



SAPIENZA  
UNIVERSITÀ DI ROMA

# Programmazione Sistemi Multicore

*Ripasso del linguaggio di programmazione C*

**Christian Cardia & Gabriele Saturni**

cardia@di.uniroma1.it – saturni@di.uniroma1.it

Dip. Informatica, st. 317 – via Salaria 113, Roma



# Perché studiare C

- Il C è un linguaggio di programmazione sviluppato nel laboratorio AT&T della Bell nel 1972
- C largamente usato in molti ambiti
  - Telecomunicazioni;
  - Controllo di processi industriali;
  - embedded systems;
  - Sviluppo e testing di protocolli di rete
  - etc.
- Il C è un linguaggio minimalista e molto efficiente



# Il primo programma in C

- Il primo esempio di programma in C sarà il classico “Hello World!”

```
1 //Questo programma stampa Hello World!  
2 #include <stdio.h>  
3 /*la funzione main () denota l'inizio dell'esecuzione  
4 del programma.  
5 VA SEMPRE INSERITO in ogni tipo di programma C*/  
6 int main () {  
7     printf("Hello World! \n");  
8     return 0;  
9 }
```



# Il primo programma in C

- Ogni istruzione in C segue la seguente sintassi:

*comando ;*

- Cosa sono `//` e `/**/` ?
  - Sono i comandi usati per inserire dei commenti.
  - `//` commenti su singola linea
  - `/* */` usati per commenti multilinea
- **I commenti sono fondamentali quando il codice diventa complesso!**



# Il primo programma in C

- **#include <stdio.h>**
- Il preprocessore viene invocato prima della compilazione ed effettua delle manipolazioni sul codice che devono essere effettuate prima della compilazione
  - Esempio: includere parti nel file prima della compilazione: header di librerie o effettuare delle sostituzioni di testo, come togliere i commenti.
- In particolare, la direttiva `stdio.h` permette l'inclusione delle funzioni di input/output (`printf`, `scanf`, etc.)
- Concetto simile alle *import* nel linguaggio JAVA



# Il primo programma in C

- Il **main** fa parte di ogni programma C. Le parentesi graffe che seguono indicano un blocco di istruzioni da eseguire.
- Un programma può contenere più funzioni ciascuna delle quali svolge un compito.
- Il main è sempre presente ed è la prima funzione eseguita. Altre funzioni possono essere invocate all'interno del main per svolgere dei sotto-compiti.

```
1 //Questo programma stampa Hello World!  
2 #include <stdio.h>  
3 /*la funzione main () denota l'inizio dell'esecuzione  
4 del programma.  
5 VA SEMPRE INSERITO in ogni tipo di programma C*/  
6 int main () {  
7     printf("Hello World! \n");  
8     return 0;  
9 }
```

# Il primo programma in C

- La `printf()` è una funzione che stampa una stringa (sequenza di caratteri) indicata tra i doppi apici.
- `\n` non viene stampato in output ma permette di andare a capo
- A breve vedremo questa funzione più nel dettaglio

```
1 //Questo programma stampa Hello World!  
2 #include <stdio.h>  
3 /*la funzione main () denota l'inizio dell'esecuzione  
4 del programma.  
5 VA SEMPRE INSERITO in ogni tipo di programma C*/  
6 int main () {  
7     printf("Hello World! \n");  
8     return 0;  
9 }
```



# Il primo programma in C

- La parola chiave **return** indica il valore restituito dalla funzione

- In questo caso `main()` chiede che venga restituito il valore intero 0

```
1 //Questo programma stampa Hello World!  
2 #include <stdio.h>  
3 /*la funzione main () denota l'inizio dell'esecuzione  
4 del programma.  
5 VA SEMPRE INSERITO in ogni tipo di programma C*/  
6 int main () {  
7     printf("Hello World! \n");  
8     return 0;  
9 }
```

- Nel `main` 0 è il valore di default. Indica che il programma è terminato con successo





# Dichiarazione di variabili

- Prima di poter utilizzare una variabile, la si deve dichiarare indicandone nome e tipo. La sintassi per dichiarare una variabile è:
  - `<tipo variabile> <nome>;`
- Ad esempio:
  - `int x;` → dichiara una variabile di nome x di tipo intero
- E' una buona abitudine inizializzare SEMPRE le variabili al momento della loro dichiarazione.
- Ad esempio:
  - `int x=0;` → assegna il valore 0 alla variabile x



# Assegnamento

- Il contenuto di una variabile si cambia usando un **assegnamento** con la sintassi:
  - `<tipo variabile> = <valore>;`
- Ad esempio:
  - `x=5;` → memorizza il valore 5 nella locazione di memoria indicata da x. Il valore della variabile cambia subito dopo l'assegnamento e rimane tale fino all'assegnamento successivo.

# Operatori aritmetici

- Per fare calcoli, si può operare su variabili e costanti usando gli stessi operatori aritmetici che si usano in matematica. Espressioni numeriche si possono scrivere in C con la seguente sintassi:
  - `<espressione> <operatore_aritmetico> <espressione>`
- In particolare il C fornisce i seguenti operatori aritmetici:
  - Somma  $\rightarrow x+10$
  - Differenza  $\rightarrow x-10$
  - Prodotto  $\rightarrow x*10$
  - Divisione intera  $\rightarrow x/10$
  - Modulo  $\rightarrow x\%10$   $\rightarrow$  fa la divisione e restituisce il resto
  - Parentesi  $\rightarrow 10*(x+2)$
  - Incremento  $\rightarrow x++$  equivalente a  $x=x+1$
  - Decremento  $\rightarrow x--$  equivalente a  $x=x-1$



# Indirizzo di variabili

- In C possiamo accedere non solo al *valore* di una variabile ma anche al suo *indirizzo* → locazione in memoria a cui la variabile si trova. Per ottenere l'indirizzo di una variabile usiamo l'operatore &:
  - &<variabile>
- Ad esempio &x si riferisce all'indirizzo della variabile x. Vedremo più avanti come usare questi indirizzi in generale, che per ora utilizzeremo per input e output.

# Input & Output: printf

- Abbiamo già visto che la printf si usa per stampare a video un messaggio...ma può essere usata anche per stampare i valori di una o più variabili.
- La funzione printf ha la seguente sintassi:  
printf(<argomento\_1>)  
printf(<argomento\_1>,<argomento\_2>)  
printf(<argomento\_1>,...,<argomento\_n>)

# Input & Output: printf

- **printf** ha uno o più argomenti. Il primo argomento è una sequenza di caratteri, delimitati da ", che indica cosa si vuole stampare. Gli altri argomenti sono variabili o espressioni di cui si vuole stampare il valore.
- Quindi:
  - per stampare un valore di una variabile di tipo int, si usa il simbolo %d nella stringa di caratteri.
  - `printf("Stampiamo in valore di x: %d", x);`

# Input & Output: scanf

- In C per inserire un input da tastiera bisogna usare la funzione **scanf**, contenuta in `stdio.h`
- Il formato della funzione `scanf` è simile alla `printf`:
  - Il primo argomento è una sequenza di caratteri di escape che indica cosa si vuole inserire in input. Gli altri argomenti sono variabili nelle quali memorizzare il valore inserito.
- Per far ciò, `scanf` si deve riferire all'indirizzo di queste variabili. Il formato è il seguente:
  - `scanf(<stringa>, <indirizzi_di_variabili>);`

Quindi, per leggere un valore `int` e salvarlo nella variabile `x` si usa il comando:

- `scanf('%d', &x);`



# Esecuzione sequenziale

- Le istruzioni C che abbiamo coperto finora (dichiarazioni e assegnamenti), vengono eseguiti in modo sequenziale, uno dopo l'altro. Ad esempio, il codice:

```
int x, y;  
x = 10;  
y = x * 5;  
x = y + 10;
```

- inizia assegnando a x il valore 10, per poi assegnare a y il valore 50 (10, memorizzato in x, per 5). Il valore di x cambia poi nell'ultima riga a 60 (50, memorizzato in y, più 10).





# Tipi primitivi

- Il C definisce vari *tipi primitivi*, che chiamiamo così perché sono definiti nel linguaggio stesso.
- In C (e in tutti gli altri linguaggi di programmazione), i numeri non hanno precisione infinita.
- Ogni variabile occupa un quantità fissa di memoria specificata dal tipo di dati.
  - Ad esempio una variabile di tipo **int** occupa 4 bytes, equivalenti a 32 bits e può rappresentare solo  $2^{32}$  valori diversi.



# Tipi primitivi

- Il C offre vari tipi per rappresentare numeri interi. In genere, useremo il tipo int

tipo	bits	bytes	min	max	printf/scanf
char	8	1	-127	127	"%hhd"
short	16	2	-32.767	32.767	"%hd"
int	32	4	-2.147.483.647	2.147.483.647	"%d"
long	32	4	-2.147.483.647	2.147.483.647	"%ld"
long	64	8	-2 <sup>63</sup>	2 <sup>63</sup>	"%lld"
long					



# Tipi primitivi

- Se non abbiamo la necessità di rappresentare numeri negativi, possiamo dedicare tutti i bit del tipo a numeri positivi, per un modesto incremento di precisione.

tipo	bits	bytes	min	max	printf/scanf
<code>unsigned char</code>	8	1	0	255	<code>"%hhu"</code>
<code>unsigned short</code>	16	2	0	65.535	<code>"%hu"</code>
<code>unsigned int</code>	32	4	0	4.294.967.295	<code>"%u"</code>
<code>unsigned long</code>	32	4	0	4.294.967.295	<code>"%lu"</code>
<code>unsigned long long</code>	64	8	0	2 <sup>64</sup> -1	<code>"%llu"</code>



# Tipi primitivi

- Per rappresentare numeri frazionari, il calcolatore usa la rappresentazione in virgola mobile (*floating point*).

tipo	bits	bytes	prec.	min e.	max e.	printf/scanf
float	32	4	6	-37	38	"%f"
double	64	8	15	-307	308	"%lf"
long double	128	16	18	-4931	4932	"%Lf"



# Promozione di tipi

- Quando una operazione ha operandi di tipo diverso, il C converte automaticamente il tipo più piccolo a quello più grande. Formalmente diciamo che avviene una *promozione* da un tipo all'altro.
- Ad esempio, se si somma una variabile **short** and una **int**, la variabile short viene convertita ad **int** automaticamente dal linguaggio

# Il cast

- A volte può interessare effettuare dei cambiamenti da un tipo *meno capiente* ad un tipo *più capiente*. In tali casi il linguaggio C mette a disposizione del programmatore un costrutto chiamato **cast**.

```
1  int uno, due;  
2  float tre;  
3  
4  uno = 1;  
5  due = 2;  
6  tre = (float) uno/due;  
7  printf("%f", tre);
```



# Controlli condizionali

## If - Else

```
if ( espressione ) {  
    istruzione 1  
    ....  
    istruzione n  
} else {  
    istruzione 1  
    ...  
    istruzione n  
}
```

## switch

```
switch ( variabile ){  
  
    case valore1:  
        istruzioni... ;  
        break;  
  
    case valore2:  
        istruzioni... ;  
        break;  
  
    default:  
        istruzioni... ;  
        break;  
}
```



# Operatori relazionali

- Per esprimere condizioni tra espressioni si usano gli operatori relazionali con la seguente sintassi:

*<espressione> <operatore\_relazionale> <espressione>*

- In particolare, il linguaggio C definisce i seguenti operatori relazionali:
  - minore:  $5 < x$ , che risulta *vero* per  $x$  uguale a 10
  - maggiore:  $5 > x$ , che risulta *falso* per  $x$  uguale a 10
  - minore o uguale:  $5 \leq x$ , che risulta *vero* per  $x$  uguale a 5
  - maggiore o uguale:  $5 \geq x$ , che risulta *vero* per  $x$  uguale a 5
  - uguale:  $5 == x$ , che risulta *vero* per  $x$  uguale a 5
  - diverso:  $5 != x$ , che risulta *falso* per  $x$  uguale a 5





# Operatori logici

- Varie condizioni di possono combinare usando gli operatori logici.
- Il C definisce gli operatori logici:
  - *AND* → `&&`
  - *OR* → `||`
  - *NOT* → `!`

# Controlli iterativi



## WHILE

```
while (condizione) {  
    istruzioni...  
}
```

## DO - WHILE

```
do {  
    istruzioni...  
} while (condizione)
```

## FOR

```
for (inizializzazione ; condizione; incremento){  
    istruzioni...  
}
```

**Es.**

```
int i;  
for (i=0; i<100 ; i++)  
    printf( "%d \n ",i);
```

**break** : esce dal ciclo o dallo switch

**continue** : salta un'iterazione del ciclo senza interromperlo



# Esercizio 1

Si scriva un programma che legge in input due numeri interi.

**1)** Il programma deve stampare se il primo numero è maggiore, minore o uguale al secondo.

**2)** Il programma deve stampare la somma e il prodotto dei due numeri.



# SOLUZIONE Esercizio 1

```
10  #include<stdio.h>
11
12  ▼ int main(){
13
14      int a = 0;
15      int b = 0;
16
17      printf("Inserire il primo numero: \n");
18      scanf("%d",&a);
19      printf("\nInserire il secondo numero: ");
20      scanf("%d",&b);
21      printf("\n");
22
23      int somma = a +b;
24      int prodotto = a * b;
25
26      if (a > b)
27          printf("%d > %d ",a,b);
28      else if (a < b)
29          printf("%d < %d ",a,b);
30      else
31          printf("%d == %d ",a,b);
32
33      printf("\nSomma: %d, prodotto: %d\n",somma,prodotto);
34
35
36      return 0;
37
```



## Esercizio 2

Si scriva un programma che legge in input un numero intero “a”.

**1)** Il programma stampa tutti i numeri da 0 ad “a” in ordine **crescente**.

**2)** Il programma stampa tutti i numeri da 0 ad “a” in ordine **decrescente**.

N.B. per svolgere l’esercizio si utilizzino cicli FOR



# SOLUZIONE Esercizio 2

```
10  #include<stdio.h>
11
12  ▼ int main(){
13
14      int numero = 0;
15
16      printf("Inserire un numero: \n");
17      scanf("%d",&numero);
18
19      int i = 0;
20
21      printf("Numeri in ordine crescente--> \n");
22  ▼  for(i=0;i<=numero;i++){
23          printf("%d - ",i);
24      }
25
26      printf("\nNumeri in ordine decrescente--> \n");
27  ▼  for(i=numero;i>=0;i--){
28          printf("%d - ",i);
29      }
30
31      printf("\nFINE PROGRAMMA \n");
32
33      return 0;
```



## Esercizio 3

Si scriva un programma che legge in input un numero intero "a".

**1)** Il programma stampa tutti i numeri da 0 ad "a" in ordine **crescente**.

**2)** Il programma stampa tutti i numeri da 0 ad "a" in ordine **decrescente**.

N.B. per svolgere l'esercizio si utilizzino cicli WHILE



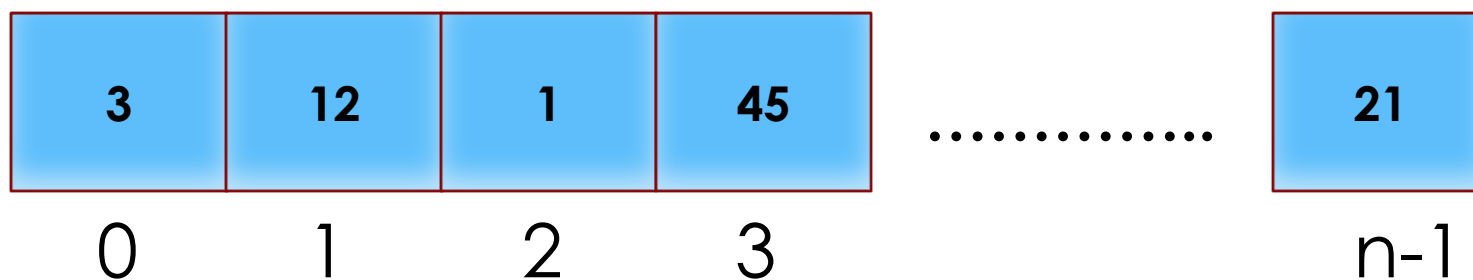
# SOLUZIONE Esercizio 3

```
12  #include<stdio.h>
13
14  ▼ int main(){
15
16      int numero = 0;
17
18      printf("Inserire un numero: \n");
19      scanf("%d",&numero);
20
21      int i = 0;
22
23      printf("Numeri in ordine crescente--> \n");
24  ▼ while(i<=numero){
25          printf("%d - ",i);
26          i++;
27      }
28
29
30      i=numero;
31      printf("\nNumeri in ordine decrescente--> \n");
32  ▼ while(i>=0){
33          printf("%d - ",i);
34          i--;
35      }
36
37      printf("\nFINE PROGRAMMA \n");
38
39      return 0;
40
```



# Gli Array

Un array è un insieme di elementi **dello stesso tipo**, che possono essere identificati tramite un indice. Dato un array di  $n$  elementi, gli indici vanno da 0 a  $n-1$



## Dichiarazione:

tipo nomeArray [dimensione];

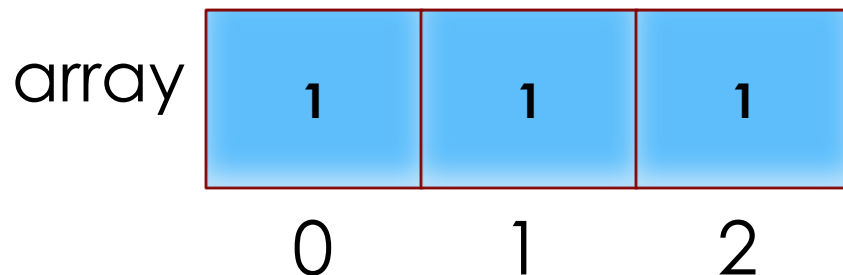
## Esempio:

```
int arrayInteri[100];
```

# Gli Array

**Esempio:** Dichiariamo un array di interi di dimensione 3 e inseriamo in tutte le posizioni il numero 1.

```
int array [3];  
array[0] = 1;  
array[1] = 1;  
array[2] = 1;
```



Cosa succede con la seguente istruzione?

```
printf ( " %d ", array[3] );
```





# Gli Array

**Nessun errore in compilazione.**

Ma ciò non significa che è corretto!

Non è possibile prevedere cosa viene stampato.  
In ogni caso non si deve mai andare a leggere  
o scrivere al di fuori dell'array!



# Dichiarazione di un array

```
11     int num[10];  
12  
13     char c[5];  
14  
15     int numeri[] = {1,5,7,3};  
16  
17     char caratteri[] = {'c','i','a','o'};  
18  
19     char caratteri2[] = "ciao";
```

# Accedere agli elementi di un array



23  
24  
25  
26 ▼  
27  
28  
29  
30  
31  
32  
33  
34  
35

```
int array[10];  
  
int i;  
for (i=0;i<10;i++) {  
    array[i] = i;  
}  
  
int a = 2;  
int b = 3;  
  
array[a+b] = 15;  
  
array[a*b] = 11;
```



# Array: un esempio

```
9  ▼ int main() {
10
11     int dimensione = 100;
12     int array[dimensione];
13
14     int i = 0;
15
16  ▼ for (i=0; i<dimensione; i++) {
17         array[i] = i;
18     }
19
20  ▼ for (i=0; i<dimensione; i++) {
21         printf("%d \n", array[i]);
22     }
23
24     return 0;
25 }
```



## Esercizio 4

Si scriva un programma che legge in input 5 numeri interi e li salva in un array

- 1) Il programma stampa il numero più grande e il numero più piccolo
- 2) Il programma stampa la somma di tutti i numeri

N.B. per svolgere l'esercizio si deve utilizzare un array e i cicli FOR


# SOLUZIONE Esercizio 4



```
13 ▼ int main() {
14
15     int array[5];
16     int i=0;
17
18 ▼     for (i=0;i<5;i++){
19         printf("Inserire l'elemento %d: \n",i);
20         scanf("%d",&array[i]);
21     }
22
23     int minimo = array[0];
24     int massimo = array[0];
25     int somma = 0;
26
27 ▼     for (i=0;i<5;i++){
28         if (array[i]<minimo)
29             minimo=array[i];
30
31         if(array[i]>massimo)
32             massimo=array[i];
33
34         somma+=array[i];
35     }
36
37     printf("MINIMO: %d - MASSIMO: %d - SOMMA: %d \n",minimo,massimo,somma);
```



# Gli array multidimensionali

- È possibile rappresentare tabelle e matrici
- Si possono rappresentare strutture che hanno più di una dimensione
- Esempio: dichiarare una matrice composta da  $N$  righe e  $M$  colonne:
  - ***int matrix[n][m]; //n=numero righe, m=numero colonne***
- Per accedere agli elementi è opportuno specificare l'indice della riga e della colonna:
  - ***matrix[0][1] = 4;*** 
  - ***printf("%d",matrix[0][1]);***

**Elemento alla riga  
0 e alla colonna 1**



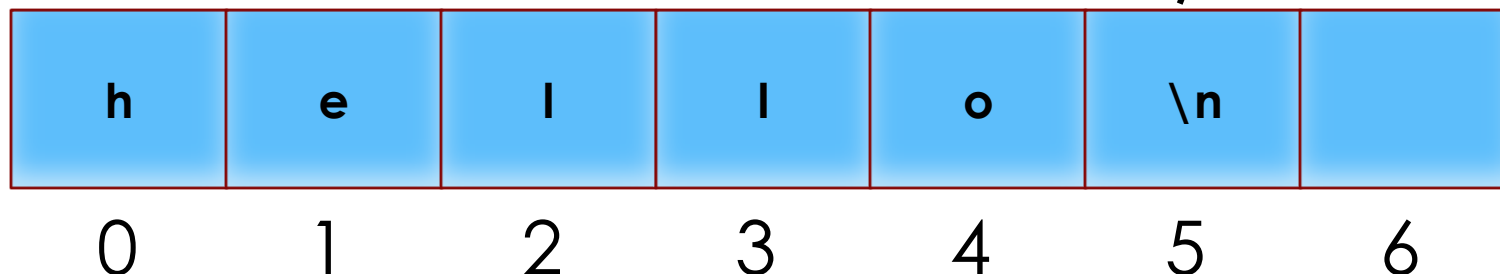
# Un esempio

```
5     int matrice[2][3];
6
7     int riga = 0;
8     int colonna = 0;
9
10    ▼ for ( riga=0; riga<2; riga++ ) {
11    ▼     for ( colonna=0; colonna<3; colonna++ ) {
12         printf("Input [%d][%d] \n",riga,colonna);
13         scanf("%d",&matrice[riga][colonna]);
14     }
15 }
16
17 ▼ for ( riga=0; riga<2; riga++ ) {
18 ▼     for ( colonna=0; colonna<3; colonna++ ) {
19         printf("%d ",matrice[riga][colonna]);
20     }
21     printf("\n");
22 }
```

# Le stringhe

- 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



# Leggere una stringa in input e stamparla



- Step per leggere una stringa in input e stamparla:

- Dichiarazione:

- `char stringa[20];`

- Lettura in input:

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

- Stampa in output:

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

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

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

Stampa ciò che è contenuto nell'array *stringa* fino al carattere `\0` (escluso)



## Esercizio 5

- Si scriva un programma che legge in input una stringa e la salva in un Array di char di 20 elementi
- Il programma deve sostituire tutte le cifre che compaiono nella stringa con il carattere \*
- Infine deve stampare la stringa modificata
- *N.B. per controllare se il char **c** è un numero:*
- $(c \geq '0' \ \&\& \ c \leq '9')$



# Soluzione esercizio 5

```
5 char array[20];
6 printf("Stringa in input: ");
7 scanf("%s",array);
8
9 int i = 0;
10 while (array[i] != '\0') {
11     if ( (array[i] >= '0') && (array[i] <= '9') ) {
12         array[i] = '*';
13     }
14     i++;
15 }
16
17 printf("%s \n",array);
```

# Copiare una stringa

- 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 →

```
#include <string.h>
```

Alcune delle funzioni:

- *strlen(stringa);*
- *strcpy(destinazione, sorgente);*
- *strcat(destinazione, sorgente);*
- *strcmp(s1, s2);*





# Le funzioni

- 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”
- Vengono create ogni volta che si entra nella funzione e distrutte (perdendo il valore) quando si esce

```
tipo nomeFunzione (parametri)  
{  
    definizione_variabili_locali  
    istruzioni  
    eventuale return  
}
```

} *corpo della funzione*



# Esempio di funzione

```
#include <stdio.h>
```

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



**Dichiarazione  
della funzione**

```
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;
}
```



**Chiamata alla funzione**



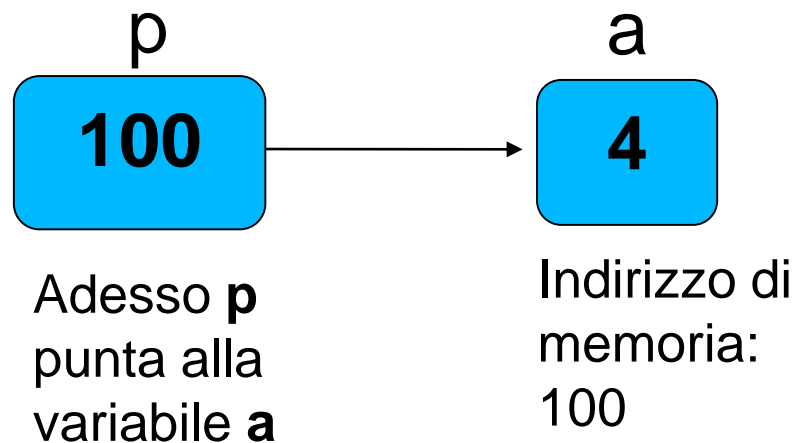
# Passaggio di parametri

- I parametri vengono passati per valore alle funzioni.
- Questo significa che nella funzione viene creata una **copia**, quindi 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.
- Come modificare i valori di variabili utilizzando delle funzioni?
- Dato che le funzioni ammettono la possibilità di restituire un solo valore, come possiamo fare se abbiamo bisogno di restituire più valori?

# I puntatori

- Un puntatore è una variabile che contiene l'indirizzo di memoria di un'altra variabile.

Esempio, dichiariamo una variabile intera "a" ed un puntatore "p" che punta ad "a".





# I 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

# I puntatori

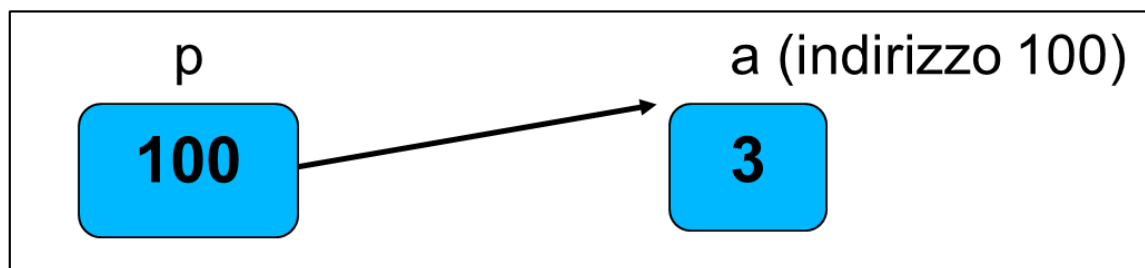
```
//variabile intera  
int a = 3;  
//puntatore a intero  
int *p;
```



- \* 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;`

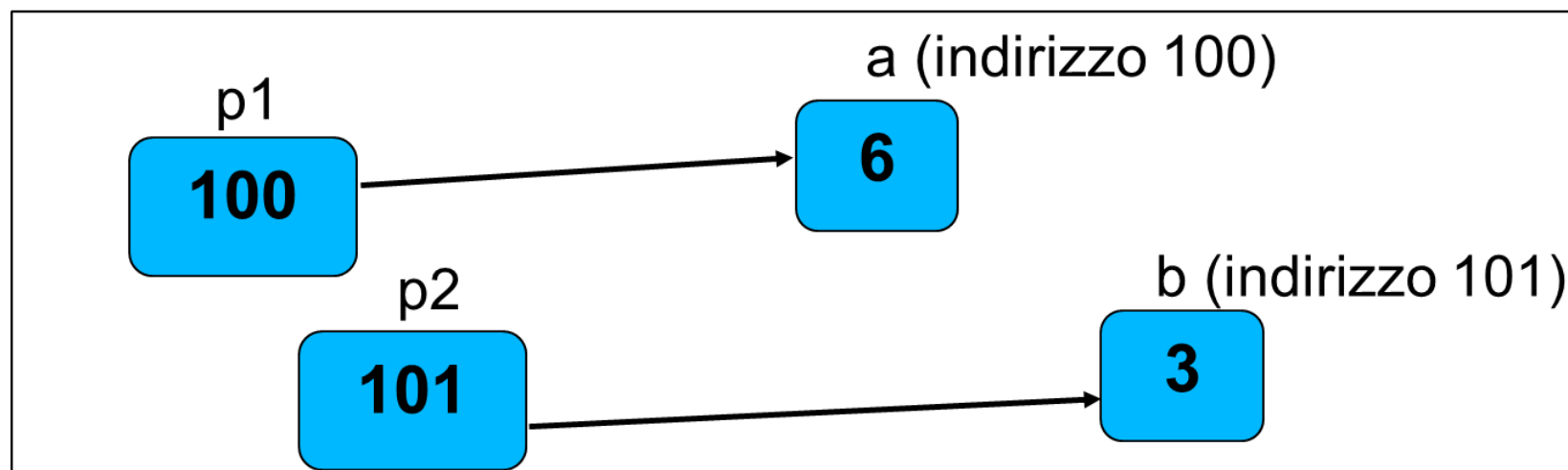


- L'operatore & restituisce l'indirizzo in memoria della variabile *a*



# I puntatori

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

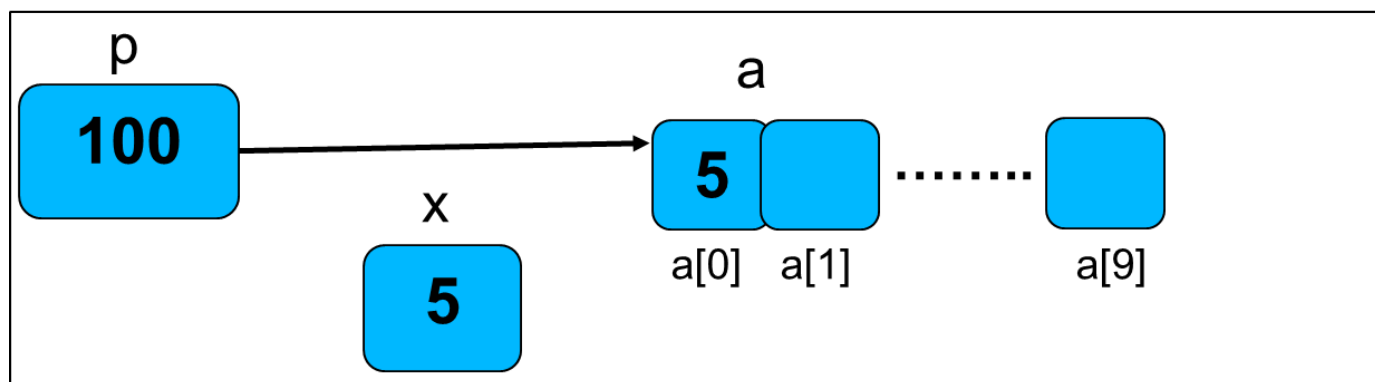


# Aritmetica dei puntatori

```
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**





# Aritmetica dei puntatori

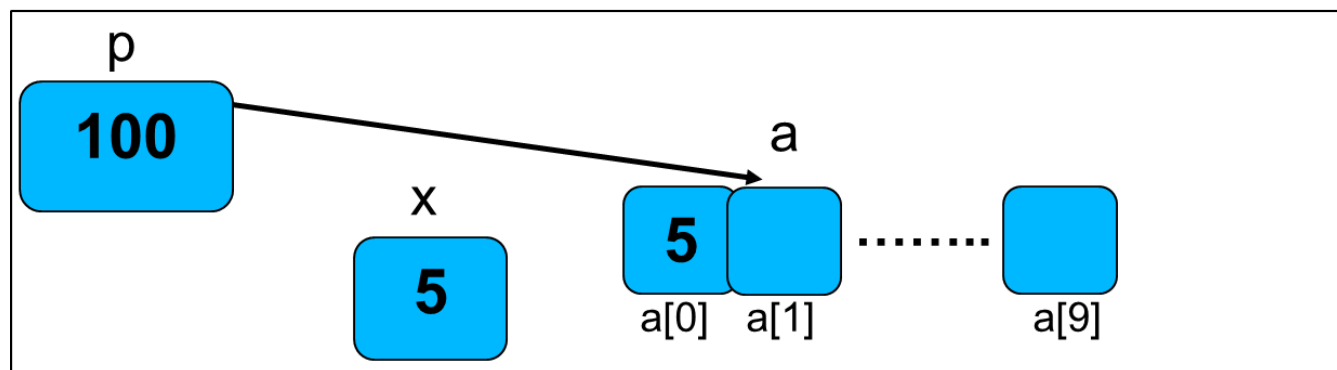
```
int a[10], x;  
int *p;  
p = &a[0];  
x = *p;  
p++;
```

- 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)



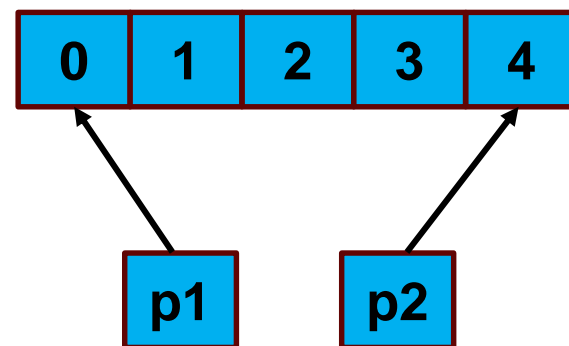
# Esercizio

```
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**

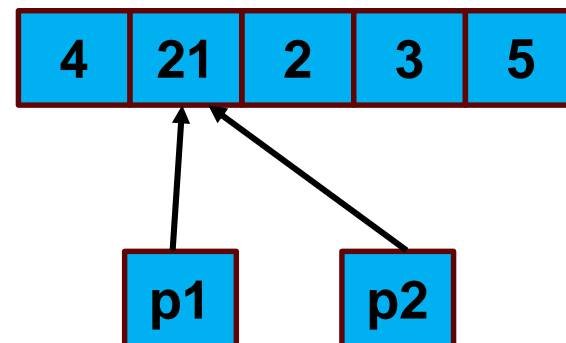
# Esercizio

```
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

```
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

- Si scriva un programma che dichiari un Array di char di 10 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 - soluzione

```
6 char stringa[10];
7 char *c;
8 c = &stringa[0];
9
10 printf("Stringa in input: ");
11 scanf("%s",stringa);
12
13 ▼ while (*c != '\0'){
14 ▼     if ( (*c >= '0') && (*c <= '9') ) {
15         *c = '*';
16     }
17     c++;
18 }
19
20 printf("\nStringa modificata: %s \n",stringa);
```



# Esercizio

- Si scriva un programma che dichiari due Array di interi (array1 e array2) di 5 elementi e permetta all'utente di riempirli
- Successivamente, tramite l'utilizzo di due puntatori, inverti 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



# Esercizio - soluzione

```
5   int array1[5], array2[5];
6   int i=0;
7   ▼ for(i=0;i<5;i++){
8       printf("Input array1[%d]: ",i);
9       scanf("%d",&array1[i]);
10      printf("\nInput array2[%d]: ",i);
11      scanf("%d",&array2[i]);
12  }
13
14  int *p1,*p2;
15  p1 = &array1[0];
16  p2 = &array2[4];
17
18  ▼ for(i=0;i<5;i++){
19      int a = *p1;
20      *p1 = *p2;
21      *p2 = a;
22      p1++;
23      p2--;
24  }
25  printf("\nStampo gli array...\n");
26  ▼ for(i=0;i<5;i++){
27      printf("Input array1[%d]: %d \n",i,array1[i]);
28      printf("Input array2[%d]: %d \n",i,array2[i]);
29  }
```



# Esercizio

Scrivere una funzione che dato un array di interi e un valore  $x$  cerca il valore  $x$  nell'array. Se il valore  $x$  è presente allora restituirà la posizione altrimenti un codice di errore 404.



# Esercizio - soluzione

```
1
2 #include <stdio.h>
3
4 int ricerca (int*, int, int);
5
6 int main() {
7
8     int array[10];
9     for(int i=0; i<10; i++) {
10         array[i]=i;
11     }
12
13     int valore = ricerca(array, 10, 4);
14
15     printf("\nIl valore restituito dalla funzione ricerca e': %d", valore);
16 }
17
18
19 int ricerca(int* array, int dim, int x) {
20
21     for(int i=0; i<dim; i++) {
22
23         if(x == array[i]) {
24             return i;
25         }
26     }
27     return 404;
28 }
29
```



# Esercizio - soluzione

Modificare la funzione di ricerca affinché restituisca anche il numero di occorrenze del valore ricercato

# Esercizio - soluzione

```
3 #include <stdio.h>
4 int ricerca (int*, int, int, int*);
5 int main() {
6
7     int array[10];
8     for(int i=0; i<10; i++) {
9         array[i]=i;
10    }
11    int n_occ=0;
12    int *ptr_n_occ=&n_occ;
13    int valore = ricerca(array, 10, 4, ptr_n_occ);
14    if(valore != 404){
15        printf("\nIl valore restituito dalla funzione ricerca e': %d il numero di occorrenze e' %d ", valore,n_occ);
16    } else {
17        printf("\nIl valore restituito dalla funzione ricerca e': %d", valore);
18    }
19
20 }
21
```



# Esercizio

```
22
23 ▾ int ricerca(int* array, int dim, int x, int *occ) {
24     bool find=false;
25     int ret_pos=404;
26     int n_occ=0;
27 ▾ for(int i=0; i<dim; i++) {
28
29 ▾     if(x == array[i]) {
30 ▾         if (!find) {
31             find=true;
32             ret_pos=i;
33         }
34         n_occ++;
35     }
36 }
37 *occ = n_occ;
38 return ret_pos;
39 }
40
```



# Esercizio

- Scrivere una funzione **int included(int A[],int n, int B[], int m)** che ritorna 1 se tutti i valori nel vettore A (di dimensione n) sono anche contenuti nel vettore B (di dimensione m), altrimenti ritorna 0. Ad esempio, se  $A = \{2, 1, 3, 2, 1\}$  e  $B = \{5, 1, 3, 2\}$  allora la funzione ritorna 1, se invece  $B = \{5, 3, 2, 0\}$  ritorna 0.



# Esercizio - soluzione

```
2  #include<stdio.h>
3
4  // Ritorna 1 se tutti i valori di A sono inclusi in B, 0 altrimenti
5  int included(int A[], int n, int B[], int m) {
6      for (int i = 0 ; i < n ; i++) {
7          int j = 0;
8          while (j < m && B[j] != A[i]) j++;
9          if (j == m) return 0;
10     }
11     return 1;
12 }
13
14
15 int main() {
16
17     int A[5]={2, 1, 3, 2, 1};
18     int B[4]= {5, 1, 3, 2};
19     int res = included(A, 5, B, 4);
20     printf("\n Il risultato di included e': %d", res);
21     return 0;
22 }
23
```