Richiami su `printf`

utilizza una stringa di formato per definire la stuttura del messaggio da produrre – anche in questo caso secondo uno schema bufferizzato!!!

la stringa di formato può contenere argomenti

per ogni argomento, un ulteriore parametro è richiesto in input da parte di `printf()`

questo parametro definisce l'espressione il cui valore dovrà comparire al posto del corrispettivo argomento nel messaggio da produrre

È una funzione di output e anche qui abbiamo una *stringa di formato* come primo parametro, per andare ad indicare qual'è il formato/la tipologia del messaggio che deve essere prodotto in output, e la stringa di formato può contenere argomenti e per ogni argomento dobbiamo passare un ulteriore parametro che è il valore dell'argomento.

Questo ulteriore parametro che noi passiamo ogni volta, va a definire cosa dovrà comparire nel messaggio di output al posto dell'argomento che è presente nella stringa di formato.

Il valore di ritorno di printf è il numero di byte che effettivamente costituiscono il messaggio prodotto.

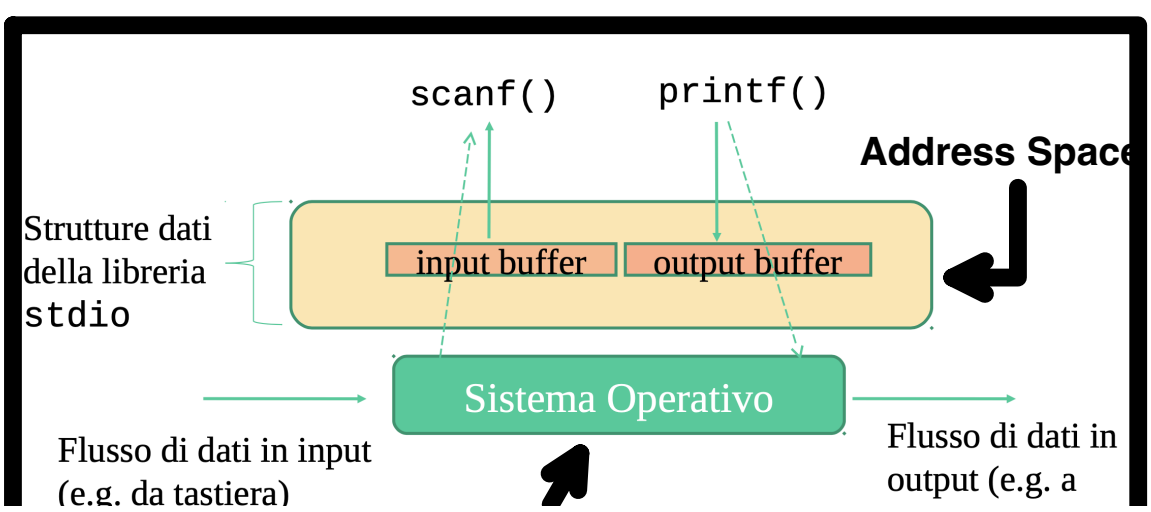
NOTA: il valore di ritorno di `printf()` corrisponde al numero di byte effettivamente costituenti il messaggio prodotto

Quindi quando diciamo alla printf di produrre un messaggio secondo un certo formato considerando certi argomenti, a seconda degli argomenti, avremo un numero di byte che andrà a comporre globalmente il messaggio.

Printf ci dice quanti byte sono stati effettivamente utilizzati.

C standard codifica il tipo `char` in ASCII (American Standard Code for Information Interchange) – 1 byte per carattere

Anche printf utilizza un approccio bufferizzato.



terminale)

Software del S.O

Il software del sistema operativo quali operazioni fa quando noi chiamiamo scanf()?

Cerchiamo di farci consegnare dei dati che arrivano da una sorgente, per esempio tastiera. È il software del sistema operativo che parla direttamente con i dispositivi. Quindi scanf chiamerà il sistema operativo e cercherà di capire se sono presenti dei dati che possono essere *consegnati* su questa chiamata.

Il sistema operativo prende il controllo e consegna questi dati in un'area di memoria che scanf ha consegnato come parametro al *sistema operativo*.

Così ora noi ci ritroveremo i dati nel cosiddetto "Input Buffer".

Ora scanf li analizza, analizza questa sequenza di caratteri che provengono dalla testiera per verificare la compliance rispetto al formato che avevo richiesto.

E dopodiché viene effettuata la consegna.

Ad esempio scrivo "87 90 54". Se nella scanf viene chiamato l'accesso ad un solo intero, allora 90 e 54 verranno mantenuti nel buffer, e il buffer viene riusata nel momento in cui noi consegniamo i dati quando scanf viene chiamata.

Quando noi chiamiamo printf e passiamo il puntatore alla stringa di formato e poi i vari argomenti, printf accede alla memoria, accedere ai parametri che vengono passati, definire la struttura del messaggio che deve essere prodotto e scriverlo nel cosiddetto "output buffer".

Soltanto in alcune circostanze i dati che vengono prodotti nell'output buffer vengono essere passati anche al sistema operativo, ad esempio per andare su terminale.

Soltanto in specifici eventi printf prende il messaggio che è scritto nell'output buffer e lo passa al sistema operativo.

Se scrivo printf("riesser \n"), queste informazioni vengono scritte nel buffer di output, però viene chiamato il S.O per fargliele prendere, e viene comunicato al modulo del S.O che sta partendo tramite la System call, qual'è l'area di memoria in cui c'è la sequenza di byte che devono essere mostrati sul terminale, questo perché è una linea completa.

La funzione che ci permette di lavorare con questi buffer di input/output si chiama fflush().

"Svuotare" il buffer di input/output

in linea di principio potrebbe essere utilizzato il servizio `fflush()` offerto dalla libreria `stdio`

però in C questo servizio ha un comportamento definito solo per quel che riguarda lo svuotamento del buffer di output (denominato `stdout`)

Su fflush() noi possiamo passare un parametro che sta ad indicare qual'è il buffer su cui vogliamo lavorare, e abbiamo la possibilità di identificare con **stdout** il buffer di uscita della libreria standard.

La funzione fflush() va a chiamare il S.O indicandogli che, tutto ciò che non è stato emesso verso il dispositivo di output, deve ORA essere emesso.
Quindi abbiamo la sicurezza di immettere in output informazioni anche se queste erano lasciate pendenti da printf.

- per il buffer di input (denominato `stdin`) conviene ridefinire il comportamento di tale servizio in modo esplicito e funzionale al proprio scopo
- a tal fine si può utilizzare la direttiva di precompilazione `#define`
- Un esempio per l'eliminazione di un'intera linea di input dal buffer

```
#define fflush(stdin) while(getchar() != '\n')
```

ESEMPIO

```c
#include <stdio.h>
#include <stdlib.h>

#ifdef FLUSH
#define fflush(stdin) while(getchar() != '\n')
#endif


int main(int a, char **b){

    int x;

    printf("%d\n",scanf("%d",&x));
    fflush(stdin);
    printf("%d\n",scanf("%d",&x));
    exit(0);

}
```

Vogliamo emettere in output un intero che è il valore di ritorno di scanf().

A scanf diciamo di acquisire un intero e di caricarlo all'interno dell'area di memoria dov'è posizionato x (&x).

```
ENT/SOFTWARE-EXAMPLES/C-BASICS>  /a.out
```

```
rtyiw
0
0
```

**Questi dati li mantiene nel buffer di input e nella prossima scanf è ancora nel buffer
di input, e quindi ottengo due volte zero.**

ORA COMPILIAMOLA DEFINENDO FLUSH

```
ENT/SOFTWARE-EXAMPLES/C-BASICS> gcc fflush-example.c -DFLUSH
```

Questa volta immettiamo una sequenza di caratteri e la prima scanf fallisce, i dati erano all'interno dell'input buffer, e a questo punto io ho consumato tutta la linea e non c'è più nulla nell'input buffer, e appena metto un numero in maniera adeguata la scanf ritornerà 1.

```
ENT/SOFTWARE-EXAMPLES/C-BASICS> ./a.out
egewuyfr
0
78
1
```

ESEMPIO

```c
#include <stdio.h>

char* text = "francesco";

int main(int a, char** b){

        printf("\n%d\n",printf("%s",text));
//      printf("%s",text);
}
```

Output:
francesco
9

Tutti i byte precedenti al terminatore.

Li emette nel buffer di stdout ed eventualmente dovremmo flushare queste informazioni verso il S.O