Fondamenti di Informatica: esercitazione di laboratorio n. 1
Scrittura dei programmi, utilizzo del compilatore e strumenti per il debugging
Pier Luca Montessoro

1. Sistema operativo e file system

Questo primo esercizio serve per imparare ad utilizzare alcune delle funzionalità di base delle interfacce a linea di comando dei sistemi operativi Linux e Windows (dette anche "finestre di teminale").
NOTA: i comandi devono essere teminati dal tasto "invio".

- Si predisponga il proprio ambiente operativo per l'utilizzo del compilatore gcc. Per l'ambiente Windows: si esegua il file Cygwin.bat di pocket cygwin. Per l'ambiente Unix: avviare Ubuntu tramite WSL di Windows 10/11, Xcode per i sistemi Apple, una macchina virtuale Linux tramite VirtualBox.
- Se non già attiva, si apra una finestra a linea di comandi ("Terminal" negli ambienti Unix).
- Si crei un direttorio di nome FPSDA all'interno del direttorio corrente (comando Windows: "`md FPSDA`"; comando Unix: "`mkdir FPSDA`") – NOTA: questo comando dovrà essere eseguito solo la prima volta; se si prosegue in tempi successivi l'esercitazione sullo stesso computer, il direttorio sarà già presente.
- Ci si sposti nel direttorio appena creato (comando "`cd FPSDA`").
- Si controlli il contenuto del direttorio appena creato (comando Windows: "`dir`"; comando Unix: "`ls -al`"). Il direttorio dovrebbe essere vuoto ad eccezione del direttorio stesso (indicato con ".") e del direttorio padre (indicato con "..").
- Le prossime operazioni permetteranno di creare il file sorgente di un programma in linguaggio C.

Il comando "`code`" seguito dl nome di un file esegue il programma Visual Studio Code che consente di scrivere file di testo (nel nostro caso, file sorgenti in linguaggio C, identificati dall'estensione .c). Se non disponibile, si può usare notepad++.

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

int main()
{
   printf ("hello, world\n");
   return EXIT_SUCCESS;

}
```

- Si esegua tale comando scrivendo come argomento sulla riga di comando il nome del file che si vuole editare: "`code hello.c`" (NOTA: i nomi dei file NON devono contenere spazi e l'estensione deve essere ".c").
  ATTENZIONE: NON lanciare Visual Studio Code dall'icona dell'interfaccia grafica. Si utilizzi la linea di comando specificando il nome del file da editare.
- Si scriva il programma riportato nel riquadro precedente.
- Si salvi il file.
- Si ritorni alla finestra di terminale e si verifichi la presenza del file hello.c nel direttorio corrente (comando Windows: "`dir`"; comando Unix: "`ls -al`").
- Si visualizzi il contenuto di tale file (comando Windows: "`type hello.c`"; comando Unix: "`more hello.c`").
- Spesso è utile scrivere un programma partendo da un programma precedente. Per fare questo è necessario copiare il programma già scritto in un nuovo file. Si effettui una copia di hello.c nel file hello_copia.c (comando Windows: "`copy hello.c hello_copia.c`"; comando Unix: "`cp hello.c hello_copia.c`")
- Si verifichi la presenza del nuovo file hello_copia.c nel direttorio corrente (comando Windows: "`dir`"; comando Unix: "`ls -al`").
- Si verifichi che il contenuto di tale file è identico all'originale (comando Windows: "`type hello_copia.c`"; comando Unix: "`more hello_copia.c`").

2. Compilatore GCC

In questo esercizio il programma sorgente C viene compilato e linkato dal compilatore GCC generando un file eseguibile, che poi viene lanciato in esecuzione. Si osservi che il parametro "`-o hello`" serve per specificare che il file eseguibile creato deve chiamarsi hello (in ambiente Linux i file eseguibili in genere non hanno estensione, mentre in ambiente Windows hanno estensione ".exe").

- Si compili il programma hello.c
  (comando Windows: "`gcc -Wall -o hello.exe hello.c`"; comando Unix: "`gcc -Wall -o hello hello.c`")
- Si verifichi la presenza del file eseguibile nel direttorio corrente (comando Windows: "`dir`"; comando Unix: "`ls -al`").
- Si lanci in esecuzione il programma eseguibile
  (comando Windows: "`hello`"; comando Unix: "`./hello`")

3. Interpretazione dei messaggi di errore del compilatore

È fondamentale imparare a comprendere i messaggi di errore forniti dal compilatore. Essi riportano anche l'indicazione della riga del programma in cui presumibilmente si trova l'errore, ma non sempre il compilatore è in grado di fornire tale indicazione in modo accurato. Per esempio, se si dimentica una parentesi graffa chiusa, è possibile che il compilatore arrivi fino al fondo del file prima di accorgersi che il conto delle partentesi aperte e chiuse non torna.

- Il seguente programma, nella versione a sinistra, contiene diversi errori: lo si scriva in un file, esattamente come riportato (con gli errori), lo si compili e si confrontino i messaggi di errore ottenuti durante la compilazione con la versione corretta del programma riportata a destra. (NOTA: il file può anche essere scaricato dalla pagina Moodle del corso all'indirizzo elearning.uniud.it)
- Si correggano gli errori uno per volta, ricompilando ogni volta, e si verifichi come ogni correzione incide sui messaggi d'errore del compilatore.

```
/* minimo_con_errori.c
   Minimo di una sequenza di interi
   (versione con errori)*/

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

integer main();
{
   integer contatore; totnum; n; min;

   printf (quanti numeri (almeno 2)? );
   do
   {
      scanf ("%d", &totnum);
      if (totnum < 2)
         printf ("quanti numeri (almeno 2)? ");
   } while (totnum < 2)

   /* leggi il primo numero, che diventa il minimo */
   printf ("%d^ numero: ", contatore);
   scanf ("%d", &min);

   /* procedi con gli altri */
   while (contatore < totnum)
   {
      contatore+ +;
      printf ("%d^ numero: ", contatore);
      scanf ("%d", &n);
      if (n < min)
         min = n;
   }

   printf ("il minimo e` %d\n", min);

   return EXIT_SUCCESS;
}
```

NOTA:
- Se non presenti sulla tastiera, i caratteri '{'e '}' possono essere inseriti mediante il tastierino numerico con SHIFT + ALT GR + [e SHIFT + ALT GR +] rispettivamente.

```
/* versione corretta */

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

int main()  /* "int", non "integer", inoltre
               l'intestazione di una funzione
               non deve terminare con il ; */
{
   int contatore, totnum, n, min;
      /* "int", non "integer", inoltre la lista dei
         nomi delle variabili richiede delle virgole
         come separatori, non ; */

   printf ("quanti numeri (almeno 2)? ");
      /* le stringhe di caratteri vanno racchiuse
         tra doppi apici */

   do
   {
      scanf ("%d", &totnum);
      if (totnum < 2)
         printf ("quanti numeri (almeno 2)? ");
   } while (totnum < 2);
               /* il do – while termina con ; */

   /* leggi il primo numero, che diventa il minimo */
   contatore = 1;
   printf ("%d^ numero: ", contatore);
   scanf ("%d", &min);

   /* procedi con gli altri */
   while (contatore < totnum)
   {
      contatore++;   /* l'operatore ++ va scritto
                        senza spazi in mezzo */

      printf ("%d^ numero: ", contatore);
      scanf ("%d", &n);
      if (n < min)
         min = n;
   }

   printf ("il minimo e` %d\n", min);

   return EXIT_SUCCESS;
}
```

4. Funzioni di input/output

Il linguaggio C fornisce diverse funzioni di libreria per l'input di dati da tastiera e la stampa di dati sul monitor.

Negli esempi visti finora potete osservare l'utilizzo di:

- printf, per stampare sul monitor una stringa di caratteri costante o il contenuto di una o più variabili
- scanf, per leggere da tastiera un dato, testuale o numerico, in un formato predefinito (nell'esempio, il %d indica un dato intero decimale). Si osservi che quando l'esecuzione arriva alla funzione scanf, il programma resta in attesa dei dati e riparte non appena i dati sono stati inseriti ed è stato premuto il tasto di invio.

Il seguente programma illustra il funzionamento di printf e scanf (il file può anche essere scaricato dalla pagina Moodle del corso all'indirizzo elearning.uniud.it) . Si provi ad eseguirlo inserendo i numeri uno per riga o più numeri sulla stessa riga. Cosa accade in quest'ultimo caso?

```
/* esempio_printf_scanf.c */

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

int main()
{
   int a, b, c, d;

   printf ("inserire due numeri interi: ");
   scanf ("%d", &a);
   scanf ("%d", &b);
   printf ("ho letto %d e %d\n", a, b);

   printf ("inserire tre numeri interi: ");
   scanf ("%d %d %d", &a, &b, &c);
   printf ("ho letto %d, %d e %d\n", a, b, c);

   do
   {
      printf ("inserire un numero intero "
              "(-1 per terminare): ");
      scanf ("%d", &d);
      printf ("ho letto %d\n", d);
   } while (d != -1);

   return EXIT_SUCCESS;
}
```

5. Debug del programma mediante printf (I)

Il seguente programma (il file può anche essere scaricato dalla pagina Moodle del corso all'indirizzo elearning.uniud.it) dovrebbe calcolare la somma di n numeri, ma non funziona (provare, per esempio, con n = 4)

```
/* Somma di una sequenza di interi */

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

int main()
{
   int contatore, n, dato, somma;

   printf ("quanti numeri? ");
   scanf ("%d", &n);

   contatore = 1;
   somma = 0;

   while (contatore < n)
   {
      printf ("numero: ");
      scanf ("%d", &dato);
      somma = somma + dato;
      contatore = contatore + 1;
   }

   printf ("la somma e` %d\n", somma);

   return EXIT_SUCCESS;
}
```

- Si inseriscano delle opportune istruzioni di printf per visualizzare le variabili e individuare l'errore. Per esempio:

```
...
   while (contatore < n)
   {
      printf ("INIZIO DEL CICLO WHILE\n")
      printf ("valore corrente di contatore: %d\n",
              contatore);
      printf ("numero: ");
      scanf ("%d", &dato);
      printf ("valore corrente di somma: %d\n",
              somma);
      somma = somma + dato;
      printf ("nuovo valore di somma: %d\n",
              somma);
      contatore = contatore + 1;
      printf ("nuovo valore di contatore: %d\n",
              contatore);
   }
...
```

- Si corregga il programma e lo si provi

6. Debug del programma mediante printf (II)

Il seguente programma (il file può anche essere scaricato dalla pagina Moodle del corso all'indirizzo elearning.uniud.it) non termina (lo si blocchi con CONTROL-C)

```
/* elevamento_a_potenza_con_errore.c
   Elevamento a potenza mediante
   moltiplicazioni successive */

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

int main()
{
   int base, esponente, potenza;

   printf ("base: ");
   scanf ("%d", &base);
   printf ("esponente: ");
   scanf ("%d", &esponente);

   potenza = 1;

   while (esponente > 0)
   {
      potenza = potenza * base;
      esponente = esponente + 1;
   }

   printf ("il risultato e` %d\n", potenza);

   return EXIT_SUCCESS;
}
```

- Si applichi la tecnica di debug tramite printf vista nell'esercizio precedente e si corregga il programma.

7. Controllo dei dati di ingresso

Si scriva un programma che legga N numeri compresi tra -50 e +50, estremi compresi, e ne calcoli la media. Se l'utente inserisce un numero non compreso nell'intervallo, il programma deve segnalare l'errore e chiedere di nuovo l'inserimento.
NOTA: la struttura del programma resta invariata rispetto a quella già vista per il calcolo della media, ma la lettura di un singolo numero (sottoproblema!) diventa più complessa.
ATTENZIONE: è importante separare (concettualmente e operativamente) il ciclo relativo alla lettura degli N numeri e quello che insiste nella richiesta di un numero nel caso quello inserito non appartenga all'intervallo valido.
SUGGERIMENTO: per rendere il programma più generale, si definiscano due costanti MINIMO e MASSIMO e si attribuiscano ad esse i valori -50 e 50. Verificare anche che il programma funzioni correttamente se si modificano tali valori.

8. Lettura di un numero esadecimale

Si scriva un programma che legga da tastiera un numero in formato esadecimale e ne stampi il valore decimale.
Suggerimento 1: l'algoritmo è lo stesso della lettura di un numero decimale, quello che cambia è il calcolo del valore rappresentato dai caratteri inseriti
Suggerimento 2: il valore di una cifra esadecimale viene calcolato in modo simile per le cifre 0-9 e per quelle A-F.