Approfondimento : printf

La funzione printf **:**

- stampa su standard input (video) dati **complessi**
- ha un formato articolato, molto potente ma spesso poco chiaro
- ha un numero di opzioni utili e poco conosciute

Printf: esempi d'uso

- Scrivere output ben spaziato “`|  1| 23|100|`”:

```
printf("|%3d|%3d|%3d|\n", a, b, c);
```

- Scrivere testo e numeri:

```
printf("%lf %c %lf\n", num1, simb, num2);
```

- Trasformare un decimale in esadecimale

```
printf("%x", numero);
```

- Stampare solo i primi 3 decimali di un float

```
printf("%.3f", numero);
```

Printf

```
int printf (const char *format, …);
```

- Il formato è una stringa costante (non può essere una variabile stringa) che può contenere :
  - whitespaces: spazi,tabulazioni, invii a capo (\n)
  - caratteri standard (qualunque sequenza di caratteri non bianchi che non inizi per %)
  - specificatori di formato (che iniziano per %)
- Per dettagli sui segnaposti, vedere

  http://it.wikipedia.org/wiki/Printf#Segnaposti_del_formato_printf

Approfondimento : scanf

La funzione scanf **:**

- legge da standard input (stdin, tastiera) dati **complessi**
- è quasi duale a printf (prende un formato e una lista di parametri) ma non del tutto
- è facilmente fonte di errori e incongruenze

Scanf: esempi d'uso

- Leggere una data (gg/mm/aaaa):

```
scanf("%2d/%2d/%4d", &giorno, &mese, &anno);
```

- Leggere un'operazione aritmetica (su una riga):

```
scanf("%lf %c %lf", &num1, &simb, &num2);
```

- Trasformare un esadecimale in decimale:

```
scanf("%x", &numero);
```

- Leggere un singolo carattere:

```
scanf(" %c", &carattere);
```

(notate lo spazio prima del %c, necessario perchè questo pattern non elimina eventuali invii a capo e spazi lasciati nel buffer da scanf precedenti).

Scanf: il formato

```
int scanf(const char *format, …);
```

- Il formato è una stringa costante (non può essere una variabile stringa) che può contenere :
  - whitespaces: spazi,tabulazioni, invii a capo (\n)
  - caratteri standard (qualunque sequenza di caratteri non bianchi che non inizi per %)
  - specificatori di formato (che iniziano per %)

Il formato

- Gli specificatori di formato sono così fatti :

```
%[*][width][modifiers]type
```

(le parentesi quadre indicano un elemento opzionale)

dove :

*****	Un asterisco fa si che i dati corrispondenti a questo argomento vengano letti dallo stdin ma ignorati (non salvati in un argomento)
width	Specifica il numero massimo di caratteri da leggere per questo argomento
modifiers	Modifica le dimensioni: **h** : short int (per d, i and n), o unsigned short int (per o, u and x) **l** : *(è una elle minuscola)* long int (per d, i and n), o unsigned long int (per o, u and x), o double (per e, f and g) **L** : long double (per e, f e g)
type	Un carattere che specifica il tipo di dato da leggere (vedi sotto)

Il funzionamento

La funzione :

- legge lo stdin e prova a riconoscere elementi così come indicati dal formato
- elimina automaticamente gli whitespace iniziali (quasi sempre)
- copia nelle variabili passate come argomento gli elementi riconosciuti con successo
- ritorna il numero di elementi riconosciuti e letti con successo (possono essere meno di quelli richiesti dal formato)

La logica

- L'input da tastiera è bufferizzato

  e resta nel buffer finchè qualcuno non lo rimuove.

- scanf cerca nel buffer i pattern (numeri decimali, floating poing, caratteri, ecc...) e “mangia" (cioè toglie dal buffer) quelli che riconosce

Il formato

- Gli specificatori di formato sono così fatti :

```
%[*][width][modifiers]type
```

(le parentesi quadre indicano un elemento opzionale)

dove :

*****	Un asterisco fa si che i dati corrispondenti a questo argomento vengano letti dallo stdin ma ignorati (non salvati in un argomento)
width	Specifica il numero massimo di caratteri da leggere per questo argomento
modifiers	Modifica le dimensioni: **h** : short int (per d, i and n), o unsigned short int (per o, u and x) **l** : *(è una elle minuscola)* long int (per d, i and n), o unsigned long int (per o, u and x), o double (per e, f and g) **L** : long double (per e, f e g)
type	Un carattere che specifica il tipo di dato da leggere (vedi sotto)

Specificatori di tipo (type)

type	descrizione	tipo argomento
c	Singolo carattere: legge il carattere successivo. Se è impostata una *width* differente da 1, legge *width* caratteri ma nessun terminatore di stringa viene aggiunto. **NON ELIMINA GLI SPAZI INIZIALI.**	**char ***
d	Intero decimale: un numero con un segno (+ o -) opzionale prima.	**int ***
e, E, f, F, g, G	Virgola mobile: numero decimale con un punto decimale, eventualmente preceduto da un segno (+ o -) ed eventualmente seguito da e (o E) ed un altro numero decimale. Per esempio: **-732.103** o **7.12e4**	**float ***
o	Intero ottale	**int ***
s	Stringa di caratteri: legge una stringa generica di caratteri fino al prossimo whitespace. E' **fortemente** consigliato di usarlo sempre con uno *width* prefissato.	**char ***
u	Intero decimale unsigned	**unsigned int ***
x, X	Intero esadecimale	**int ***

Esempi d'uso

- Leggere una data (gg/mm/aaaa):

```
scanf("%2d/%2d/%4d", &giorno, &mese, &anno);
```

- Leggere un'operazione aritmetica (su una riga):

```
scanf("%lf %c %lf", &num1, &simb, &num2);
```

- Trasformare un esadecimale in decimale:

```
scanf("%x", &numero);
```

- Leggere un singolo carattere:

```
scanf(" %c", &carattere);
```

(notate lo spazio prima del %c, necessario perchè questo pattern non elimina eventuali invii a capo e spazi lasciati nel buffer da scanf precedenti).

Particolari da tenere a mente

Ci sono alcuni punti che è bene tenere a mente e ricordare.

1. La funzione scanf ritorna il numero di elementi correttamente letti: se leggete più di un elemento per volta, dovreste controllare il valore ritornato per assicurare che tutte le variabili siano **inizializzate**.
2. Per leggere un singolo carattere **DOVETE** premettere il %c (nel formato) con uno spazio bianco: come spiegato prima, il %c non elimina gli spazi bianchi rimasti sul buffer ed in particolare ogni **invio a capo** residuo da scanf precedenti!