

# INPUT/OUTPUT

---

- L'immissione dei dati di un programma e l'uscita dei suoi risultati avvengono attraverso operazioni di **lettura e scrittura**
- **C non ha istruzioni predefinite** per l'input/output
- In ogni versione ANSI C, esiste una **Libreria Standard (stdio)** che mette a disposizione alcune funzioni (dette *funzioni di libreria*) per effettuare l'input e l'output

1

# INPUT/OUTPUT

---

- Le dichiarazioni delle funzioni messe a disposizione da tale libreria devono essere incluse nel programma:  
**#include <stdio.h>**
  - **#include** è una direttiva per il **preprocessore C**
  - nella fase precedente alla compilazione del programma ogni direttiva “#...” viene eseguita, provocando delle modifiche testuali al programma sorgente. Nel caso di **#include <nomefile>** viene sostituita l'istruzione stessa con il contenuto del file specificato
- **Dispositivi standard di input e di output:**  
per ogni macchina, sono periferiche predefinite (generalmente tastiera e video)

2

# INPUT/OUTPUT

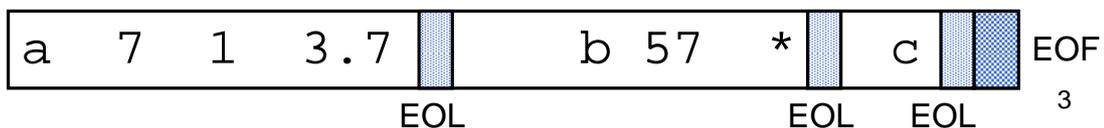
---

A default, C vede le informazioni lette/scritte da/verso i dispositivi standard di I/O come file *sequenziali*, cioè **sequenze di caratteri** (o stream). Vedremo più avanti la possibilità di fare anche I/O in cosiddetto formato binario...

- Gli *stream* di input/output possono contenere dei caratteri di controllo:
  - End Of File (EOF)
  - End Of Line (EOL)

**Sono disponibili funzioni di libreria per:**

- Input/Output a caratteri
- Input/Output a stringhe di caratteri
- Input/Output con formato



# INPUT/OUTPUT CON FORMATO

---

Nell'I/O con formato occorre specificare il **formato** (*tipo*) dei dati che si vogliono leggere oppure stampare

Il **formato** stabilisce:

- **come interpretare** la sequenza dei caratteri immessi dal dispositivo di ingresso (nel caso della lettura)
- con quale sequenza di caratteri **rappresentare** in uscita i valori da stampare (nel caso di scrittura)

## LETTURA CON FORMATO: scanf

---

È una **particolare forma di assegnamento**: scanf() assegna i valori letti alle variabili specificate come argomenti (nell'ordine di lettura)

```
scanf(<stringa-formato>, <sequenza-variabili>);
```

Ad esempio:

```
int X;  
float Y;  
scanf("%d%f", &X, &Y);
```

5

## LETTURA CON FORMATO: scanf

---

scanf() legge una serie di valori in base alle specifiche contenute in *<stringa-formato>* e memorizza i valori letti nelle variabili

- restituisce il **numero di valori letti** e memorizzati, oppure EOF in caso di *end of file*
- gli **identificatori** delle variabili a cui assegnare i valori sono sempre preceduti dal **simbolo &** (ne parleremo diffusamente...)
- la *<stringa\_formato>* può contenere dei caratteri qualsiasi (scartati durante la lettura), che si prevede vengano immessi dall'esterno, insieme ai dati da leggere

```
scanf("%d:%d:%d", &A, &B, &C);
```

richiede che i tre dati vengano immessi separati dal carattere ":"  
6

## SCRITTURA CON FORMATO: `printf`

---

- `printf()` viene utilizzata per fornire in uscita il valore di una variabile o, più in generale, il risultato di una espressione
- Anche in scrittura è necessario specificare (mediante una *stringa di formato*) il formato dei dati che si vogliono stampare

```
printf(<stringa-formato>, <sequenza-elementi>)
```

7

## SCRITTURA CON FORMATO: `printf`

---

- `printf` scrive una serie di valori in base alle specifiche contenute in *<stringa-formato>*
- I valori visualizzati sono i risultati delle espressioni che compaiono come argomenti
- `printf` restituisce il numero di caratteri scritti
- La stringa di formato della `printf` può contenere sequenze costanti di caratteri da visualizzare

8

# FORMATI COMUNI

---

- **Formati più comuni**

<code>int</code>	<code>%d</code>
<code>float</code>	<code>%f</code>
<code>carattere singolo</code>	<code>%c</code>
<code>stringa di caratteri</code>	<code>%s</code>

- **Caratteri di controllo**

<code>newline</code>	<code>\n</code>
<code>tab</code>	<code>\t</code>
<code>backspace</code>	<code>\b</code>
<code>form feed</code>	<code>\f</code>
<code>carriage return</code>	<code>\r</code>

- Per la stampa del carattere ' % ' si usa: `%%`

9

---

## ESEMPIO

---

```
main()  
{  
  int    k;  
  scanf("%d",&k);  
  printf("Quadrato di %d: %d",k,k*k);  
}
```

Se in ingresso viene immesso il dato:

3 viene letto tramite la `scanf` e assegnato a `k`

`printf()` stampa:

Quadrato di 3: 9

10

# Esempio

```
int main()
{
printf("Prime sei potenze di 2:  %d, %d,
      %d, %d, %d, %d", 1, 2, 4, 8, 16, 32);
}
```

11

## ESEMPIO

---

```
scanf("%c%c%c%d%f", &c1,&c2,&c3,&i,&x);
```

Se in ingresso vengono dati:

ABC 3 7.345

`scanf()` effettua i seguenti assegnamenti:

char c1	'A'
char c2	'B'
char c3	'C'
int i	3
float x	7.345

12

## ESEMPIO

---

```
char Nome='F';
char Cognome='R';
printf("%s\n%c. %c. \n\n%s\n",
       "Programma scritto da:",
       Nome, Cognome, "Fine");
```

---

vengono stampate le seguenti linee

```
Programma scritto da:
```

```
F. R.
```

```
Fine
```

13

## scanf: STRINGA DI FORMATO

---

```
#include <stdio.h>
main()
{
    int intero1, intero2;
    float reale1;
    char car1, car2;

    scanf("%d%d",&intero1,&intero2);
    printf("%d,%d",intero1,intero2);
}
```

14

## scanf: STRINGA DI FORMATO

---

```
scanf("%d%d",&intero1,&intero2);
```

Inserire due interi separati da uno o più spazi:

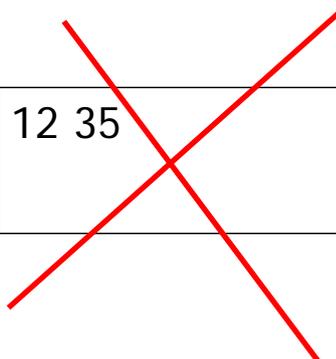
15

## scanf: STRINGA DI FORMATO

---

```
scanf("%d,%d",&intero1,&intero2);
```

Inserire due interi separati da una (e una sola) virgola (eventuali spazi sono scartati):



16

## scanf: STRINGA DI FORMATO

---

Regole:

- la stringa di formato descrive esattamente quello che deve esserci in input
- lo spazio bianco viene considerato un separatore e viene scartato

17

## PERO'...

---

Lo spazio bianco è a tutti gli effetti un carattere...  
quindi nella lettura di caratteri

```
scanf("%d%c%c",&interol,&car1,&car2);  
printf("%d,%c,%c",interol, car1,car2);
```

```
12 A B  
12, ,A
```

La scanf ha preso lo spazio come se fosse il carattere inserito !

18

## UNA SOLUZIONE

---

Usare un separatore (anche lo spazio stesso)

spazio  
↓  
`scanf("%d %c %c",&interol1,&car1,&car2);`  
`printf("%d,%c,%c",interol1, car1,car2);`

```
12 A B
12,A,B
```

19

## UN'ALTRA TRAPPOLA

---

```
printf("Inserire un numero reale: ");
scanf("%f",&reale1);
printf("\nInserire un carattere: ");
scanf("%c",&car1);
printf("\nLetti: %f,%c",reale1, car1);
```

Questo frammento di programma sembra corretto...

20

## UN'ALTRA TRAPPOLA

---

...ma il risultato è questo:

Inserire un numero reale:

12.4

Inserire un carattere:

Letti: 12.400000,

21

## MOTIVO

---

L' I/O è bufferizzato: i caratteri letti da tastiera sono memorizzati in un buffer.

In architetture Windows, il tasto di INVIO corrisponde a 2 (DUE!) caratteri (CR LF): il primo è interpretato come separatore, ma il secondo rimane nel buffer ed è preso come carattere inserito dall'utente.

22

## UNA SOLUZIONE

---

Leggere il carattere "spurio"

```
printf("Inserire un numero reale: ");
scanf("%f",&reale1);
scanf("%*c"); /* letto e buttato via */
printf("\nInserire un carattere: ");
scanf("%c",&car1);
printf("\nLetti: %f,%c",reale1, car1);
```

23

## UNA SECONDA SOLUZIONE

---

Vuotare il buffer:

```
printf("Inserire un numero reale: ");
scanf("%f",&reale1);
fflush(stdin); /* Si vuota il buffer */
printf("\nInserire un carattere: ");
scanf("%c",&car1);
printf("\nLetti: %f,%c",reale1, car1);
```

24

## PRECISAZIONE

---

Questo problema si verifica solo con la lettura di caratteri.

Negli altri casi il doppio carattere nel buffer e' considerato come sequenza di separatori e scartato.

25

## CHE COSA SUCCEDDE?

---

```
a=0;
printf("ciao e %d\n", a=1);

a=0;
printf("ciao e %d\n", a==1);

a=0;
printf("ciao e %d\n", a==0);

a=0;
printf("ciao e a=%d\n", a);

a=1;
if (a=4)
    printf("ciao e a=%d\n", a);
```

26

## CHE COSA SUCCUDE?

---

<code>a=0; printf("ciao e %d\n", a=1);</code>	ciao e 1
<code>a=0; printf("ciao e %d\n", a==1);</code>	ciao e 0
<code>a=0; printf("ciao e %d\n", a==0);</code>	ciao e 1
<code>a=0; printf("ciao e a=%d\n", a);</code>	ciao e a=0
<code>a=1; if (a=4)     printf("ciao e a=%d\n", a);</code>	ciao e a=4

27

## getchar() e putchar()

- **char getchar( );**
  - *Legge un carattere da standard input e lo restituisce*
  - Prima di cominciare a leggere, attende la pressione del tasto invio
  - La lettura termina quando viene restituito il carattere di fine linea (10)
- **void putchar(char c);**
  - *Scrive un carattere su standard output*

28

## esercizio

- Tentare di costruire le funzioni di lettura/scrittura (**scanf**, **printf**) partendo dalle funzioni base di lettura caratteri:
  - **char getchar( );**
  - **void putchar(char c);**