Sviluppo in C++

strumenti in ambiente Linux

Indice

Introduzione....2
Il processo di sviluppo....2
Terminologia di GNU Emacs....3
Interfaccia di GNU Emacs....5
Editing di un file generico....7
Editing di un sorgente C++....9
Compilazione ed esecuzione....10
Il debugging dei programmi....11
Sessioni di debugging....12
Sviluppo di progetti complessi e GNU Make....15
Makefile: un esempio pratico....15
Flessibilità del Makefile: esempio di uso delle variabili....17

Introduzione

In questi appunti viene trattata una introduzione all'uso di alcuni strumenti, esistenti in ambiente Linux, utilizzabili per lo sviluppo di programmi in C++. Più specificatamente si tratterà di ***GNU Emacs*** editor nonché vero e proprio ambiente di sviluppo all'interno del quale possono essere utilizzati e integrati altri strumenti utili per la programmazione come quelli trattati in questi appunti, ***GCC*** (GNU Compiler Collection) suite di compilatori comprendente, fra l'altro, anche un compilatore C++, ***GDB*** (GNU Debugger) il *debugger* per la correzione degli errori di programmazione, ***GNU Make*** strumento per l'automatizzazione della compilazione, in progetti software che richiedono la stesura di parecchi file contenenti sorgenti. Si tratta, in tutti i casi, di software GPL sviluppati all'interno del progetto GNU dalla FSF di Richard Stallman (Free Software Foundation, http://www.fsf.org). La sigla GNU è un acronimo ricorsivo (Gnu's Not Unix) ed è associata ad un progetto che prevedeva, in origine, anche lo sviluppo di un intero sistema operativo. In seguito Linus Torvalds, il creatore di Linux, rilasciò il suo prodotto con licenza GPL (spesso si parla di GNU/Linux) e la fondazione ha di fatto abbandonato l'idea di un nuovo S.O. concentrandosi sullo sviluppo di strumenti ed utilità che fossero da corredo al S.O. stesso.

Il disegno riportato come intestazione di queste note è il logo del progetto.

Il processo di sviluppo

Scelto un linguaggio di programmazione (per esempio C o C++) e codificato un programma, il problema che si pone è quello di fare diventare tale programma eseguibile da un computer. A tale scopo è necessario utilizzare un ***compilatore*** (nel nostro caso un compilatore C++) che è un particolare software che a partire da un programma scritto in un determinato linguaggio di programmazione (il *programma sorgente* C++) può generare un *programma oggetto eseguibile*. Tutto ciò è ottenuto dal compilatore traducendo in codice binario comprensibile dalla macchina le istruzioni contenute nel sorgente. Tale processo, che può variare e prevedere diverse fasi in dipendenza del compilatore stesso e del suo funzionamento, richiede dal programmatore l'esecuzione di alcune operazioni che possono essere ricondotte in breve alle seguenti:

- Scrittura del sorgente e salvataggio su disco del sorgente stesso. Tale operazione viene effettuata con l'ausilio di un *editor*: software che permette appunto di inserire il testo con le istruzioni del programma e salvarlo su memoria di massa. In questa fase il programma è semplicemente una sequenza di caratteri priva di significato per il computer e, quindi, il programma può essere scritto utilizzando un qualsiasi programma che generi *testo puro*. Non va bene, in genere, un programma di elaborazione testi perché verrebbero inseriti nel testo anche i caratteri di formattazione (grassetto, corsivo, dimensioni caratteri, colori, ecc...) a meno che il software non preveda (e in genere è possibile) la generazione di testo puro. Conviene però utilizzare un editor poiché si tratta di uno strumento creato per chi sviluppa software e quindi provvisto di facilitazioni e aiuti per i programmatori.
- Compilazione del sorgente. Tale fase richiede appunto un *compilatore* che è in grado di tradurre il sorgente in codici eseguibili *per una determinata CPU* e *per un determinato Sistema Operativo*. Se il compilatore non è in condizione di generare il codice eseguibile perché è stato commesso qualche errore di sintassi, viene generata la lista degli errori. In questo caso, anche se qualche compilatore produce il codice eseguibile, non se ne può tenere conto avendo, questo, comportamenti imprevedibili. Bisogna riprendere l'editing del sorgente, correggere le istruzioni

che hanno generato errori e ripetere la compilazione con il programma modificato.

- Esecuzione e test del programma. Una volta generato il codice eseguibile è necessario verificare se il programma fornisce i risultati attesi. A questo punto, scelti i dati sui quali effettuare i test, si lancia il programma per tutti i dati previsti dai test. Se i risultati prodotti sono quelli aspettati: bene! In genere il caso più frequente è quello in cui si verificano situazioni inattese: risultati non coerenti rispetto alle attese, blocchi del programma. Si rende necessario effettuare un monitoraggio dell'esecuzione del programma per vedere quali sono le situazioni che non permettono al programma stesso di funzionare nella modalità corretta. In questi casi è necessario utilizzare un *debugger* che è un software che permette, per esempio, di controllare come variano, nel corso dell'esecuzione, i valori contenuti nelle variabili utilizzate dal programma o fare eseguire le singole istruzioni controllando una per una le modifiche effettuate.

In definitiva, da quanto esposto, per completare il processo di compilazione per la produzione del codice eseguibile, è necessario utilizzare una serie di strumenti software. Quanto meno bisogna utilizzare: editor, compilatore, debugger. In alcuni casi, per completare l'opera del compilatore è necessario un *linker*, in altri casi, in dipendenza del tipo di compilatore, occorrono anche altri strumenti software, per esempio per l'esecuzione del programma compilato. Se gli strumenti riescono ad integrarsi fra di loro consentendo, per esempio, all'editor di dialogare con il compilatore, si può più rapidamente cercare e correggere la riga del sorgente dove è stato commesso un errore. Gli strumenti del progetto GNU hanno questa capacità di integrazione: Emacs è fornito di librerie che gli consentono di interagire in maniera proficua con GCC e GDB, consentendo una maggiore efficienza e velocità, da parte del programmatore, nel processo di sviluppo.

Terminologia di *GNU Emacs*

La prima fase dello sviluppo di un programma prevede la scrittura del sorgente con l'utilizzo di un editor.

Nel nostro caso si utilizzerà, come editor, GNU Emacs: si tratta di uno strumento estremamente potente (e complesso) che, a definirlo solo un editor, sicuramente si pecca di approssimazione. Tanto per avere un idea si tenga presente che il manuale d'uso, disponibile presso il sito della FSF, è composto da più di 600 pagine. In realtà può essere utilizzato come un ambiente completo di sviluppo, come uno strumento per la navigazione all'interno dell'albero delle directory, come interfaccia verso il sistema operativo, come agenda, come piattaforma per giochi, client di posta elettronica ecc.. e inoltre è completamente personalizzabile ed espandibile: basta conoscere e sapere utilizzare il linguaggio con cui è sviluppato. In definitiva si tratta di un ambiente per l'editing e l'esecuzione di macroistruzioni in Lisp (il nome deriva da Editor MACroS).

Tutta questa potenza, naturalmente, ha un prezzo. Da molti è considerato un prodotto *ostico* da imparare ad usare ed effettivamente anche la terminologia adottata non agevola l'utilizzo, specie per chi lo usa per le prime volte o chi, per esempio, è abituato alle interfacce dei programmi che girano in ambienti grafici. Occorre considerare che si tratta di un prodotto nato in ambienti non grafici e

che conserva alcuni modi di operare di quegli ambienti. Per poterlo utilizzare proficuamente è necessario entrare nella sua *filosofia*.

Inizialmente l'interazione con Emacs, alla stessa stregua di tutti i programmi nati in epoca dove ancora le interfacce grafiche erano inesistenti o poco diffuse, avveniva, e tuttora può avvenire, con comandi che vengono impartiti utilizzando delle combinazioni di caratteri. Ora Emacs si è dotato pure di una barra con i pulsanti, per le operazioni più frequenti, e nei sistemi desktop la maggior parte delle operazioni possono essere effettuate con il mouse, così come verrà evidenziato in queste note. Le combinazioni di caratteri possono essere utilizzate come *scorciatoie* dei comandi e i comandi, impartiti nel minibuffer, servono per avere il pieno controllo di un ambiente totalmente personalizzabile.

In generale i caratteri associati ai comandi sono accompagnati da tasti modificatori:

- ***Control*** *(indicato con* `C`*),* ***Meta*** *(indicato con* `M`*)*: il primo è il tasto *Ctrl* che si trova in doppia copia in tutte le tastiere moderne a lato della barra spaziatrice, il secondo è il tasto *Alt* anche questo in doppia copia accanto a *Ctrl* . Il nome *Meta* deriva dalla stampigliatura presente nei tasti nei sistemi Unix. Nelle tastiere frequentemente utilizzate la stampigliatura è stata sostituita, appunto, da *Alt*. Vanno usati in accoppiata con un altro tasto. Per esempio, se si legge la combinazione `C-x`, vuol dire premere uno dei tasti *Ctrl* e, senza rilasciarlo, premere il tasto *x* e quindi rilasciare i due tasti. `C-x C-f` (*Apri file* in Emacs) vuol dire premere *Ctrl* e, senza rilasciarlo, premere in sequenza *x* e poi *f*. `C-x d` (*Apri Directory*) si esegue premendo *Ctrl* e, senza rilasciarlo, si preme *x*; si rilasciano quindi i due tasti e si preme *d*. Allo stesso modo si agisce se la combinazione da eseguire è `M-w` (*Copia selezione*): si preme *Alt* e quindi il tasto *w*. Il tasto *Alt* può essere sostituito dal tasto *Esc* presente nelle tastiere in alto a sinistra. Nell'esempio di prima `M-w` può essere eseguito, in alternativa a quanto esposto prima, digitando *Esc*, rilasciandolo e digitando poi *w*.

L'interazione con l'ambiente è possibile anche tramite vari menù da cui si possono selezionare le operazioni desiderate.

Sono usati anche altri termini che è bene chiarire:

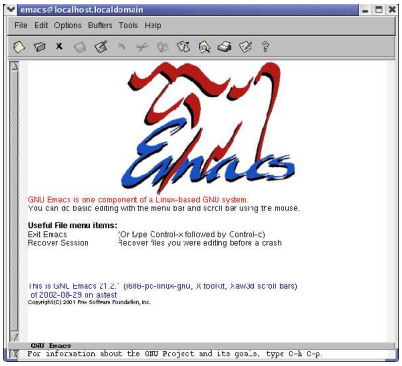
- ***Buffer***: si tratta della zona di memoria centrale che contiene il testo che si sta editando. Un file di testo che si vuole editare viene caricato in un buffer. Se poi si vuole salvare su disco il testo, si effettuerà la selezione, per esempio, dal menù *File* > *Save buffer*.
- ***Window***: si tratta della sola parte di schermo che visualizza il buffer. Negli ambienti grafici viene anche chiamata *area di lavoro*, laddove, in genere, per *finestra* si intende la parte di schermo dentro la quale gira l'applicazione. In Emacs la *finestra* è la parte dentro cui è mostrato il testo (l'area di lavoro) e quindi esiste, per esempio, un comando per dividere la finestra (*File* > *Split Window*) e in questo modo l'area di lavoro viene suddivisa in due parti in modo, per esempio, da permettere l'editazione di un file diverso in ogni window. *File* > *Unsplit Window* riporta il buffer corrente a coprire l'intera area di lavoro.
- ***Frame***: si tratta in questo caso di tutta la zona dello schermo che contiene sia l'applicazione sia il testo, quella che negli ambienti desktop è definita finestra. Se si esegue, per esempio *File* > *New Frame*, ci si ritroverà con un duplicato di tutta l'applicazione. In realtà non si tratta di una nuova istanza di Emacs poiché se, per esempio, in un frame si apre un nuovo testo, lo si ritroverà anche nell'altro. Quindi in definitiva si tratta di due visioni separate dello stesso ambiente

Emacs, quando gira in un ambiente grafico, fa uso dei pulsanti del mouse. Anche in questo caso è bene chiarire la terminologia utilizzata:

- ***Mouse-1***, ***Mouse-2***, ***Mouse-3***: si tratta rispettivamente della pressione dei pulsanti sinistro, centrale, destro del mouse. Nei mouse forniti di soli due tasti, il pulsante centrale è emulato dalla pressione contemporanea dei due tasti presenti. La pressione di uno dei tasti del mouse può essere anche accompagnata dalla pressione dei tasti modificatori. Per esempio `C-Mouse-1` si esegue tenendo premuto il tasto *Ctrl* e premendo il tasto sinistro del mouse (in Emacs mostra l'elenco dei buffer).

Interfaccia di *GNU Emacs*

Quando viene avviato Emacs all'interno di un ambiente desktop viene visualizzata una schermata simile a quella seguente:



- Nella parte superiore si trovano la barra dei menù e una barra di pulsanti per effettuare, in modo rapido, alcune operazioni di uso frequente. Se si sposta il puntatore del mouse su uno di questi pulsanti viene visualizzato un piccolo riquadro contenente una sintetica spiegazione dell'azione legata al pulsante stesso. La barra dei menù può visualizzare voci diverse in relazione al tipo di buffer corrente.
- La zona centrale è la *window* su cui verrà visualizzato il *buffer* corrente. L'area di lavoro nella quale si potrà editare il testo. Ogni area di lavoro è chiusa da una riga contenete informazioni sulle modifiche effettuate al testo, la posizione del cursore, ecc. (la *mode line*)
- L'ultima riga in basso è la *Echo Area* utilizzata per visualizzare il *Minibuffer*. Questa riga è dedicata alle comunicazioni con Emacs: per esempio in questa riga si scriverà il nome del file che si vuole editare. Nella *Echo Area* vengono anche visualizzati i caratteri digitati per i comandi composti da più di un carattere. Si accede, direttamente, alla echo area digitando `M-x` e si esce da questa digitando tre volte il tasto *Esc*.

Quando si edita un file, un rettangolino nero lampeggiante indica il *punto di inserimento* cioè il punto in cui verranno visualizzati i caratteri introdotti da tastiera. Si tenga inoltre presente che si possono caricare più file da editare. Il buffer su cui si sta lavorando è il *buffer corrente.* Se ci sono più finestre nell'area di lavoro il buffer corrente è quello dove il punto di inserimento è evidenziato dal blocchetto pieno lampeggiante; negli altri buffer, eventualmente visualizzati, il punto di inserimento è evidenziato come un rettangolino fisso vuoto.

Alcune fra le operazioni di più frequente utilizzo possono essere avviate dalla barra dei pulsanti. In alcuni casi la pressione del tasto scelto porta nella *Echo Area* dove si dovranno inserire altre informazioni affinché sia possibile l'esecuzione del comando.

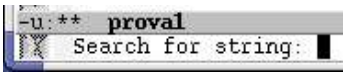
Per cominciare una sessione di editing con un nuovo file si può selezionare la relativa scelta dal menù (*File* > *Open File...*) o premere il pulsante *Read a file*. In ogni caso il cursore passa nell'area dei comandi dove viene richiesto di inserire il nome del file da editare. Viene presentata come posizione, dove conservare il file, la *Home* dell'utente. Si può scrivere di seguito il nome del file o specificare un cammino diverso: per esempio `~/programmi/cpp/primo.cpp`

Se si preme il pulsante *Read a directory* (equivale alla voce di menù *File* > *Open Directory...*), inserendo il nome di una directory o, semplicemente premendo <*Invio*>, viene mostrato nell'area di lavoro l'elenco dei file contenuti nella directory stessa. Utilizzando il tasto centrale del mouse si può navigare lungo l'albero delle directory: `Mouse-2` o la pressione del tasto *Invio* sul nome di un file ne permette il caricamento in un buffer, sul nome di una directory apre una nuova area di lavoro in cui vengono mostrati i file contenuti in essa

I pulsanti, nell'ordine da sinistra verso destra, eseguono le operazioni:

- Chiusura del buffer corrente (quello su cui si trova il punto di inserimento, il blocchetto lampeggiante). Se il file è stato modificato viene chiesta conferma (dal menù *File* > *Close (current buffer)*).
- Salva il buffer corrente con il nome specificato all'apertura (*File* > *Save (current buffer)*).
- Come il precedente ma permette di assegnare un nome al file (*File* > *Save Buffer As...*).
- Tasto di *Undo*: serve per annullare le ultime modifiche effettuate.

Pulsanti *Taglia*, *Copia*, *Incolla*. Selezionata una parte di testo, il primo elimina il testo dalla posizione in cui si trova, il secondo ne fa una copia in memoria. Il terzo pulsante permette di incollare nella posizione in cui si trova il punto di inserimento, il testo tagliato o copiato in precedenza.



Il pulsante *Cerca*, necessita di avere specificato, nell'area dei comandi, la stringa da cercare. Inserita la stringa viene effettuata una ricerca dalla posizione del punto di inserimento in avanti. Il punto di inserimento si sposta nella posizione dove si è trovata la stringa cercata.

Gli ultimi tre pulsanti permettono di stampare il contenuto del buffer attivo, di personalizzare Emacs e di aprire un menù da cui, per esempio, può essere avviato un tutorial o leggere il manuale.

La navigazione all'interno della gerarchia di directory può anche avvenire con un altro strumento disponibile in Emacs: la *Speedbar*. Si tratta di un frame a parte in cui viene visualizzata la directory corrente. Lo strumento si attiva selezionando *Tools* > *Display Speedbar*.

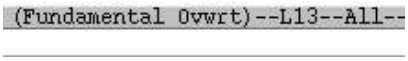
Nello screenshot riportato è visualizzata la directory `prove` contenuta nella *home* dell'utente. La directory `.xvpics` è contenuta in `prove` così come gli altri tre file visualizzati. In realtà non sono mostrati tutti i file che si trovano nella directory, ma solo i file di testo. Per poter vedere tutti i file della directory si può premere `Mouse-3` dentro il frame della *Speedbar*, in modo da rendere disponibile il menù relativo, e selezionare *Show All Files*. Il menù è visualizzabile anche premendo `Mouse-1` nella mode line della speedbar.

Si può passare ad una sottodirectory o caricare in un buffer un file, con `Mouse-2` o il tasto *Invio* sulla directory o sul nome del file. Il doppio clic sul tasto sinistro del mouse produce lo stesso effetto.

Il file con il nome sottolineato (nell'esempio `prova.cpp`) è quello caricato nel buffer corrente.

Editing di un file generico

Quando si comincia ad editare un file il punto di inserimento viene mostrato come un rettangolino scuro e tutto ciò che è digitato da tastiera viene inserito nella posizione del punto di inserimento, il testo già esistente nella posizione del punto di inserimento o alla sua destra, viene spostato per lasciare posto ai nuovi caratteri digitati.

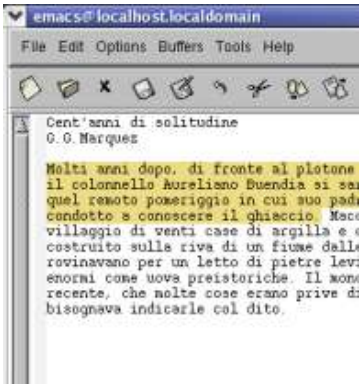


Il tasto *Ins* è un interruttore che permette di passare da *modo inserimento* (quello selezionato in automatico quando si avvia Emacs) a *modo sovrapposizione*. In questa ultima modalità, evidenziata dalla scritta *Ovwrt* (Overwrite) nella *mode line*, ogni carattere introdotto da tastiera sostituisce il carattere nella posizione del punto di inserimento.

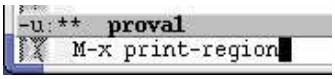
Per tornare al modo inserimento basta premere nuovamente il tasto *Ins*.

Per selezionare una parte di testo (*Region* nel linguaggio utilizzato da Emacs) si porta il punto di inserimento nel primo carattere della parte di testo interessata (basta spostare il mouse nella posizione desiderata e premere il pulsante sinistro `Mouse-1`), e, tenendo premuto il pulsante, si trascina il mouse stesso fino ad una posizione successiva a quella finale.

In alternativa, una volta portato il punto di inserimento nel primo carattere del testo da selezionare, si può premere `Mouse-3` nel punto finale della selezione che si vuole effettuare.

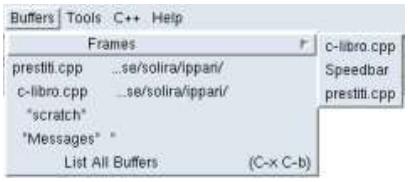


- Per cancellare il testo selezionato basta premere il tasto *Canc*
- Il testo si può spostare o copiare in una nuova posizione. Le operazioni possono essere effettuate con l'ausilio dei pulsanti della barra. Per esempio per spostare il testo in una nuova posizione, si preme il tasto *Taglia*, si sposta il punto di inserimento nella nuova posizione e si preme il tasto *Incolla*. Un modo ancora più rapido è quello di sfruttare le caratteristiche di *Taglia*, *Copia*, *Incolla* dei tasti del mouse. Quando si è selezionato il testo si è già effettuata l'operazione di Copia. Se si vuole spostare il testo basta premere *Canc*: il testo a questo punto scompare. Si sposta il mouse nella posizione del nuovo inserimento e si preme `Mouse-2`. Selezionando *Edit* > *Select and Paste*, si può scegliere, fra tutte le selezioni effettuate nella sessione di lavoro, quale incollare nel punto di inserimento attuale. Il menù *Edit* è accessibile anche mediante la combinazione `C-Mouse-3`.



- Il testo selezionato può essere mandato in stampa selezionando *File* > *Print Region*. In alternativa si può mandare in stampa direttamente la parte di testo utilizzando i comandi riconosciuti da Emacs: nel caso specifico `M-x` (fa passare al Minibuffer) e quindi digitando `print-region`. `M-x` nel minibuffer è scritto da Emacs quando si preme la combinazione di tasti (è la combinazione di tasti che fa passare al Minibuffer per l'introduzione del comando da eseguire), `print-region` invece bisogna digitarlo.

Per mandare in stampa l'intero file si può selezionare l'icona della stampante dalla barra dei pulsanti di avvio rapido.



Dall'elenco, visualizzato dal menù *Buffers*, si possono raggiungere i file caricati e, qualora esistano, i vari frames attivati. Nell'elenco sono presenti due ulteriori buffer uno utilizzato da Emacs per visualizzare i messaggi concernenti le operazioni effettuate (*Messages*), l'altro messo a disposizione per scrivere brevi note da non salvare (*scratch*).

Per rendere attivo un buffer e quindi poter editare il file inserito, si seleziona il nome dal menù.

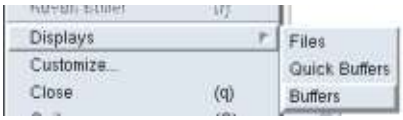
La selezione *List All Buffers* ha come effetto l'apertura di una area di lavoro con l'elenco di tutti i buffer presenti. Si può selezionare il buffer corrente, quello cioè su cui si vuole lavorare, con `Mouse-2` o con il tasto *Invio* sul nome scelto.

Il menu *Buffers*, senza però indicazione sui frames, è disponibile anche con la combinazione `C-Mouse-1`.

Se ci sono più buffer, per visualizzare nella window corrente il buffer desiderato, si può scorrere la lista anche con `Mouse-1` (buffer precedente) o `Mouse-3` (buffer successivo), sul nome del buffer

correntemente visualizzato, nella mode line.

Anche la Speedbar può essere utilizzata per navigare fra i buffer:

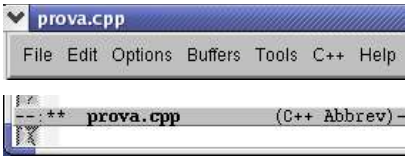


Dal menù della Speedbar (`Mouse-1` sulla mode line della Speedbar stessa), selezionando, da *Display*, l'opzione *Buffers*, si può fare in modo che la Speedbar possa essere usata per navigare fra i vari buffer presenti.

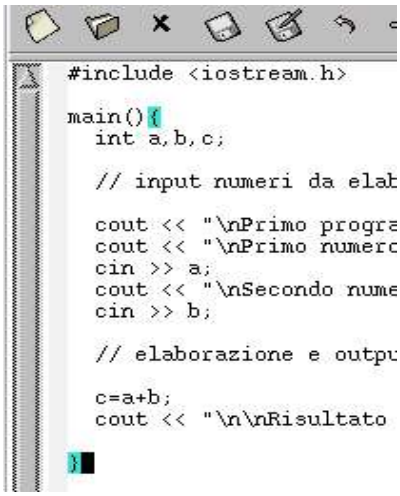
Riselezionando *Files*, il frame della Speedbar consente la navigazione fra le directory.

Editing di un sorgente C++

Essendo Emacs concepito come editor per programmatori, mette a disposizione tutta una serie di facilitazioni utili quando il file che si edita è un sorgente di un programma scritto in un determinato linguaggio di programmazione.



- ***Riconoscimento automatico dell'estensione***: quando ad un file in editazione viene assegnato un nome che termina con il suffisso `.cpp`, `.cc` o `.C` Emacs aggiunge ai menù la voce *C++* e passa alla *modalità C++*. Tutto ciò è evidenziato nella *mode line* dove compare, appunto, l'indicazione *C++*. Tale modalità attiva delle facilitazioni legate alla sintassi del linguaggio per cui, per esempio, se si seleziona *Options* > *Syntax Highlighting*, si attiva la colorazione della sintassi per cui le parole chiavi sono colorate in un modo, i commenti in un altro, le stringhe in un altro ancora e così via.



- ***Indentazione automatica e abbinamento di parentesi***: il riconoscimento del tipo di sorgente da editare, consente a Emacs di gestire le giuste indentazioni, caratteristica questa molto utile quando si scrivono programmi. Quando, per esempio, si introduce il carattere `{` e si inserisce una nuova linea, questa, appena terminata (digitazione del carattere ; o altro carattere delimitatore), viene indentata in maniera corretta così come avviene anche se il carattere che si introduce è quello di fine blocco (`}`). Se si attiva *Option* > *Paren Match Highlighting*, non appena si sposta il cursore dopo una parentesi chiusa, viene evidenziata con un colore diverso la coppia di parentesi che racchiude il blocco così come, nell'esempio, quello della funzione `main`.

L'indentazione è una facilitazione importante per la programmazione e il fatto che Emacs la metta a disposizione, gestendola in modo automatico, può mettere al riparo di alcuni errori che frequentemente vengono commessi come il dimenticare qualche parantesi o il carattere di fine linea (il `;`).

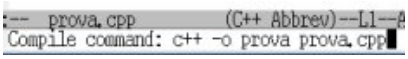
La indentazione può essere più non corrispondente se, per esempio, si cancella una struttura esistente o se ne aggiunge una nuova. Anche in questi casi Emacs mette a disposizione comandi per risolvere il problema.

- ***Indentazione singola riga***: per indentare in modo corretto (così come risulta dalle parentesi messe nel sorgente), basta posizionare il punto di inserimento in qualsiasi posizione della linea da indentare e premere il tasto *Tab*.
- ***Indentazione blocchi di linee consecutive***: si seleziona il blocco e si sceglie, dal menù *C++* l'opzione *Indent Line or Region*. Lo stesso effetto può essere ottenuto inviando il comando `M-x indent-region`. Come specificato altre volte si tratta di premere la combinazione di tasti `M-x` per far passare il punto di inserimento nel Minibuffer e digitare `indent-region` seguito dal tasto *Invio*.

Compilazione ed esecuzione

Il compilatore C++ presente in Linux è, come accennato in precedenza, contenuto nella suite GCC. Si avvia normalmente con un comando, dalla shell, ma è fornita la possibilità di avviare la compilazione di un programma all'interno di Emacs stesso.

In generale è possibile lanciare l'esecuzione di una linea di comando della shell, selezionando *Tools > Shell Command*, digitando il comando nella Echo Area e premendo *Invio*. Una nuova area di lavoro visualizzerà i risultati della esecuzione del comando.



Nel caso della compilazione è conveniente selezionare *Tools > Compile* e quindi inserire la riga di comando dal *Minibuffer*. La compilazione può essere lanciata anche digitando `M-x compile`. In ogni caso bisogna poi inserire la linea di comando.

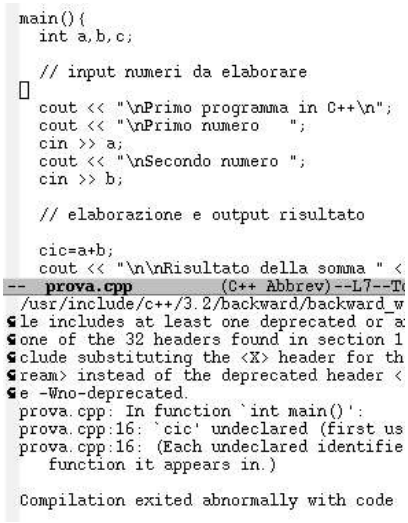
Se, come evidenziato nell'esempio riportato, il file sorgente ha il nome `prova.cpp` e si vuole che il file oggetto si chiami `prova`, la riga di comando da digitare nel *Minibuffer* è:

```
c++ -o prova prova.cpp
```

- `c++` è il nome dell'eseguibile per la compilazione di programmi C++.
- l'opzione `-o` serve per specificare il nome che dovrà avere il file oggetto. Se non specificato il nome sarà `a.out`.
- Chiude la linea il nome del file sorgente

Se la compilazione va a buon fine con il messaggio `Compilation finished`, il file oggetto è stato creato correttamente nella directory del sorgente. Si può chiudere il buffer con i messaggi della compilazione, portare il punto di inserimento nuovamente nel sorgente, premere `Mouse-2` sulla *mode line* dell'area di lavoro in modo da espandere detta area. Anche selezionando *File > Unsplit Window* si espande la window corrente (quella con il cursore di inserimento attivo) in modo da riempire l'intero frame.

Se la compilazione non è andata a buon fine, nella window dedicata ai messaggi del compilatore, viene visualizzato l'elenco dei messaggi di errore con indicazioni sulla linea del sorgente. In questo caso il messaggio visualizzato è del tipo `Compilation exited abnormally`.



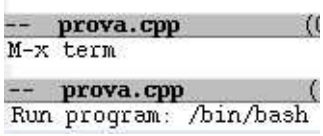
Nell'esempio viene evidenziato che alla linea 16 è usata una variabile non dichiarata, infatti nella penultima riga visualizzata del sorgente è usata una variabile `cic` non dichiarata.

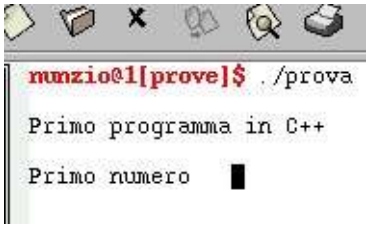
Non è necessario sapere quale è la linea 16 per andare a correggere l'errore, basta, nell'area di lavoro dei risultati della compilazione, premere `Mouse-2` nella riga dell'errore e ci si ritrova immediatamente nella riga incriminata del sorgente.

Si può andare ad una specifica linea di un testo in editazione anche selezionando *Edit* > *Go To* > *Goto Line* e scrivendo il numero della linea richiesta. La *mode line* inoltre riporta il numero di linea su cui è posizionato il punto di inserimento (`L7` dell'esempio riportato accanto).

Corretti tutti gli errori rilevati dal compilatore e salvata su disco la nuova versione del sorgente, si può procedere con una nuova compilazione.

Una volta che la compilazione del programma va a buon fine, bisogna lanciarne l'esecuzione per provare se i risultati forniti coincidono con quelli attesi. Naturalmente il lancio dell'eseguibile può essere effettuato da una shell avviata da fuori, ma può risultare più comodo effettuare tale operazione direttamente da Emacs.





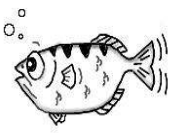
- Portandosi sulla riga del *Minibuffer* con `M-x` e digitando `term`, si richiede di lanciare l'emulazione di terminale.
- Confermando la riga di richiesta del tipo di terminale (viene proposta per default la bash)
- Viene avviata, in un nuovo buffer, l'emulazione di terminale richiesta

Si tratta di una emulazione che funziona esattamente come qualunque altra avviata esternamente. Sono, quindi, accessibili tutti i comandi e le caratteristiche della Bash (storico, completamento linea con *Tab*)

Si può eseguire il programma direttamente all'interno della shell dalla quale si possono mandare *segnali* al programma in esecuzione.

Come accennato in precedenza, Emacs, in relazione al tipo di buffer corrente, aggiunge nuove voci alla barra dei menù. Nel caso il buffer corrente contiene una emulazione di terminale, viene aggiunto il menù *Signals* dal quale, per esempio, selezionando *Signals* > *QUIT* si può interrompere l'esecuzione del programma avviato dal terminale o, selezionando *Signals* > *KILL*, si può interrompere la shell stessa.

Il debugging dei programmi



Un programma durante l'intera fase del suo sviluppo, dal sorgente alla compilazione e generazione del file oggetto, al testing, può evidenziare errori e malfunzionamenti che possono essere, orientativamente, ricondotti a due categorie:

- ***Errori rilevati in fase di compilazione***: si tratta degli errori rilevati dal compilatore e derivati da un uso errato delle parole chiavi o delle strutture sintattiche del linguaggio utilizzato per la scrittura del sorgente. Il compilatore, per vari motivi, non è in condizioni di generare il codice oggetto. Sono questi, tutto sommato, gli errori meno preoccupanti perché semplici da rintracciare e correggere: il compilatore fornisce una indicazione di massima sul tipo di errore commesso e sulla localizzazione della riga del sorgente dove è stato rilevato. Il messaggio di errore fornito può essere ambiguo e la riga indicata può essere anche la successiva a quella nella quale è localizzato effettivamente l'errore. Il compilatore genera l'errore quando non sa come tradurre una istruzione e questo può avvenire anche dopo la linea errata: in seguito all'errata interpretazione ci si aspetta qualcosa che, in effetti, non c'è e quindi l'errore viene individuato in quello che manca quando invece è quello che c'è che non va. In ogni caso, presa confidenza con i messaggi generati dal compilatore, è abbastanza agevole, in generale, risolvere l'errore.
- ***Errori rilevati in fase di run-time***: sono gli errori che vengono in evidenza quando si esegue il programma. I risultati prodotti non sono quelli attesi. Sono gli errori più insidiosi perché spesso difficili da rintracciare principalmente perché le uniche cose che il programmatore può controllare sono gli output, ma in realtà si tratta solamente dell'ultimo anello della catena: gli output sono semplicemente il risultato, visualizzato su schermo per esempio, di calcoli, che evidentemente sono errati in qualche punto, ma che non sono immediatamente controllabili.

Quello che è necessario per la correzione degli errori di run-time è avere a disposizione strumenti per il monitoraggio dell'esecuzione del programma, qualcosa che permetta, per esempio, di *vedere dentro il computer* gli effetti dell'esecuzione delle istruzioni. I debugger sono programmi che forniscono strumenti per il controllo dell'esecuzione di un programma. GDB è il debugger del progetto GNU di cui verranno esaminati i comandi di uso più comune e la sua interazione con l'ambiente Emacs.

Prima di entrare nello specifico di una *sessione di debugging*, è bene chiarire alcuni termini di uso comune.

- ***Breakpoint***: nelle sessioni di debugging si indica con questo termine il punto in cui si deve bloccare l'esecuzione del programma. Settare il breakpoint su una riga del programma comporta il blocco dell'esecuzione del programma immediatamente prima dell'esecuzione della riga.
- ***Watchpoint***: punti di osservazione delle variabili. Settare il watchpoint su una variabile o una espressione comporta il ricevere informazioni su come e quando varia il contenuto di una variabile o il valore dell'espressione.
- ***Step***: si tratta in questo caso di un modo di eseguire il programma un passo per volta. Ogni volta che vengono eseguite una o più istruzioni, a seconda del comando impartito al debugger, l'esecuzione del programma si blocca permettendo così di controllare le modifiche che la stessa ha apportato, per esempio, nei valori contenuti in una variabile.

Sessioni di debugging

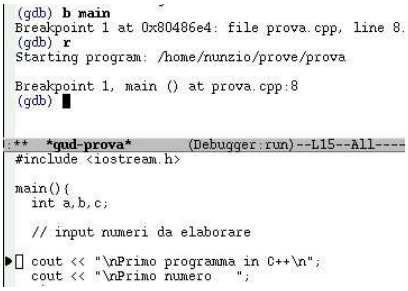
Per prima cosa se si vuole effettuare il debugging di un programma è necessario compilarlo con una opzione che permette, al compilatore stesso, di inserire nell'oggetto le informazioni utili per il debugging. Tali informazioni, infatti, poiché occupano spazio facendo aumentare le dimensioni del file oggetto, di solito non vengono aggiunte. Per aggiungere tali informazioni bisogna utilizzare il flag `-g` nella riga di comando della compilazione. In definitiva, per la compilazione del file sorgente

`prova.cpp` con le informazioni per il debugging, la linea di comando da digitare, da Emacs selezionando *Tools* > *Compile*, è:

```
c++ -g -o prova prova.cpp
```

In questo caso viene generato l'oggetto `prova` contenete le informazioni che servono al debugger.

Emacs è dotato di librerie che permettono l'interazione con i debugger. Si può avviare la sessione di debugging e, contemporaneamente, osservare l'esecuzione delle singole linee del sorgente. Per avviare la sessione di debugging si può selezionare *Tools* > *Debugger (GUD)* e confermare l'uso di `gdb`, inserito di default nel minibuffer, aggiungendo il nome dell'oggetto (`prova` nell'esempio).



- L'area di lavoro in cui è in esecuzione GDB mostra il prompt (`gdb`) ad indicare che si è in attesa di un comando.
- Con `b main` si setta un breakpoint nella prima istruzione eseguibile della funzione `main`. I breakpoint si possono settare anche specificando la linea (es: `b 8`) o, direttamente nel sorgente, con la combinazione di tasti `C-x` <*Spazio*> nella linea.
- Con `r` si avvia l'esecuzione del programma che, come si può notare nella window contenete il sorgente, viene bloccata nella prima `cout`. Una freccetta sulla sinistra dell'istruzione nel sorgente, indica la prossima istruzione che verrà eseguita.

Ogni volta che il programma in esecuzione è bloccato da un breakpoint si possono, per esempio, esaminare i valori assunti dalle variabili in quel momento.

- Si setta un watchpoint sulla variabile somma. Istruzione `watch main`
- Si ordina di continuare l'esecuzione del programma. Istruzione `c`. L'esecuzione procede fino al prossimo breakpoint o alla fine del programma
- L'esecuzione del programma viene sospesa alla linea 17 (come evidenziato nella mode line del sorgente e anche nella finestra della sessione di debugging) perché si è appena eseguita una istruzione che modifica il valore della variabile osservata
- La modifica del valore della variabile `somma` produce un breakpoint: l'esecuzione viene sospesa e viene mostrata la variazione del valore. Dal valore casuale (la variabile, precedentemente, era stata solo dichiarata) al valore `0` settato dall'istruzione precedente.

Dopo un breakpoint l'esecuzione può riprendere secondo diverse modalità a seconda dell'obiettivo che si vuole raggiungere.

Di seguito è riportata una tabella con i comandi di uso più frequente che possono essere impartiti dal prompt di GDB.

Comando	*Significato ed esempi d'uso*
`run` abbreviabile con `r`	Lancia l'esecuzione del programma specificato all'avvio. Se non ci sono breakpoint l'esecuzione termina con la fine del programma, altrimenti viene bloccata al primo breakpoint
`watch`	Permette di specificare le variabili o le espressioni da tenere sotto osservazione: 1. `watch alfa` (mostra come viene modificato il valore contenuto nella variabile `alfa` tutte le volte che viene modificato) 2. `watch alfa*100/gamma` (permette di monitorare il valore dell'espressione)
`break` abbreviabile con `b`	Fissa un punto di sospensione ad una determinata linea: 1. `break 16` (fissa un breakpoint alla linea 16 del sorgente. L'esecuzione del programma viene sospesa *prima* dell'esecuzione della istruzione scritta in tale linea) 2. `break main` (l'esecuzione del programma viene sospesa prima della prima istruzione della funzione `main`)
`print` abbreviabile con `p`	Visualizza il valore della variabile o dell'espressione specificate: 1. `print beta` (mostra l'*attuale* valore contenuto nella variabile specificata) 2. `print beta+10` (mostra il valore dell'espressione. Il valore attuale della variabile viene aumentato di `10`)
`info break`	Fornisce informazioni sui breakpoint: il numero progressivo del break e la linea del sorgente dove è posto
`info program`	Fornisce informazioni sullo stato del programma nella sessione: se è in esecuzione o no
`delete`	Elimina uno o tutti i breakpoint impostati: 1. `delete` (permette l'eliminazione di tutti i break impostati) 2. `delete 2` (elimina il break individuato dal numero `2`. Vedere `info break` per conoscere la linea del programma cui si riferisce)
`continue` abbreviabile con `c`	Continua l'esecuzione del programma dopo un breakpoint: 1. `continue` (continua l'esecuzione del programma a partire dall'istruzione contenuta nella riga del breakpoint e fino al prossimo break o alla fine del programma)
`next` abbreviabile con `n`	Esegue la prossima linea di programma. Se l'istruzione è una chiamata di funzione, viene eseguita come singola istruzione. Funzionalità utile se non si vuole entrare nella funzione e quando si tratta di chiamate a funzioni di libreria.
`step` abbreviabile con `s`	Esegue la prossima istruzione. Se l'istruzione è una chiamata ad una funzione, viene eseguita la prima istruzione della funzione.
`quit` abbreviabile con `q`	Permette l'uscita da GDB e fa terminare la sessione di debugging.

Sviluppo di progetti complessi e GNU Make

Quando lo sviluppo di un progetto software prevede l'utilizzo di più file sorgenti, emergono problematiche di gestione di tale progetto:

- L'esistenza di più file sorgenti, la dipendenza di un sorgente da altri, porta come conseguenza, ad ogni modifica, la ricompilazione di tutti i file interessati dal progetto. Risulta estremamente difficoltosa, se non impossibile, date le dipendenze che possono esistere fra sorgenti diversi, anche scrivere una procedura automatica per la ricompilazione dei soli file interessati, quando su questi sono state effettuate modifiche.
- La necessità di ricompilare tutti i file sorgenti, anche quelli non interessati dalle modifiche effettuate, può comportare tempi lunghi dipendenti dalla quantità e dimensioni dei file da ricompilare.

Per ovviare a questi inconvenienti, e velocizzare il processo di sviluppo, si può utilizzare il programma GNU Make.

Make compila i sorgenti per come sono specificati nel `Makefile` (è questo il nome di default. Naturalmente si può usare anche un altro nome, ma è convenzione diffusa lasciarlo così). Il modo in cui sono specificati i file e le dipendenze, all'interno del Makefile, permette di effettuare una *compilazione intelligente*. In pratica, mediante un confronto fra date ed orari registrati nei file oggetto e file sorgenti, Make è in grado di effettuare la compilazione soltanto dei file sorgenti che hanno un timestamp successivo a quello dei rispettivi file oggetto, e dei sorgenti che li ammettono come prerequisiti.

I concetti fondamentali su cui si basa il Makefile sono:

- ***target***. Indica l'obiettivo da raggiungere. Nel caso generale può essere il nome del file oggetto che si vuole generare
- ***prerequisites***. Indicano le cose necessarie, che devono essere verificate, per poter essere in condizioni di raggiungere l'obiettivo
- ***command***. Ammesso che tutti i prerequisiti siano verificati, indica il modo come raggiungere l'obiettivo

Makefile: un esempio pratico

Si consideri, a titolo di esempio, una semplice procedura di prestito di libri in una biblioteca, scritta in C++ utilizzando il paradigma ad oggetti. La procedura, semplificata, potrebbe essere composta da:

- il file `c-libro.h` contenete l'interfaccia della classe libro. Potrebbe, a titolo di esempio essere:

```
/*
  Definizione classe Libro con proprieta' (titolo, editore, ..)
  e metodi (registra, infoLibro, ecc...)
*/

class libro {
 public:
   void registra();
   void infoLibro();
```

```
    bool inPrestito();
    bool pEffettuato();
    bool lRitornato();
 private:
    char titolo[50];
    char autore[20];
    char editore[20];
    long int prezzo;
    bool prestato;
};
```

- il file `c-libro.cpp` contenete l'implementazione dei metodi della classe:

```
#include <iostream.h>
#include "c-libro.h"

//  Implementazione metodi della classe

void libro::registra(){
 ...
};

void libro::infoLibro(){
 ...
};
...
```

- il file `prestiti.cpp` contenete la procedura di prestiti che utilizza oggetti della classe libro:

```
// Gestione prestiti libri di una biblioteca

#include <iostream.h>
#include "c-libro.h"

int main(){
  libro biblio[10];
  int n,i,tipoop,quale;
  ...
  // Gestione prestiti libri

  for(;;){
    cout << "\n1 - prestito";
    cout << "\n2 - restituzione";
    cout << "\n3 - info libro";
    cout << "\n0 - fine";
    cout << "\nOperazione ? ";
    cin >> tipoop;
    if(!tipoop) break;
  ....
```

In questo caso il Makefile potrebbe essere:

```
# Esempio Makefile
# l'obiettivo prestiti ha due dipendenze
# riga successiva (inizia con tab) comando per la generazione obiettivo

prestiti: prestiti.o c-libro.o
        c++ -o prestiti prestiti.o c-libro.o
```

```
# obiettivo dipende da prestiti.cpp

prestiti.o: prestiti.cpp
        c++ -c prestiti.cpp

# obiettivo dipende da due file
# comando ripetuto se cambia uno dei due file da cui dipende

c-libro.o: c-libro.cpp c-libro.h
        c++ -c c-libro.cpp

# fine Makefile
```

Il flag `-c` nella compilazione di `prestiti` e `c-libro` ha l'effetto di creare soltanto i file oggetto. Ottenuti i file oggetto, questi si collegano assieme per generare l'oggetto `prestiti` (l'obiettivo principale).

A parte le righe precedute da # che sono righe di commento, ogni riga del file può assumere due formati:

- *righe con obiettivo*. La riga inizia specificando l'obiettivo, subito dopo a seguire il carattere :, si trovano i prerequisiti. Per esempio l'obiettivo finale (il primo specificato), l'ottenimento dell'oggetto `prestiti`, richiede come prerequisiti `prestiti.o` e `c-libro.o`.
- *righe con comandi*. Ammesso che i prerequisiti siano soddisfatti, nelle righe sono specificati i comandi per il raggiungimento dell'obiettivo. Nel caso presentato ogni obiettivo è seguito da una sola riga di comando, ma potrebbero essere più di una. Tutte le righe di comando cominciano con il *Tab*. È la presenza di questo carattere che identifica la riga come riga di comando.

La compilazione si avvia, al solito, scegliendo *Compile...* dal menù *Tools* di Emacs, ma specificando nel minibuffer, stavolta, `make`. In questo caso non è necessario specificare altro: il makefile è conservato con il nome di default `Makefile`.

Se non è mai stata avviata la compilazione, questa, in coerenza con i dettami del makefile, procederà dall'obiettivo `prestiti.o`, all'obiettivo `c-libro.o`, per arrivare, infine, all'obiettivo `prestiti` compilando i programmi necessari.

Se viene modificato, per esempio il file `c-libro.cpp`, una successiva esecuzione del make avvierà la compilazione di `c-libro.cpp` e `prestiti` poiché quest'ultimo lo ha fra i suoi prerequisiti, ma non avvierà la compilazione di `prestiti.cpp` perché il timestamp del file sorgente `prestiti.cpp` e del file `prestiti.o` coincidono, non c'è stata alcuna modifica del sorgente rispetto all'ultima compilazione effettuata e, quindi, non è necessaria alcuna ricompilazione.

Flessibilità del *Makefile*: esempio di uso delle variabili

La capacità di GNU Make di usare e gestire, nelle righe di comando, variabili che possono anche fare riferimento a variabili di ambiente, aumenta notevolmente la flessibilità e la velocità di gestione dei progetti, che consente il Makefile.

A titolo di esempio si vedrà come si possano controllare i flag di compilazione di un progetto, per aggiungere o togliere informazioni per il debugging, senza essere costretti a modificare ogni volta il Makefile, ma, molto più semplicemente, invocando make con parametri diversi.

- Per prima cosa si modificano, nel Makefile, le righe di comando, in modo da poter utilizzare la variabile che serve:

```
...
c++ $(CFLAGS) -o prestiti prestiti.o c-libro.o
...
c++ $(CFLAGS) -c prestiti.cpp
...
c++ $(CFLAGS) -c c-libro.cpp
...
```

  le variabili hanno un nome scelto dal programmatore, sono inserite fra parentesi e sono precedute dal simbolo `$`. Nell'esempio proposto controllando il contenuto della variabile, si è in grado di aggiungere il flag per il debugging. Il nome `CFLAGS` per la variabile è quello tradizionalmente utilizzato per specificare i flag di compilazione.

- Si aggiunge, alla fine del Makefile, un nuovo obiettivo:

```
...
clean:
        rm c-libro.o
        rm prestiti.o

# fine Makefile
```

  il raggiungimento di questo obiettivo comporta la rimozione dei file oggetto generati dalla compilazione (il flag `-c`). In questo modo, richiamando l'obiettivo, si ha la possibilità, cancellando gli oggetti, di ricompilare il progetto utilizzando flag diversi anche senza la necessità di modifica dei sorgenti. Si ricorda che se il timestamp non è modificato, i sorgenti non vengono ricompilati.

  Il comando `make`, utilizzato da solo, si occupa del raggiungimento dell'obiettivo principale. Se invece viene invocato seguito dal nome di un obiettivo, salta direttamente a quello specificato. L'invocazione `make clean`, non essendoci prerequisiti, lancia l'esecuzione dei due comandi di rimozione associati all'obiettivo, dopodiché, make, termina il proprio lavoro.

Lanciando normalmente il comando `make`, viene eseguita una compilazione come se `$(CFLAGS)` non ci fosse. Ed, effettivamente, la variabile, non essendo definita né nel Makefile né nelle variabili di ambiente, non verrà sostituita da alcun valore.

Se, invece, si invoca make con il comando `make CFLAGS=-g`, si fa in modo di passare a make il valore della variabile indicata come opzione. Il nuovo valore sostituirà qualunque altro valore della stessa variabile sia definito nel Makefile (nel caso in esempio non esiste alcun valore da sostituire) e verrà avviata la compilazione con il flag delle opzioni di debugging.