Introduzione al C

Lez. 2

Funzioni e Puntatori

Esercizio 1: test primalità

Scrivere un programma che prenda in input un intero *n*, e stampa “SI” se *n* è primo, “NO” altrimenti. (N.B.: un intero *n* è primo se è solo i suoi unici divisori interi sono 1 e *n*)

Esercizio 1: test primalità

```c
#include <stdio.h>

int main()
{
  int n, fattore, primo = 1;
  int limite;

  scanf("%d", &n );
  fattore = 2;

  while( primo && fattore < n ){
    if ( (n % fattore) == 0 ) primo = 0;
    fattore++;
  }

   if ( primo ) printf("SI\n");
   else printf("NO\n");

   return 0;
}
```

Esercizio 1: test primalità

```c
#include <stdio.h>
#include <math.h>
int main()
{
  int n, fattore, primo = 1;
  int limite;

  scanf("%d", &n );
  fattore = 2;
  limite = (int)sqrt(n);
  while( primo && fattore <= limite ){
    if ( (n % fattore) == 0 ) primo = 0;
    fattore++;
  }

   if ( primo ) printf("SI\n");
   else printf("NO\n");

   return 0;
}
```

Esercizio 2: inversione array

Scrivere un programma che esegua i seguenti 3 passi in sequenza:
• Legga in input un intero *n* e inizializzi un'array A con *n* valori presi in input dall'utente.
• *Inverta l'array A in loco,* ossia scambi il contenuto della prima e dell'ultima cella, della seconda e della penultima, etc...
• Stampi l'array invertito in output

Esempio:

Input: n= 5 e A = {3, 1, 4, 0, 0}
Output: 0 0 4 1 3

Esercizio 2: inversione array

```
#include <stdio.h>
#define MAXSIZE 10000

int main() {

 int n,i,j,scambio;

 /* Lettuta input da tastiera */
 scanf("%d", &n );
 if ( n < 1 || n > MAXSIZE )  {
   printf("Errore: l'array deve avere dimensione tra 1 e %d\n",
        MAXSIZE );
    return 1;
 }

 for ( i = 0; i < n; i++ )
   scanf("%d", &input[i]);



 ....
```

Esercizio 2: inversione array

```
 ...
 /* INVERSIONE IN LOCO     */

 for ( i = 0; i < n/2; i++ )   {
       j = (n-1)-i;

      scambio = A[i];
      A[i] = A[j];
      A[j] = scambio;
    }

 /* OUTPUT     */

 for ( i = 0; i < n; i++ )   printf("%d ", output[i]);
   printf("\n");

    return 0;
}
```

Esercizio

Scrivere un programma che richieda in input un intero n (assumendo n < 100), e inizializzi un array con n valori casuali binari (0 e 1). Quindi si chiede all'utente di immettere n valori 0/1 e si restituisce il numero di entry che coincidono tra i due array.

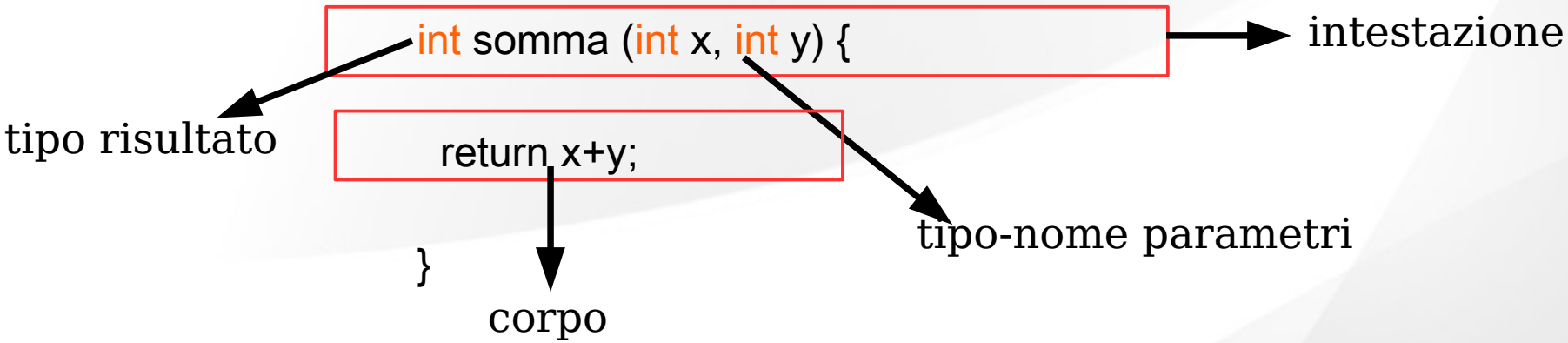
Strumenti utili:
- `srand( time(NULL) );` // rimpiazzate time(NULL) con una variabile letta da input come seme se usate un sistema Windows
- `(rand() % 2)`

Funzioni in C

Il C permette di strutturare un progamma in *funzioni*

Una definizione di funzione si compone di:

- intestazione: tipo del risultato e lista dichiarazioni parametri
- corpo: blocco di istruzioni terminato da *return*



• Il tipo di ritorno è void se la funzione non restituisce nessun valore

Funzioni in C

Per invocare una funzione occorre che questa sia stata precedentemente definita (o soltanto dichiarata):

Esempio:

```
/*somma(int,int): definizione */
int somma (int x, int y)
    { return x+y; }

int main() {
 ...
 printf("La somma di x e y è", somma(x,y));
 return 0;

}
```

Funzioni in C

Per invocare una funzione occorre che questa sia stata precedentemente definita (o soltanto dichiarata)

Esempio (*forward-declaration)*:

```
/*somma(int,int): forward-declaration */
int somma (int x, int y);
```

dichiarazione

```
int main() {
 ...
 printf("La somma di x e y è:", somma(x,y));
 return 0;

}
```

definizione

```
/* qui la definizione  */
int somma (int x, int y)
    { return x+y; }
```

Funzioni ricorsive in C

Il C permette che una funzione chiami se stessa ricorsivamente:

```
/* power(int,int): definita ricorsivamente */

int power(int n, int x){
  if (n == 0) return 1;
  else return x*power(n-1,x);
}


int main() {
 printf("x elevato a y : %d", power(y,x));
 return 0;
}
```

Funzioni in C

Ogni funzione è dotata di un *ambiente locale*:

- Variabili locali non sono visibili all'esterno
- Allocate/deallocate automaticamente all'ingresso/uscita della funzione

```
int somma(int x, int y){
  int tmp;
  tmp = x+y;
  return tmp;
}
```

variabile locale

In C il passaggio di parametri è **sempre** **per valore**:

- La funzione riceve *una copia del parametro passato*
- Quindi eventuali modifiche **NON** *si propagano al chiamante*
- Nella prossima lezione vedremo che altre modalità di passaggio (i.e., per riferimento) si possono simulare mediante l'uso dei puntatori

Puntatori

Variabile tradizionale

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

Una variabile (puntatore) può contenere un indirizzo

Es:

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

?

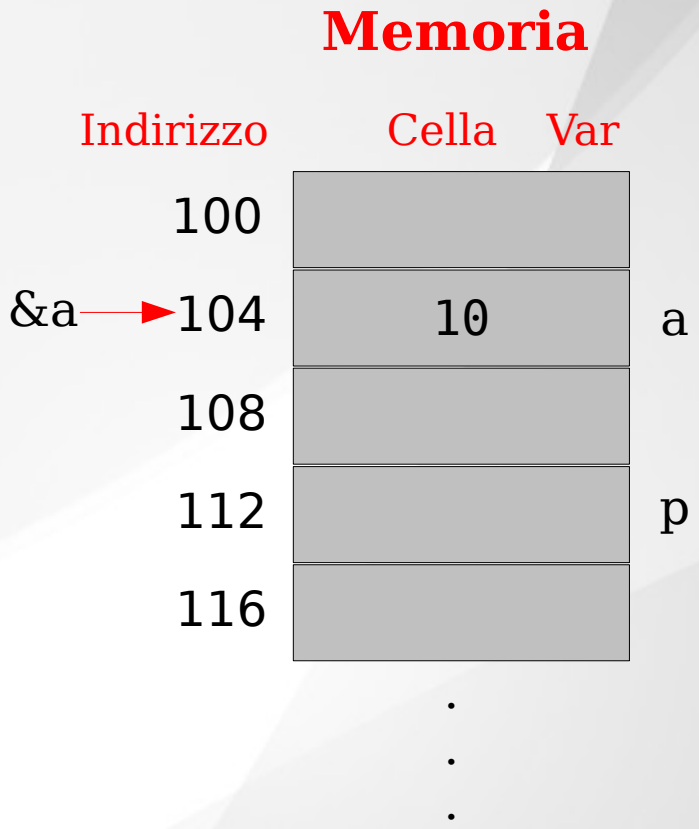
Operazioni con i puntatori

Referenziazione: &a

Denota l'indirizzo di memoria di a

Dereferenziazione: *p

Denota la cella puntata da p



Operazioni con i puntatori

Referenziazione: &a

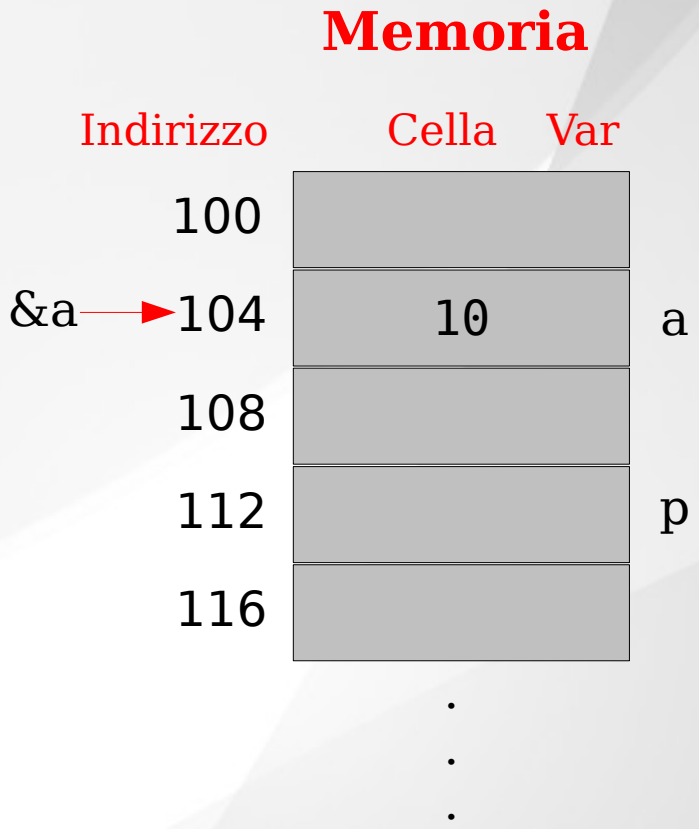
Denota l'indirizzo di memoria di a

Dereferenziazione: *p

Denota la cella puntata da p

Esempio

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

p = &a;          ?
```



Operazioni con i puntatori

Referenziazione: &a

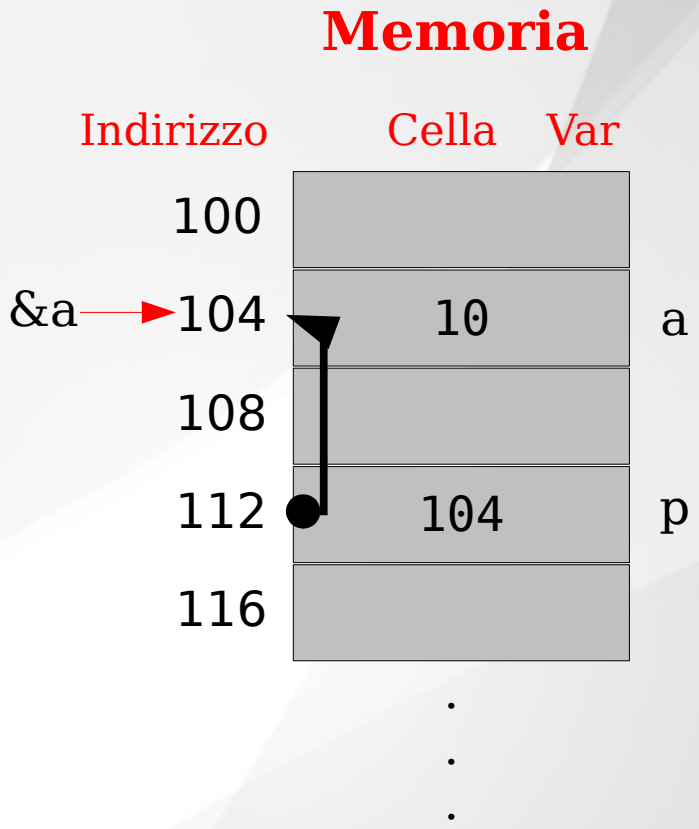
Denota l'indirizzo di memoria di a

Dereferenziazione: *p

Denota la cella puntata da p

Esempio

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

p = &a;
```



Operazioni con i puntatori

Referenziazione: &a

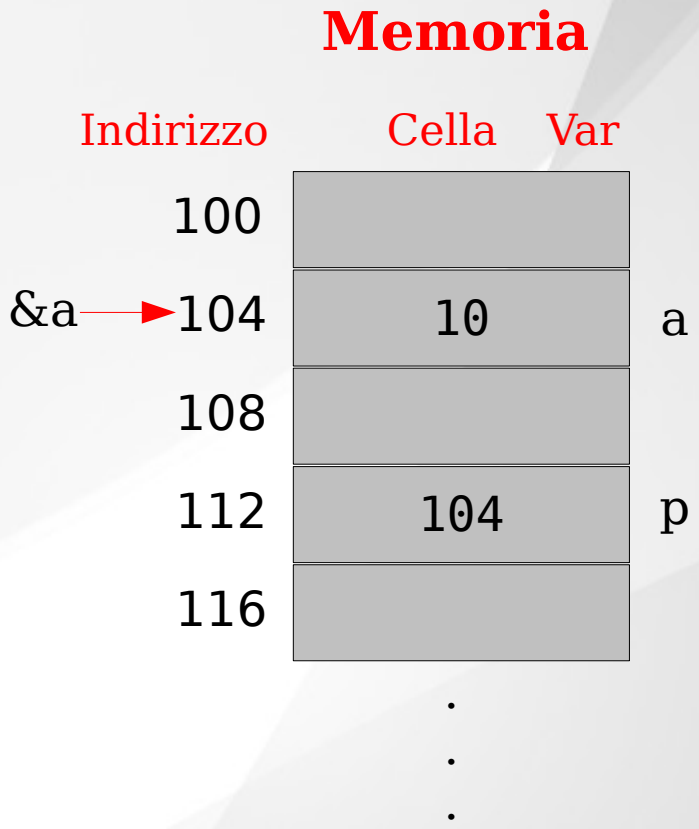
Denota l'indirizzo di memoria di a

Dereferenziazione: *p

Denota la cella puntata da p

Esempio

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

p = &a;
&p                ?
```



Operazioni con i puntatori

Referenziazione: &a

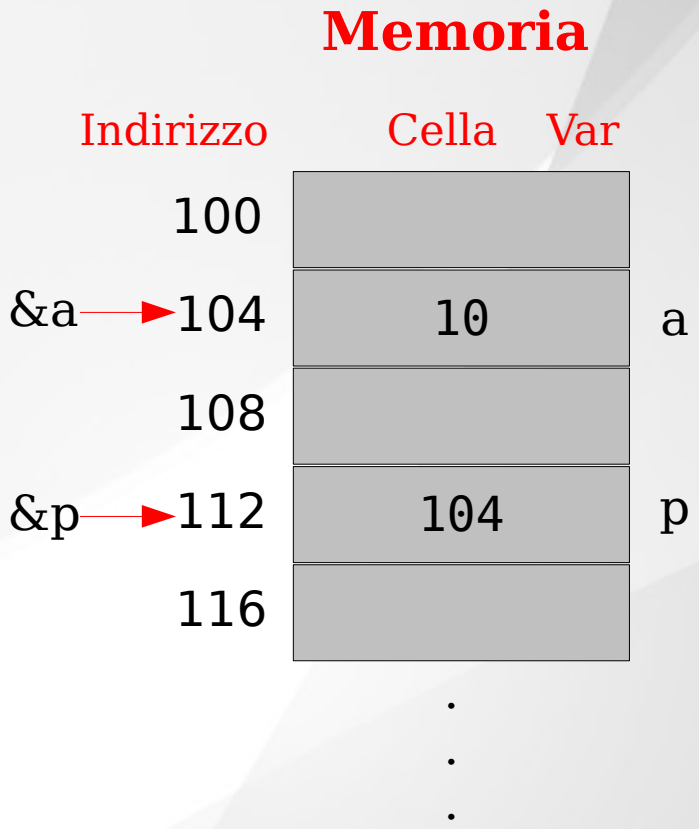
Denota l'indirizzo di memoria di a

Dereferenziazione: *p

Denota la cella puntata da p

Esempio

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

p = &a;
&p
```



Operazioni con i puntatori

Referenziazione: &a

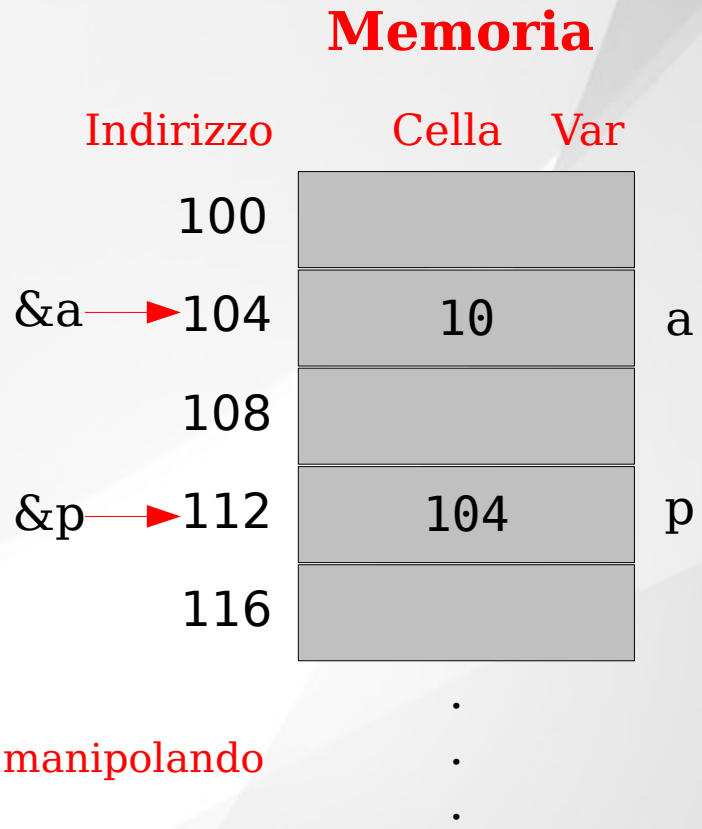
Denota l'indirizzo di memoria di a

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

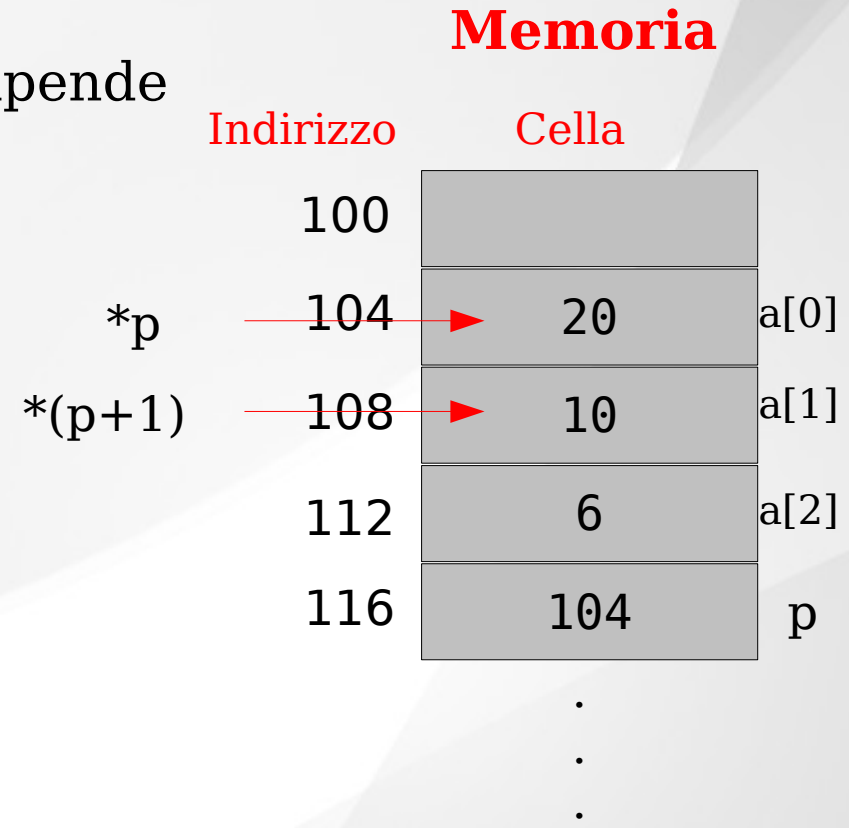
Si possono utilizzare espressioni puntatore che includono gli usuali operatori aritmetici (+, -, ++, –-, ecc.)

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.)

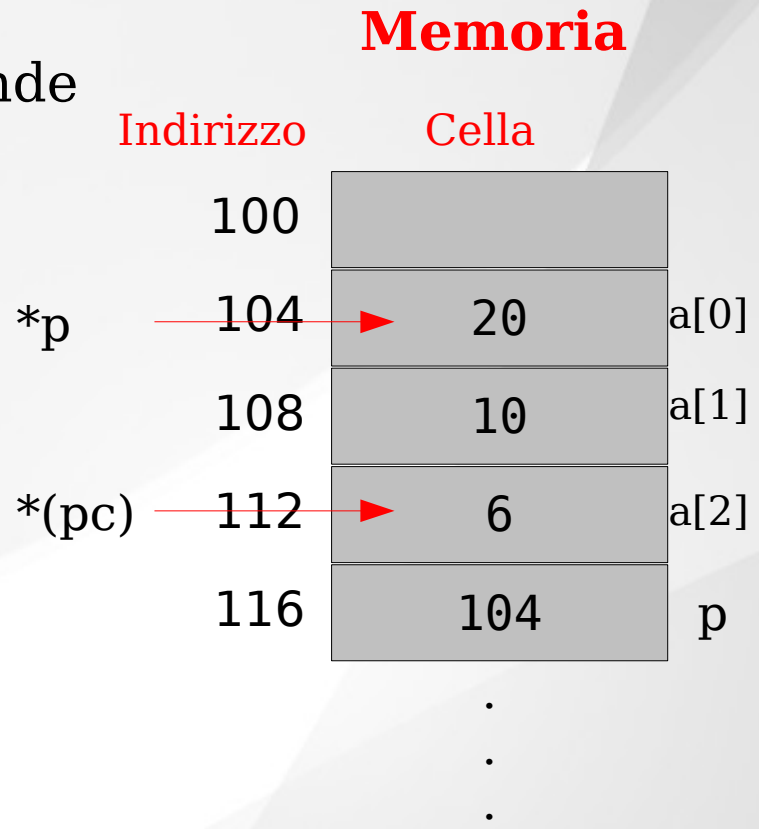
L'incremento, in termini di indirizzo, dipende dal tipo puntato.

```
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

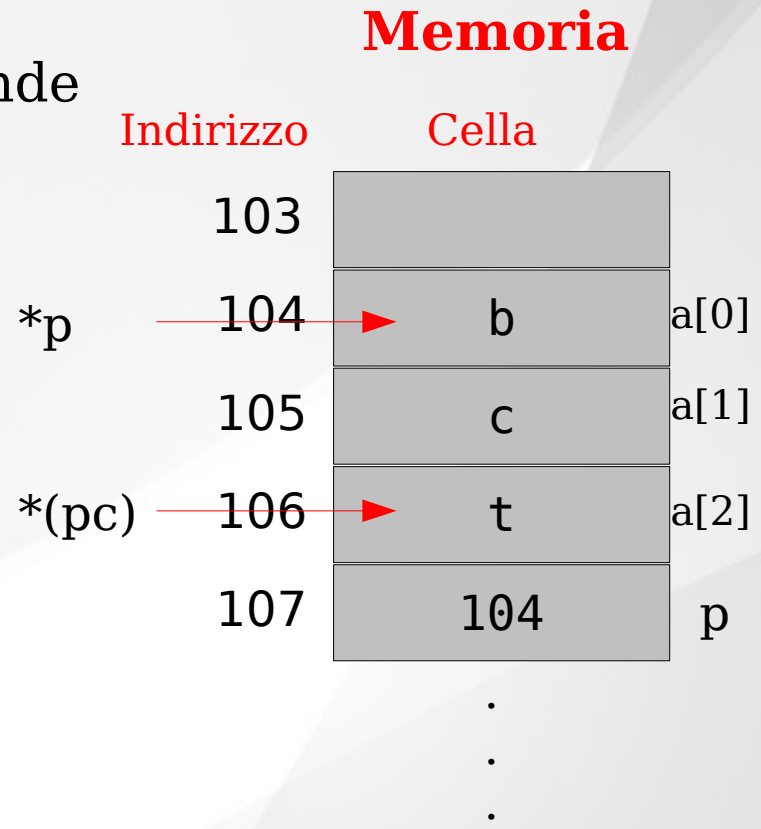
Si possono utilizzare espressioni puntatore che includono gli usuali operatori aritmetici (+, -, ++, --, ecc.)

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
```

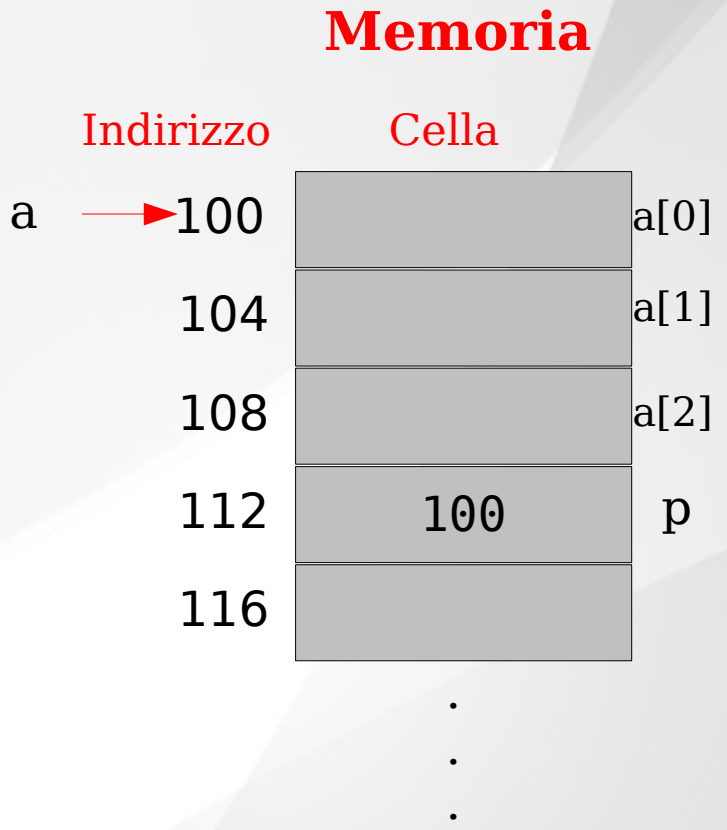


Un char occupa un byte.
pc punta ancora al terzo elemento di a ma l'indirizzo è diverso

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]

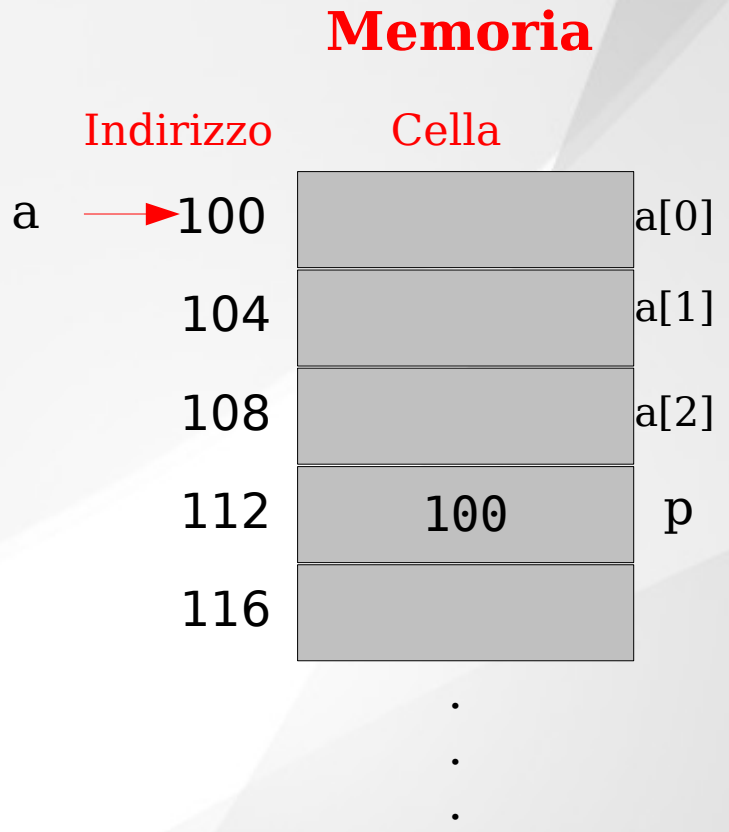
```
int a[3], *p;

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

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;
```

Esercizi

1) Scrivere una funzione ricorsiva f che, dato un intero N, restituisca la somma dei primi N interi dispari. Scrivere un semplice programma per testare la funzione che prenda in input un intero x e stampi il valore di f(x)

2) 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.

3) 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. Provare i diversi modi equivalenti visti nel lucido precedente

Funzioni utili:

- `srand( time(NULL) );`
- `(rand() % 101)`

Un indirizzo si stampa con il segnaposto %p ad esempio

```
printf(“L'indirizzo di x è %p”, &x);
```