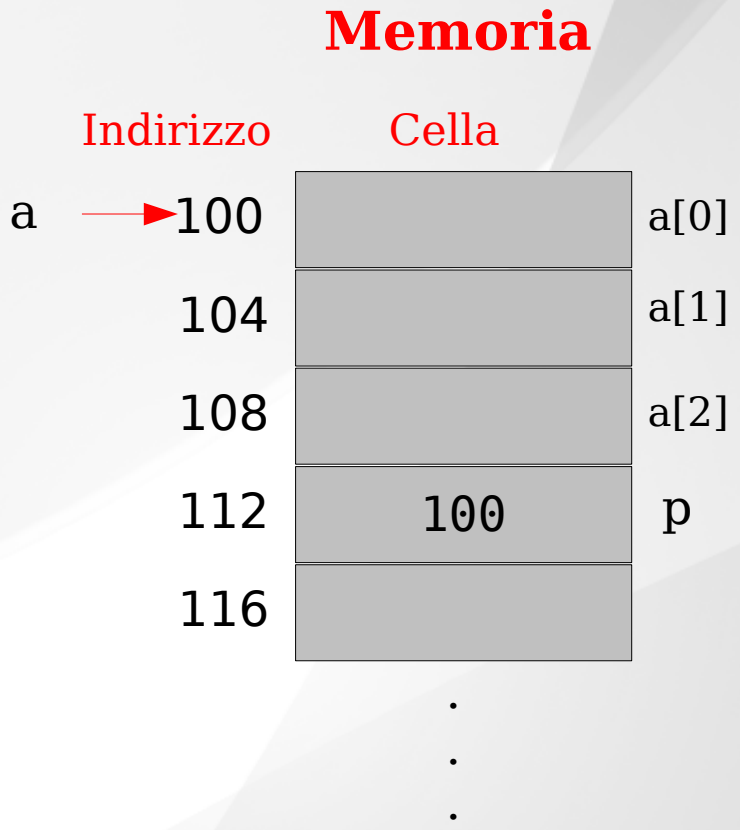


Un char occupa un byte.
pc punta ancora al terzo elemento di a ma l'indirizzo è diverso

Array vs Puntatori

Il nome di un array, è il puntatore (costante) al primo elemento dell'array. a <====> &a[0]

```
int a[3], *p;

p = a; // si può assegnare a p
```



Array vs Puntatori

Il nome di un array, è il puntatore (costante) al primo elemento dell'array. a <===> &a[0]

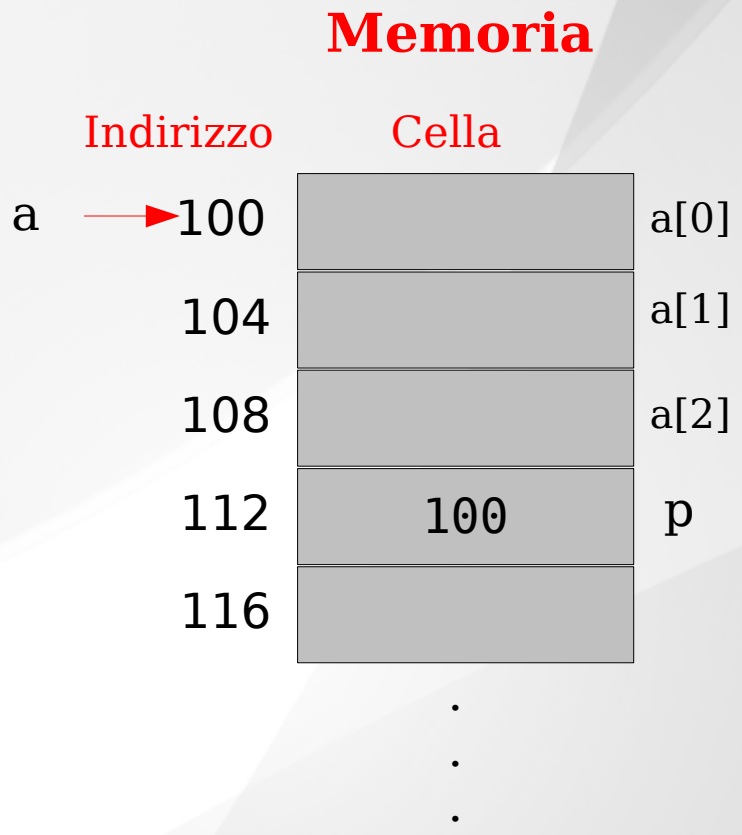
```
int a[3], *p;

p = a; // si può assegnare a p
```

L'operatore [-] è un'abbreviazione...

`a[2]` e `*(a+2)` sono equivalenti.
Entrambi denotano il terzo elemento di `a`.

Si può usare [-] con qualunque variabile puntatore.

```
p[2] <===> a[2];
```



Array vs Puntatori

Il nome di un array, è il puntatore (costante) al primo elemento dell'array. a <====> &a[0]

```
int a[3], *p;

p = a; // si può assegnare a p
```

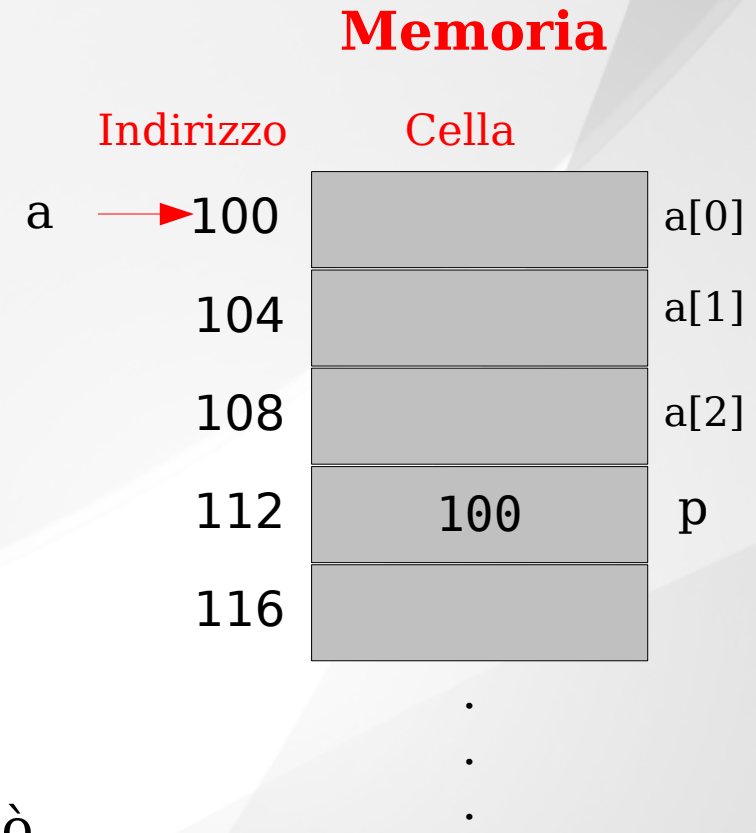
L'operatore [-] è un'abbreviazione...

a[2] e *(a+2) sono equivalenti.
Entrambi denotano il terzo elemento di a.

Si può usare [-] con qualunque variabile puntatore.

C'è un'unica differenza tra a e p: non si può cambiare l'indirizzo puntato da a.

```
int a[3], b[3];
a=b; No!
```



Array vs Puntatori (2)

4 frammenti equivalenti per la somma dei valori di un array:

```
int i, sum = 0, a[3], *p = a;
…

1)
for(i = 0; i < 3; i++)
    sum += a[i];
```

Array vs Puntatori (2)

4 frammenti equivalenti per la somma dei valori di un array:

```
int i, sum = 0, a[3], *p = a;
…

1)
for(i = 0; i < 3; i++)
    sum += a[i];

2)
for(i = 0; i < 3; i++)
    sum += *(a+i);
```

Array vs Puntatori (2)

4 frammenti equivalenti per la somma dei valori di un array:

```
int i, sum = 0, a[3], *p = a;
…

1)
for(i = 0; i < 3; i++)
    sum += a[i];

2)
for(i = 0; i < 3; i++)
    sum += *(a+i);

3)
for(i = 0; i < 3; i++)
    sum += p[i];
```

Array vs Puntatori (2)

4 frammenti equivalenti per la somma dei valori di un array:

```
int i, sum = 0, a[3], *p = a;
…

1)
for(i = 0; i < 3; i++)
    sum += a[i];

2)
for(i = 0; i < 3; i++)
    sum += *(a+i);

3)
for(i = 0; i < 3; i++)
    sum += p[i];

4)
for(p = a; p < a + 3; p++)
    sum += *p;
```

Esercizio 1

Scrivere un programma che crea un array di 10 interi casuali nell'intervallo `[0,100]` e stampa le coppie indirizzo-valore per ogni cella dell'array a.

Strumenti utili:
- `srand( time(NULL) );` // rimepiazzate time(NULL) con una variabile letta da input come seme se usate un sistema Windows
- `(rand() % 101)`
- un indirizzo si stampa con `printf("%p", p);` dove p è un indirizzo

Provare ad eseguirlo due volte e notare che gli indirizzi cambiano.

Funzioni e passaggio di parametri

Tutti i parametri delle funzioni C sono sempre passati per valore.

- il loro valore viene copiato al momento del passaggio

- eventuali modifiche nel corpo della funzione non si ripercuotono nell'originale!

Funzioni e passaggio di parametri

Tutti i parametri delle funzioni C sono sempre passati per valore.

- il loro valore viene copiato al momento del passaggio

- eventuali modifiche nel corpo della funzione non si ripercuotono nell'originale!

Esempio:

```
void foo(int a) {
   a++;
}

void main() {
   int x = 10;
   foo(x);
   printf(“%d”, x); // 10 o 11?
}
```

Funzioni e passaggio di parametri

Tutti i parametri delle funzioni C sono sempre passati per valore.

- il loro valore viene copiato al momento del passaggio

- eventuali modifiche nel corpo della funzione non si ripercuotono nell'originale!

Esempio:

```
void foo(int a) {
   a++;
}

void main() {
   int x = 10;
   foo(x);
   printf("%d", x); // 10 o 11?
}
```

10! foo modifica una copia di x!

Funzioni e passaggio di parametri

Come posso fare in modo che le modifiche siano visibili fuori dal corpo della funzione chiamata?

Si simula il passaggio per riferimento con i puntatori!

- Si passa un puntatore anziché il suo valore
- questo viene copiato nel parametro formale
- si può modificare il valore nella cella puntata

Funzioni e passaggio di parametri

Come posso fare in modo che le modifiche siano visibili fuori dal corpo della funzione chiamata?

Si simula il passaggio per riferimento con i puntatori!

- Si passa un puntatore anziché il suo valore
- questo viene copiato nel parametro formale
- si può modificare il valore nella cella puntata

Esempio:

```
void foo(int *a) {
   (*a)++;
}

void main() {
   int x = 10;
   foo(&x);
   printf(“%d”, x); // 10 o 11?
}
```

Funzioni e passaggio di parametri

Come posso fare in modo che le modifiche siano visibili fuori dal corpo della funzione chiamata?

Si simula il passaggio per riferimento con i puntatori!

- Si passa un puntatore anziché il suo valore
- questo viene copiato nel parametro formale
- si può modificare il valore nella cella puntata

Esempio:

```
void foo(int *a) {
   (*a)++;
}

void main() {
   int x = 10;
   foo(&x);
   printf("%d", x); // 10 o 11?
}
```

11! foo modifica il valore contenuto in x attraverso una copia di &x!

Funzioni e passaggio di parametri

Funzione che scambia il contenuto di due variabili

```
void scambio(int a, int b) {
   int tmp;
   tmp = a;
   a = b;
   b = tmp;
}

void main() {
   int m, n;
   m = 1;
   n = 2;
   scambio(m, n);
   printf("m = %d n = %d\n");
}
```

Output?

1) m = 2 n = 1

2) m = 1 n = 2

Funzioni e passaggio di parametri

Funzione che scambia il contenuto di due variabili

```
void scambio(int a, int b) {
   int tmp;
   tmp = a;
   a = b;
   b = tmp;
}

void main() {
   int m, n;
   m = 1;
   n = 2;
   scambio(m, n);
   printf("m = %d n = %d\n");
}
```

Output?

1) ~~m = 2 n = 1~~

2) m = 1 n = 2

Funzioni e passaggio di parametri

Funzione che scambia il contenuto di due variabili

```
void scambio(int a, int b) {
   int tmp;
   tmp = a;
   a = b;
   b = tmp;
}

void main() {
   int m, n;
   m = 1;
   n = 2;
   scambio(m, n);
   printf("m = %d n = %d\n");
}
```

Output?

1) ~~m = 2 n = 1~~

2) m = 1 n = 2

NON FUNZIONA! Lo scambio viene fatto sulle copie!

Funzioni e passaggio di parametri

Funzione che scambia il contenuto di due variabili

```
void scambio(int *a, int *b) {
    int tmp;
    tmp = *a;
    *a = *b;
    *b = tmp;
}

void main() {
    int m, n;
    m = 1;
    n = 2;
    scambio(&m, &n);
    printf("m = %d n = %d\n");
}
```

Output?

1) m = 2 n = 1

2) m = 1 n = 2

Funzioni e passaggio di parametri

Funzione che scambia il contenuto di due variabili

```
void scambio(int *a, int *b) {
   int tmp;
   tmp = *a;
   *a = *b;
   *b = tmp;
}

void main() {
   int m, n;
   m = 1;
   n = 2;
   scambio(&m, &n);
   printf("m = %d n = %d\n");
}
```

Output?

1) m = 2 n = 1

2) ~~m = 1 n = 2~~

Funzioni e passaggio di parametri

Funzione che scambia il contenuto di due variabili

```
void scambio(int *a, int *b) {
   int tmp;
   tmp = *a;
   *a = *b;
   *b = tmp;
}

void main() {
   int m, n;
   m = 1;
   n = 2;
   scambio(&m, &n);
   printf("m = %d n = %d\n");
}
```

Output?

1) m = 2 n = 1

2) ~~m = 1 n = 2~~

Perché ora è corretta?

Funzioni e passaggio di parametri: Array

Gli array sono sempre passati per riferimento!
Ciò che viene passato (e copiato) è il puntatore al primo elemento dell'array.

Funzioni e passaggio di parametri: Array

Gli array sono sempre passati per riferimento!
Ciò che viene passato (e copiato) è il puntatore al primo elemento dell'array.

```
void init(int a[], int len) { // passare lunghezza!

   int i;
   for(i=0; i<len; i++;)
      a[i] = 0;              // == *(a+i) = 0;

}

void main() {
   int a[10];
   init(a, 10);
   // qui a[i] = 0 per ogni i
}
```

Funzioni e passaggio di parametri: Array

Le due scritture

```
void init(int a[], int len) {
   int i;
   for(i=0; i<len; i++;)
      a[i] = 0;
}
```

e

```
void init(int *a, int len) {
   int i;
   for(i=0; i<len; i++;)
      a[i] = 0;
}
```

sono equivalenti. Si preferisce la prima per leggibilità.

Esercizio 2

Scrivere una funzione

```
void minmax(int a[], int len, int *min, int *max)
```

che, dato un array `a` e la sua lunghezza `len`, scrive il valore del massimo e del minimo elemento di `a` nelle celle puntate da `min` e `max`.

Scrivere un programma che utilizzi `rand()` per la generazione di un array di 10 elementi interi casuali in `[0,100]` e provare la funzione stampando i valori del massimo e del minimo.

Esercizio 3

Scrivere una funzione

```
int * minimo(int a[], int len)
```

che, dato un array `a` di lunghezza `len`, restituisca un puntatore al minimo valore contenuto in `a`.

Scrivere un programma che utilizzi `rand()` per la generazione di un array di 10 elementi interi casuali in `[0,100]` e provare la funzione stampando il valore del minimo e il suo indirizzo.

Esercizio 4 (pomeriggio)

All'interno del main dichiarare due array a e b di 10 elementi ciascuno. Stampare a video “contigui” (risp. “`non contigui`”) se le celle di memoria di a e b sono contigue in memoria. Stampare inoltre il nome dell'array che tra i due ha l'indirizzo più piccolo.

Esercizio 5 (pomeriggio)

Scrivere una funzione

```
void converti(int tempo, int *h, int *m, int *s)
```

che, dato un orario `tempo` espresso in secondi, calcoli l'orario corrispondende espresso in ore, minuti e secondi e lo memorizzi nelle tre variabili `h`, `m` e `s`.

Scrivere un programma che riceva da stdin `tempo` ed esegua `converti` stampando a video il risultato.