

LIBRERIE STANDARD in C

- La **libreria standard** del C è in realtà **un insieme di librerie**
- Per usare una libreria, **non occorre inse-
-rirla esplicitamente nel progetto**: ogni
ambiente di sviluppo sa già dove cercarle
- Ogni file sorgente che ne faccia uso deve
però **includere header opportuno** che
contiene le **dichiarazioni** necessarie

1

LIBRERIE STANDARD in C

Le librerie standard

- **input/output** **stdio.h**
- funzioni matematiche **math.h**
- gestione di stringhe **string.h**
- operazioni su caratteri **cctype.h**
- gestione dinamica della memori
a **stdlib.h**
- ricerca e ordinamento **stdlib.h**
- ... e molte altre

2

IL MODELLO DI INPUT/OUTPUT

- Libreria standard **stdio**
- **Input** avviene di norma dal **canale standard
di input (stdin)**
- **Output** avviene di norma sul **canale
standard di output (stdout)**
- Input e output avvengono sotto forma di una
sequenza di caratteri
- tale sequenza di caratteri è terminata dallo
speciale carattere EOF (End-Of-File), la cui
rappresentazione può variare da un SO ad un altro
(spesso **EOF=-1**; ANSI C prescrive un qualunque intero
negativo; definizione di **EOF** in `<stdio.h>`)

3

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**

4

MODELLO di BASE per I/O

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**

5

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 il carattere letto, oppure EOF in caso la sequenza di input sia finita o in caso di errore

Entrambe le funzioni leggono/scrivono un carattere convertito in int

6

ESEMPIO

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

```
#include <stdio.h>
int 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 Win, CTRL+D in Unix

7

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 già disponibili nella libreria di I/O standard

8

I/O con FORMATO

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 con successo**

9

I/O con FORMATO

Le funzioni **printf()** e **scanf()** possono avere **un numero variabile di parametri**. Inoltre, 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

10

OUTPUT con FORMATO: 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

11

OUTPUT con FORMATO: 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

Formati per numeri:

tipi reali	<i>n</i>	<i>x</i>	<i>espt.</i>	<i>varia</i>
float	%f	%e	%g	
double	%lf	%le	%lg	
long double	%Lf	%Le	%Lg	

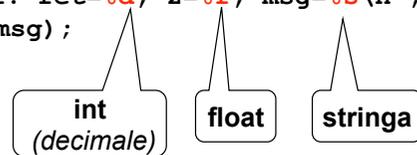
tipi interi	<i>normale</i>	<i>short</i>	<i>long</i>
(signed) int	%d	%hd	%ld
unsigned int (decimale)	%u	%hu	%lu
unsigned int (ottale)	%o	%ho	%lo
unsigned int (esadecimale)	%x	%hx	%lx

12

ESEMPIO 1

```
#include <stdio.h>

int 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);
}
```



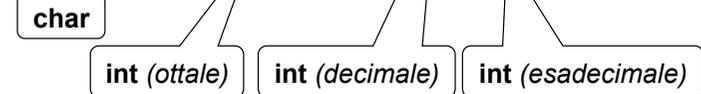
13

ESEMPIO 2

```
#include <stdio.h>

int 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);
}
```

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



14

INPUT con FORMATO: 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

15

INPUT con FORMATO: scanf()

La stringa di formato *frm[]*

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

specifica esattamente 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

16

INPUT con FORMATO: 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	fi xed-pt. esp.	varia
float	%f	%e %g
double	%lf	%le %lg
long double	%Lf	%Le %Lg

17

ESEMPIO 3

```
#include <stdio.h>

int main() {
    float x; int ret, i; char name[50];
    printf("Inserisci un numero decimale, ");
    printf("un float e 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 (piccola variante)

```
#include <stdio.h>

int main() {
    float x; int ret,
    int *pi = &i;
    float *px = &x;

    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", 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*

19

scanf(): PRECISAZIONI

La stringa di formato *frm[]* è 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 di output**

20

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:

<code>scanf("%c %c %c", &ch1, &ch2, &ch3);</code>	si, ma...
<code>scanf("%c%c%c", &ch1, &ch2, &ch3);</code>	SI

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

21

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*

Che cosa succede se manca anche solo una barra?

- **scanf() elimina automaticamente gli spazi di separazione fra i campi**

Piccolo esercizio: come usare scanf() per leggere un char dopo aver letto un altro dato in precedenza?

22

ESEMPIO 4

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

```
#include <stdio.h>
int 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.cognome, p.nome, p.g, p.m, p.a);
}
```

Gli spazi di separazione sono eliminati automaticamente