Esercitazione di Laboratorio - 1

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
- Input e Output (printf e scanf)
- Esercizi

Compilazione : make e gcc

- Un programma eseguibile è prodotto dal compilatore – nel nostro caso il gcc
- Una tipica invocazione del gcc, per esempio:

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

  include il nome del file **test.c** e dell'eseguibile prodotto **test**, oltre ad alcune opzioni.
- Per non doverla 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 -Wno-return-type -Wno-implicit-int -g -O -lm
       -pedantic -Wformat=2 -Wextra
```

- Indica
  - il compilatore da usare (il gcc)
  - le opzioni da invocare (da scrivere tutte di seguito su una sola riga)
- 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 -Wno-return-type -Wno-implicit-int -g -O -lm
       -pedantic -Wformat=2 -Wextra
```

- -Wall → attiva la maggior parte degli warning
- -Wno-* → rimuovono alcuni warning particolari
- -g → aggiunge informazione per il debug
- -O → ottimizza il programma compilato
- -pedantic → forza l'uso di C standard
- -Wformat=2 → controlla il formato di printf
- -Wextra → attiva extra warning

Il primo programma: un main vuoto

```
#include <stdio.h>

main() {

}
```

- Abituatevi subito a scrivere le intestazioni correttamente
- Includete sempre almeno `<stdio.h>`
- Salvate il file come `vuoto.c`

Compilazione di un main vuoto

```
# makefile
CC=gcc
CFLAGS=-Wall -Wno-return-type -Wno-implicit-int -g -O -lm
       -pedantic -Wformat=2 -Wextra
```

```
#include <stdio.h>

main() {

}
```

- Compiliamo con `make vuoto`

```
[rama]:olivia [~/es01] ->  make vuoto
gcc -Wall -Wno-return-type -Wno-implicit-int -g -O -lm
-pedantic -Wformat=2 -Wextra    vuoto.c   -o vuoto
[rama]:olivia [~/es01] ->
```

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>

main() {

        int my_var

        myvar = my_var + 1;

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

}
```

- 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 -Wno-return-type -Wno-implicit-int -g -O -lm
-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.)
make: *** [sbagliato] Error 1
```

Un main sbagliato : leggere gli errori di gcc

```
[rama]:olivia [~/es01] -> make sbagliato
gcc -Wall -Wno-return-type -Wno-implicit-int -g -O -lm
-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.)
make: *** [sbagliato] Error 1
```

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

Un main sbagliato (2)

```
#include <stdio.h>

main() {

        int my_var;

        myvar = my_var + 1;

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

}
```

- Ne abbiamo corretto uno.
  E otteniamo un errore diverso da prima:
  - `myvar undeclared`
- Correggiamo anche questo

Un main sbagliato (3)

```
#include <stdio.h>

main() {

        int my_var;

        my_var = my_var + 1;

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

}
```

- L'errore (o warning) adesso è più interessante :

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

- Perché? Qual'è il problema logico?

Un main (non più) sbagliato

```
#include <stdio.h>

main() {

        int my_var = 0;

        my_var = my_var + 1;

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

}
```

- Finalmente compila senza warning.
- Tutti i vostri programmi dovranno compilare correttamente e senza warning!

Eseguiamo “sbagliato” corretto

```
[rama]:olivia [~/es01] -> make sbagliato
gcc -Wall -Wno-return-type -Wno-implicit-int -g -O -lm
-pedantic -Wformat=2 -Wextra    sbagliato.c   -o sbagliato
[rama]:olivia [~/es01] -> ./sbagliato
my_var e' 1
```

- Come ci attendevamo, il programma esegue correttamente.
- Ma è giusto pensare che tutto vada bene solamente perché 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 ovvi!
- Alcuni saranno palesi durante l'esecuzione.
- Altri saranno errori logici, in cui tutto funziona anche se non come dovrebbe.

Problemi di esecuzione

```
#include <stdio.h>

main() {

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

        risultato = dividendo / divisore;

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

}
```

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

Problemi di esecuzione

```
[rama]:olivia [~/es01] -> make divisione
gcc -Wall -Wno-return-type -Wno-implicit-int -g -O -lm
-pedantic -Wformat=2 -Wextra    divisione.c   -o divisione

[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 stampa a video in punti strategici permettono di individuarli rapidamente

Problemi di esecuzione : errori logici

```
#include <stdio.h>

main() {

    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);

}
```

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

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!

Obiettivo: nessun warning

- Ovviamente i vostri programmi dovranno compilare correttamente (quindi senza errori).
- Ma dovrete anche cercare di far si che il compilatore non emetta alcun Warning!
- Gli warning sono avvertimenti che state facendo qualcosa di sbagliato ma che non compromette la compilazione → è comunque un errore!
- Ricordate **DOVETE** compilare usando il **`makefile`** dato!