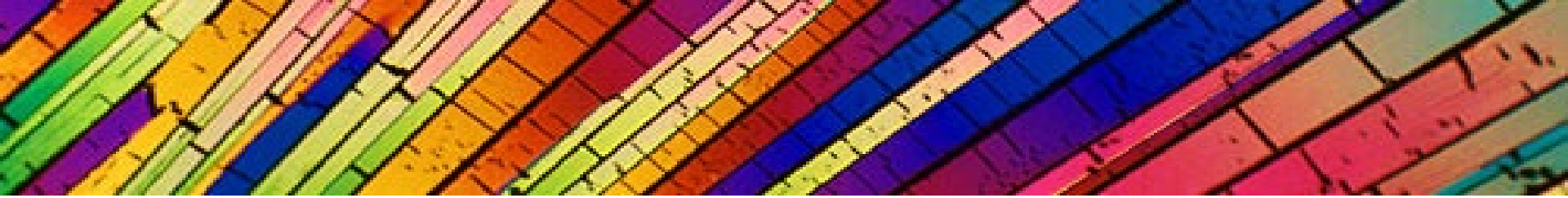

Input e output di valori numerici

Ver. 3

Funzioni di I/O per numeri

- Per utilizzare le funzioni di Input/Output bisogna includere il *file di intestazione* (*header file*) denominato `<stdio.h>`
- `<stdio.h>` contiene la sintassi d'uso delle funzioni per mezzo dei *prototipi* delle funzioni (descritti in altre slide) in modo che il compilatore possa controllare che l'utilizzo delle funzioni sia corretto
- `<stdio.h>` contiene anche la definizione di alcune costanti simboliche (`#define`), tra queste la `EOF` usata da alcune funzioni per segnalare la fine dell'input o del file

Stream preesistenti

- Quando un programma viene mandato in esecuzione, il sistema operativo gli fornisce 3 ***stream*** (flussi di dati) già *aperti* (pronti per essere utilizzati):
  - `stdin` [standard input] collegato alla tastiera
  - `stdout` [standard output] collegato al video per l'output *normale*, ossia per mostrare i dati prodotti dall'elaborazione
  - `stderr` [standard error] collegato al video per l'output di diagnostica, ossia per riportare informazioni relative al funzionamento del programma, ad esempio segnalazioni errori (che non sono i dati prodotti dall'elaborazione)

Input formattato – scanf

- La `scanf` è la più completa tra le funzioni di input da tastiera (**scan** **f**ormatted = scansione con formato)
- Sintassi:

  `scanf(`*stringa di formato*`,` *lista di variabili*`)`
- Legge valori dalla tastiera e li assegna alle variabili indicate nella *lista di variabili*
- Per ciascuna variabile deve esserci il corrispondente tipo nella *stringa di formato*

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

Input formattato – scanf

- Più in dettaglio:
  `scanf(`*stringa di formato*`,` *lista di variabili*`)`
  - La *lista di variabili* è un elenco di variabili a cui assegnare i dati letti dalla tastiera, queste:
    - sono separate da virgole
    - sono assegnate nell'ordine in cui sono elencate
    - precedute (ciascuna) dal carattere '`&`' (per ricavarne il *puntatore*)
  - La *stringa di formato* contiene un elenco di ***specifiche di conversione***, queste:
    - sono composte da una lettera preceduta dal carattere '`%`'
    - specificano il tipo di dato atteso per ciascuna delle variabili della *lista di variabili*

Input formattato – scanf

- Esempi
  - `scanf("%d", &a);`
    legge 1 valore di tipo `int` (come richiesto dalla specifica `%d`) e lo assegna alla var `a` (che deve essere stata definita di tipo `int`)
  - `scanf("%d%d", &a, &b);`
    legge 2 valori di tipo `int` (come richiesto dalle 2 specifiche `%d`) e li assegna ad `a` e `b` (di tipo `int`)
  - Quest'ultima equivale a:
    ```
    scanf("%d", &a);
    scanf("%d", &b);
    ```
  - Negli ultimi 2 esempi l'utente può inserire i 2 valori in input tutti insieme (separati da spazi) con un unico Invio finale o uno per volta con un Invio dopo ciascun valore

Input formattato – scanf

- Principali specifiche di conv. per val. numerici:

`%d`	→ `int`
`%u`	→ `unsigned int`
`%x`	→ `unsigned int` in esadecimale
`%hd`	→ `short int`
`%ld`	→ `long int`
`%lu`	→ `unsigned long int`
`%f`, `%e`, `%g`	→ `float` (equivalenti)
`%lf`	→ `double`
`%Lf`	→ `long double`

- `h`, `l` e `L` sono detti *modificatori di tipo*
- I valori con parte frazionaria si scrivono con il punto decimale e non con la virgola

Input formattato – scanf

- Tipicamente la *stringa di formato* contiene **solo** specifiche di conversione, senza altri caratteri in mezzo (neppure spazi): `"%d%d%d"`
- La *stringa di formato* può contenere anche caratteri ordinari, in questo caso la `scanf` si aspetta l'introduzione proprio di quei caratteri in quella esatta sequenza. I caratteri letti sono scartati ad uno ad uno finché corrispondono a quelli indicati nella stringa di formato, quando si verifica una non corrispondenza la `scanf` termina e rimette l'ultimo carattere a disposizione delle prossime letture

Input formattato – scanf

- Esempio
  Lettura di una data:
  `scanf("%d/%d/%d", &giorno, &mese, &anno);`
  legge date tipo “25/2/1997”
  ma in un input quale “25 febbraio 1997” fa leggere solo il 25 e al primo spazio si ferma, la successiva lettura riprende da questo spazio (incluso)
- Per indicare nella stringa di formato il carattere ‘%’ (non come specifica di conversione) è necessario raddoppiarlo: %%

Input formattato – scanf

- Nella *stringa di formato* i *white space* (tra cui: spazio, Tab e '`\n`') sono caratteri ordinari gestiti in modo diverso
- Un qualsiasi *white space* nella stringa di formato indica alla `scanf` che ci si aspetta di trovare zero o più *white space* (di qualunque tipo) consecutivi in quel punto dell'input (da scartare)
- Errore frequente è indicare il carattere '`\n`' nella stringa di formato pensando di passare così alla lettura della riga successiva

Input formattato – scanf

- La `scanf` legge caratteri dalla tastiera e in base alla specifica di conversione '`%`*x*' indicata nella *stringa di formato* li converte in un valore del tipo corrispondente
  - Ad esempio, con la specifica `%d` la `scanf` legge i caratteri 1 e 2 e li trasforma nel numero 12 (`int`)
- Il gruppo di caratteri letti di volta in volta dalla `scanf` per essere trasformati in un unico valore viene detto *campo* (*field*) o *token*
- La `scanf` salta i *white space* iniziali del campo che legge (è per questo che i valori possono essere immessi su più righe: i '`\n`' vengono saltati alla ricerca del primo dato)

Input formattato – scanf

- La quantità di caratteri utilizzati per costituire un *campo* dipende dalla specifica di conversione utilizzata:
  - viene prelevato il massimo numero di caratteri compatibili con il tipo richiesto dalla specifica
  - non appena si incontra un carattere non compatibile con la specifica si ferma
  - i caratteri rimanenti rimangono a disposizione per le successive letture ("restano nel buffer della tastiera", ossia in memoria)

    Se si inserisce dalla tastiera la sequenza di caratteri "12 23 34", una `scanf("%d", &a);` legge il 12 e lo mette in `a`, mentre " 23 34" resta in memoria e verrà utilizzato (ad es.) dalla successiva `scanf`

Input formattato – scanf

- Se non ci sono caratteri compatibili con il tipo richiesto dalla specifica, il campo è vuoto: la `scanf` termina e la variabile corrispondente (e le altre successive in quella `scanf`) non viene modificata, tutti i caratteri restano nel buffer di tastiera tranne eventuali spazi
  - Se si inserisce dalla tastiera la sequenza di caratteri "12 abc 3", una `scanf("%d%d", &a, &b, &c)` legge solo il 12 perché cerca un valore numerico per `b` e invece trova " abc 3": *consuma lo spazio davanti* ad 'a' e poi quando legge 'a' si accorge che non è una cifra e si ferma, il contenuto di `b` e quello di `c` non vengono modificati, la successiva lettura riprende da 'a' (lo spazio non c'è più)

Input formattato – scanf

- La funzione `scanf` restituisce un valore `int`:
  - se è ≥ 0 indica quanti valori sono stati letti e *assegnati* alle variabili della lista
  - oppure è il valore costante `EOF` che indica che si è verificato un errore o che non c'è nulla da leggere
- La `scanf` produce il valore `EOF` quando l'utente inserisce il *carattere di Fine Input* da tastiera su una riga vuota:
  - *Windows* → premere Control-Z e poi Invio
  - *Linux/Unix/OSX* → premere Control-D
- Si noti che il codice ASCII del carattere di Fine Input non è il valore della costante `EOF`

Input formattato – scanf

- Esempio

```
n = scanf("%d%d", &a, &b);
```

  - se `n` vale 2: la `scanf` ha letto 2 valori e li ha assegnati ad `a` e `b`
  - se `n` vale 1: la `scanf` ha letto un solo valore (assegnato ad `a`), mentre `b` non è stata modificata (ha il vecchio valore)
  - se `n` vale 0: nessun valore è stato letto, né `a` né `b` sono stati modificati, ma ci sono altri caratteri pronti per essere letti dalla prossima istruzione di input
  - se `n` vale `EOF`: nessun valore è stato letto, né `a` né `b` sono stati modificati e non ci sono altri caratteri pronti per essere letti dalla prossima istruzione di input

Input formattato – scanf

- Esempi
  Se in input (da tastiera) la funzione `n=scanf("%d", &a)` legge i caratteri:
  - "12" ➔ questi producono per `a` = 12, `n` = 1
  - "12 23" ➔ `a` = 12, `n` = 1, i caratteri " 23" restano a disposizione della successiva operazione di input
  - "12abc" ➔ `a` = 12, `n` = 1, i caratteri "abc" restano a disposizione della successiva operazione di input
  - "abc" ➔ non producono nulla per `a` (che mantiene il vecchio valore), `n` = 0, i caratteri "abc" restano a disposizione della successiva operazione di input
  - "abc12" ➔ non producono nulla per `a` (mantiene il vecchio valore), `n` = 0, i caratteri "abc12" restano a disposizione della successiva operazione di input

Input formattato – scanf

- Una specifica contenente un '`*`' tra il '`%`' e la lettera indica che il corrispondente campo deve essere letto, ma il valore non deve essere assegnato ad alcuna variabile (*soppressione dell'input*)
- Il valore restituito dalla `scanf` considera solo i valori letti *e assegnati*
- Esempio

```
n = scanf("%d%*d%d", &a, &b);
```

Se si dà in input: "12 23 34" si ottiene: `a`=12, `b`=34, `n`=**2**
mentre il valore 23 viene letto e scartato

Input formattato – scanf

- Si noti che se la `scanf` legge almeno un valore (anche se c'è soppressione dell'input) NON dà EOF
- Esempio

```
n = scanf("%*d");
```

  - Se si dà in input: "12" si ottiene:
    `n` = **0**
    mentre il valore 12 viene letto e scartato
  - Se si dà in input: Control-Z (+ tasto Invio) si ottiene:
    `n` = **–1**
    (supponendo che –1 sia il valore della costante EOF nel compilatore in uso)

Input formattato – scanf

- Per specificare la *quantità massima* di caratteri da leggere per costituire il campo corrispondente ad un dato (e per poi convertirlo in numero) tra il `%` e la lettera si indica un numero intero
- Esempio
  La specifica `%4d` indica che il campo da leggere per essere convertito in un valore di tipo `int` può essere costituito da *al massimo* 4 caratteri

Input formattato – scanf

- Se ce ne sono di meno, vengono usati solo i caratteri pertinenti il tipo:
  - `scanf("%4d%d", &a, &b);`
    se si dà in input "12 23" si ottiene `a`=12, `b`=23
    in quanto lo spazio delimita il primo campo prima del 4º carattere
- Se ce ne sono di più, vengono letti solo tanti caratteri quanti sono indicati nella specifica (gli altri restano per le letture successive) :
  - `scanf("%4d%d", &a, &b);`
    se si dà in input "123456"
    si ottiene `a`=1234, `b`=56

Output formattato – printf

- La `printf` è la più completa tra le funzioni di output su video (**print** **f**ormatted = stampa con formato)
- Sintassi:
  `printf(`*stringa di formato*`,` *lista di espressioni*`)`
- Visualizza il testo indicato nella *stringa di formato* e valori elencati nella *lista di espressioni*
- Ciascun valore della *lista di espressioni* viene visualizzato sostituendolo alla corrispondente direttiva di conversione nella *stringa di formato*

```
printf("Lat: %d, Lon: %d\n", a, b);
```

Output formattato – printf

- Più in dettaglio:
  `printf` (*stringa di formato*, *lista di espressioni*)
  - La *stringa di formato* contiene:
    - caratteri ordinari, vengono visualizzati tali e quali (per visualizzare il carattere %, lo si deve indicare raddoppiato: `%%`)
    - specifiche di conversione (es. `%d`) che indicano come visualizzare l'espressione corrispondente della *lista di espressioni*
  - La *lista di espressioni* è l'elenco delle espressioni di cui visualizzare il risultato:
    - ciascun risultato viene visualizzato al posto della corrispondente specifica di conversione (devono corrispondere per numero e tipo)
    - le espressioni sono separate da virgole

Output formattato – printf

- Specifiche di conversione per valori numerici

  `%d` → `int`
  `%u` → `unsigned int`
  `%x` → `unsigned int` in esadecimale
  `%hd` → `short int`
  `%lu` `%ld` → [`unsigned`] `long int`
  `%f` → `double` in formato decimale con 6 cifre dopo la virgola
  `%e` → `double` in formato esponenziale
  `%g` → `double` in formato decimale o esponenziale, rimuove zeri in coda
  `%Lf` → `long double` in formato decimale

- Per i `float` si usano le specifiche dei `double`

Output formattato – printf

- Restituisce un valore `int` che:
  - se ≥ 0: è il numero di caratteri scritti a video
  - se < 0: è una segnalazione di errore

```
n = printf("Somma = %d", a);
```

Con `a` pari a 5, `n` vale 9

- Le ***sequenze di escape*** permettono di inserire singoli caratteri speciali nella *stringa di formato*:
  - `\n` inserisce un ritorno a capo (si mette **in fondo** alla stringa di formato, di solito)
  - `\t` inserisce un carattere Tab
  - `\"` inserisce un carattere doppie virgolette '"'
  - `\'` inserisce un carattere apice '''
  - `\\` inserisce un carattere backslash '\'

Output formattato – printf

- Tra il carattere '`%`' e la specifica di conversione possono esserci, nell'ordine da sinistra:
  - Opzioni di *allineamento*
  - L'*ampiezza minima* del campo
  - Un punto se si vuole specificare la precisione
  - La *precisione*
  - Un *modificatore* di tipo (`h`,`l`,`L`)

Output formattato – printf

- L'*ampiezza minima* indica il numero di caratteri riservato (*campo*) per rappresentare il valore (compresi l'eventuale segno '–' e il punto decimale '.').
  Se il valore visualizzato richiede meno caratteri viene allineato a destra aggiungendo spazi a sinistra (␣), se ne richiede di più viene occupato lo spazio necessario senza considerare l'ampiezza minima
- `printf("*%4d*%4d*", 12, 123456);` visualizza "`*␣␣12*123456*`"

Output formattato – printf

- Le opzioni di *allineamento* permettono di:
  - '`+`' far precedere il numero dal suo segno
  - '`-`' allineare a sinistra il valore nel campo
  - '`0`' riempire il campo inutilizzato con 0 e non con spazi

  e possono essere indicati in qualsiasi ordine se ne vengono usati più di uno
- `printf("*%+d*%-4d*%04d*",12,12,12);` visualizza "`*+12*12␣␣*0012*`"

Output formattato – printf

- La *precisione* è un numero intero e indica:
  - **per i floating-point:** in `%f` è il numero di cifre da visualizzare *dopo il punto decimale* (di default: 6), ma attenzione che i valori hanno internamente il numero di cifre proprio del tipo a cui appartengono, ad es. i `float` sono in genere memorizzati con 7 cifre, i `double` con 14;
    in `%g` è il numero totale di cifre significative
  - **per gli interi:** è il numero minimo di caratteri da visualizzare (aggiungendo 0 in testa se necessario)
- Il *modificatore* di tipo serve per indicare tipi `short` (`h`), `long` (`l`), `long double` (`L`)

Output formattato – printf

- Esempi
  - `%8d` → `int` su *almeno* 8 caratteri
    `n=printf("X%8dX", 12);`
    visualizza "`X␣␣␣␣␣␣12X`", `n` vale 10
  - `%12f` → `float/double` su almeno 12 char
    `n=printf("X%12fX", 12.4);`
    visualizza "`X␣␣␣12.400000X`", `n` vale 14
  - `%.2f` → `float/double` con 2 caratteri decimali
    `n=printf("X%.2fX", 12.4);`
    visualizza "`X12.40X`", `n` vale 7
  - `%8.2f` → `float/double` su *almeno* 8 caratteri e con 2 caratteri dopo la virgola
    `n=printf("X%8.2fX", 12.4);`
    visualizza "`X␣␣␣12.40X`", `n` vale 10
    `n=printf("X%8.2fX", 121212.4);`
    visualizza "`X121212.40X`", `n` vale 11

Esercizi

1. Scrivere un programma che chieda 4 numeri `double` e ne calcoli la media (`double`) con 2 decimali.
2. Scrivere un programma che chieda un valore `double` e lo visualizzi con le 3 specifiche di conversione `%f`, `%e` e `%g`.