Appunti sulla funzione scanf

Violetta Lonati

Scanf è una funzione della libreria standard; per usarla è necessario includere `stdio.h`

Ecco alcuni esempi di programmi che presentano comportamenti apparentemente bizzarri, dovuti in realtà ad un cattivo utilizzo della funzione scanf.

Esempio 1 - il comportamento di questo programma potrebbe essere riassunto con la frase: "salta un giro"

```
#include <stdio.h>
int main( void ) {
    char risposta;
    for ( ; ; ) {
        printf( "Inserisci una lettera: " );
        scanf( "%c", &risposta );
        printf( "Hai inserito la lettera: %c\n", risposta );
    }
    return 0;
}
```

Esempio 2 - il comportamento di questo programma potrebbe essere riassunto con la frase: "resta in sospeso"

```
#include <stdio.h>
int main( void ) {
    char risposta;
    printf( "Inserisci una lettera: " );
    scanf( "%c ", &risposta );
    }
    return 0;
}
```

Esempio 3 - il comportamento di questo programma potrebbe essere riassunto con la frase: "aspetta il passaggio successivo"

```
#include <stdio.h>
int main( void ) {
    char risposta;
    for ( ; ; ) {
        printf( "Inserisci una lettera: " );
        scanf( "%c ", &risposta );
        printf( "Hai inserito la lettera: %c\n", risposta );
    }
    return 0;
}
```

Per comprendere correttamente il funzionamento della scanf è importante avere chiari alcuni elementi.

NB: nel seguito uso i simboli `]` *e* `°` *per indicare rispettivamente l'a-capo e lo spazio, quando compaiono all'interno del flusso di input.*

Flusso di input e output sono indipendenti

Poiché il flusso di input e quello di output appaiono entrambi sul monitor, abbiamo l'impressione che ci sia un'interazione tra i due; in realtà si tratta di due flussi totalmente indipendenti. Sul flusso di output troviamo tutti i caratteri prodotti dalle funzioni di standard output (tra cui anche i messaggi del tipo `"inserisci un numero"!`); sul flusso di input troviamo tutti i caratteri inseriti dall'utente tramite tastiera (o rediretti da file sullo standard input), *inclusi i caratteri di a-capo*.

Da cosa deriva l'effetto di interazione tra programma e utente?

Tutte le volte che una funzione di lettura dell'input deve essere eseguita, va considerato il flusso di input. Se questo è vuoto, il prompt viene passato all'utente, che può inserire caratteri. Il prompt resta all'utente fino a quanto inserisce un a-capo, a questo punto il controllo torna al programma in esecuzione, e l'esecuzione della funzione di lettura dell'input può proseguire. I caratteri non consumati restano sul flusso di input e saranno considerati dalle chiamate successive di lettura dell'input.

Quindi è sbagliato pensare che ad ogni chiamata di `scanf` o `getchar` corrisponda esattamente un'azione di inserimento di input da parte dell'utente. Possono infatti verificarsi queste situazioni:

- una chiamata di `scanf` richiede all'utente di fare più volte l'inserimento di caratteri

Esempio:

Se il flusso di input è vuoto, la chiamata `scanf( "%d %d", &a, &b )` passa il prompt all'utente. se l'utente scrive `17]`
(cioè: 17 seguito da a-capo), allora il prompt tornerà all'utente (l'effetto apparente è che il prompt resti all'utente anche dopo che ha digitato l'a-capo).*

- una chiamata di `scanf` non richiede nessun intervento dell'utente

Esempio:

Se il flusso di input è contiene la sequenza di caratteri `17°14°`, la chiamata `scanf( "%d %d", &a, &b )` consuma i caratteri `17°14` senza che il prompt passi all'utente (NB: lo spazio finale non viene consumato!). L'effetto apparente è che la `scanf` non venga eseguita perché l'utente non viene interpellato.*

Interpretazione della stringa di formato

La `scanf` cerca nel flusso di input, a partire dal primo carattere disponibile, una sequenza di caratteri (anzi, *la più lunga sequenza di caratteri*) corrispondente ad un certo formato. Il formato è specificato dalla stringa di formato, primo parametro della `scanf`.

La stringa di formato può contenere tre tipi diversi di cose: - specifiche di conversione (che iniziano con `%`) tipo quelle usate con la printf. Le specifiche di conversione non considerano (e quindi non consumano) gli spazi bianchi finali. Invece, quasi tutte le specifiche di conversione (ad esempio quella per le stringhe `%s%` o quelle numeriche come `%d`, `%f`, o la `%s`) ignorano (cioè consumano ma non considerano) gli spazi bianchi iniziali - "spazi bianchi" (//white-space character//), es: spazio, a-capo \n, tabulazione \t): uno spazio bianco corrisponde ad una sequenza (la più lunga possibile) di spazi bianchi. - altri caratteri non bianchi: un carattere che non è uno spazio bianco corrisponde esattamente a quel carattere nel flusso di input.

Esempio:

Se il flusso di input è `]°°]z]` (cioè: a-capo seguito da due spazi seguito da a-capo seguito da z seguito da a-capo e la stringa di formato è `" %c"` e (oppure, equivalentemente `"\n%c"`), allora i caratteri `]°°]` sono fatti corrispondere allo spazio bianco e il carattere z corrisponde a `%c`. L'a-capo finale *non* viene consumato.

In altre parole, la specifica `" %c"` può essere intesa come *"prendi il primo carattere che non è uno spazio bianco"*.

Esempio:

Se la stringa di formato è `"ID %d-%d-%d"` allora l'input m dovrà contenere, in questo ordine:

- le lettere `ID` (senza spazi in mezzo!)
- eventualmente degli spazi bianchi
- un numero (eventualmente preceduto da spazi bianchi)
- il carattere –
- un numero (eventualmente preceduto da spazi bianchi)
- il carattere –
- un numero (eventualmente preceduto da spazi bianchi)

La `scanf` consuma solo i caratteri che corrispondono alla stringa di formato.

La `scanf` consuma solo i caratteri che riesce effettivamente a far corrispondere alla stringa di formato; eventuali caratteri in più verranno considerati dalle successive chiamate di funzioni di lettura dell'input.

Quando la `scanf` trova un carattere che non corrisponde alla stringa di formato, si interrompe

senza consumarlo. Le parti già consumate perché conformi alle parti precedenti della stringa di formato risultano correttamente assegnate, ma potrebbero rimanere delle variabili non assegnate.

Esempio:

Se il flusso di input è `10-12` e la chiamata è `scanf( "%d:%d", &a, &b )` con `a` e `b` variabili intere,

allora i caratteri 10 sono fatti corrispondere al primo `%d`, ma il carattere – non corrisponde al carattere atteso `:` e la scanf si interrompe. La variabile `a` avrà valore 10, la variabile `b` non viene assegnata.