Introduzione5 Richiami su scanf e printf

sabato 4 ottobre 2025 18:07

È funzione di libreria standard che vuole una stringa di formato che identifica un numero generico di informazioni da consegnare al chiamante e la relativa tipologia. Per ogni informazione da consegnare specificata come parametro della stringa di formato, scanf() **vuole il relativo indirizzo di memoria dove effettuare la consegna**. La specifica dell'informazione da determina il numero di byte che verranno consegnati al rispettivo indirizzo, il quale può essere anche non noto. Vediamo esempio:

```
int x;
void get_input_value(void){
        Scanf("%d",&x);
}
```

Vediamo un esempio :

```c
#include <stdio.h>

char format_line[4096];

//x=2^20
unsigned long x = 2<<20;

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

        while(1){
                scanf("%s",format_line);
                printf(format_line,x);
                printf("\n");
                //cosi stampa il valore di x
                //se %d ritorna x
                //se %p indirizzo in memoria del puntatore

        }
}
```

Dove il risultato è il seguente : con il modificatore per l'intero del valore di x, mentre il secondo è il formato pointer (indirizzo nella memoria).

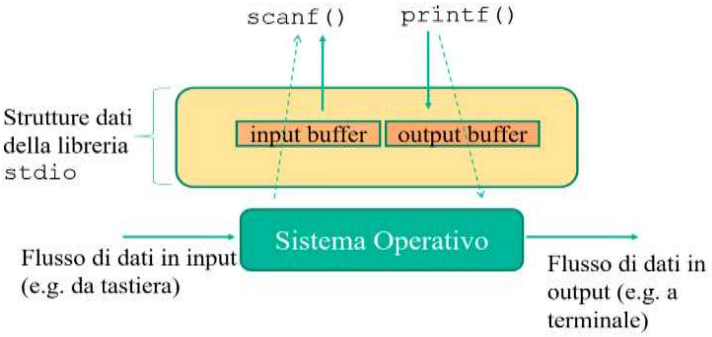
```
%d
2097152
%p
0x200000
```

Vediamo ora una variante : la **printf parametrica**:

```c
#include <stdio.h>

//text è array di 3 elementi
//gli elementi sono dei puntatori a char  ( char *)
char* text[3] = {"ennio - matricola %d\n",
                 "luigi - matricola %d\n",
                 "mario - matricola %d\n"};


int matricola[3] = {0, 1, 2};

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

        int i;

        for(i=0; i < 3 ;i++)
                printf(text[i],matricola[i]);
}
```

```
ennio - matricola 0
luigi - matricola 1
mario - matricola 2
```

Quindi andiamo vedere altri aspetti di scanf: **ritorna un codice numerico** ovvero il numero delle informazioni consegnate in base all'ordine richiesto. Attenzione però se ci sono informazioni incompatibili? Rimangono "appesi" in chiamate successive : quindi le info di scanf vengono gestite in modo **bufferizzato** .
Dualmente andiamo a vedere il printf : funzione di output, formata da un pointer come primo parametro (stringa di formato) e come successivi parametri si hanno gli argomenti. **Ritorna un valore che rappresenta il numero di byte effettivamente costituenti il messaggio**. Vediamo ora lo schema di bufferizzazione :



Il quale è diviso in due address space e sistema operativo. Vediamo il caso dell'input : scanf -> scanf chiama SO e cerca di capire se ci sono dati presenti , poi passa il controllo al SO che li passa all'area memoria nella quale vengono mandati questi dati, prima che il chiamante li ottenga , poi scanf accede a questa area e se i dati sono compliant , li consegna al chiamante. Analogamente per printf : i messaggi vengono scritti nel buffer, i quali vengono poi passati al SO , che poi magari li manda in output sul terminale. Il carattere terminatore è "\n" . Tornando allo "svuotamento" del buffer , si ha una funzione dedicata : **fflush()** , che prende in input il buffer su cui vogliamo lavorare. **In generale per input buffer si intendono stdout (buffer per output) e stdin (buffer per input). Occhio che fflush non si può usare su stdin (ritorna il controllo al chiamante)**. Invece se vogliamo eliminare una riga dall'input buffer possiamo usare questa macro :

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

Vediamo degli esempi :

```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);

}
```

```
fgvbhj
0
0
```

Da notare che qui c'è un problema : scanf vuole degli interi ma io passo una stringa , quindi scanf non consegna i dati. Ricordandoci che scanf ritorna il numero dei parametri matchati, in questo caso in nessuna delle due chiamate riesce a soddisfare. Ultima nota : questa versione è stata compilata la libreria di sistema fflush : ignora completamente la definizione della macro.

Andiamo ora a compilare il sorgente aggiungendo la macro da noi definita:

```
gcc fflush-example.c -DFLUSH
```

```
vfyvgyvgvg
0
95
1
```

Dove nel primo caso usando la nostra macro elimino il primo valore inserito nello stdin, che essendo errore da 0 , mentre nel secondo tentativo va tutto ok in quanto scanf chiede un intero ed io consegno un intero