Lezione 2: esercizio pratico

La lezione è centrata su una variante dell'Esercizio 2 della dispensa `Lez02-Esercizi-IO.pdf` e ha lo scopo di ripassare i seguenti argomenti:

- *parsing* degli *stream*
  - interpretazione dei dati in ingresso, provenienti da tastiera, file o stringhe;
  - formattazione dei risultati in uscita, diretti a video, file o stringhe;
- gestione delle stringhe
- gestione dei file
- uso dei parametri della linea di comando

Nota *Prego chi notasse eventuali incoerenze o errori, oppure avesse dubbi sul contenuto di queste pagine e dei codici, di segnalarmeli per contribuire a migliorare i materiali del corso.*

Esercizio

Si scriva un programma `catalogo.c` che riceve da tastiera una serie di articoli, nel formato

articolo prezzo giorno mese anno

dove *articolo* è una singola parola, *prezzo* un numero reale positivo, *giorno* e *anno* sono numeri interi che indicano il giorno e l'anno di una data e *mese* il nome di uno dei dodici mesi (scritto in minuscolo). Per segnalare il termine della serie di articoli, l'utente inserirà da tastiera la parola chiave `STOP`.

Dopo l'inserimento di ciascun articolo, il programma deve stamparne le informazioni a video su tre colonne, rispettivamente di 10, 9 e 11 caratteri, secondo il formato seguente:

```
Articolo1      12.50 21/05/2010
Articolo2     199.90 01/04/2010
```

Si scriva poi una versione del programma che riceve i dati da file di testo, e che riceve il nome del file di testo come parametro attraverso la linea di comando.

Traccia della risoluzione

La lezione comincia con un ripasso delle principali procedure per la gestione delle stringhe e delle procedure `scanf` e `printf` per l'interpretazione dei dati inseriti da tastiera e la scrittura formattata a video (pagine da 1 a 14 dei lucidi).

Per completare la lezione precedente, basata sull'approccio *top-down*, in questa lezione adottiamo un approccio *bottom-up*, nel quale si parte dalla soluzione di problemi semplificati, per costruire poi la soluzione al problema complessivo. L'esercizio è diviso in fasi successive, ciascuna delle quali risolve un problema leggermente più sofisticato della precedente, aggiungendo alcune righe al codice, in modo tale da generare sempre un programma sintatticamente corretto e che fa qualcosa di sensato. Gli studenti dovrebbero leggere la descrizione di ciascuna fase, provare a realizzarla partendo dalla precedente e poi verificare la soluzione disponibile e ragionare sulle eventuali differenze. Il punto di partenza dell'intero esercizio è il file `catalogo0.c`, che fornisce un programma vuoto, con un `main` dichiarato, ma vuoto, e alcune direttive di base.

Prima fase (`catalogo1.c`) Partendo dal file `catalogo0.c` realizziamo nel `main` una versione semplificata dell'algoritmo finale. Questa deve trattare un solo articolo, eseguendo:

1. *parsing* di un articolo inserito da tastiera (`scanf`)
2. verifiche varie di correttezza, con la stampa di messaggi di errore sullo `stderr`
3. conversione del mese da forma estesa a indice numerico (cascata di costrutti `if...else` e `strcmp`)
4. stampa formattata (`printf`)

Seconda fase (`catalogo2.c`) Per muovere verso la soluzione del problema complessivo, raccogliamo le istruzioni in una funzione *GestisceArticolo*, aggiungendone la chiamata dentro il `main`, il prototipo prima del `main` e la definizione dopo il `main`. Questo ci servirà a chiamare la funzione su ciascun articolo iterativamente. In un approccio *top-down* saremmo invece partiti da un ciclo che chiamasse la funzione, inizialmente definita vuota, e poi avremmo realizzato il corpo della funzione. In generale questa seconda strada è preferibile, ma può capitare di accorgersi a posteriori che vi siano pezzi di codice utili che meritano di trasformarsi in funzioni, e quindi è utile vedere almeno un esempio di questo tipo. I due approcci sono in effetti complementari.

Terza fase (`catalogo3.c`) Avviciniamoci quindi a risolvere il problema complessivo, inglobando la funzione *GestisceArticolo* in una funzione *GestisceCatalogo* che ripetutamente la chiama finché l'utente non inserisca la parola chiave `STOP`. La funzione *GestisceArticolo* va modificata per riconoscere e comunicare alla funzione chiamante se l'utente ha inserito la parola chiave. In questo caso, bisognerà terminare l'esecuzione; altrimenti, bisognerà procedere richiamando la funzione per leggere l'articolo successivo. Per pulizia stilistica, definiremo un risultato di tipo logico. Siccome in C90 questo tipo non esiste, aggiungiamo fra le direttive i simboli `boolean`, `TRUE` e `FALSE` per simulare il tipo logico con il tipo intero e i due valori convenzionali 1 e 0. Infine, racchiuderemo la chiamata a *GestisceArticolo* in un ciclo a condizione finale (`do ... while (...);` che usa il risultato della funzione come condizione di permanenza.

Quarta fase (`catalogo4.c`) Il programma precedente è scorretto, perché non si arresta quando l'utente inserisce `STOP`. Questo succede perché la funzione `scanf` si aspetta cinque dati, anziché uno solo. Un possibile ripiego è usare la redirezione dell'input per passare i dati da file: siccome il file termina con un carattere speciale `EOF`, la funzione `scanf` termina in quanto tale carattere non è coerente con il secondo dato da leggere. I lucidi da pagina 15 a 21, che trattano il *parsing* di stringhe e file e la gestione dei file, forniscono una soluzione più elegante e generale.Se il programma legge un'intera riga alla volta in una stringa (`gets`), e poi fa il *parsing* della stringa (`sscanf`) anziché dello `stdin`, il problema è superato. Infatti, la funzione `sscanf` non rimane bloccata in attesa di altri dati, dato che la stringa è inequivocabilmente terminata, mentre l'ingresso da tastiera è potenzialmente illimitato. Volendo, si può usare una stringa anche come passaggio intermedio nella stampa (`sprintf` e `puts`).

Quinta fase (`catalogo5.c`) Ora si può sostituire la lettura da tastiera con la lettura da un file. Per cominciare, il file avrà un nome fisso (`"catalogo.txt"`). Questo comporta la definizione di una variabile per accedere al file (`FILE *`), l'apertura (`fclose`) e la chiusura (`fclose`) del file e la sostituzione delle procedure `scanf` e `printf`, `gets` e `puts`, con le corrispondenti procedure `fscanf` e `fprintf`, `fgets` e `fputs`.

Sesta fase (`catalogo6.c`) I lucidi da pagina 22 a 24 trattano i parametri della linea di comando. Aggiungiamo una funzione *InterpretaLineaComando* che acquisisce dalla linea di comando (`argv`) il nome del file di testo contenente il catalogo e lo inserisce in una stringa opportuna (`strcpy`).