LA LIBRERIA STANDARD DEL C

- **La *libreria standard* del C è in realtà *un insieme di librerie***
- **Per usare una libreria, *non occorre inserirla esplicitamente nel progetto*: ogni ambiente di sviluppo *sa già* dove cercarle**
  - **Rhide: C:\DJGPP\LIB**
  - **Turbo C: vedi *Options/Project/Directories***
- **Ogni file sorgente che ne faccia uso deve però *includere lo header opportuno,* che contiene le *dichiarazioni* necessarie.**

LA LIBRERIA STANDARD DEL C

Le librerie standard

- **input/output** ***stdio.h***
- funzioni matematiche *math.h*
- gestione di stringhe *string.h*
- operazioni su caratteri *ctype.h*
- gestione dinamica della memoria *stdlib.h*
- ricerca ed ordinamento *stdlib.h*
- …
- ... e molte altre.

IL MODELLO DI INPUT/OUTPUT DEL C

- **Libreria standard `stdio`**
- **l'input avviene di norma dal canale standard di input (*stdin*)**
- **l'output avviene di norma sul canale standard di output (*stdout*)**
- **input e output avvengono sotto forma di una *sequenza di caratteri***
- **tale sequenza è terminata dallo *speciale carattere EOF* (*End-Of-File*), che varia da una piattaforma all'altra.**

I CANALI STANDARD

Di norma:

- **il canale standard di input, *stdin*, coincide con la tastiera**
- **il canale standard di output, *stdout*, coincide con il video**

Esiste inoltre un altro canale di output, riservato ai messaggi di errore: *stderr.*

- **anch'esso di norma coincide con il video**

IL MODELLO DI I/O DI BASE

- **Poiché sui canali di I/O fluiscono *sequenze di caratteri,* il modello di I/O prevede *due operazioni base:***
- ***scrivere un carattere sul canale di output***

```
putchar(ch);
```

- ***leggere un carattere dal canale di input***

```
ch = getchar();
```

- **Ogni altro tipo di I/O può essere costruito a partire da queste *operazioni primitive*.**

I/O A CARATTERI

- `int putchar(int ch);`
  - scrive un carattere sul canale di output
  - restituisce il carattere scritto, o EOF in caso di errore.
- `int getchar(void);`
  - legge un carattere dal canale di input
  - restituisce in carattere letto, o EOF in caso la sequenza di input sia finita, o in caso di errore.

Entrambe le funzioni leggono/scrivono un carattere convertito in int

ESEMPIO

Ricopiare l'input standard sull'output standard, carattere per carattere.

```
#include <stdio.h>

main(){
 int c;
 while(( c=getchar() ) != EOF)
  putchar(c);
}
```

Attenzione: getchar() inizia a produrre caratteri solo *dopo* aver premuto INVIO.

Per chiudere l'input producendo un EOF da tastiera, CTRL+Z in sistemi Windows, CTRL+D in Unix.

I/O DI TIPI PRIMITIVI

Ogni altro tipo di I/O può essere costruito sulle due primitive putchar() e getchar().

Esempi

- **scrivere o leggere *stringhe di caratteri***
- **scrivere o leggere *la rappresentazione di un numero* (naturale, intero, reale) sotto forma di stringa, in una base data.**

Queste funzionalità sono fornite già pronte nella libreria di I/O standard.

I/O FORMATTATO

La libreria standard offre due funzioni di I/O *di uso generale*, che compendiano tutte le necessità precedenti: printf() e scanf()

- **`int printf(...);`**
  - scrive sul canale di output una serie di valori, *effettuando le conversioni richieste ove necessario*
  - restituisce il numero di caratteri emessi
- **`int scanf(...);`**
  - legge dal canale di input una serie di *campi*, *effettuando le conversioni richieste ove necessario*
  - restituisce il numero di *campi* letti.

I/O FORMATTATO

Le funzioni `printf()` e `scanf()`:

- **possono avere *un numero variabile di parametri***
- **possono scrivere/leggere**
  - **singoli caratteri**
  - **stringhe di caratteri *formattate nel modo indicato dall'utente***
  - **interi, con o senza segno, in base 8, 10, 16**
  - **reali (float o double) in vari formati**

OUTPUT FORMATTATO: `printf()`

Sintassi:

```
int printf(char frm[], e1,…, eN)
```

- **la funzione scrive sul canale di output *i risultati delle espressioni* `e1`, *…,* `eN` *nel formato specificato dalla stringa* `frm[]`**
- **restituisce il numero di caratteri scritti, o EOF in caso di errore.**

OUTPUT FORMATTATO: `printf()`

La stringa di formato `frm[]`

```
int printf(char frm[], e1,…, eN)
```

è una stringa che può contenere *specifiche di formato*, del tipo `%`*carattere*

Formati per caratteri e stringhe:

tipi carattere e stringhe	*(formato unico)*
carattere singolo	`%c`
stringa di caratteri	`%s`

OUTPUT FORMATTATO: `printf()`

Formati per numeri:

tipi interi	*normale*	*short*	*long*
(signed) int	`%d`	`%hd`	`%ld`
unsigned int (decimale)	`%u`	`%hu`	`%lu`
unsigned int (ottale)	`%o`	`%ho`	`%lo`
unsigned int (esadecimale)	`%x`	`%hx`	`%lx`

tipi reali	*fixed-pt.*	*esp.*	*varia*
float	`%f`	`%e`	`%g`
double	`%lf`	`%le`	`%lg`
long double	`%Lf`	`%Le`	`%Lg`

ESEMPIO 1

```
#include <stdio.h>

main() {
   float z = 3.1415;
   int ret = 5;
   char msg[50] = "Finalmente si stampa!";
   printf("Valori: ret=%d, z=%f, msg=%s\n",
          ret, z, msg);
}
```

int *(decimale)* **float** **stringa**

ESEMPIO 2

```
#include <stdio.h>

main() {
   int a;
   printf("Immettere un carattere: ");
   a = getchar();
   printf("\n%c rappresenta %d come intero"
      "decimale, %o in ottale e %x in hex",
       a, a, a, a);
}
```

La stringa di formato può essere per comodità *spezzata in più stringhe*, che vengono concatenate automaticamente.

char **int** *(ottale)* **int** *(decimale)* **int** *(esadecimale)*

INPUT FORMATTATO: `scanf()`

Sintassi:

```
int scanf(char frm[], add1,…, addN)
```

- **la funzione legge dal canale di input *tanti campi quanti ne specifica la stringa di formato* `frm[]`, *e li pone in memoria agli indirizzi denotati da* `add1`, *…*, `addN`**
- **restituisce il numero di *campi* letti (0 se non ha letto nulla), o EOF in caso di errore.**

INPUT FORMATTATO: `scanf()`

La stringa di formato `frm[]`

```
int scanf(char frm[], add1,…, addN)
```

specifica <u>esattamente</u> ciò che ci si aspetta in input*, tramite specifiche *%carattere

Formati per caratteri e stringhe:

tipi carattere e stringhe	*(formato unico)*
carattere singolo	`%c`
stringa di caratteri	`%s`

NB: scanf considera finita la stringa *al primo spazio o separatore* che incontra. Quindi *non si può usare scanf* per leggere una stringa contenente spazi.

INPUT FORMATTATO: `scanf()`

Formati per numeri:

tipi interi	*normale*	*short*	*long*
(signed) int	`%d`	`%hd`	`%ld`
unsigned int (decimale)	`%u`	`%hu`	`%lu`
unsigned int (ottale)	`%o`	`%ho`	`%lo`
unsigned int (esadecimale)	`%x`	`%hx`	`%lx`

tipi reali	*fixed-pt.*	*esp.*	*varia*
float	`%f`	`%e`	`%g`
double	`%lf`	`%le`	`%lg`
long double	`%Lf`	`%Le`	`%Lg`

ESEMPIO 3

```
#include <stdio.h>

main() {
   float x; int ret, i; char name[50];
   printf("Inserisci un numero decimale, ");
   printf("un float ed una stringa con meno");
   printf("di 50 caratteri e senza spazi: ");
   ret = scanf("%d%f%s", &i, &x, name);
   printf("%d valori letti: %d, %f, %s",
                            ret, i, x, name);
}
```

indirizzo di una variabile ***int***

indirizzo di una variabile ***float***

nome di una stringa *(è già un indirizzo)*

ESEMPIO 3 (variante)

```
#include <stdio.h>

main() {
   float x; int ret,
   int *pi = &i;
   float *px = &x;
   printf("Inserisci un nu    o decimale, ");
   printf("un float ed una    ringa con meno");
   printf("di 50 caratteri    senza spazi: ");
   ret = scanf("%d%f%s", pi, px, name);
   printf("%d valori letti: %d, %f, %s",
                            ret, i, x, name);
}
```

Gli indirizzi possono essere passati a scanf() anche attraverso *idonei puntatori.*

`scanf()`: PRECISAZIONI

- **In scanf(), la stringa di formato `frm[]` è tipicamente una sequenza di specifiche *%carattere* *senza spazi o altri caratteri intermedi*. Ad esempio:**

```
scanf("%d%d%f", ...)
```

- ***non deve contenere messaggi!***

```
scanf("inserire un int: d%", …)
```

Infatti, la stringa di formato descrive *esattamente* quello che ci deve essere *in input*, ***non ha nulla a che fare con i messaggi che si vogliono in output!!***

`scanf()`: PRECISAZIONI

- **Inserire spazi o altri caratteri nella stringa di formato di scanf() è quindi possibile, ma ha il significato di *richiedere che tali caratteri siano obbligatoriamente presenti in input:* se mancano, scanf() dà errore.**
- **Ad esempio:**

`scanf("%c %c %c", &ch1, &ch2, &ch3);`	***sì, ma…***
`scanf("%c%c%c", &ch1, &ch2, &ch3);`	**SI**

Nella prima forma, i caratteri *devono* essere separati da uno spazio, altrimenti...

`scanf()`: PRECISAZIONI

- **Questa caratteristica può essere sfruttata per leggere dati formattati in modo particolare,** come ad esempio una data (gg/mm/aa)

```
scanf("%d/%d/%d", &g, &m, &a);
```

- **In questo modo, scanf() *filtra automaticamente i dati, eliminando le barre* e "catturando al volo" gli interi che interessano.**
  Se anche solo una barra manca → errore
- ***Dulcis in fundo, scanf() elimina automaticamente gli spazi di separazione fra i campi.***

ESEMPIO 4

Leggere (e poi riscrivere) nome, cognome, e data di nascita di una persona.

```
#include <stdio.h>
main() {
  struct { char cognome[20], nome[20];
           int g, m, a;
  } p;
  printf("Cognome, nome e data di nascita: ");
  scanf("%s%s%d/%d/%d", p.cognome, p.nome,
                &p.g, &p.m, &p.a);
  printf("                è nato il %d/%d/%d\n",
     p.cog                nome, p.g, p.m, p.a);
}
```

Gli spazi di separazione sono eliminati automaticamente