

Richiami su SCANF

Scarf è una funzione che vuole una stringa di formato come primo parametro, questa va ad identificare un numero di informazioni da consegnare al chiamante.

Non identifichiamo solo il numero, ma anche la tipologia delle informazioni

Ad esempio vogliamo che scarf ci consegna un intero, quindi venga scritta una locazione intera da qualche parte in memoria

Per ogni parametro che la stringa di scarf deve consegnare, va specificato il relativo indirizzo di memoria dove effettuare la consegna (&).

Quando noi chiediamo una consegna di un certo tipo, essa va a determinare il numero di byte che verranno consegnati a partire dall'indirizzo che noi abbiamo indicato a scarf.

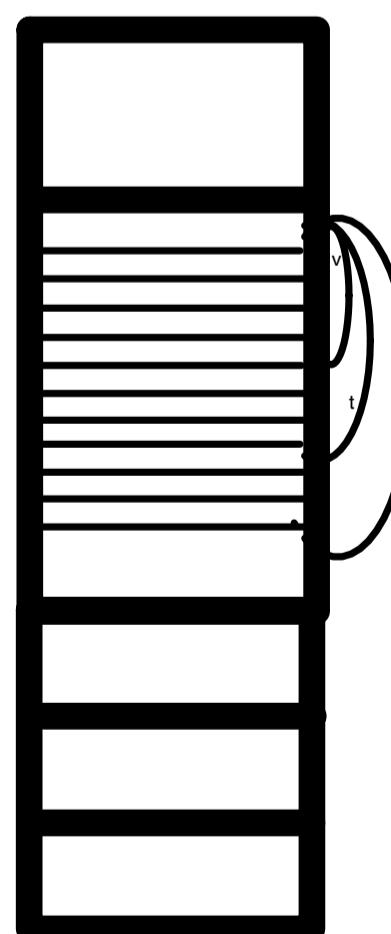
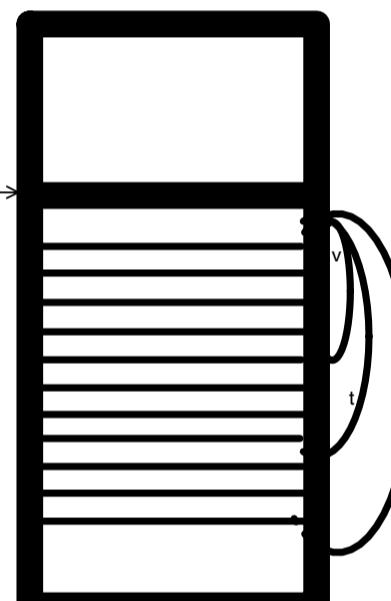
Il numero dei byte che scarf va a scrivere in memoria può NON essere noto: supponiamo che noi vogliamo ricevere una stringa di informazioni usando scarf, la stringa è una sequenza di caratteri chiusa da un terminatore '\0', la stringa potrebbe avere N,K o T caratteri.

Quando noi cerchiamo di usare scarf per far sì che il nostro programma, passando a scarf un certo indirizzo di memoria che corrisponde ad una specifica zona dell'address space, cerchi di ottenere una stringa in quella zona, noi non sappiamo se verranno utilizzati K,T o V byte per le operazioni che scarf va ad eseguire.

Se noi chiamiamo una funzione passando un indirizzo di memoria, ma non sappiamo come la funzione, a partire da questo indirizzo, quanti byte effettivamente andrà ad aggiornare, abbiamo il rischio di andare ad aggiornare una quantità di byte che vadano a sovrascrivere altre informazioni all'interno dell'address space.

```
int x;      X è la variabile dove vogliamo il dato
            e la sua posizione è &x.
            "%d" è una stringa di formato.

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



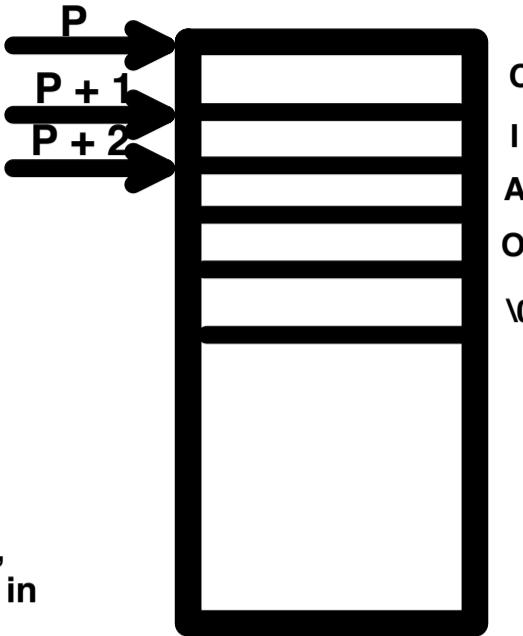
In realtà in C il tipo stringa NON esiste.

Però le stringhe sono un modo per rappresentare dei dati in memoria. Tipicamente per rappresentare sequenze di caratteri.

Per identificare una stringa in memoria devo utilizzare l'indirizzo del primo dei caratteri che rompono la stringa stessa, quindi uso dei "pointer".

Per mantenere questo indirizzo di memoria utilizzo un puntatore a char:
char *p;

Quando noi mettiamo appunto il puntatore, mettiamo l'indirizzo di dove è memorizzata in memoria questa informazione: quello che passiamo è l'indirizzo di memoria di dove è memorizzata quell'informazione.
La posizione viene scelta a livello del compilatore.
Può stare anche nella stack area di una funzione determinata. Dipende.



ESEMPIO

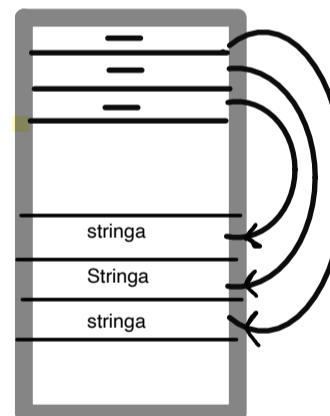
```
parametric-printf.c
1 #include <stdio.h>
2
3 char* text[3] = {"ennio - matricola %d\n",
4                 "luigi - matricola %d\n",
5                 "mario - matricola %d\n"};
6
7 int matricola[3] = {0, 1, 2};
8
9 int main(int a, char** b){
10
11     int i;
12
13     for(i=0; i < 3 ;i++)
14         printf(text[i],matricola[i]);
15 }
```

```
scan-print -- bash -- 80x24
(base) MacBook-Pro-di-Simone:scan-print simoneremoli$ ls
dynamic-printf.c      get-secret.c      printf-test.c
fflush-example.c      parametric-printf.c
(base) MacBook-Pro-di-Simone:scan-print simoneremoli$ pwd
/Users/simoneremoli/Downloads/C-BASICS/scan-print
(base) MacBook-Pro-di-Simone:scan-print simoneremoli$ gcc parametric-printf.c -o file
(base) MacBook-Pro-di-Simone:scan-print simoneremoli$ ./file
ennio - matricola 0
luigi - matricola 1
mario - matricola 2
(base) MacBook-Pro-di-Simone:scan-print simoneremoli$ |
```

Nel nostro address space da qualche parte in memoria, abbiamo text che è formato da 3 elementi, e che sono tutti puntatori a carattere.

Quindi puntiamo da qualche parte all'interno dell'address space.
In qualità di puntatori a carattere, puntiamo a stringhe di carattere.

Molto Semplice



TEXT

ESEMPIO

```
dynamic-printf.c
1 #include <stdio.h>
2
3 char format_line[4096];
4
5 unsigned long x = 2<<20;
6
7 int main(int a, char** b){
8
9     while(1){
10         scanf("%s",format_line);
11         printf(format_line,x);
12         printf("\n");
13     }
14 }
```

Vogliamo poter prendere %s, ossia una stringa. Questa stringa deve essere scaricata in memoria a partire dall'indirizzo "format_line". Format_line è una zona di memoria che funge da pointer parametro di scanf, in cui abbiamo 4096 caratteri di tipo char.

Nella printf andiamo a specificare una stringa di formato utilizzando un pointer: stiamo passando l'indirizzo di memoria di format_line, dove avevamo scaricato prima le informazioni con scanf.

Nella printf diciamo che il valore di x, ossia 2^{20} , deve essere messo in output con un formato che non è noto all'inizio, ma è un formato in funzione di quello che la scanf ha deciso di andare ad acquisire in termini di stringa all'interno dell'area di memoria.

```
scan-print -- bash -- 80x24
(base) MacBook-Pro-di-Simone:scan-print simoneremoli$ ls
dynamic-printf.c      get-secret.c      printf-test.c
fflush-example.c      parametric-printf.c
(base) MacBook-Pro-di-Simone:scan-print simoneremoli$ gcc dynamic-printf.c -o file
(base) MacBook-Pro-di-Simone:scan-print simoneremoli$ ./file
ciao
ciao
sono
sono
una
una
stampa
```



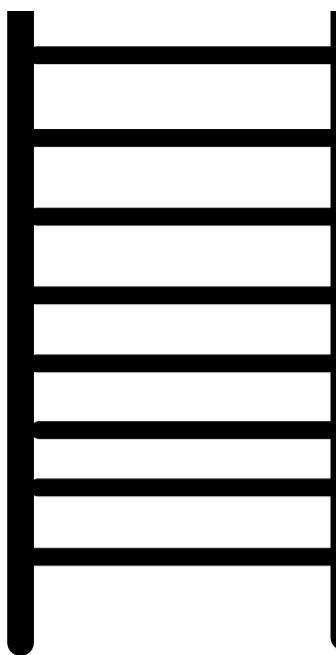
```
stampa
^Z
[1]+ Stopped . ./file1
(base) MacBook-Pro-di-Simone:scan-print simoneremolisi |
```

Noi abbiamo una `format_line` con un certo numero di byte, che corrisponde quindi ad un array.

Noi chiamiamo `scanf` per scaricare una stringa in questo array.

Il primo parametro che stiamo passando a queste `printf` e `scanf`, è un parametro di tipo pointer.

Questa è una linea di formato che contiene tutte le formattazioni sino a 4096 elementi



La stringa che scarichiamo la utilizziamo poi in `printf` e quando chiamiamo `printf`, non facciamo altro che passare l'indirizzo di memoria dove abbiamo scaricato la stringa che rappresenta il formato dei dati, che vogliamo emettere in output.

`Scanf` attende da noi l'acquisizione di una stringa.

Possiamo inserire: "%d". Quello che esce è la rappresentazione intera del valore di x, ossia 2097152.

Possiamo inserire: "%p". Quello che esce è un indirizzo.

```
OFTWARE-EXAMPLES/C-BASICS> ./a.out
%d
2097152
%p
0x200000
```

Scarf ritorna anche un valore, che è effettivamente il numero di informazioni consegnate.

In base all'ordine di consegna definito dalla stringa di formato.

Potrebbe anche non consegnarle, magari l'informazione da consegnare non è disponibile con il tipo di informazione che io vorrei ricevere in input.

La testiera immette una sequenza di caratteri che `scanf` valuta, e nel momento in cui le valuta, consegna le informazioni rappresentandole poi nella maniera adeguata.

Ad esempio se la richiesta è di un intero, e le informazioni che provengono dalla tastiera indicano un numero, ad esempio 19, in realtà questo numero viene tradotto nella sequenza di caratteri 1 e 9, e vorrei avere questo intero in una zona della memoria.

Scarf traduce la sequenza 1 e 9 nella relativa rappresentazione binaria del numero 19 che deve essere consegnato nella zona di memoria.

Ma ATTENZIONE:

Supponiamo che io avessi chiesto qualcosa che non era compatibile con la consegna di un intero, avessi chiesto un intero e la situazione fosse stata questa:

`scanf("%d")`
Ed inserisco da tastiera AX

Che fine fanno questi caratteri?



Questi caratteri rimangono li ancora per essere consegnati. Quando? In chiamate successive che l'applicazione potrebbe inoltrare a scanf.

Le informazioni vengono gestite da scanf secondo uno schema bufferizzato.

L'area di memoria mantiene i dati bufferizzati.

ovvero i dati che consegna sono preventivamente prelevati dalla sorgente tramite specifiche system call offerte dal sistema operativo, e bufferizzati in aree di memoria gestite dalla stessa libreria di I/O

dati bufferizzati e non ancora consegnati rimangono validi per prossime consegne

Scanf interagisce veramente col nostro sistema operativo e acquisisce dei dati che vengono bufferizzati dentro una certa area di memoria, quindi in questo momento scanf ritornerà il valore 0, perché non può ritornare nulla.

Alla prossima invocazione di scanf, sempre quei dati avremo da consegnare.