Appendice al Cap. 4

Gestione dell'input

Molto spesso accade che il valore di una variabile sia assegnato attraverso un input da tastiera per mezzo della funzione `scanf`.

Un problema molto comune consiste nel valutare se i valori immessi attraverso la tastiera da parte dell'utente sono corretti. Ad esempio, se si richiede un numero attraverso un'istruzione del tipo

```
scanf("%d", a);
```

si vuole controllare che l'utente non immetta, per errore, uno o piú caratteri non ammessi (un carattere alfabetico, una virgola, etc.). Allo scopo si può sfruttare il fatto che la funzione `scanf` restituisce un numero intero pari al numero di conversioni eseguite con successo. Nel caso in esame, immettendo un numero intero attraverso la tastiera, la funzione restituisce 1 perché esegue la conversione di una sequenza di caratteri in un `int` una volta. Nel caso invece in cui l'utente dovesse immettere un carattere (ad esempio `f`), la funzione restituisce 0, non riconoscendo il carattere `f` come un carattere valido per convertire la sequenza in input in un `int`.

La porzione di codice che verifica la correttezza dell'input sarebbe dunque

```
int c;
int a;
...
printf("Inserisci un numero intero: ");
do {
  c = scanf("%d", &a);
  if (c == 0) {
    printf("Attenzione: input non valido.\n");
  }
} while (c == 0);
```

Sfortunatamente, su molti sistemi, se si esegue questa porzione di codice e si immette un carattere al posto di un numero, il programma entra in un ciclo infinito nel quale, apparentemente, la funzione `scanf` non è piú invocata, ma compare continuamente la riga che segnala l'errore.

Il motivo di tale comportamento apparentemente incomprensibile consiste nel fatto che la funzione `scanf` usa una porzione della memoria (*input buffer*) nella quale sono copiati i bit corrispondenti ai codici ASCII dei caratteri inseriti da tastiera. Nel caso in cui i caratteri digitati siano caratteri validi ai fini della conversione (i segni `+` e `-`, e tutte le cifre da `0` a `9`) la funzione `scanf` elimina un carattere per volta dall'*input buffer* e converte la sequenza di caratteri in una sequenza di bit nella rappresentazione opportuna (in questo caso nella notazione in complemento a 2).

Qualora la conversione non riesca (perché ad esempio è stato inserito il carattere `f`), l'*input buffer* non è svuotato e la variabile `c` assume valore 0. In questo caso la funzione `scanf` viene ripetuta, ma trovando nel *buffer* di input ancora dei caratteri, non attende che questi vengano nuovamente inseriti dalla tastiera e tenta di convertirli, ovviamente non riuscendovi, innescando il ciclo infinito.

Per ovviare a questo inconveniente è necessario svuotare il *buffer* prima di un'eventuale rilettura. Per questo si può modificare il codice nel modo seguente :

```
int c;
int a;
...
printf("Inserisci un numero intero: ");
do {
  c = scanf("%d", &a);
  if (c == 0) {
    scanf("%*[^\n]");
    printf("Attenzione: input non valido.\n");
  }
} while (c == 0);
```

L'aggiunta della riga `scanf("%*[^\n]");` produce lo svuotamento del *buffer* perché, quando viene eseguita, legge qualunque carattere (indicato nel descrittore come `*`) escluso il carattere di "a capo" `\n`. I caratteri tra parentesi quadre infatti indicano i possibili caratteri da riconoscere, mentre il carattere `^` funge da operatore di negazione e definisce la lista di caratteri tra parentesi quadre come quelli da non riconoscere.

Un esempio chiarirà meglio il significato dell'espressione in parentesi quadre. La riga `c = scanf("%[abcd]", &ch);` con `ch` definito come variabile carattere accetta in input i caratteri `a`, `b`, `c` o `d`. Se si inserisce il carattere `f` la variabile `c` assume il valore 0 e alla variabile `ch` non è assegnato alcun valore. Viceversa, la riga `c = scanf("%[^abcd]", &ch);` permette di far assumere alla variabile `ch` qualunque carattere purché diverso dai quattro elencati in parentesi quadre.

Quando si esegue la funzione `scanf` nella porzione di codice sopra riportata, questa estrae tutti i bit dal *buffer* fino al carattere di "a capo" (l'ultimo carattere digitato dall'utente), cosí che il sistema è di nuovo pronto a riceverne di nuovi. Non essendovi variabili indicate come secondo argomento di `scanf` l'eventuale risultato della conversione è semplicemente ignorato.

Una versione alternativa consiste nell'acquisire, attraverso la funzione `scanf`, una

stringa, da trasformare successivamente in un numero, come nel Listato 4.1.

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

extern int errno;

int main() {
  int n;
  char s[256];
  do {
    printf("Inserisci un numero intero: ");
    scanf("%s", s);
    errno = 0;
    n = (int)strtol(s, (char **)NULL, 10);
    if (errno != 0) {
      printf("Errore: il dato inserito non e' un numero intero.\n");
    }
  } while (errno);
  ...
}
```

Listato 4.1 Un metodo per controllare l'input di numeri.

In questo listato si acquisisce una stringa di caratteri, passata poi alla funzione `strtol`. La funzione converte i primi caratteri della stringa in un `long int`, interpretandoli come un numero espresso nella base data dal suo ultimo parametro (in questo caso 10) e restituisce il risultato della conversione. La stringa può contenere zero o piú spazi iniziali, i caratteri `+` o `-`, seguiti dai caratteri che rappresentano le cifre. Gli eventuali spazi iniziali sono ignorati. La conversione si ferma non appena trova un carattere non valido. Il secondo parametro rappresenta l'indirizzo di una stringa nella quale `strtol` copia i caratteri non convertiti. Nel caso in cui tale indirizzo sia `NULL`, come nel Listato 4.1, gli eventuali caratteri in eccesso sono scartati.

Qualora `strtol` non riesca ad eseguire la conversione correttamente, assegna un valore diverso da zero a `errno`, una variabile globale i cui valori sono definiti in `errno.h`.

Oltre alla funzione `strtol`, in `stdlib.h` sono definite anche `strtoll` che restituisce un `long long int`, `strtof`, `strtod` e `strtold` che restituiscono, rispettivamente, un `float`, un `double` e un `long double`.