

# 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

# 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)

# 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...

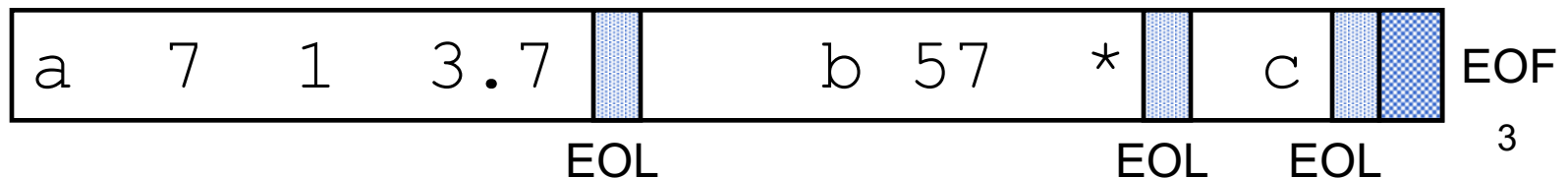
- Gli *stream* di input/output possono contenere dei **codici** di controllo:
  - End Of File (EOF)
  - End Of Line (EOL)

NOTA: i file non contengono EOF/EOL... Questi sono codici di controllo inseriti nello stream! Non sono caratteri, ad esempio non sono elencati nella tabellina ASCII...

Gli stream possono però contenere i caratteri LF (Line Feed) e CR (Carriage Return).

**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`

---

Può essere intesa come 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);
```

# 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 il codice EOF in caso di *end of file*
- gli **identificatori** delle variabili a cui assegnare i valori sono sempre (salvo una eccezione) 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

## Esempio:

```
scanf ("%d:%d:%d", &a, &b, &c) ;
```

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

# 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>)
```

# 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



# FORMATI COMUNI

---

- ***Formati più comuni***

int	%d
float	%f
carattere singolo	%c
stringa di caratteri	%s

- ***Caratteri di controllo***

newline	\n
tab	\t
backspace	\b
form feed	\f
carriage return	\r

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

# ESEMPIO

---

```
#define _CRT_SECURE_NO_DEPRECATE
#include <stdio.h>
int main() {
    int k;
    scanf("%d",&k);
    printf("Quadrato di %d: %d", k, k*k);
    return 0;
}
```

Se in ingresso viene immesso il dato:

3 viene letto tramite la scanf e assegnato a **k**  
**printf()** stampa:

Quadrato di 3: 9

# Esempio

```
#define _CRT_SECURE_NO_DEPRECATE
```

```
int main() {  
    printf("Prime sei potenze di 2: %d,%d,%d,  
        %d,%d,%d", 1, 2, 4, 8, 16, 32);  
    return 0;  
}
```

# 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:

<code>char c1</code>	<code>'A'</code>
<code>char c2</code>	<code>'B'</code>
<code>char c3</code>	<code>'C'</code>
<code>int i</code>	<code>3</code>
<code>float x</code>	<code>7.345</code>

# 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**

# scanf: STRINGA DI FORMATO

---

```
#define _CRT_SECURE_NO_DEPRECATED

#include <stdio.h>

int main() {
    int intero1, intero2;
    float reale1;
    char car1, car2;

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

    return 0;
}
```

## scanf: STRINGA DI FORMATO

---

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

Inserire due interi separati da uno o più spazi:



12 35



12 35

## scanf: STRINGA DI FORMATO

---

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

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

12,35

12, 35

~~12 35~~



# `scanf`: STRINGA DI FORMATO

---

Regole:

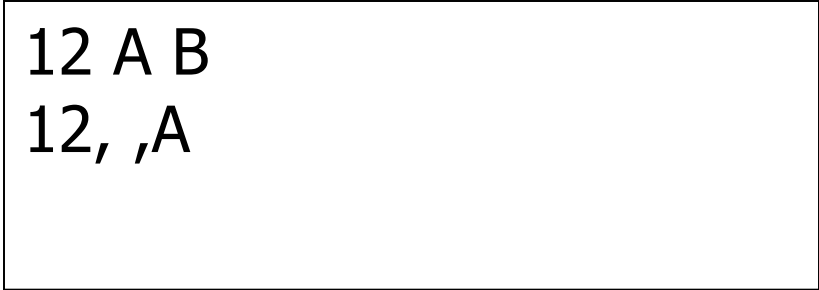
- la stringa di formato descrive esattamente quello che deve esserci in input
- lo spazio bianco viene considerato un separatore e viene scartato
- Caratteri separatori:
  - Se legge un formato diverso da `%c`, ignora i caratteri separatori (spazi, tab, invio, etc.)
  - Se legge un formato `%c`, allora restituisce anche i caratteri separatori

## Caratteri separatori...

---

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

```
scanf("%d%c%c", &intero1, &car1, &car2);  
printf("%d,%c,%c", intero1, car1, car2);
```



12 A B  
12, ,A

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

# UNA SOLUZIONE

---

Usare un separatore (anche lo spazio stesso)

spazio



```
scanf("%d %c %c", &intero1, &car1, &car2);  
printf("%d,%c,%c", intero1, car1, car2);
```

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

# 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...

# UN'ALTRA TRAPPOLA

---

...ma il risultato è questo:

Inserire un numero reale:  
12.4

Inserire un carattere:  
Letti: 12.400000,

# MOTIVO

---

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

Il separatore posto in fondo ad un campo con formato (ad esempio un intero o float) potrebbe non essere consumato ma rimanere nel 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 (solo da file).

# 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);
```

# PRECISAZIONE

---

Questo problema si verifica solo con la lettura di caratteri.

Negli altri casi sono considerati come separatori e scartati.



## La scanf è deprecata...

---

La funzione `scanf(...)` da alcuni anni è considerata “**deprecata**”, e il suo utilizzo ne viene sconsigliato.

Al suo posto, negli standard più recenti del linguaggio C, si suggerisce l'uso della funzione `scanf_s(...)`

Perché `scanf(...)` è deprecata? Motivi di sicurezza, ne comprenderemo il vero motivo quando affronteremo le stringhe e la loro rappresentazione in C.

## La scanf è deprecata e VS non compila...

---

Da diversi anni i compilatori non compilano più programmi contenenti funzioni deprecate.

Visual Studio, a partire dalla versione 2015, non compila più programmi contenenti invocazioni alle funzioni scanf / fscanf / sscanf.

Si può *forzare* la compilazione, aggiungendo in testa ad ogni file, come primissime direttive per il pre-compilatore:

```
#define _CRT_SECURE_NO_DEPRECATE
```

In alternativa:

```
#define _CRT_SECURE_NO_WARNINGS
```

## **scanf versione *secured*: scanf\_s(...)**

---

I problemi di sicurezza della scanf sono relativa in particolare a:

- lettura di caratteri
- lettura di stringhe

scanf\_s(...) propone una modifica, quando si vanno a leggere singoli caratteri o stringhe:

dopo il campo destinazione, è necessario indicare anche il numero di caratteri che verranno letti

## **scanf versione *secured*: scanf\_s(...)**

---

scanf\_s(...) di caratteri o stringhe:

dopo il campo destinazione, è necessario indicare anche il numero di caratteri che verranno letti

```
char mioChar;
```

```
scanf_s("%c", &mioChar, 1);
```

```
char mioChar;
```

```
scanf("%c", &mioChar);
```

## **scanf versione *secured*: scanf\_s(...)**

---

scanf\_s(...) di caratteri o stringhe:

dopo il campo destinazione, è necessario indicare anche il numero di caratteri che verranno letti

```
char miaStringa[10];  
scanf_s("%s", miaStringa, 10);
```

```
char mioStringa[10];  
scanf("%s", miaStringa);
```

## **scanf versione *secured*: scanf\_s(...)**

---

scanf\_s(...) di interi o razionali:

**nessun cambiamento**

```
int mioNum;  
scanf_s ("%d", &mioNum) ;
```

```
char mioNum;  
scanf ("%d", &mioNum) ;
```

## scanf\_s(...) vs. scanf(...)

---

```
int num1, num2;  
int char c1, c2;  
int float f1;  
scanf_s("%d%c%d%c%f",  
        &num1, &c1, 1, &num2, &c2, 1, &f1);
```

```
int num1, num2;  
int char c1, c2;  
int float f1;  
scanf("%d%c%d%c%f",  
      &num1, &c1, &num2, &c2, &f1);
```

## Es. 1: `scanf_s(...)` vs. `scanf(...)`

---

Realizzare un programma che legga da standard input un valore intero, seguito da un carattere (separato da uno spazio).

- Il valore intero deve essere inteso come il saldo di un conto corrente espresso in centesimi di una certa valuta.
- Il carattere può essere 'E': in tal caso, il saldo è in euro; se il carattere è 'D', allora il saldo è in dollari.

Il programma poi provveda a stampare a video il valore intero in euro o in dollari (senza i centesimi), seguito dalla lettera specificata.

Si realizzi una prima versione del programma che usi la `scanf_s`; se ne realizzi poi una seconda che usa la `scanf`.



## Es. 1: scanf\_s(...) vs. scanf(...) – Soluzione

---

```
#include <stdio.h>

int main() {
    int valore;
    char valuta;
    scanf_s("%d %c", &valore, &valuta, 1);
    printf("%d %c", valore/100, valuta);
    return 0;
}
```

## Es. 1: scanf\_s(...) vs. scanf(...) – Soluzione

---

```
#define _CRT_SECURE_NO_DEPRECATE

#include <stdio.h>

int main() {
    int valore;
    char valuta;
    scanf("%d %c", &valore, &valuta);
    printf("%d %c", valore/100, valuta);
    return 0;
}
```

# getchar() e putchar()

- **int 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
  - Restituisce il carattere letto (cast a int), o EOF in caso di errore
- **int putchar (char c) ;**
  - *Scrive un carattere su standard output*
    - Restituisce il carattere letto (cast a int), o EOF in caso di errore