Corso di laurea in fisica
Anno Accademico 2020/2021
Laboratorio di informatica 1

Esercitazione 1

Per svolgere questi esercizi, è necessario studiare le *Note sull'interazione fra utente e sistema operativo* nella sezione *Esercitazioni* del sito del corso, ed è utile leggere il file `vigcc.pdf`, nella cartella *Materiale per la prima esercitazione*.

Note Un esercizio di questa esercitazione, e la maggior parte degli esercizi futuri, consistono nella risoluzione di un problema per via algoritmica: dato un problema, ad esempio risolvere equazioni di secondo grado nel campo reale, dovrete ideare un algoritmo che lo risolve, e tradurlo in un programma in C.

La trafila sarà dunque questa: (1) analisi del problema; (2) scelta delle strutture dei dati e ideazione dell'algoritmo risolutivo; (3) stesura del programma che traduce l'algoritmo; (4) compilazione del programma, cioè traduzione del codice sorgente in linguaggio-macchina; (5) esecuzione. Per ora, i primi due punti si risolvono immediatamente: la nozione di *struttura dei dati*, che sarà trattata a fondo in lezioni successive, coinciderà con quella, più semplice, di tipo delle variabili; l'algoritmo risolutivo sarà dato esplicitamente, oppure consisterà in una semplice formula o procedimento.

La compilazione si svolge richiamando, dal terminale su cui lavorate, un programma specifico, il compilatore appunto. Supponendo che il codice sorgente sia stato salvato nel *file* `bluff.c`, il comando di compilazione sarà

```
gcc bluff.c -Wall -std=c99 -o bluff
```

oppure, per chi usa Macos,

```
clang bluff.c -Wall -std=c99 -o bluff
```

Il comando è formato dal nome del programma da eseguire (`gcc` o `clang`), dall'*argomento*, cioè il nome del programma da compilare (in questo caso: `bluff.c`), e dalle *opzioni*, che permettono di specificare varianti del comando. In questo caso, le opzioni sono tre: (1) `-Wall`, che richiede al compilatore di segnalare non solo gli eventuali errori grammaticali nel codice sorgente ma anche i cosiddetti *warning*, cioè punti "sospetti" del codice, nei quali potrebbe esserci un errore sostanziale; (2) `-std=c99`, che indica al compilatore di prendere come riferimento la grammatica del linguaggio C definita nel 1999; (3) `-o`, che introduce il nome che si vuole attribuire al *file* eseguibile generato dal compilatore; in questo esempio, si è seguita la convenzione secondo la quale il *file* eseguibile ha lo stesso nome del *file* con il codice sorgente, ma senza l'estensione `.c`: potete però scegliere qualsiasi altro nome.

Se la compilazione va a buon fine, il compilatore non emette nessun messaggio, e la shell si rimette in attesa del prossimo comando. Per eseguire il programma compilato, dovete digitare (supponendo che il file eseguibile sia stato chiamato `bluff`, come nell'esempio qui sopra):

```
./bluff
```

Il significato dei caratteri `./` in questo contesto verrà spiegato in seguito.

Il resto di questa nota introduttiva ha lo scopo di illustrare la struttura generale di un programma e di dare alcune nozioni che non sono state ancora trattate in aula, ma che sono indispensabili per comporre un programma completo. Si suppongono note le nozioni di variabile, tipo, assegnamento, sequenza, scelta e iterazione.

Lo schema che segue rappresenta un tipico programma in C; vale solo per programmi semplici, e corrisponde al genere di programmi che tratteremo in questo corso. L'aggettivo *semplice* qui non è sinonimo di banale: i programmi che scriveremo corrisponderanno a problemi reali. Le varie parti dello schema saranno descritte una per una. I numeri di riga non fanno parte del testo del programma, e sono riportati solo per facilitare i riferimenti nella spiegazione successiva.

```
 1  /**
 2   * Commento iniziale
 3   */
 4  #include <stdio.h>
 5  #include <stdlib.h>
 6
 7  /* Altre direttive */
 8  /* Dichiarazione delle variabili globali */
 9  /* Definizione delle funzioni */
10
11  int main (int argc, char * argv[])
12  {
13      /* Dichiarazione delle variabili locali */
14      /* Corpo della funzione main */
15      exit (EXIT_SUCCESS);
16  }
```

Le parti del programma comprese fra la combinazione di caratteri `/*` e la combinazione `*/` sono *commenti* che il programmatore inserisce per agevolare la lettura del codice, giustificare e documentare le scelte che ha compiuto, chiarire il senso delle parti. I commenti sono ignorati dal compilatore, e sono quindi superflui dal punto di vista dell'esecuzione del programma; sono però fondamentali nell'arco dello sviluppo del programma e per le successive modificazioni, correzioni e migliorie. Dovrete imparare a commentare adeguatamente un programma; abituatevi, fin dai primi esercizi, a inserire un commento iniziale che espone sinteticamente lo scopo del programma, e aggiungete commenti là dove vi sembra che possa aiutare la lettura altrui.

Le righe 4 e 5 contengono *direttive*, che istruiscono il compilatore a compiere alcune operazioni preliminari alla traduzione vera e propria del codice. Ne riparleremo; per ora, consideratele come parti fisse, da inserire in ogni programma. In seguito, vedremo altri tipi di direttive utili.

I commenti alle righe 8 e 13 indicano i punti nei quali si inseriscono le dichiarazioni delle variabili su cui il programma lavora. Nei primi esercizi non userete le variabili dette *globali*.

In termini generali, si può dire che un programma in C è costituito da una serie di *funzioni*. Il concetto di funzione nella programmazione dei calcolatori è così importante da meritare una trattazione approfondita, che rimandiamo alle prossime lezioni. Per ora, basti sapere che ogni programma deve contenere una funzione speciale, di nome *main*, il cui corpo (nello schema, le righe da 13 a 15) contiene le istruzioni che il programma eseguirà.

Dal punto di vista operativo, i programmi che consideriamo agiscono in tre fasi distinte: (1) "lettura" dei dati, (2) elaborazione, (3) stampa dei risultati.

La lettura, o acquisizione, dei dati può avvenire in diversi modi; oggi ci limitiamo al caso in cui i dati sono digitati alla tastiera durante l'esecuzione del programma. A questo scopo, dovrete utilizzare istruzioni di un tipo particolare, descritte qui di seguito. Si tratta in effetti di funzioni, che appartengono a una raccolta di funzioni predefinite, di uso generale. In attesa di studiare più a fondo l'argomento, potete limitarvi a usarle come se fossero istruzioni primitive del linguaggio.

Per leggere un valore battuto alla tastiera si può usare la funzione `scanf`. Gli esempi che seguono dovrebbero coprire le necessità degli esercizi odierni. Osservate che in tutti i casi il carattere `&` precede l'identificatore della variabile.

```
1. scanf("%d", &n);
2. scanf("%f", &x);
```

L'esempio 1 legge un valore di tipo intero e lo assegna alla variabile `n`, che dev'essere stata dichiarata di tipo `int`; l'esempio 2 legge un valore di tipo decimale e lo assegna alla variabile `x`, che dev'essere di tipo `float`.

Per stampare i risultati, o qualsiasi messaggio, durante l'esecuzione di un programma, adoprerete la funzione `printf`. Anche in questo caso, mi limito per ora a esemplificarne l'uso in casi semplici.

```
1. printf("Messaggio qualsiasi.\n");
2. printf("%d", num);
3. printf("%f", x);
4. printf("x: %8.2f, n: %d\n", x, n);
```

L'esempio 1 stampa un messaggio letteralmente; la combinazione `\n` equivale al carattere "a capo"; l'esempio 2 stampa il valore della variabile `num`, che dev'essere di tipo `int`; l'esempio 3 stampa il valore della variabile `x`, che può essere di tipo `double` o `float`. L'ultimo esempio mostra come costruire frasi che mescolano parti letterali e valori di variabili; ogni sottoespressione introdotta da `%` indica il punto in cui sarà inserito il valore di una variabile. La sequenza compresa tra virgolette è seguita dal'elenco delle variabili cui si fa riferimento, separate da virgole. La combinazione `8.2` specifica che il valore della variabile `x` va stampato in modo che occupi almeno otto posizioni, delle quali due devono seguire il punto decimale. Qualche esperimento, compiuto variando i parametri numerici, vi aiuterà a capirne esattamente il significato.

1 A partire dalla vostra cartella personale (*home directory*), create una nuova cartella, di nome *labinfo1*. All'interno di questa cartella, createne altre tre, di nome *A*, *B*, e *C*. Nella cartella *C*, create due cartelle, di nome *D* e *E*. Popolate ogni cartella con uno o due *file* contenenti un semplice testo. Spostatevi fra le cartelle, osservando ogni volta il contenuto e i permessi di accesso dei *file*.

Spostate nella cartella *D* tutti i *file* contenuti in *B*, ed eliminate *B*. Create, nella cartella *C*, una nuova cartella di nome *A*. Quali sono i percorsi assoluti delle due cartelle di nome *A*?

Sperimentate i comandi `cp`, `mv` e `rm`, osservando via via il loro effetto per mezzo del comando `ls`.

2 Con l'aiuto di un *text editor*, copiate il codice sorgente del programma `fahrcels.c` in un nuovo *file*. Compilate ed eseguite il programma. Modificate il testo del programma, variando i valori estremi della tabella e il passo. Ricompilate e rieseguite il programma.

Per fare una prima conoscenza con i messaggi d'errore del compilatore, modificate il programma cancellando il punto e virgola alla riga 21. Ricompilate e osservate l'effetto. Infine, reinserite il punto e virgola.

3 Procuratevi, dal sito del corso, i due *file* `comp1.txt` e `comp2.txt`, e collocateli in una nuova cartella. Sfruttando i comandi `sort` e `grep`, generate un elenco, ordinato alfabeticamente, di tutti coloro che praticano lo sci e che compaiono in uno dei due file. Generate un elenco di tutti coloro che parlano il tedesco, ordinato secondo l'età.

Quale effetto ha, se ne ha, l'ordine in cui si eseguono le operazioni `sort` e `grep`?

4 Supponete che il costo computazionale di tre algoritmi, ciascuno dei quali risolve un problema diverso, sia espresso dalle funzioni $f(x) = (x^2 - x)/10$, $g(x) = 10x\log(x)$ e $h(x) = 2^{x/50}$. Se disponete di un programma in grado di tracciare il grafico di una funzione, osservate i tre grafici.

Supponendo che un calcolatore sia in grado di eseguire un milione di istruzioni al secondo, calcolate quanto tempo impiegherebbe per eseguire i tre algoritmi fittizi, per i seguenti valori di x: 10, 100, 100, 1000000, ed esprimete il tempo in unità di tempo opportune (dal millisecondo al secolo).

5 La serie di Gregory-Leibniz è una serie che converge al valore di $\pi/4$, ed definita cosí:

$$\sum_{n=0}^{\infty} \frac{(-1)^n}{2n+1}$$

Scrivete un programma che calcola la somma dei primi K termini delle serie, e stampa il valore ottenuto, moltiplicato per 4. Fissate nel codice il valore di K. Modificate poi il programma in modo che il valore di K venga letto all'inizio dell'esecuzione.