Problemi con scanf, e come evitarli

Quando si esegue la lettura di un dato da tastiera con la funzione *scanf*, l'utente deve inserire da tastiera il dato e farlo seguire dalla pressione del tasto <enter>.
Il dato inserito viene interpretato secondo il *formato* specificato dal corrispondente carattere di conversione %... inserito nella stringa di formattazione presente nella chiamata a scanf.
A seconda del formato associato al dato letto, parte dell'input inserito via tastiera dall'utente può venire trascurata da *scanf*.

Problema 1: uso di scanf per la lettura di caratteri dopo che sono già state eseguite altre letture con scanf
Per la lettura di singoli carattri si utilizza il carattere di conversione *%c*. Si consideri l'istruzione

```
[1]  scanf (“%c”, &VariabileDiTipoChar);
```

il carattere di conversione *%c* specifica che va letto un char. Se l'utente, prima di premere <enter>, digita più di un carattere, solo il primo di essi viene effettivamente considerato. Tuttavia GLI EVENTUALI CARATTERI SUPERFLUI IMMESSI NON VENGONO ELIMINATI, ma restano "a disposizione" della successiva chiamata a *scanf*, che può quindi produrre risultati inaspettati. Tra questi caratteri che restano a disposizione è incluso IL CARATTERE CORRISPONDENTE AL COMANDO DI "NEWLINE" (chiamato Line Feed, o LF, e rappresentato come '\n'). Infatti la pressione del tasto <enter> produce l'immissione di un carattere: il carattere LF, appunto.

Più in generale: tra i caratteri inseriti via tastiera dall'utente *scanf* usa esclusivamente quelli necessari per il tipo di dato specificato dal carattere di formattazione; e comunque, non utilizza mai il carattere LF inserito dall'utente per indicare la fine dell'input.
La cosa non costituisce un problema se quella considerata è l'unica istruzione *scanf* di tutto il programma. Non è un problema nemmeno se esiste una successiva *scanf*, ma questa si riferisce alla lettura di dati numerici: in questo caso, infatti, i caratteri che non possono far parte di un numero (come LF) vengono ignorati.
Se però esiste una istruzione *scanf* successiva e quest'ultima riguarda la lettura di un carattere o di una stringa di caratteri, sorge un problema. Infatti in questo caso il primo (o unico) carattere letto sarà proprio il LF “lasciato indietro” dalla precedente *scanf*. In particolare, se la nuova *scanf* prevede la lettura di un char, all'utente non verrà nemmeno chiesto di inserire nulla da tastiera.

Uno dei modi per evitare ciò è, per ogni *scanf* relativa ad un char o ad una stringa

che sia stata preceduta da un'altra scanf, eliminare il LF rimasto dalla *scanf* precedente. Ciò si può ottenere aggiungendo in coda alla stringa di controllo della prima scanf o in testa alla stringa di controllo della seconda scanf il carattere di conversione *%*c*, che esegue una lettura di carattere *ed elimina il risultato di tale lettura senza inserirlo da nessuna parte*. Più in generale, inserire "*" tra '%' e il carattere che indica il tipo di dato ('c', 'd', 'f', ...) indica che il dato va eliminato.
Ad esempio se all'istruzione [1] segue, ovunque nel programma, una lettura di carattere, perché quest'ultima operi correttamente essa andrà scritta nella forma

```
scanf ("%*c%c", &AltraVariabileDiTipoChar);
```

Il carattere di conversione %*c può essere utilizzato anche in altri modi. Ad esempio quando viene eseguita una serie di letture da tastiera di singoli caratteri (con uso ripetuto di *scanf*) è possibile eliminare il '\n' associato alla pressione del tasto <enter> immediatamente dopo il prelievo del carattere utile, direttamente tramite l'istruzione *scanf*: in questo caso la sintassi utilizzata sarà del tipo

```
scanf ("%c%*c", &VariabileDiTipoChar);
```

Ad ogni esecuzione di questa istruzione, il calcolatore richiede all'utente di inserire un carattere seguito dalla pressione di <enter>, e dei due caratteri così inseriti (quello utile e il '\n' corrispondente a <enter>) il secondo viene eliminato senza venire memorizzato.

Problema 2: lettura di stringhe di testo con scanf
scanf può essere utilizzata anche per eseguire in modo semplice la lettura da tastiera di una stringa di testo, da memorizzare in un array di caratteri. Per fare questo si usa il carattere di conversione *%s*.
scanf memorizza le stringhe secondo il formato standard del C, ovvero come *C-string*: in questo formato, i caratteri utili sono memorizzati nell'array in ordine, a partire dall'elemento di indice 0, e arrivano fino al primo carattere '\0' (escluso) presente nell'array.
Per leggere una stringa di testo e memorizzarla in un array di caratteri chiamato ArrChar si usa la sintassi

```
[2]  scanf ("%s", ArrChar);
```

La mancanza della & di fronte al nome della variabile array non è un errore: è una conseguenza del legame tra array e puntatori (argomento che verrà trattato più avanti[1]).
Durante l'esecuzione dell'istruzione [2], i caratteri vengono prelevati dalla tastiera uno dopo l'altro e copiati nella variabile array ArrChar. La copia si interrompe

1 In effetti al posto di ArrChar è possibile utilizzare un qualsiasi puntatore a carattere: ad esempio uno che punti ad un elemento intermedio di un array di caratteri. In questo caso scanf comincia a scrivere in memoria dalla cella il cui indirizzo è contenuto nel puntatore.

quando l'utente inserisce un carattere '\n', il che corrisponde alla pressione del tasto <enter>. Dopo aver memorizzato nell'array l'ultimo carattere utile inserito (quello che precede il '\n'), *scanf* scrive subito dopo di esso un carattere '\0', in modo da rendere il contenuto dell'array conforme al formato C-string. Il carattere '\n' non viene memorizzato nell'array.

Il problema che può verificarsi nell'uso di *scanf* per leggere da tastiera e memorizzare stringhe è che scanf non conosce la dimensione dell'array in cui la stringa viene memorizzata, e pertanto continua a scrivere caratteri in memoria fino a che l'utente non inserisce da tastiera un '\n' (premendo <enter>). Se l'utente tarda troppo a farlo, *scanf* riempie l'array per intero e continua a scrivere nelle celle di memoria RAM successive all'area occupata dall'array. Ciò genera sempre dei problemi, che possono essere di due tipi:

1. se l'area di memoria che segue l'array contiene dati non di proprietà del programma, il sistema operativo blocca l'esecuzione di quest'ultimo e segnala il malfunzionamento con un messaggio di *segmentation fault*;
2. se l'area di memoria che segue l'array contiene dati di proprietà del programma (altre variabili), questi dati vengono **sovrascritti** da i nuovi byte scritti da scanf, e il loro valore precedente viene perso. In questo caso il comportamento successivo del programma diventa imprevedibile, perché il programma elabora dati alterati. Nonostante ciò, il sistema operativo non è in grado di rendersi conto che esiste un problema e dunque non blocca il programma. Dunque è possibile che il programmatore e/o l'utente non si rendano conto di quanto sta avvenendo fino a molto tempo dopo, quando i danni alle variabili provocati da scanf si manifestano.

Entrambe le categorie di problemi vanno evitate assolutamente.
Il modo corretto per farlo è non usare mai *scanf* per eseguire la lettura di linee di testo eccetto che in programmi di prova in cui l'utente è lo stesso programmatore, e il cui uso non sarà mai permesso a qualcun altro. Questo è l'unico caso in cui è ammissibile usare scanf per leggere stringhe. Naturalmente nell'usare il programma il programmatore/utente dovrà fare attenzione a non inserire da tastiera più caratteri di quelli che l'array è in grado di contenere.
Si ricordi che un array composto da N caratteri è in grado di contenere una C-string composta al massimo da N-1 caratteri: infatti occorre garantire che il '\0' di terminazione (che *scanf* inserisce sempre) si trovi all'interno dello spazio di memoria occupato dall'array.