Esercitazione di Laboratorio - 1

Pagina del corso :
http://didawiki.cli.di.unipi.it/doku.php/fisica/inf/start

Oggi vediamo i primi programmi in C

- Introduzione agli strumenti di compilazione
- Errori
  - di compilazione
  - di esecuzione
  - logici
- Esercizi

Elementi di compilazione : make e gcc

- Un programma eseguibile è prodotto dal compilatore gcc
- Una tipica invocazione del gcc

  ```
  gcc -Wall -g -O test.c -o test
  ```

  include il nome del file **test.c** e dell'eseguibile prodotto **test**, oltre ad un numero di opzioni.
- Per non doverle ricordare a mente e riscrivere ogni volta, prepariamo un **makefile**
- E poi usiamo `make`, un'utility di automazione, che legge il makefile ed invoca il gcc per noi

Un semplice makefile

- Un makefile elementare ma completo (per noi)

```
# makefile
CC=gcc
CFLAGS=-Wall -g -O -pedantic -Wformat=2 -Wextra
```

- Indica
  - il compilatore da usare (il gcc)
  - le opzioni da invocare
- Va salvato in un file chiamato **makefile** che deve essere locale (nella stessa directory) ai file da compilare

Un semplice makefile: dettagli

```
# makefile
CC=gcc
CFLAGS=-Wall -g -O -pedantic -Wformat=2 -Wextra
```

- -Wall → attiva la maggior parte degli warning
- -g → aggiunge informazione per il debug
- -O → ottimizza lo spazio occupato
- -pedantic → forza l'uso di C standard
- -Wformat=2 → controlla il formato di printf
- -Wextra → extra warning (variabili non usate, corpi vuoti, valori di ritorno mancanti, ecc...)

Il primo programma: un main vuoto

```
#include <stdio.h>

int main(int argc, char** argv) {

 return 0;

}
```

- Abituatevi subito a scrivere le intestazioni correttamente
- Mettete subito il return finale ad ogni funzione
- Includete sempre almeno `<stdio.h>`
- Salvate il file come `vuoto.c`

Compilazione di un main vuoto

```
# makefile
CC=gcc
CFLAGS=-Wall -g -O -pedantic -Wformat=2 -Wextra
```

```
#include <stdio.h>

int main(int argc, char** argv) {

 return 0;

}
```

- Compiliamo con `make vuoto`

```
[rama]:olivia [~/es01] -> make vuoto
gcc -Wall -g -O -pedantic -Wformat=2 -Wextra    vuoto.c   -o vuoto
vuoto.c:3: warning: unused parameter argc
vuoto.c:3: warning: unused parameter argv
```

Esecuzione di vuoto

```
[rama]:olivia [~/es01] -> ./vuoto
[rama]:olivia [~/es01] ->
```

- Come era lecito attendersi, il risultato è “vuoto”
- Il prompt ritorna immediatamente e senza errori
- Il programma è lecito e l'eseguibile prodotto corretto
- Ovviamente non fa altro che uscire subito dopo essere stato avviato
- Notate il comando con cui viene invocato: `./vuoto`

Un main sbagliato

```
#include <stdio.h>

int main(int argc, char** argv) {

        int my_var

        myvar = my_var + 1;

        printf("my_var e' %d\n", my_var);

        return 0;
}
```

- Quanti errori ci sono?
- Quali sono?
- Salviamo il file come `sbagliato.c` e proviamo a compilarlo

Un main sbagliato : leggere gli errori di gcc

```
[rama]:olivia [~/es01] -> make sbagliato
gcc -Wall -g -O -pedantic -Wformat=2 -Wextra sbagliato.c -o sbagliato
sbagliato.c: In function main:
sbagliato.c:7: warning: ISO C forbids nested functions
sbagliato.c:7: error: syntax error before myvar
sbagliato.c:9: error: my_var undeclared (first use in this function)
sbagliato.c:9: error: (Each undeclared identifier is reported only
once
sbagliato.c:9: error: for each function it appears in.)
sbagliato.c: At top level:
sbagliato.c:3: warning: unused parameter argc
sbagliato.c:3: warning: unused parameter argv
make: *** [sbagliato] Error 1
```

- gcc ci segnala 2 errori:
  - `syntax error before myvar`
  - `my_var undeclared`

Un main sbagliato (2)

```
#include <stdio.h>

int main(int argc, char** argv) {

        int my_var;

        myvar = my_var + 1;

        printf("my_var e' %d\n", my_var);

        return 0;
}
```

- E adesso? Un errore diverso da prima:
  - `myvar undeclared`
- Correggiamo anche questo

Un main sbagliato (3)

```
#include <stdio.h>

int main(int argc, char** argv) {

        int my_var;

        my_var = my_var + 1;

        printf("my_var e' %d\n", my_var);

        return 0;
}
```

- L'errore adesso è più interessante :

  sbagliato.c:7: warning: my_var is used uninitialized in this function

- Perchè?

Un main (non più) sbagliato

```
#include <stdio.h>

int main(int argc, char** argv) {

        int my_var = 0;

        my_var = my_var + 1;

        printf("my_var e' %d\n", my_var);

        return 0;
}
```

- Finalmente compila, con i soliti warning su argc e argv inutilizzati (questi sono gli unici warning che potete ignorare).

Eseguiamo “sbagliato” corretto

```
[rama]:olivia [~/es01] -> make sbagliato
gcc -Wall -g -O -pedantic -Wformat=2 -Wextra    sbagliato.c
-o sbagliato
sbagliato.c:3: warning: unused parameter argc
sbagliato.c:3: warning: unused parameter argv
[rama]:olivia [~/es01] -> ./sbagliato
my_var e' 1
```

- Come ci attendevamo, il programma esegue correttamente.
- Ma è giusto pensare che tutto vada bene perchè un programma compila senza errori?

Problemi di esecuzione

- I problemi di esecuzione sono quasi sempre molto più subdoli e difficili da individuare di quelli sintattici
- Il compilatore vi avverte di ogni errore “ovvio” nei vostri programmi
- Quindi quelli che restano (e ce ne sono SEMPRE) sono gli errori non ovvii!
- Alcuni saranno palesi durante l'esecuzione
- Altri saranno errori logici, in cui tutto funziona anche se non come dovrebbe

Problemi di esecuzione : ciclo infinito

```
#include <stdio.h>

int main(int argc, char** argv) {

        int my_var = 0;

        while(my_var == 0) {
                printf("my_var e' %d\n", my_var);
        }

        return 0;
}
```

- Questo programma compila?
- E' un programma corretto?
- Che succede quando viene eseguito?

Problemi di esecuzione (2)

```
#include <stdio.h>

int main(int argc, char** argv) {

        int dividendo = 42;
        int divisore = 0;
        int risultato;

        risultato = dividendo / divisore;

        printf("Il risultato e' %d\n", risultato);

        return 0;
}
```

- Programma `divisione.c`
- Cosa c'è che non va stavolta?
- Compila? Esegue?

Problemi di esecuzione (2)

```
[rama]:olivia [~/es01] -> make divisione
gcc -Wall -g -O -pedantic -Wformat=2 -Wextra    divisione.c
-o divisione
divisione.c:3: warning: unused parameter argc
divisione.c:3: warning: unused parameter argv

[rama]:olivia [~/es01] -> ./divisione
Floating exception
```

- Non poi così sorprendente, vero?
- Eppure anche questi errori sono “facili” da individuare, anche se le informazioni su cosa è andato storto sono poche
- Spesso un'ispezione visiva del codice e qualche printf in punti strategici permettono di individuarli rapidamente

Problemi di esecuzione : errori logici

```
#include <stdio.h>

int main(int argc, char** argv) {

    int addendo1 = 44;
    int addendo2 = 2;

    int risultato;

    risultato = addendo1 - addendo2;

    printf("Gli addendi sono %d e %d\n", addendo1, addendo2);
    printf("Il risultato della somma e' %d\n", risultato);

    return 0;
}
```

- Il programma si chiama somma.c
- Cosa fa? E' corretto? Perchè?

Problemi di esecuzione : errori logici

```
[rama]:olivia [~/es01] -> ./somma
Gli addendi sono 44 e 2
Il risultato della somma e' 42
```

- Sembra ovvio vedere cosa c'è di sbagliato
- Ma questo tipo di errori è il più difficile da trovare e correggere
- Spesso non ci accorgiamo nemmeno che c'è un errore
- Notate come scegliere nomi di variabili SIGNIFICATIVI aiuti a trovare il problema!

Esercizi proposti

Nella propria home directory creare una sottodirectory chiamata es01, in cui metteremo tutti i file C di oggi.

1) Scrivere un programma **nome** che stampa il proprio nome e sulla riga successiva il proprio cognome.

2) Scrivere un programma **cornice** che stampa il proprio nome su una riga racchiuso da una cornice, così:

```
**************
* Aureliano *
**************
```

Riuscite a scrivere il programma utilizzando un solo comando di output?

Esercizi proposti (2)

3) Scrivere un programma **tipo_elementare** che, per ciascun tipo elementare già visto a lezione, stampa una riga contenente il nome del tipo e la dimensione, in byte, di una variabile di quel tipo.

4) Scrivere un programma **area_rettangolo** che dichiara due variabili che rappresentano i lati di un rettangolo, assegna a tali variabili due valori e stampa il perimetro e l'area del rettangolo risultante.

5) Modificare il programma precedente in modo da leggere dall'input i valori delle dimensioni del rettangolo.
Eseguire il programma per un rettangolo di dimensioni 3 x 2.

Esercizi proposti (3)

6) Scrivere un programma **valore** che calcola e stampa il valore di una villetta composta da due piani, ciascuno dei quali include:
   - salotto (dimensioni 3m x 5m)
   - cucina (dimensioni 4m x 4m)
   - camera da letto (dimensioni 5m x 5m)
   - bagno (dimensioni 2m x 3m)

   sapendo che il costo per metro quadro di quella zona è di 1250 euro.

   Per rendere leggibile il codice, definire opportune costanti METROQUADRO, SALOTTO, CUCINA, CAMERA e BAGNO.

   Suggerimento: *inizializzare le costanti SALOTTO, CUCINA, CAMERA e BAGNO in funzione della costante METROQUADRO.*

   (Il valore calcolato dovrebbe essere 155000).