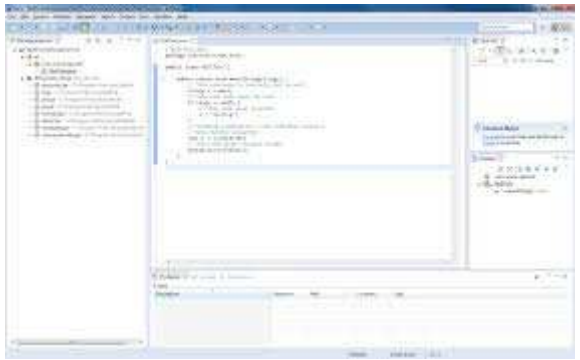
L'ambiente UNIX/Linux

Strumenti per la programmazione C

Stefano Quer
Dipartimento di Automatica e Informatica
Politecnico di Torino

IDE

- Il software si sviluppa usualmente in un
  - Integrated Development Environment (IDE)
- Tra gli IDE "free" ricordiamo
  - Netbeans
    - C, C++
    - https://www.netbeans.org/
  - Code::Blocks
    - C, C++, Fortran
    - http://www.codeblocks.org/



IDE

- Eclipse
  - Per Java, C, C++ e altri linguaggi
  - http://www.eclipse.org/
- CodeLite
  - Open source, cross platform specializzato in C e C++
- Geany
  - Molto semplice, pochi plug-in, completamente in italiano
- MonoDevelop
  - Realizzato per C# e altri linguaggi .NET
- Anjuta
  - Per CD, C++, Java e Python dal progetto GNOME

Editor

- Editor più utilizzati in ambienti UNIX/Linux
  - Sublime
  - Atom
  - **Vim (vi)**
  - **GNU Emacs**
  - Gedit
  - Nano
  - Brackets
  - Bluefish
  - Spacemacs

Editor: VIM (VI)

- Editor di testo
  - Presente in tutti i sistemi BSD e Unix
  - Sviluppato a partire dal 1976
  - Ultima versione (8.1) del 2018
- Versione base
  - VI = "Visual in ex"
    - Commuta l'editor di linea ex in modalità visuale
  - Non molto funzionale per operazioni estese
  - Utile in caso di problemi con altri editor (e.g., editing remoto)

Editor: VI (VIM)

- Nel tempo è stato ampliato e migliorato
  - VIM "VI Improved"
  - Estensione per editing di progetti complessi
    - Multi-level undo, multi-window, multi-buffer, etc.
    - On-line help, syntax hightlighting, etc.
- Insieme a emacs è uno dei protagonisti della "guerra degli editor"
- Estensioni
  - Permettono di incrementare le feature dell'editor
  - Bvi (Binary VI), Vigor (VI con Vigor Assistant), VILE (VI Like Emacs)

Editor: VI (VIM)

- Si invoca con il commando
  - vi nomeFile
- Prevede diverse modalità operative
  - Command Mode
    - Cursore posizionato nel testo
    - La tastiera è utilizzata per impartire comandi
  - Input Mode
    - Modalità di inserzione testo
    - La tastiera è utilizzata per inserire testo
  - Directive Mode
    - Cursore posizionato sull'ultima riga del video
    - La tastiera è utilizzata per direttive di controllo

Editor: VI (VIM)

Documentazione
Help locale: man vim
Risorse on-line: http://www.vim.org/docs.php
Risorse in PDF: ftp://ftp.vim.org/pub/vim/doc/book/vimbook-OPL.pdf

Command Mode	Comando
Spostamento cursore	←,↑,→,↓ (h, j, k, l)
Modalità inserimento (dal cursore)	i
Modalità inserimento (a inizio linea)	I
Modalità append (dal cursore)	a
Modalità append (fine linea)	A
Modalità sovrascrittura	R
Passa (ritorna) in modalità comandi	esc
Cancella una riga	dd
Cancella un carattere singolo	x

Anche
0-g
n-g

Anche
n-dd
n-x

Editor: VI (VIM)

Command Mode (continua)	Comando
Inserisce ultima cancellazione	P
Annulla l'ultima operazione (undo)	U
Ripristina l'ultima modifica (redo)	Ctrl-r

Directive Mode	Comando
Passa in modalità direttive (ultima riga)	:
Mostra numeri di riga	:set num
Salva il file	:w!, :w fileNAME
Esce senza salvare le ultime modifiche	:q!
Entra nell'help on-line	:help

Imparare VIM (da Google): VIM Adventures

Editor: emacs

- Editor di testo libero
  - Emacs = Editor MACroS
  - Sviluppato a partire dal 1976 da Richard Stallman
  - Ultima versione (26.2) dell'aprile 2019
- Molto popolare tra i programmatori avanzati, potente, estendibile e versatile
- Ne esistono diverse versioni ma le più popolari sono la
  - GNU Emacs
  - Xemacs = next generation EMACS

Editor: emacs

- Disponibilile per
  - GNU, GNU/Linux
  - FreeBSDm NetBSD, OpenBSD
  - Mac OS X
  - MS Windows
  - Solaris

Editor: emacs

- Vantaggi
  - Molteplici funzionalità ben oltre il semplice editor di testi
  - Completamente customizzabile
  - Esecuzione veloce di operazioni complesse
- Svantaggi
  - Curva di apprendimento lenta
  - Scritto in Lisp

Editor: emacs

- Comandi base disponibili a
  - Menù
  - Sequenze di tasti
    - Comandi control: control + lettera (c-key)
    - Comandi meta: alt + lettera (m-key)

Documentazione
Help locale: man emacs
Risorse on-line: http://www.gnu.org/software/emacs/manual/emacs.html
Risorse in PDF: http://www.gnu.org/software/emacs/maanual/pdf/emacs.pdf

Compiler e Debugger

- Compiler
  - GCC
  - G++
  - Makefile
  - Configure
- Debugger
  - GDB

Compiler: gcc

- Open-Source GNU project
  - Compilatore e linker gcc
  - Supporta C e C++

Documentazione
Help locale: man gcc
Risorse on-line: http://www.gnu.org

Compiler: gcc

```
gcc <opzioni> <argomenti>
```

- Comando di compilazione e linker generico
  - Opzioni
    - Elenco di flag che controllano il compilatore e il linker; ci sono opzioni per la sola compilazione, per il solo linker o per entrambi
  - Argomenti
    - Elenco di file che gcc legge e trasforma in maniera dipendente dalle opzioni

Compiler: gcc

- Compilazione di singoli file e poi link dei file oggetto in un unico eseguibile

```
gcc -c file1.c
gcc -c file2.c
gcc -c main.c

gcc -o myexe file1.o file2.o main.o
```

Run: myexe versus ./myexe
→ echo $PATH
→ PATH=$PATH:./
→ echo $PATH

- Contestuale compilazione di diversi file sorgente, link e creazione dell'eseguibile

```
gcc -o myexe file1.c file2.c main.c
```

Compiler: gcc

Opzioni			
Formato		Significato	Effetto
Compatto	Esteso		
-c file			Esegue la compilazione non il linker
-o file			Specifica il nome di output; in genere indica il nome dell'eseguibile finale (linkando)
-g			Indica a gcc di non ottimizzare il codice e di inserire informazioni extra per poter effettuare il debugging (i.e., vedere gdb)
-Wall			Stampa warning per tutti i possibili errori nel codice

Compiler: gcc

Opzioni			
Formato		**Significato**	**Effetto**
Compatto	**Esteso**		
-Idir			Specifica ulteriori direttori in cui cercare gli header file. Possibile specificare più direttori (-Idir1 –Idir2 …). N.B. Non ci sono spazi tra –I e il nome del direttorio.
-lm			Specifica utilizzo libreria matematica
-Ldir			Specifica direttori per ricercare librerie preesistenti

Non inserire spazi

Esempio

```
gcc -Wall -g -I. -I/myDir/subDir -o myexe \
  myMain.c \
  fileLib1.c fileLib2.c file1.c \
  file2.c file3.c -lm
```

- Contestuale compilazione di diversi file sorgente, link e creazione dell'eseguibile
  - Comando su più righe
  - Fornisce "All Warnings"
  - Non ottimizzare il codice (debug)
  - Preleva gli header in due direttori
  - Inserisce la libreria matematica

Makefile

- Tool di supporto allo sviluppo di progetti complessi
- Sviluppato a partire dal 1998
- Costituito dalle utility
  - Makefile
  - Make
- Fornisce un mezzo conveniente per automatizzare le fasi di compilazione e linker
- Help
  - man make

Primo linguaggio di scripting che analizziamo

Strumento estremamente duttile, ma punto di forza principale è la verifica delle dipendenze

Makefile

- Makefile ha due scopi principali
  - Effetturare operazioni ripetitive
  - Evitare di (ri)fare operazioni inutili
    - Mediante la verifica di **dipendenze** e l'istante dell'ultima **modifica** evita di gestire ripetutamente operazioni inutili (e.g., ricompilare file non modificati)
- Si procede in due fasi
  - Si scrive un file Makefile
    - File di testo simile a uno script (di shell o altro)
  - Si interpreta il file Makefile con l'utility make
    - In questo modo si effettuano compilazione e link

Makefile

Opzioni			
Formato		**Significato**	**Effetto**
Compatto	**Esteso**		
-n			Non esegue i comandi ma li stampa solo
-i	--ignore-errors		Ignora gli eventuali errori e va avanti
-d			Stampa informazioni di debug durante l'esecuzione
	--debug=[options]		Opzioni: a = print all info, b = basic info, v = verbose = basic + altro, i = implicit = verbose + altro

Makefile: Opzioni

- Il comando make può eseguire sorgenti (Makefile) diversi dal file standard ovvero
  - Il comando make esegue, di default
    - Il file makefile se esiste
    - Oppure il file Makefile se makefile non esiste
  - -f <nomeFile> (oppure –file <nomeFile>)
    - Permette di eseguire il Makefile di nome specificato
    - make --file <nomeFile>
    - make --file=<nomeFIle>
    - make –f <nomeFile>

Makefile: Formato

Carattere di tabulazione

```
target: dependency
  <tab>command
```

- Ogni Makefile include
  - Righe bianche
    - Esse sono ignorate
  - Righe che iniziano per '#'
    - Esse sono commenti e sono ignorate
  - Righe che specificano regole
    - Ogni regola specifica un obiettivo, delle dipendenze e delle azioni e occupa una o più righe
    - Righe molto lunghe possono essere spezzate inserendo il carattere "\" a fine riga

Makefile: Formato

```
target: dependency
  <tab>command
```

- Quando si esegue un Makefile (con il comando make)
  - Il default è eseguire la prima regola
    - Ovvero quella che compare prima nel file
  - Nel caso esistano più regole può però esserne eseguita una specifica
    - make <nomeTarget>
    - make –f <myMakefile> <nomeTarget>

Makefile: Formato

```
target: dependency
  <tab>command
```

- Ogni regola è costituita da
  - Nome del target
    - Spesso il nome di un file
    - Talvolta il nome di un'azione
  - Lista delle dipendenze da verificare prima di eseguire i comandi relativi alla regola
  - Comando o elenco di comandi
    - Ogni comando è preceduto da un carattere di **tabulazione**, invisibile ma **necessario**

Si definisce "phony" target

Esempio 1: Target singolo

```
target:
  <tab>gcc –Wall –o myExe main.c -lm
```

- Specifica
  - Un'unica regola all'interno del file Makefile di nome **target**
  - Il target non ha dipendenze
- Eseguendo il Makefile
  - Viene eseguito il target
  - Il target non ha dipendenze, quindi l'esecuzione del target corrisponde all'esecuzione del comando di compilazione

Esempio 2: Target multiplo

```
project1:
  <tab>gcc –Wall –o project1 myFile1.c

project2:
  <tab>gcc –Wall –o project2 myFile2.c
```

- Specifica più regole
  - Occorre scegliere quale target eseguire
  - Il default consiste nell'eseguire il primo target
- Eseguendo
  - make
    - Viene eseguito il target project1
  - make project2
    - Viene eseguito il target project2

Esempio 3: Target e azioni multiple

```
target:
  <tab>gcc -Wall -o my \
  <tab>  main.c \
  <tab>  bst.c list.c queue.c stack.c
  <tab>cp my /home/myuser/bin

clean:
  <tab>rm -rf *.o *.txt
```

Comando su più righe

- Specifica più regole
  - Le regole non hanno dipendenze
  - Il primo target esegue due comandi (gcc e cp)
    - Esso viene eseguito con i comandi
      - make
      - make target

Esempio 3: Target e azioni multiple

```
target:
  <tab>gcc –Wall –o my \
  <tab>  main.c \
  <tab>  bst.c list.c queue.c stack.c
  <tab>cp my /home/myuser/bin

clean:
  <tab>rm –rf *.o *.txt
```

Comando su più righe

- Il secondo target rimuove tutti i file di estensione .o e tutti quelli di estenzione .txt
  - Esso viene eseguito con
    - make clean

Esempio 4: Dipendenze

```
target: file1.o file2.o
  <tab>gcc –Wall –o myExe file1.o file2.o

file1.o: file1.c myLib1.h
  <tab>gcc –Wall –g –I./dirI –c file1.c

file2.o: file2.c myLib1.h myLIb2.h
  <tab>gcc –Wall –g –I./dirI –c file2.c
```

- Esecuzione di target multipli in presenza di dipendenze
  - Si verifica se le dipendenze del target sono più recenti del target stesso
  - In tale caso si eseguono le dipendenze prima dei comandi procedendo in maniera ricorsiva

Esempio 4: Dipendenze

```
target: file1.o file2.o
  <tab>gcc –Wall –o myExe file1.o file2.o

file1.o: file1.c myLib1.h
  <tab>gcc –Wall –g –I./dirI –c file1.c

file2.o: file2.c myLib1.h myLIb2.h
  <tab>gcc –Wall –g –I./dirI –c file2.c
```

- Target ha come dipendenze file1.o e file2.o
  - Si verifica la regola file1.o
    - Se file1.c (oppure myLib1.h) è più recente di file1.o si esegue tale regola, ovvero il comando gcc
    - Altrimenti non si esegue tale regola
  - Si procede analogamente per la regola file2.o
  - Al termine si esegue il target **se** è necessario

Esempio 4: Dipendenze

Nome azione (phony target)

```
target: file1.o file2.o
  <tab>gcc –Wall –o myExe file1.o file2.o

...

file2.o: file2.c myLib1.h myLIb2.h
  <tab>gcc –Wall –g –I./dirI –c file2.c
```

Nome file

- Se il target non è il nome di un file è un ”phony” target che dovrebbe sempre essere eseguito
- Per essere sicuri venga sempre eseguito
  - .PHONY : target

Indipendentemente dall’esistenza di un file con lo stesso nome più recente delle dipendenze

Regole implicite e modularità

- Esistono regole molto potenti per modularizzare e rendere più efficiente la scrittura dei makefile
  - Utilizzo di macro
  - Utilizzo di regole implicite
    - La dipendenza tra .o e .c è automatica
    - La dipendenza tra .c e .h è automatica
    - Le dipendenze ricorsive sono analizzate automaticamente
    - etc.

Esempio 5: Macro

```
CC=gcc
FLAGCS=-Wall -g
SRC=main.c bst.c list.c util.c

project: $(SRC)
  <tab>$(CC) $(FLAGS)-o project $(SRC) -lm
```

Definizione macro:
macro=nome
(con o senza spazi)

Utilizzo della macro:
$(macro)

- Le macro permettono di definire
  - Simboli
    - Compilatore, flag di compilazione, etc.
  - Elenchi
    - File oggetto, eseguitili, direttori, etc.

Esempio 6: Multi-folder

```
CC=gcc
FLAGCS=-Wall -g
SDIR=source
HDIR=header
ODIR=obj


project: $(ODIR)/main.o $(ODIR)/bst.o
  <tab>$(CC) $(FLAGS)-o $@ $^

$(ODIR)/main.o: $(SDIR)/main.c $(HDIR)/main.h
  <tab>$(CC) $(FLAGS) -c $^

$(ODIR)/bst.o: $(SDIR)/bst.c $(HDIR)/bst.h
  <tab>$(CC) $(FLAGS) -c $^
```

La macro $@ copia il "target name" corrente

La macro $^ copia l'elenco dei file nella lista delle dipendenze

La macro $< copierebbe il primo file dell'elenco nella lista delle dipendenze

Debugger: gdb

- Pacchetto software utilizzato per analizzare il comportamento di un altro programma allo scopo di individuare e eliminare eventuali errori (bug)
- Disponibile per quasi tutti i sistemi operativi
  - Spesso è integrato in IDE grafiche
- Può essere utilizzato
  - Come tool "stand-alone"
    - Utilizzo particolarmente scomodo
  - Completamente integrato con molti editor (e.g., emacs)
- I vari comandi possono essere forniti in maniera completa o abbreviata

Debugger: gdb

Azione	Comandi
Comandi esecuzione step-by-step	run (r) next (n) next <numeroStep> step (s) step <numeroStep> stepi (si) finish (f)
Comandi per breakpoint	continue (c) info break break (b), ctrl-x-blank break numeroLinea break nomeFunzione fileName:numeroLinea disable numeroBreak enable numeroBreak

Debugger: gdb

Azione	Comandi
Comandi di stampa	print (p) print espressione display espressione
Operazioni sullo stack	down (d) up (u) info args info locals
Comandi di listing del codice	list (l) list numeroLinea list primaLinea, ultimaLinea
Comandi per operazioni varie	file fileName exec fileName kill