Esercitazione di Reti degli elaboratori

Prof.ssa Chiara Petrioli

SAPIENZA
UNIVERSITÀ DI ROMA

Corso di C

Luca Iezzi
iezzi@diag.uniroma1.it

Cosa vedremo in questa lezione?

- **Le funzioni**
- **Le stringhe**
- **I puntatori**

Le funzioni

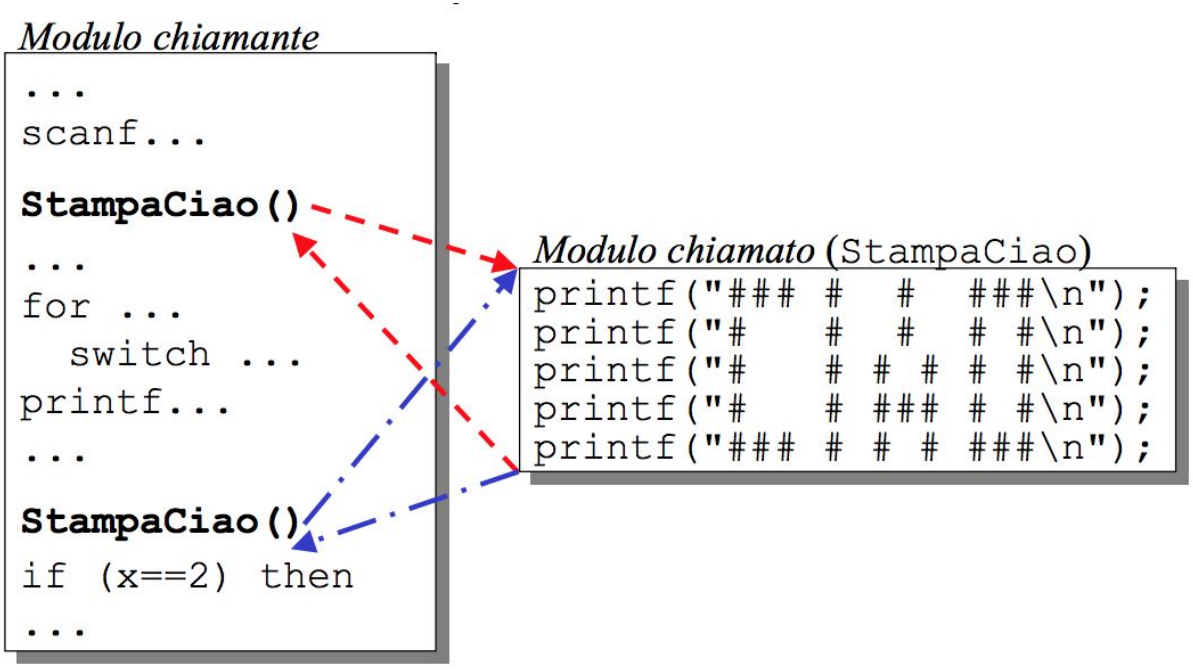
- Una funzione è un blocco di codice che può essere richiamato ed eseguito da diversi punti in un programma
- Strumento fondamentale per la scrittura di programmi complessi
- Consentono di suddividere il programma in più sottoprogrammi
- È possibile eseguire lo stesso sottoprogramma più volte, semplicemente richiamando la relativa funzione
- Fino ad ora abbiamo visto programmi composti da una sola funzione: la funzione *main*

Le funzioni
Struttura modulare

- Per semplificare la struttura di un programma complesso è possibile suddividerlo in funzioni
- Una funzione è un blocco di codice che assolve ad un compito preciso (ad es. calcola la radice quadrata) e a cui è stato dato un nome
- Un programma è composto dalla funzione *main* ed eventuali altre funzioni di supporto
- Quando una funzione richiama un'altra funzione, il chiamante viene sospeso finché il chiamato non ha terminato la sua esecuzione
- Ogni funzione può richiamare (far eseguire) qualsiasi altra funzione (anche se stessa)

Le funzioni
Struttura modulare

- Le due chiamate della funzione *StampaCiao* fanno eseguire ogni volta le istruzioni che la costituiscono (sospendendo il chiamante)



Le funzioni
I vantaggi

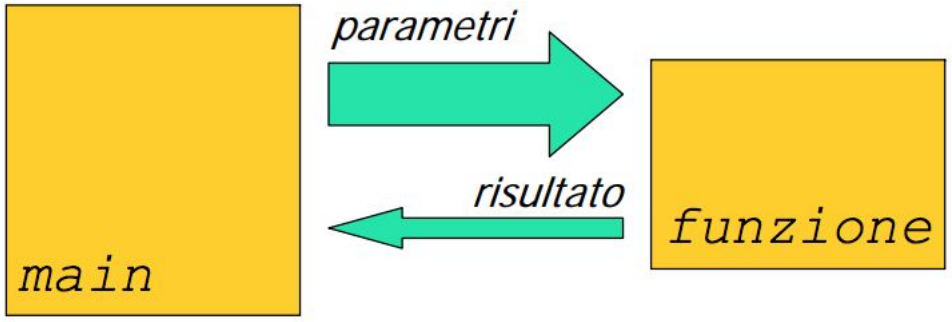
- Migliorano la leggibilità del codice organizzando le istruzioni in gruppi logici (funzioni) che risolvono problemi specifici;
- Riducono la duplicazione del codice raggruppando gruppi di istruzioni di uso comune in funzioni che vengono chiamate, invece di copiare ed incollare il codice comune;
- Facilitano l'individuazione degli errori dato che, una volta che si è individuata la funzione problematica, il programmatore si può focalizzare solo su quella parte di codice;
- Facilitano le modifiche e estensioni dato che il codice è organizzato in modo più logico;
- Rendono possibile la riusabilità del codice attraverso librerie di funzioni che si possono usare in programmi distinti.

Le funzioni
Variabili locali

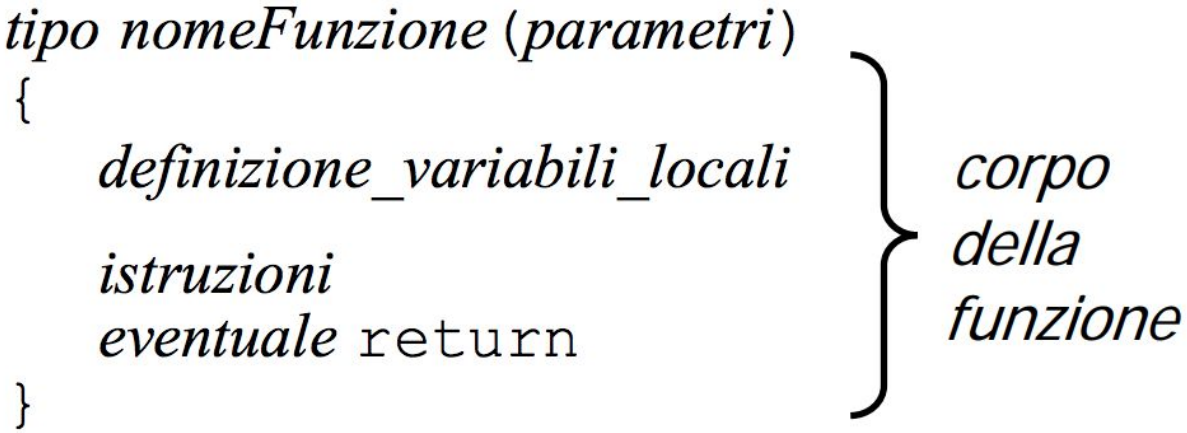
- Ogni funzione è un piccolo programma indipendente dalle altre funzioni
- All'interno di una funzione possono essere definite delle variabili locali: le altre funzioni non le "vedono"
- Variabili con lo stesso nome in funzioni diverse sono quindi completamente non correlate (possono dunque anche essere di tipo diverso)
- Vengono create ogni volta che si entra nella funzione e distrutte (perdendo il valore) quando si esce
- Le inizializzazioni avvengono ad ogni chiamata, senza inizializzazione il contenuto è indefinito

Le funzioni
Parametri e valore di ritorno

- Essendo le variabili interne locali, per passare ad una funzione i dati da elaborare è necessario utilizzare variabili speciali dette parametri.
- La funzione comunica al modulo chiamante il risultato della sua elaborazione producendo un unico valore detto valore restituito o valore di ritorno



Le funzioni
Definizione



- ***tipo*** indica il tipo del valore restituito (double, float, int, ecc…)

- Se la funzione non restituisce valori (ad esempio *StampaCiao* stampa soltanto) bisogna indicare il tipo *void*: *void StampaCiao(.....)*

Le funzioni
Chiamata di funzione

- Si chiama una funzione indicandone il nome seguito da una coppia di parentesi contenenti i valori da elaborare (separati da virgole). Es. *somma(3,5);*
- Se la funzione non richiede parametri, le parentesi sono vuote, ma devono esserci: *StampaCiao();*

Le funzioni
Chiamata di funzione

- Il valore restituito può essere assegnato ad una variabile o utilizzato in un'espressione, altrimenti viene semplicemente scartato.

Esempio:

```
int a = 3;
int b = 5;
int risultato = somma(a,b);
stampa(risultato);
```

Il valore di ritorno viene salvato in una variabile

Valore di ritorno "*void*"

Le funzioni
Ritorno di una funzione

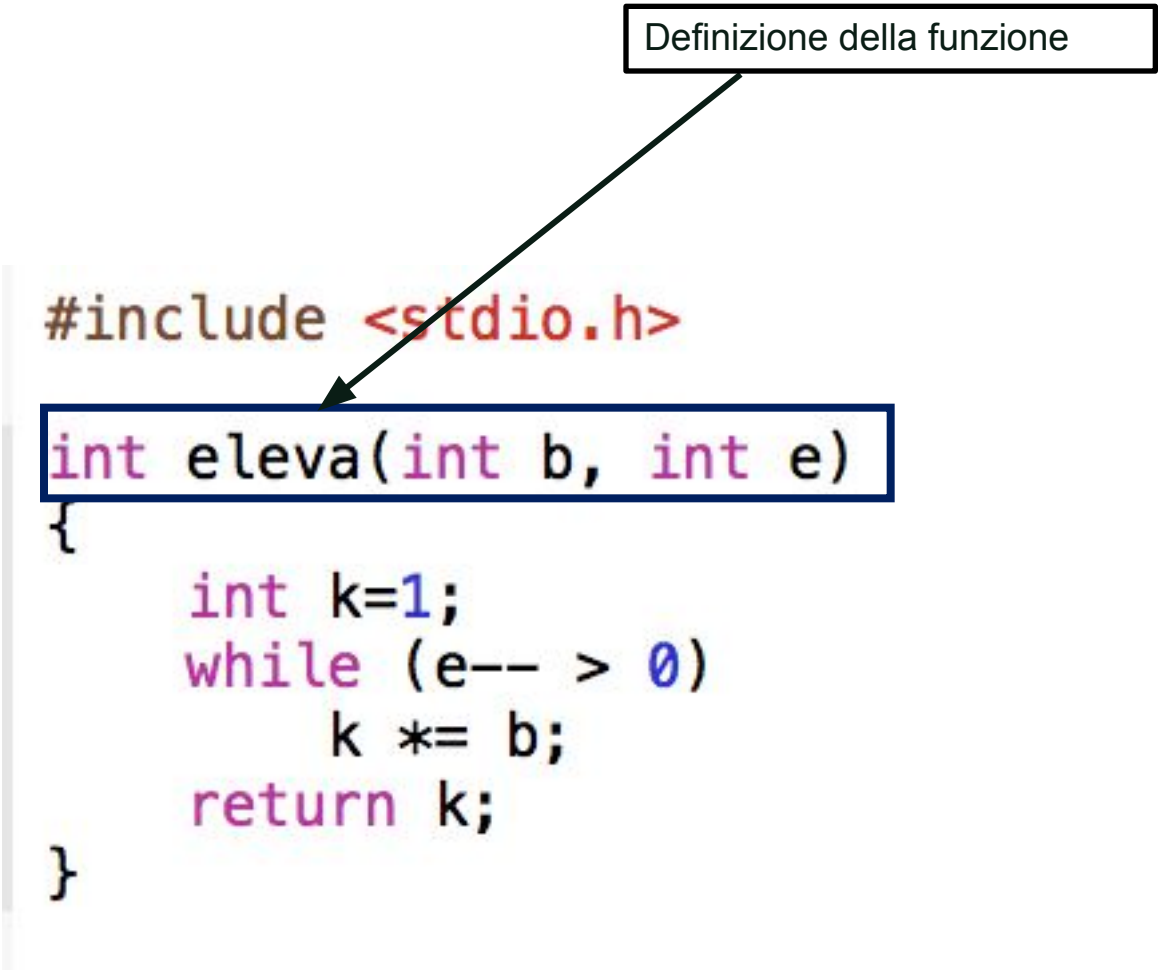
- La funzione termina (cioè l'esecuzione torna al modulo chiamante) quando viene eseguita l'istruzione ***return*** *valore*;
- Una funzione può avere più istruzioni ***return***;
- Se il tipo restituito dalla funzione è **void,** la *return* può essere omessa;
- I valori delle variabili locali vengono persi.

Le funzioni
Un esempio

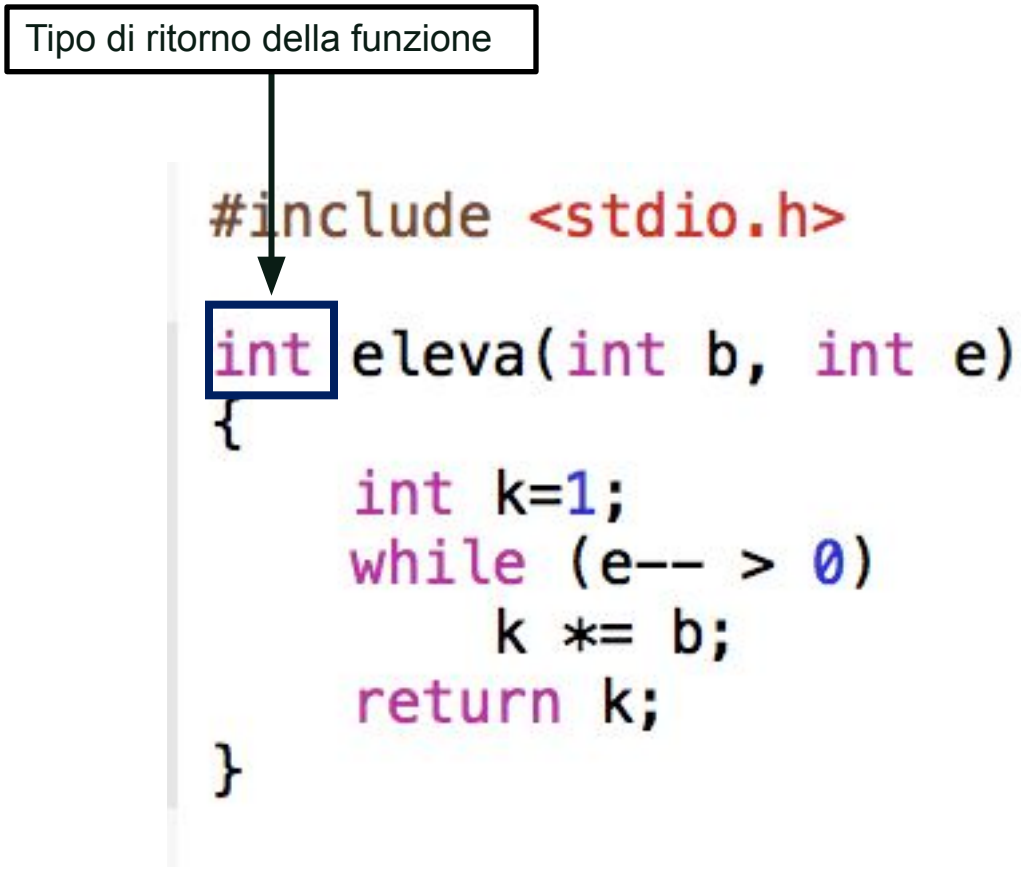
Funzione che eleva un numero *b* per un esponente *e*

```c
#include <stdio.h>

int eleva(int b, int e)
{
    int k=1;
    while (e-- > 0)
        k *= b;
    return k;
}

int main()
{
    int x=0, y=0, z=0;
    printf("Introduci numero: ");
    scanf("%d", &x);
    printf("Introduci la potenza: ");
    scanf("%d", &y);
    z = eleva(x, y);
    printf("%d^%d = %d\n", x,y,z);
    return 0;
}
```

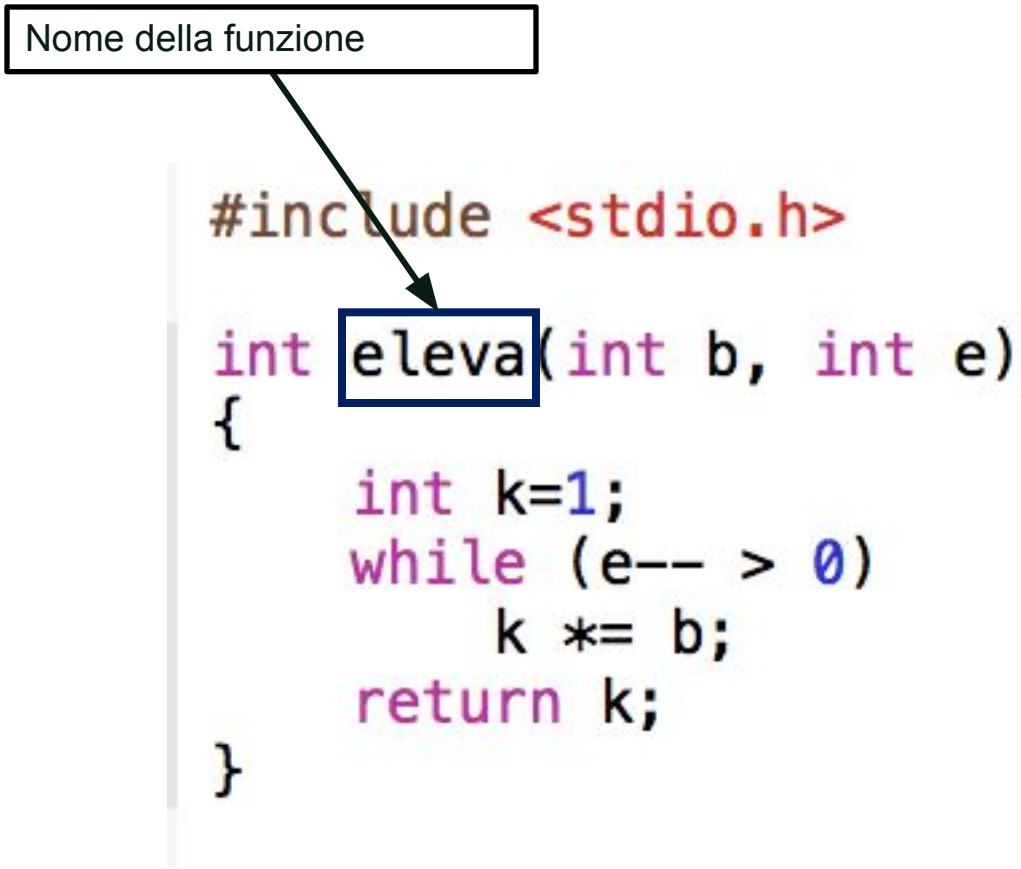
Le funzioni
La funzione *eleva*



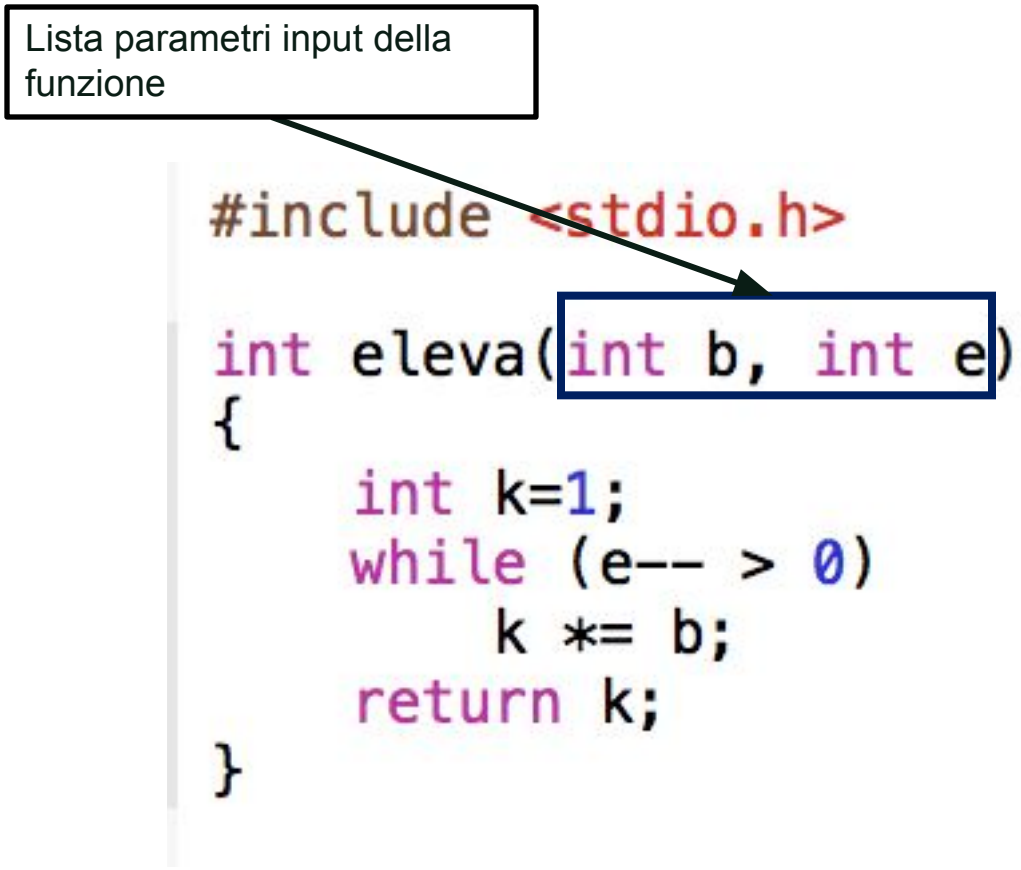
Le funzioni
La funzione *eleva*



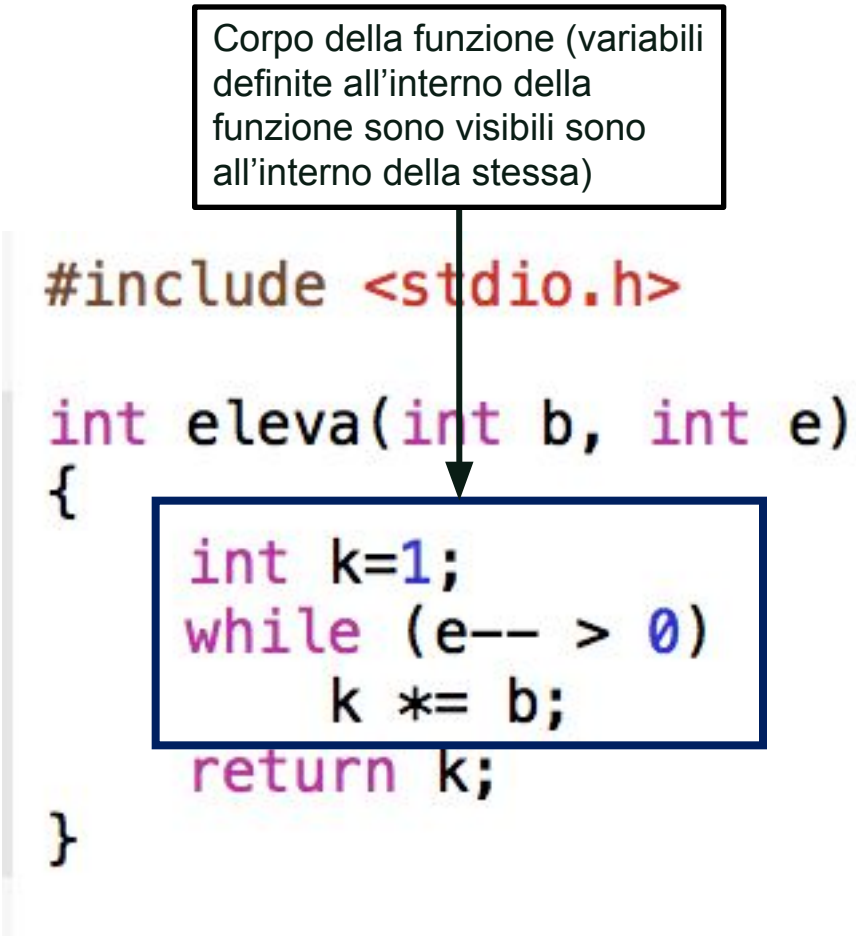
Le funzioni
La funzione *eleva*



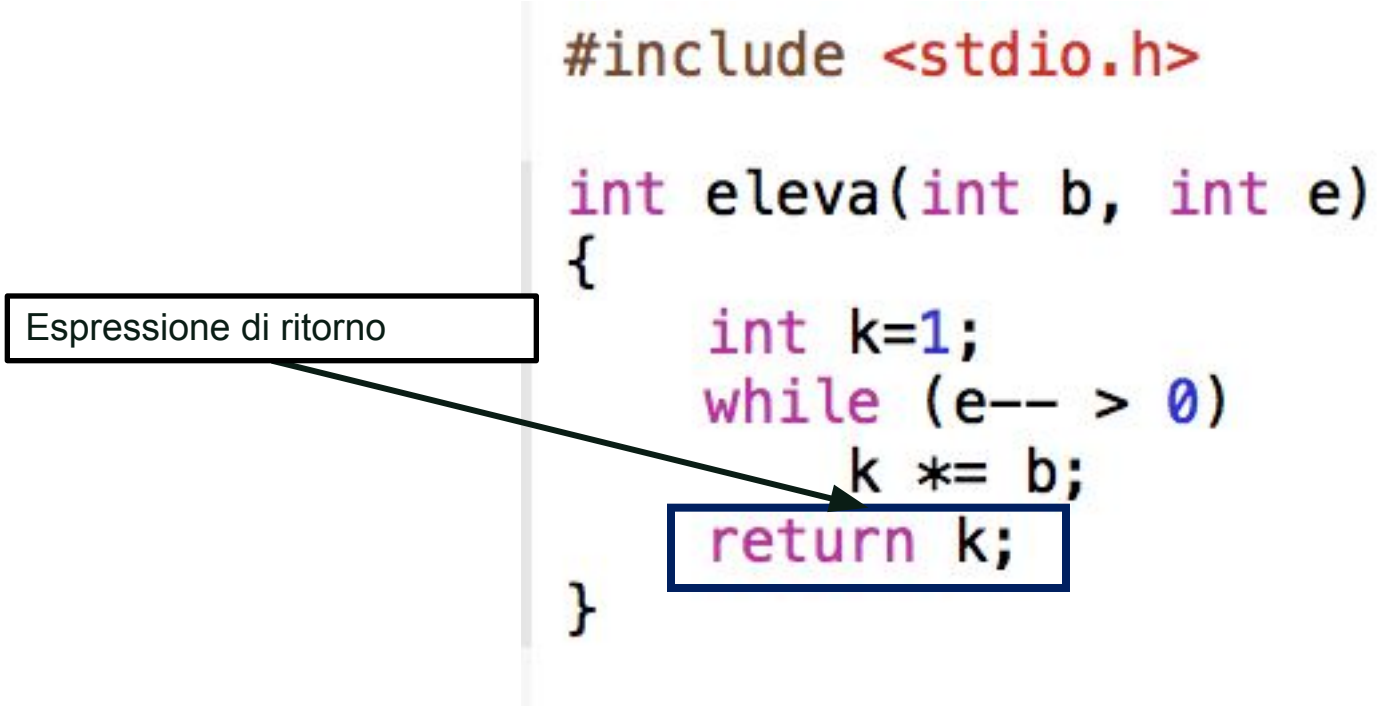
Le funzioni
La funzione *eleva*



Le funzioni
La funzione *eleva*



Le funzioni
La funzione *eleva*

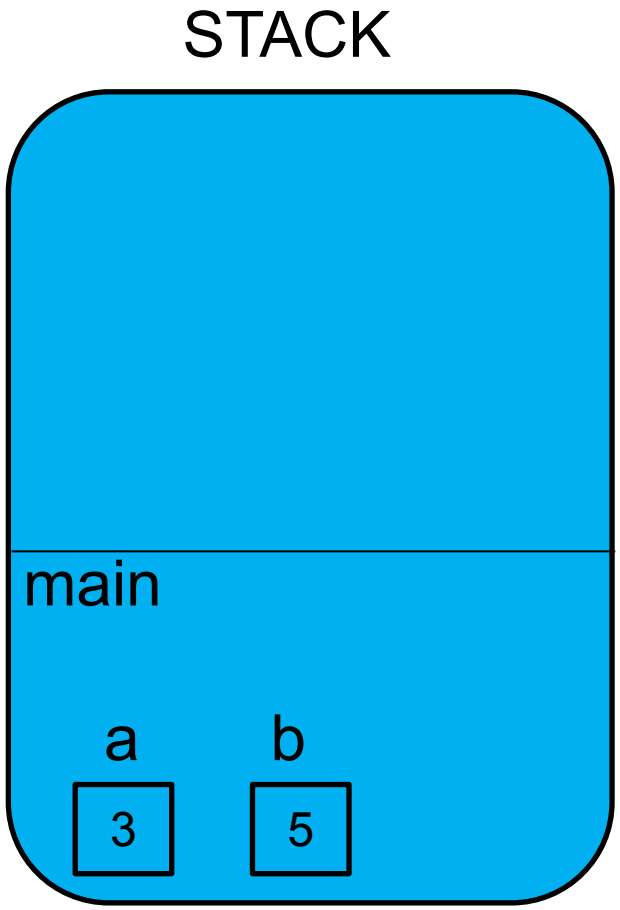


Le funzioni
Come vengono gestite nella memoria?

```
 8   main ()
 9   {
10       int a = 3;
11       int b = 5;
12       int risultato = somma(a,b);
13       printf("Risultato:%d",risultato);
14   }
15
16   int somma(int x, int y){
17       int ris = x + y;
18       return ris;
19   }
```

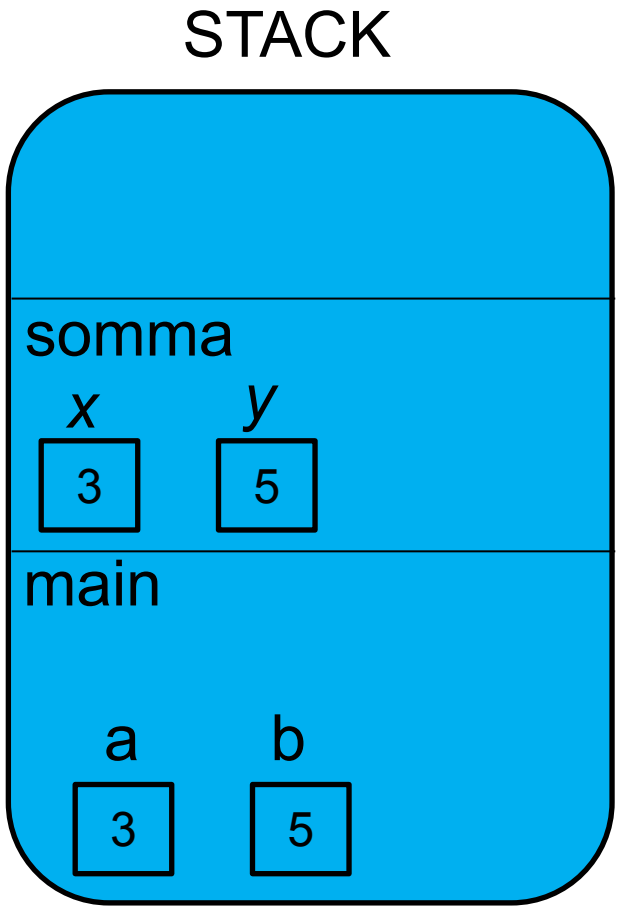
Le funzioni
Come vengono gestite nella memoria?

```
 8   main ()
 9   {
10       int a = 3;
11       int b = 5;
12       int risultato = somma(a,b);
13       printf("Risultato:%d",risultato);
14   }
15
16   int somma(int x, int y){
17       int ris = x + y;
18       return ris;
19   }
```



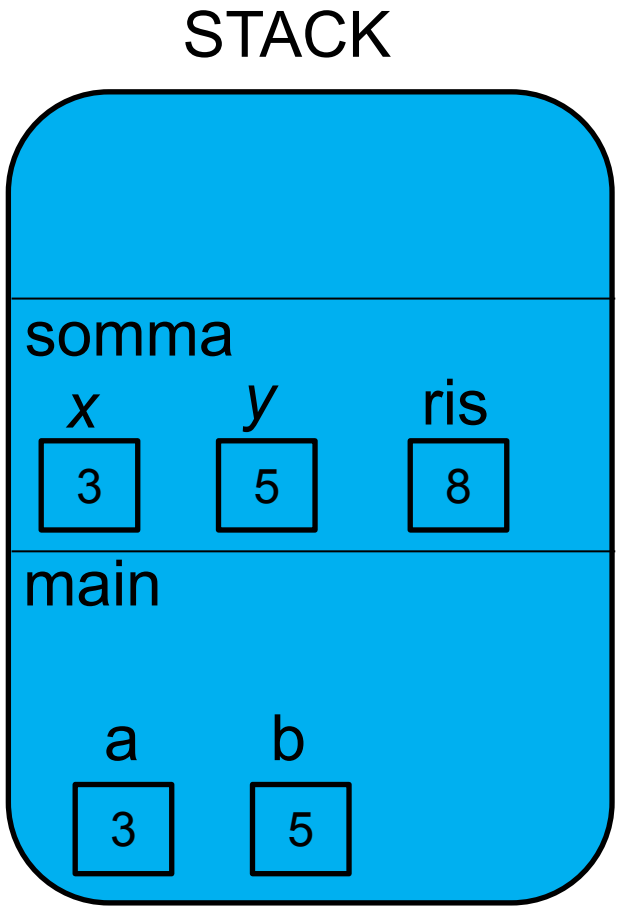
Le funzioni
Come vengono gestite nella memoria?

```
 8  main ()
 9  {
10      int a = 3;
11      int b = 5;
12      int risultato = somma(a,b);
13      printf("Risultato:%d",risultato);
14  }
15
16  int somma(int x, int y){
17      int ris = x + y;
18      return ris;
19  }
```



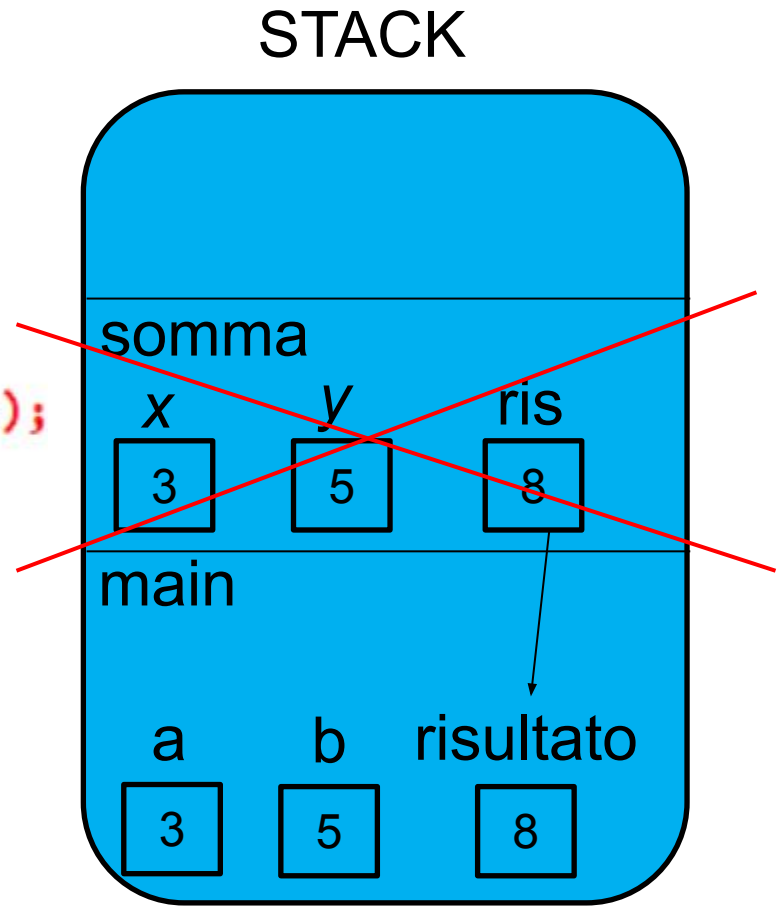
Le funzioni
Come vengono gestite nella memoria?

```
 8  main ()
 9  {
10      int a = 3;
11      int b = 5;
12      int risultato = somma(a,b);
13      printf("Risultato:%d",risultato);
14  }
15
16  int somma(int x, int y){
17      int ris = x + y;
18      return ris;
19  }
```



Le funzioni
Come vengono gestite nella memoria?

```
 8  main ()
 9  {
10      int a = 3;
11      int b = 5;
12      int risultato = somma(a,b);
13      printf("Risultato:%d",risultato);
14  }
15
16  int somma(int x, int y){
17      int ris = x + y;
18      return ris;
19  }
```



Le funzioni
Il prototipo di una funzione

- Il prototipo di una funzione è una dichiarazione della funzione (tipo di ritorno, nome, parametri). Viene inserito:
  - prima del main
  - In file header separato

```
int eleva(int b, int e);
```

- I prototipi sono simili alle definizioni, salvo che:
  - manca il corpo
  - i nomi dei parametri possono essere omessi (ma i tipi devono essere presenti!)
  - ha un punto e virgola alla fine

Le funzioni
Esempio con prototipo

```c
#include <stdio.h>

int eleva(int b, int e); //prototipo

int main()
{
    int a=3, b=2;
    int risultato=eleva(a,b);
    printf("%d^%d = %d\n", a,b,risultato);
    return 0;
}

int eleva(int b, int e)
{
    int k=1;
    while (e-- > 0)
        k *= b;
    return k;
}
```

Le funzioni
I parametri

- I parametri di tipo numerico (intero e virgola mobile) vengono ***passati per valore*** alle funzioni. Questo significa che se il parametro viene cambiato durante l'esecuzione della funzione, questo non ha effetto al di fuori della funzione stessa.
- I parametri di tipo array sono invece ***passati per riferimento***. Questo significa che se i valori delle componenti dell'array sono alterati durante l'esecuzione della funzione, gli stessi valori sono alterati nell'array che viene passato come argomento durante la chiamata.

Le funzioni
Passaggio di parametri - esempio

- Funzione che ritorna la somma dei valori di un array.

```
float sum(float[] v, int n) {
    float s = 0;
    for(int i = 0; i < n; i ++){
        s += v[i];
    }
    return s;
}
```

Le funzioni
Passaggio di parametri - esempio

- Funzione che imposta gli elementi di un array ad un valore dato

```
void set(float[] v, int n, float value) {
    for(int i = 0; i < n; i ++) {
        v[i] = value;
    }
}
```

Le funzioni
Esercizio 1

Si scriva una funzione con il seguente prototipo:

int somma(int num1, int num2);

La funzione prende come parametri due numeri interi e ritorna la loro somma.

Il main deve permettere all'utente di inserire due numeri interi e poi stampare il valore restituito dalla funzione *somma*.

Le funzioni
Esercizio 2

Si scriva una funzione con il seguente prototipo:

int minimo(int array[ ], int lunghezza);

La funzione prende come parametri un array e la sua dimensione. Deve ritornarne l'elemento minimo nell'array.

Il main deve permettere all'utente di inserire 5 numeri interi e grazie alla chiamata alla funzione *minimo* deve stampare l'elemento minore.

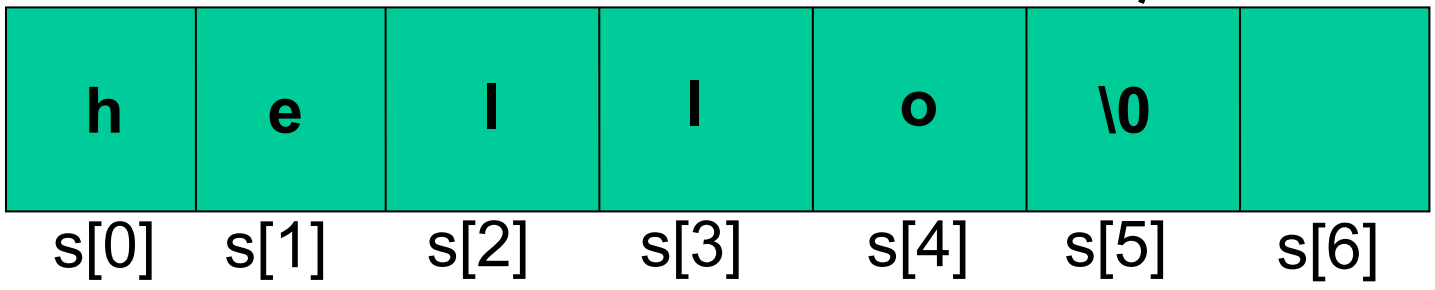
Le stringhe…
Array di char

- Una stringa viene rappresentata come sequenza di caratteri che termina con ‘\0’ (carattere terminatore)
- Quindi è un Array di *char* che termina con il carattere terminatore
- Dichiarazione:

```
char s[7] = "hello";
```

Cosa succede in memoria?

Il carattere "terminatore stringa" viene aggiunto automaticamente dal compilatore



Le stringhe…

Array di char

Step per leggere una stringa in input e stamparla:

- **Dichiarazione:**

```
char stringa[20];
```

Il vettore deve essere sufficientemente grande da contenere la stringa in input

- **Lettura in input:**

```
scanf("%s",stringa);
```

Legge i caratteri dall'input fino al primo spazio e li inserisce in *stringa*, aggiunge anche \0

- **Stampa in output:**

```
printf("%s \n", stringa);
```

Stampa ciò che è contenuto in *stringa* fino al carattere \0 (escluso)

Le stringhe

Esercizio 3

- Si scriva un programma che legge in input una stringa e la salva in un Array di char di 20 elementi
- Il programma, tramite una funzione modificaStringa, deve sostituire tutte le cifre che compaiono nell'array con il carattere *
- Infine deve stampare la stringa modificata
- *N.B. per controllare se il char **c** è un numero:*
- *(c >= '0' && c <= '9')*

Le stringhe…

Come copiare una stringa?

```
//…dati due Array di char A e B
A = B;
```

Questo assegnamento non produce una copia

È opportuno copiare ogni singolo elemento alla volta:

```
for (i = 0; i <Dim; i++) {
    A[i] = B[i];
}
```

La libreria *string.h*

- Libreria di C che consente di manipolare le stringhe
- Per poter utilizzare le sue funzioni, nel nostro programma è opportuno importarla attraverso la direttiva:
- `#include <string.h>`
- Alcune delle funzioni che vedremo:

`strlen` `strcpy` `strcmp` `strcat`

strlen (char stringa[ ])

Restituisce un intero, ovvero la lunghezza della stringa passata come parametro:

```
#include <string.h>

int main ( ) {

char s[10]=“pippo”;

int l;

l = strlen(s);

printf(“La lunghezza della stringa è: %d”,
l);

return 0;

}
```

strcpy (char s1[ ] , char s2[ ])

Copia s2 in s1:

```
#include <string.h>

int main ( ) {

char sorgente[10]=“pippo”;

char destinazione[10];

int l;

strcpy(destinazione, sorgente);

printf(“Il valore di destinazione è: %s”,
destinazione);

return 0;

}
```

L' eventuale contenuto di "*destinazione*" precedente all'assegnamento viene perso

strcat (char s1[ ] , char s2[ ])

Accoda a s1 il contenuto di s2

```
#include <string.h>

int main ( ) {

char sorgente[10]="world";

char destinazione[10]="hello";

int l;

strcat(destinazione, sorgente);

printf("Il valore di destinazione è: %s",
destinazione);

return 0;

}
```

Assicurarsi che la dimensione massima della stringa "*destinazione*" sia sufficiente ad ospitare la nuova stringa

strcmp (char s1[ ] , char s2[ ]) (1/2)

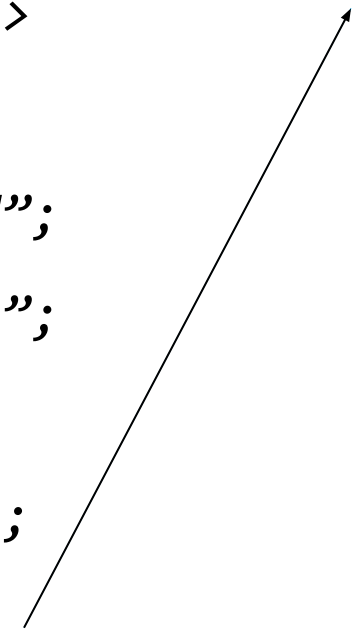
- Confronta le due stringhe (*s1* e *s2*) passate come parametro
- Restituisce:
- **(0)** se le due stringhe sono identiche
- **(< 0)** se *s1* è minore di *s2*
- **(> 0)** se *s1* è maggiore di *s2*

- *N.B. La relazione d'ordine tra stringhe è definita dalla relazione d'ordine della codifica ASCII dei caratteri che la compongono –*
- *- '0' < '9' < 'A' < 'Z' < 'a' < 'z'*

strcmp (char s1[ ] , char s2[ ]) (2/2)

Esempio:

```
#include <string.h>
int main ( ) {
char s1[10]="world";
char s2[10]="hello";
int c;
c = strcmp(s1, s2);
```



```
if (c==0) {
printf("UGUALI");
} elseif (c<0) {
printf("s1 < s2");
} else {
printf("s1 > s2");
}
return 0;
}
```

Esercizi sulle stringhe…

Esercizio 4

- Si scriva una funzione con il seguente prototipo:

void invertiStringa(char array[ ]);

Che prende come parametro un array di char e inverte i caratteri all'interno.

La funzione main deve permettere all'utente di inserire una stringa, salvarla in un array di 20 caratteri e, dopo aver chiamato la funzione *invertiStringa*, stampare la stringa invertita

Le stringhe

Esercizio 5

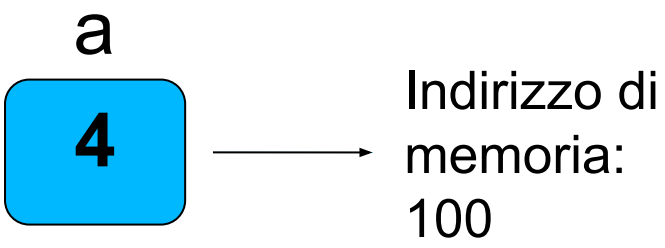
- Si scriva un programma che dichiara due Array (*s1* e *s2*) di char di 20 caratteri ognuno e permetta all'utente di inserire in input due stringhe che verranno salvate nei due rispettivi Array
- Il programma deve confrontare le due stringhe.
- **Se s1 > s2**, crea un nuovo array di char s3 composta dalla concatenazione ***s1* + *s2***.
- **Se s1<s2,** crea il nuovo Array s3 composto dalla concatenazione **s2 + s1**.
- **Se invece s1=s2**, memorizza in s3 solo una delle due stringhe
- Alla fine si stampi la stringa s3 sullo schermo

Puntatori

- Cosa succede quando dichiariamo una variabile?

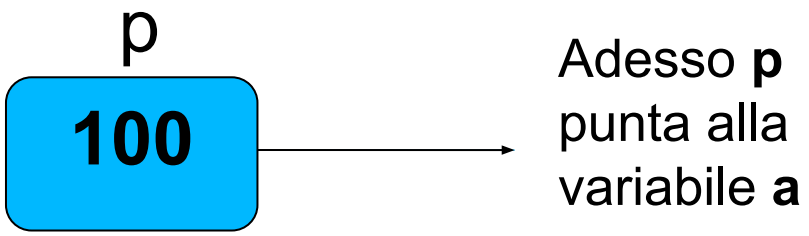
  Es. `int a = 4;`

  - Viene allocato uno spazio in memoria per contenere quel tipo di variabile. Inoltre si ha un riferimento alla variabile (nel nostro caso **a**) e un indirizzo di memoria (**100**).



- **<u>Un puntatore è una variabile che contiene l'indirizzo di memoria di un'altra variabile.</u>**

  - Es. dichiariamo un puntatore **p** alla variabile intera **a.** Cosa succede?



Puntatori

- Per dichiarare un puntatore è opportuno anteporre al nome della variabile un asterisco (*)
  - Esempio dichiariamo un puntatore a **int** e un puntatore a **char**:

```
int *pa;
char *pc;
```

- Quindi **pa** è un puntatore che può contenere l'indirizzo in memoria di qualsiasi variabile intera
- Invece **pc** può contenere l'indirizzo in memoria di qualsiasi variabile di tipo char

Puntatori

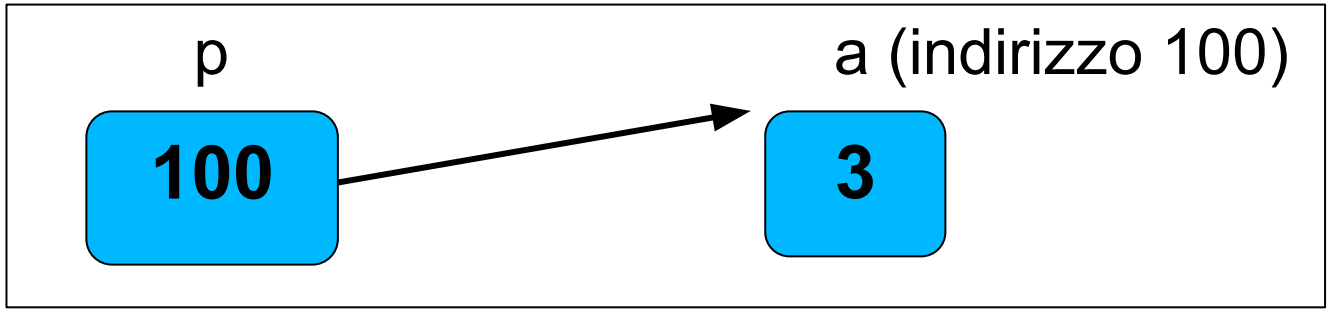
```
//variabile intera
int a = 3;
//puntatore a intero
int *p;
```

- * operatore che restituisce il contenuto dell'oggetto puntato
- **&** operatore che restituisce l'indirizzo in memoria della variabile

- **Adesso facciamo puntare a *p* la variabile *a***
- **(supponiamo che la variabile *a* abbia indirizzo di memoria 100)**

p = &a;

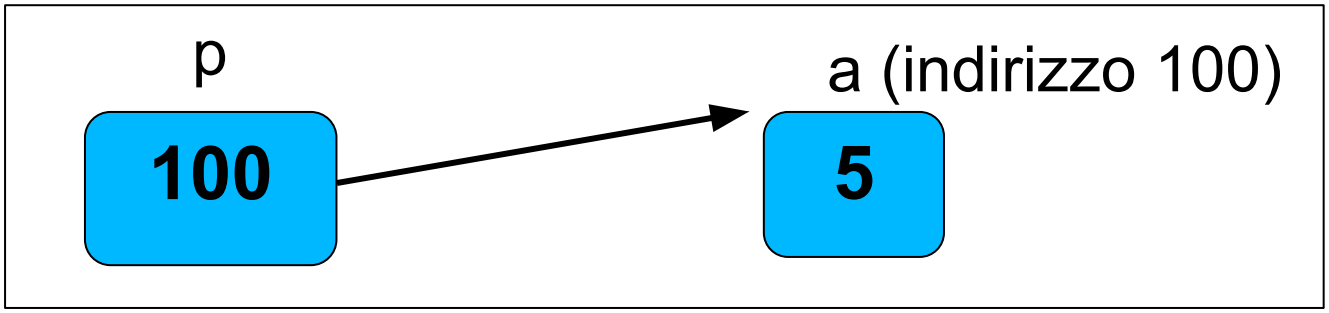
- L'operatore & restituisce l'indirizzo in memoria della variabile *a*



Puntatori

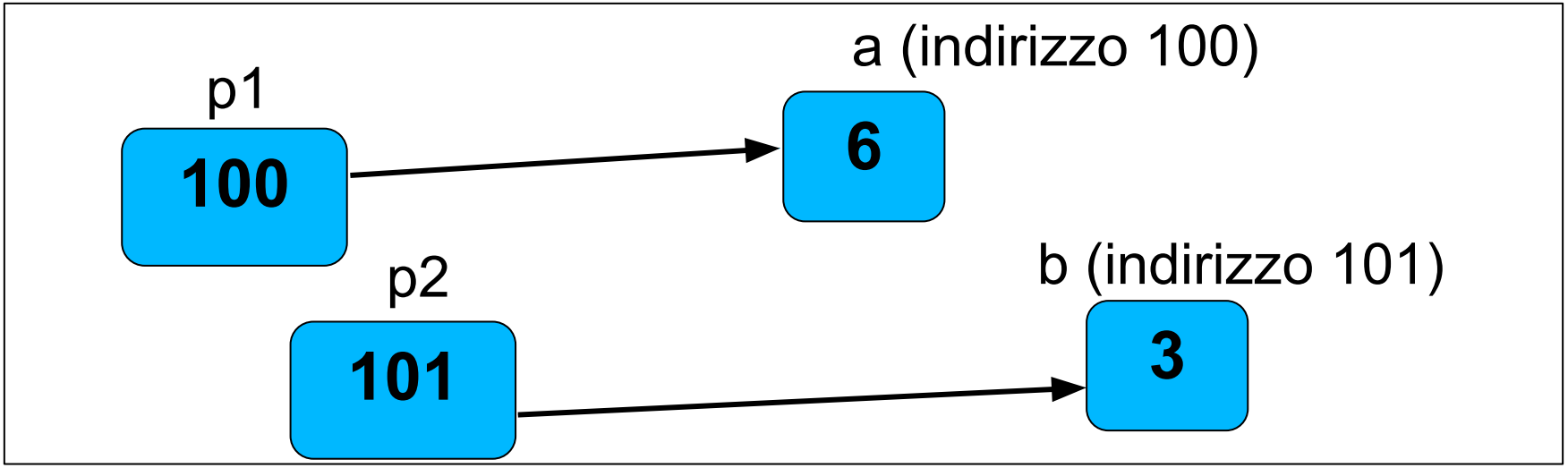
```
//variabile intera
int a = 3;
//puntatore a intero
int *p;
p = &a;
*p = 5;
```

- Viene inserito in *p* l'indirizzo in memoria della variabile *a* (*p* punta ad *a*)
- Viene modificato il contenuto della variabile puntata da *p*
- *(quindi il contenuto di a adesso vale 5)*



Puntatori (un altro esempio 1/2)

```
int *p1, *p2;
int a = 6;
int b = 3;
p1 = &a;
p2 = &b
```

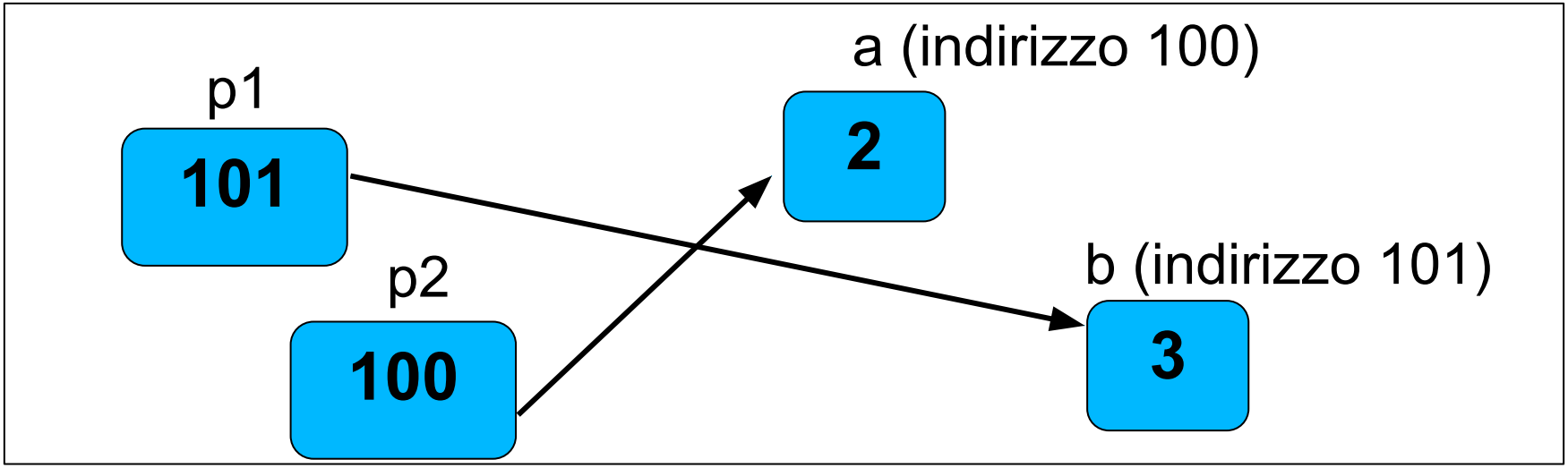


Puntatori (un altro esempio 2/2)

```
int *p1, *p2;
int a = 6;
int b = 3;
p1 = &a;
p2 = &b
*p1 = 2;
p1 = &b;
p2 = &a;
```

Modifica il valore puntato da *p1* con l'intero 2

- Modifica l'indirizzo a cui punta *p1* (adesso punta alla variabile *b*)
- Modifica l'indirizzo a cui punta *p2* (adesso punta alla variabile a)



Esercizio 6.1

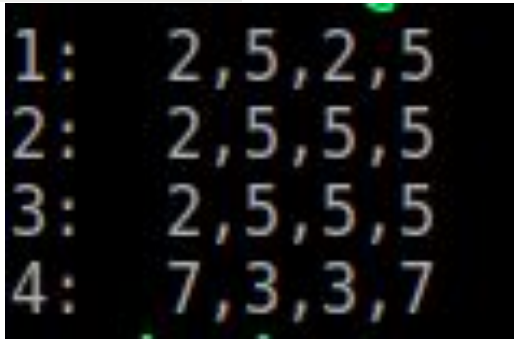
Cosa stampano le 4 printf?

```
 5      int a = 2; int b = 5;
 6      int *p1; int *p2;
 7      p1 = &a;
 8      p2 = &b;
 9      printf ("1:  %d,%d,%d,%d \n",a,b,*p1,*p2);
10      p1 = &b;
11      printf ("2:  %d,%d,%d,%d \n",a,b,*p1,*p2);
12      *p2 = *p1;
13      printf ("3:  %d,%d,%d,%d \n",a,b,*p1,*p2);
14      p2 = &a;
15      *p2 = 7;
16      b = 3;
17      printf ("4:  %d,%d,%d,%d \n",a,b,*p1,*p2);
18
```

Puntatori

Soluzione…

```
 5      int a = 2; int b = 5;
 6      int *p1; int *p2;
 7      p1 = &a;
 8      p2 = &b;
 9      printf ("1:  %d,%d,%d,%d \n",a,b,*p1,*p2);
10      p1 = &b;
11      printf ("2:  %d,%d,%d,%d \n",a,b,*p1,*p2);
12      *p2 = *p1;
13      printf ("3:  %d,%d,%d,%d \n",a,b,*p1,*p2);
14      p2 = &a;
15      *p2 = 7;
16      b = 3;
17      printf ("4:  %d,%d,%d,%d \n",a,b,*p1,*p2);
18
```



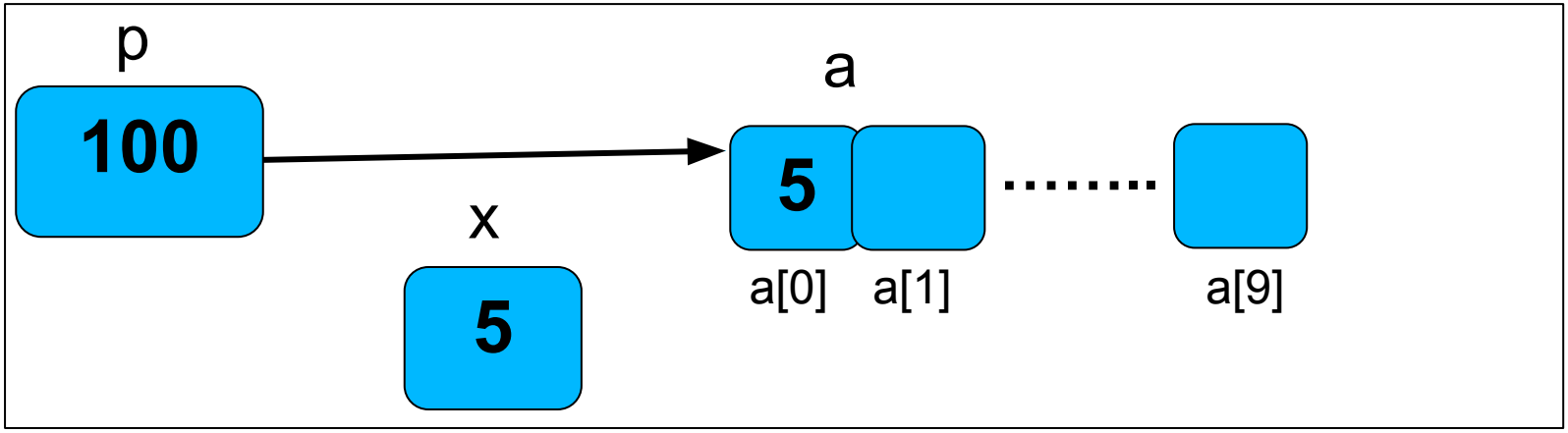
Puntatori e Array

…aritmetica dei puntatori

```
int a[10], x;
int *p;
p = &a[0];
x = *p;
```

- Viene assegnato al puntatore *p* *l'indirizzo in memoria del primo elemento dell'array a*
- Viene assegnato alla variabile *x*, il valore contenuto nella locazione di memoria puntata da *p*
- *N.B.* `x = a[0];`

Avrebbe avuto lo stesso risultato



Puntatori e Array

…aritmetica dei puntatori

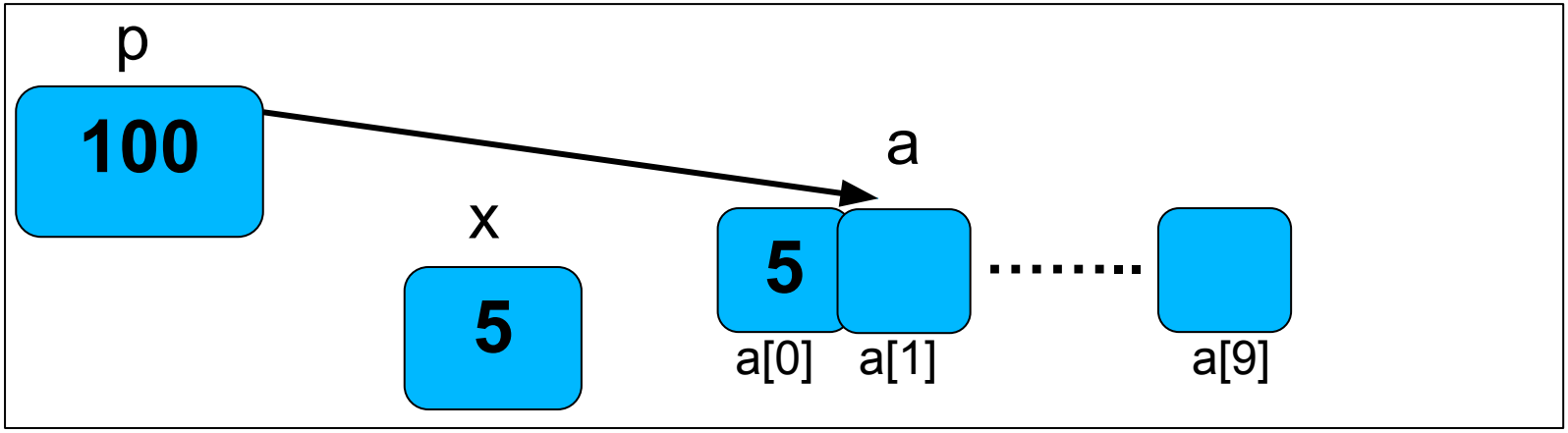
```
int a[10], x;
int *p;
p = &a[0];
x = *p;
p++;
```

- Viene incrementato l'indirizzo a cui punta *p*

(in questo caso del numero di byte con cui viene rappresentato un int).

Adesso *p* punterà alla seconda posizione dell'Array *a*

- N.B. Se *p* fosse stato un puntatore a char, l'indirizzo a cui punta sarebbe stato incrementato di 1 byte (ovvero la dimensione di un char)



Puntatori e Array

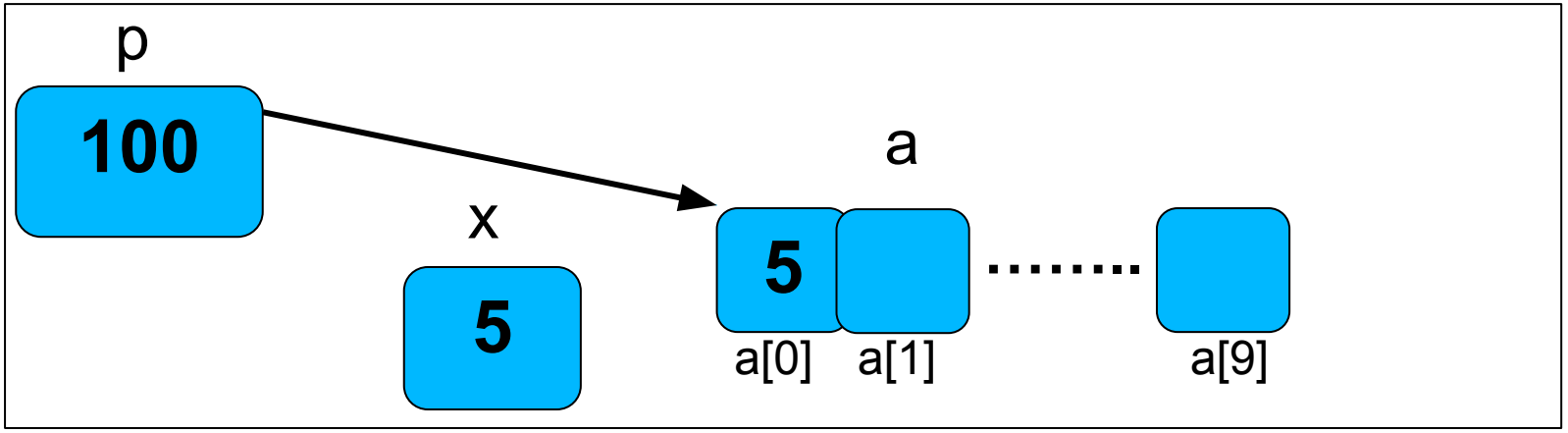
…aritmetica dei puntatori

```
p--;
```

- Viene decrementato l'indirizzo a cui punta *p*

(in questo caso del numero di byte con cui viene rappresentato un int).

Adesso *p* punterà nuovamente alla prima posizione dell'Array *a*



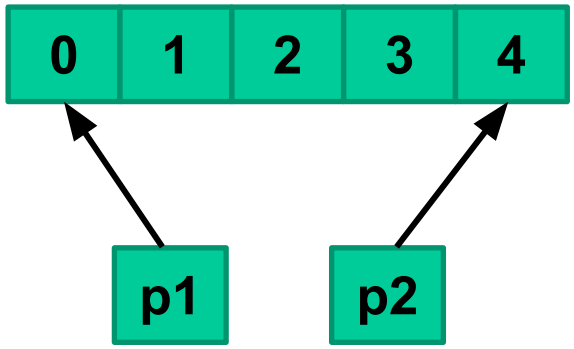
Esercizio 6.2

```
 5      int array[5];
 6      int i=0;
 7      for(i=0;i<5;i++){
 8          array[i] = i;
 9      }
10      int *p1,*p2;
11      p1 = &array[0];
12      p2 = &array[4];
13      *p1 = 4;
14      p1++;
15      *p1 = 5;
16      *p2 = *p1;
17      p2 = p1;
18      *p2 = 21;
```

Illustrare i valori contenuti dall'Array dopo che si esegue il codice

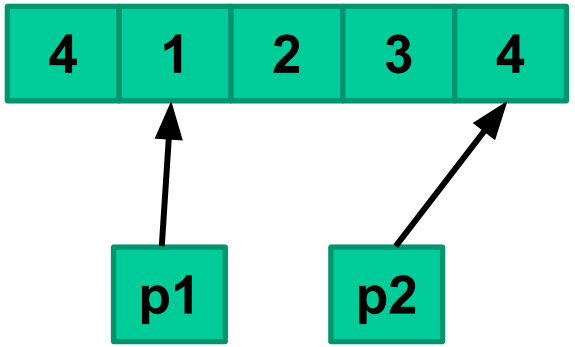
Esercizio 6.2

```
 5      int array[5];
 6      int i=0;
 7      for(i=0;i<5;i++){
 8          array[i] = i;
 9      }
10      int *p1,*p2;
11      p1 = &array[0];
12      p2 = &array[4];
13      *p1 = 4;
14      p1++;
15      *p1 = 5;
16      *p2 = *p1;
17      p2 = p1;
18      *p2 = 21;
```



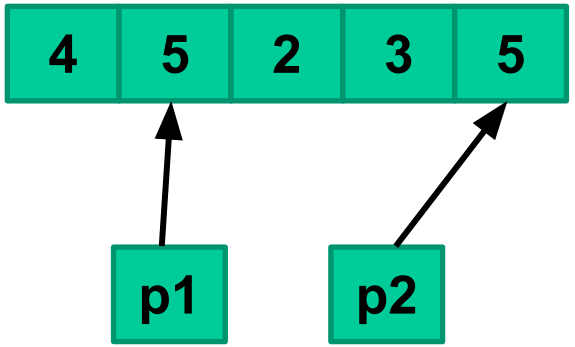
Esercizio 6.2

```
 5      int array[5];
 6      int i=0;
 7      for(i=0;i<5;i++){
 8          array[i] = i;
 9      }
10      int *p1,*p2;
11      p1 = &array[0];
12      p2 = &array[4];
13      *p1 = 4;
14      p1++;
15      *p1 = 5;
16      *p2 = *p1;
17      p2 = p1;
18      *p2 = 21;
```



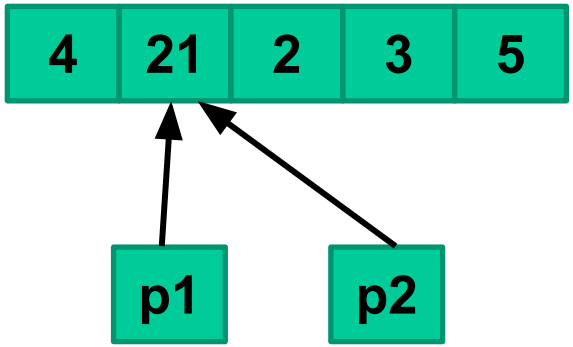
Esercizio 6.2

```
 5      int array[5];
 6      int i=0;
 7      for(i=0;i<5;i++){
 8          array[i] = i;
 9      }
10      int *p1,*p2;
11      p1 = &array[0];
12      p2 = &array[4];
13      *p1 = 4;
14      p1++;
15      *p1 = 5;
16      *p2 = *p1;
17      p2 = p1;
18      *p2 = 21;
```



Esercizio 6.2

```
 5      int array[5];
 6      int i=0;
 7      for(i=0;i<5;i++){
 8          array[i] = i;
 9      }
10      int *p1,*p2;
11      p1 = &array[0];
12      p2 = &array[4];
13      *p1 = 4;
14      p1++;
15      *p1 = 5;
16      *p2 = *p1;
17      p2 = p1;
18      *p2 = 21;
```



Passaggio di parametri per riferimento

- È possibile passare ad una funzione l'indirizzo in memoria di una variabile (anziché il suo valore come visto in precedenza)
- Quindi la funzione chiamata, potrà modificare direttamente il valore della variabile della funzione chiamante
- La funzione chiamata dovrà dichiarare i parametri (passati per riferimento) come puntatori
- La funzione chiamante dovrà chiamare la funzione passando come parametro l'indirizzo in memoria delle variabili che si vogliono passare per indirizzo

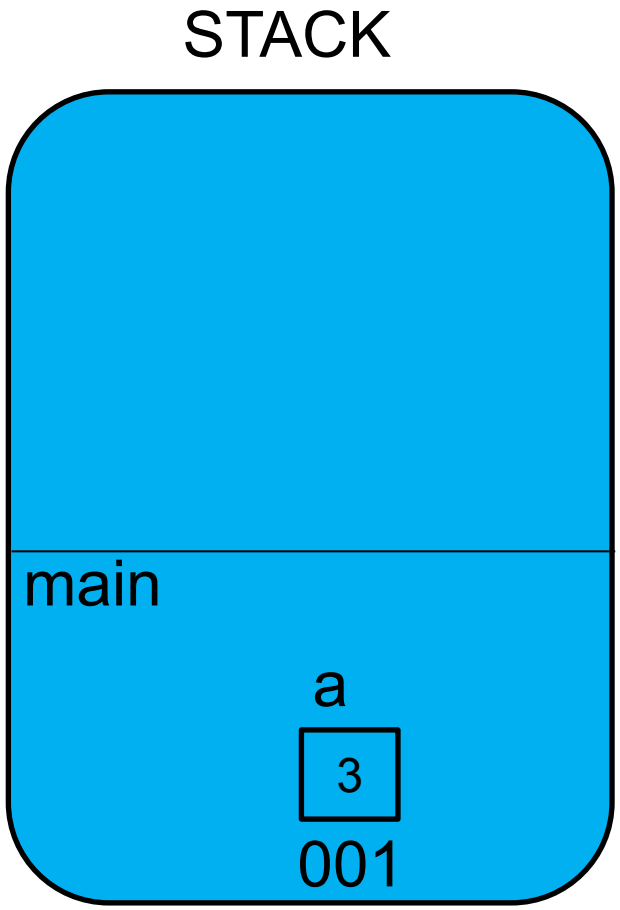
Passaggio di parametri per riferimento

- Esempio:

```c
#include <stdio.h>

void funzione(int *ptr);

int main(){
    int a = 3;
    printf("a: %d \n",a);
    funzione(&a);
    printf("a: %d \n",a);

    return 0;
}

void funzione(int *ptr){
    *ptr = 5;
}
```

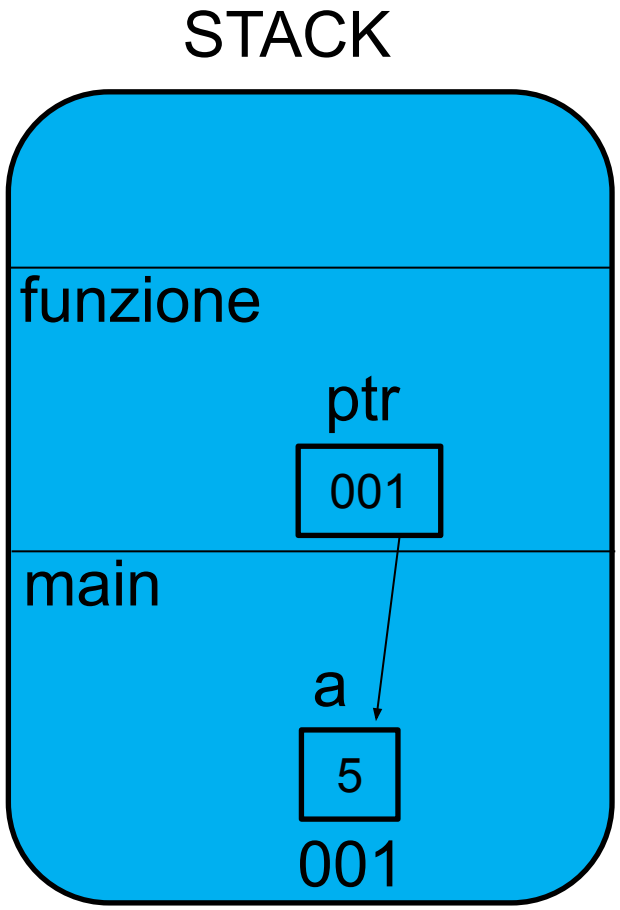
Passaggio di parametri per riferimento

```c
#include <stdio.h>

void funzione(int *ptr);

int main(){
    int a = 3;
    printf("a: %d \n",a);
    funzione(&a);
    printf("a: %d \n",a);

    return 0;
}

void funzione(int *ptr){
    *ptr = 5;
}
```



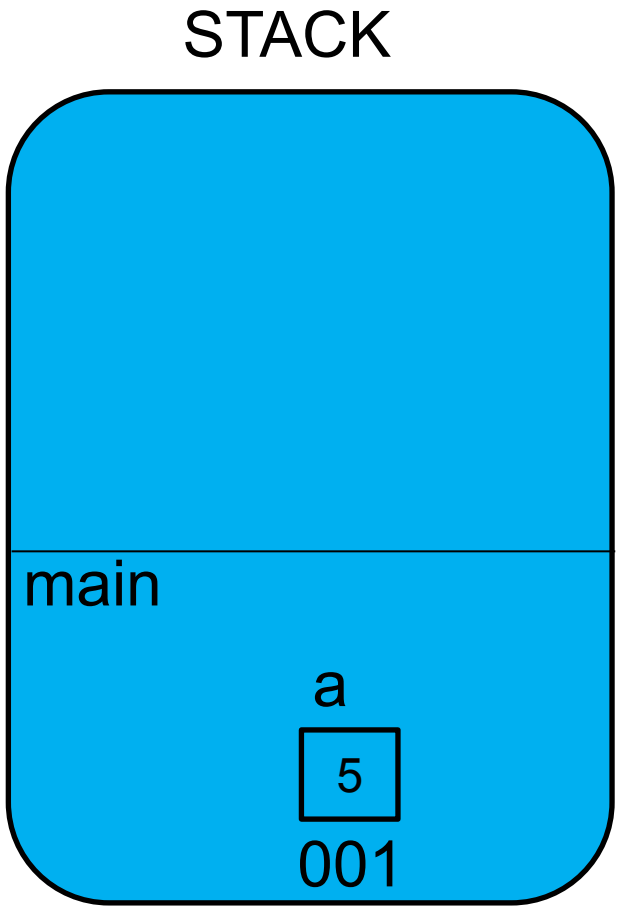
Passaggio di parametri per riferimento

```
1   #include <stdio.h>
2
3   void funzione(int *ptr);
4
5   int main(){
6       int a = 3;
7       printf("a: %d \n",a);
8       funzione(&a);
9       printf("a: %d \n",a);
10
11      return 0;
12  }
13
14  void funzione(int *ptr){
15      *ptr = 5;
16  }
```



Passaggio di parametri per riferimento

```c
#include <stdio.h>

void funzione(int *ptr);

int main(){
    int a = 3;
    printf("a: %d \n",a);
    funzione(&a);
    printf("a: %d \n",a);

    return 0;
}

void funzione(int *ptr){
    *ptr = 5;
}
```



Si scriva un programma che permetta all'utente di inserire in input due valori interi che verranno salvati nelle due variabili *a* e *b*. Dopo aver stampato il loro valore, il programma chiama la funzione *inverti* passandogli l'indirizzo delle due variabili. Dopodiché deve stampare nuovamente i valori delle due variabili invertiti dalla funzione *inverti*.

Il prototipo della funzione *inverti* è il seguente:

*void inverti(int *p1, int *p2);*

Esercizio 7

- Si scriva un programma che dichiara un Array di char di 20 caratteri e permetta all'utente di inserire in input una stringa che verrà salvata nell'Array
- Il programma deve dichiarare un puntatore di tipo char che punta all'Array
- Successivamente, **attraverso il puntatore**, deve sostituire ogni elemento dell'Array che è una cifra con il carattere *
- Infine stampare l'Array di char

Esercizio 8

- Si scriva un programma che legge da tastiera una sequenza di numeri interi positivi (legge gli interi fino a che l'utente non inserisce un intero negativo)
- Il programma, a partire dal primo intero inserito, stampa ogni volta la media di tutti gli interi inseriti.

Esercizio 9

- Si scriva un programma che legge da tastiera una stringa
- Il programma, dopo aver chiamato la funzione *int palindroma (char array[ ])* deve stampare un messaggio che indica se la stringa è palindroma o meno (in base alla funzione *palindroma* se ritorna 1 o 0 rispettivamente).
- *N.B. Una stringa è palindroma se leggibile nello stesso modo da sinistra a destra e viceversa*

Esercizio 10

Si scriva una funzione con prototipo:

*int funzione(int array[ ], int size, int *minimo, int *massimo);*

che ritorna la somma dei valori nell'array, mentre nei puntatori *minimo* e *massimo* memorizza rispettivamente il valore minimo e massimo dell'array.

Il programma deve permettere all'utente di inserire i valori nell'array e dopo aver chiamato la funzione, stampa la somma, il minimo e il massimo calcolati dalla funzione stessa.

Esercizio 11

Si scriva un programma che dichiara due Array di interi (array1 e array2) di 5 elementi e permette all'utente di riempirli

- Successivamente, tramite l'utilizzo di due puntatori, inverte il primo elemento di array1 con l'ultimo di array2, il secondo elemento di array1 con il penultimo di array2 e così via...
- Infine si stampino i due Array modificati