

## Lezione 1: esercizio pratico

La lezione è centrata su un semplicissimo esercizio introduttivo, che ha lo scopo di ripassare i seguenti argomenti:

- struttura di un programma in C
  1. direttive di precompilazione
  2. prototipi (dichiarazioni) delle funzioni secondarie
  3. funzione principale (`main`)
  4. definizioni delle funzioni secondarie

**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** L'esercizio è introdotto nel lucido 20 della lezione 1: si scriva un programma `hello.c` che chiede all'utente di indicare un carattere e un saluto e stampa a video il saluto, con una cornice rettangolare costituita dal carattere. I lucidi 20-23 presentano l'applicazione dell'approccio *top-down* all'esercizio e l'albero delle funzioni che ci serviranno, supponendo di avere già una libreria per la gestione dell'ingresso da tastiera e dell'uscita a video. I lucidi 24-27 introducono la divisione del codice in file distinti, che corrisponde alla constatazione che il livello inferiore della decomposizione del problema consiste di funzioni probabilmente utili in altri contesti, e che quindi è conveniente isolare per poterle riutilizzare. Il lucido 28 presenta la struttura generale di un codice C. Alternando i lucidi seguenti con la visione dei codici, si possono ora aprire e scorrere i file della libreria `advio.c` e `advio.h` e il file `hello0.c`, che costituisce il punto di partenza dell'esercizio.

**Traccia** La lezione comincia osservando la struttura del codice `hello0.c`, che è già abbastanza evidente, anche se per ora incompleta, dato che mancano dichiarazioni e definizioni delle funzioni secondarie.

Si eseguono quindi uno per uno i tre passi della compilazione e si osservano i risultati, poi si esegue la compilazione in blocco

Si riempie il `main` con le tre funzioni individuate nella decomposizione *top-down*, e descritte nei commenti:

1. `AcquisisceCornice` è una normale funzione che restituisce un risultato di tipo `char`, anche se non riceve dati (arrivano da tastiera, cioè dall'esterno);
2. `AcquisisceSaluto` è una funzione "strana", dato che apparentemente non restituisce risultati e riceve un dato; in realtà, non ha dati (che arrivano ancora da tastiera), ma il risultato viene gestito come se fosse un dato; i motivi, assolutamente tecnologici, verranno discussi in una lezione successiva;
3. `StampaSaluto` è una normale funzione con due dati, e nessun risultato (il risultato è un'azione di stampa a video).

L'osservazione principale è lo stile della stesura:

1. scriviamo la chiamata di ciascuna funzione, definendo le opportune variabili per conservare i suoi dati e risultati;

2. aggiungiamo il prototipo/dichiarazione nella parte 2 del codice, dichiarando i tipi dei dati e dei risultati, per garantire che lo spazio per contenerli in memoria sia preparato correttamente;
3. aggiungiamo la definizione vuota nella parte 4 del codice (per l'acquisizione della cornice definiamo un valore di default da restituire con l'istruzione `return`: per convenzione, useremo la costante simbolica `SPAZIO`, fornita dalla libreria.

Il codice compila perfettamente. Non fa niente, dato che le funzioni sono vuote. La cosa importante, però, è che compila perfettamente: conviene verificare frequentemente che non si stiano introducendo errori.

Ora si può passare a riempire la definizione di `AcquisisceCornice`, decomponendola in due funzioni già fornite dalla libreria. Ancora una volta, il codice compila, e in più fa qualche cosa. Volendo verificare che la lettura sia stata corretta, possiamo chiamare la funzione `StampaCarattere` e vedere che cosa si è effettivamente letto.

Poi riempiamo l'acquisizione del saluto, sulla quale non c'è molto di più da dire.

Infine riempiamo la stampa, divisa in sottofunzioni prese dalla libreria.

1. determinare la larghezza della cornice (lunghezza del saluto più 4) e salvarla in una variabile `larghezza`;
2. andare a capo (per separare bene le istruzioni all'utente dalla stampa finale);
3. stampare la cornice superiore, cioè stampare `larghezza` volte il carattere `cornice`;
4. andare a capo;
5. stampare la cornice laterale sinistra, cioè stampare una volta il carattere `cornice`;
6. stampare uno spazio separatore, cioè stampare una volta il carattere `SPAZIO`;
7. stampare il saluto, cioè stampare la stringa `saluto`;
8. stampare uno spazio separatore, cioè stampare una volta il carattere `SPAZIO`;
9. stampare la cornice laterale destra, cioè stampare una volta il carattere `cornice`;
10. andare a capo;
11. stampare la cornice inferiore, cioè stampare `larghezza` volte il carattere `cornice`.

Come osservato nei lucidi, i compiti ripetuti sono molto numerosi, il che testimonia la bontà della decomposizione fatta.

Il risultato della lezione è il file `hello.c`.

Completato l'esercizio, si può usare il codice per fare esempi di redirectione dell'input e output:

- `hello < input.txt` consente di ricevere il carattere e il saluto da file, anziché da tastiera, il che è un notevole vantaggio, se si deve eseguire molte volte il codice (per esempio, perché lo si sta ancora completando e verificando);

- `hello > output.txt` consente di stampare su file il saluto incorniciato, invece che a video (in realtà vengono anche stampati i messaggi di acquisizione<sup>1</sup>).

---

<sup>1</sup>Per evitare che le istruzioni all'utente si mescolino alla stampa, distorcendola, basta aggiungere un a capo prima di cominciare la stampa della cornice, così la fase di acquisizione e quella di stampa sono ben distinte