Programmazione 1 e Laboratorio

Corso A : prof. Roberto Barbuti

- www.di.unipi.it/~barbuti

Corso B : prof. Paolo Mancarella

- www.di.unipi.it/~paolo

Esercitatore : Aureliano Rama

- www.di.unipi.it/~rama

Editor di testo

- Il primo strumento del programmatore è l'editor di testo
- Proprio per questo ci sono letteralmente centinaia di editor diversi
- Uno dei più conosciuti e usati è Emacs

```
emacs nomefile &
```

  - la `&` lancia il processo in background, liberandovi la shell per altro uso

Alcune cose da sapere su Emacs

- Emacs fa un backup del file chiamandolo `nomefile~`
- Quando modifichi un file con Emacs, in realtà modifichi una copia del file (nell'esempio, un file chiamato `#nomefile#`). Le modifiche vanno salvate per renderle permanenti.
- *Ctrl-x Ctrl-s* salva il file
- *Ctrl-x Ctrl-c* esce dal programma

Editor a linea di comando

- Esistono diversi comodi editor a linea di comando:
  - vi (si legge vu-ai)
  - nano
  - emacs -nw (no window)
  - ne (nice editor, non presente di default)
- Un editor a linea di comando è utile in quei casi in cui si stia lavorando da remoto o comunque su una connessione lenta

Scriviamo il primo programma

- Un programma vuoto:

```
int main() {
    return 0;
}
```

- questo programma, sintatticamente corretto, esegue solo l'operazione di ritorno dalla procedura principale, *main*, chiudendo quindi immediatamente l'esecuzione.
- Salvate il programma in `vuoto.c`

Compiliamo ed eseguiamo il primo programma

- Compiliamo vuoto.c

```
gcc vuoto.c -o vuoto
```

- L'opzione `-o` ci permette di rinominare l'output (che altrimenti si chiamerebbe `a.out`)
- Con `ls -l` controllate la creazione di un file di nome vuoto, con flag eseguibile settato
- Eseguitelo con il comando `./vuoto`
- Il comando ritorna subito, senza errori

Il makefile

- `make` è un tool di automazione molto potente e molto utile che ci permette di rendere la compilazione più semplice
- `make` obbedisce alle impostazioni incluse in un file chiamato `makefile`, presente nella directory corrente. Scriviamo quindi il nostro:

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

Compilare con make

- Per compilare adesso basta eseguire il comando make vuoto :

```
-> make vuoto
gcc -Wall -g -O -pedantic -Wformat=2 -Wextra -lm vuoto.c
-o vuoto
->
```

- Se il programma è corretto, la compilazione va a buon fine senza errori
- A volte ci sono degli warning (avvertimenti): vogliamo compilare il più possibile senza warning!

Errori di compilazione

- Errori di sintassi

```
[rama]:olivia [~/PRL1112/1] -> cat err1.c
int main() {
        rreturn 0;
}
[rama]:olivia [~/PRL1112/1] -> make err1
make: Warning: File `makefile' has modification time 1.2e+02 s
in the future
gcc -Wall -g -O -pedantic -Wformat=2 -Wextra -lm err1.c -o err1
err1.c: In function “main”:
err1.c:4: error: “rreturn” undeclared (first use in this
function)
err1.c:4: error: (Each undeclared identifier is reported only
once
err1.c:4: error: for each function it appears in.)
err1.c:4: error: syntax error before numeric constant
err1.c:6: warning: control reaches end of non-void function
make: *** [err1] Error 1
```

Errori di compilazione

- (Semplici) errori semantici

```
[rama]:olivia [~/PRL1112/1] -> cat err2.c
int main() {
        int c = 3/0;
        return 0;
}
[rama]:olivia [~/PRL1112/1] -> make err2
make: Warning: File `err2.c' has modification time 1.4e+02 s in
the future
gcc -Wall -g -O -pedantic -Wformat=2 -Wextra -lm err2.c -o err2
err2.c: In function “main”:
err2.c:4: warning: division by zero
err2.c:4: warning: unused variable “c”
make: warning:  Clock skew detected.  Your build may be
incomplete.
```

Risorse d'aiuto

- C Reference Card (ANSI) :

  http://www.di.unipi.it/~rama/didattica/PRL1112/c_reference_card_ansi_2.2.pdf

- Gdb Quick Reference :

  http://www.di.unipi.it/~rama/didattica/PRL1112/gdb_quick_reference.pdf