Compilatori

Per Linux esistono molti compilatori C, C_{++} e $FORTRAN$. Il compilatore C_{++} si chiama g_{++} e per compilare il programma prog.c bisogna dare il comando

g_{++} prog.c

che crea il file eseguibile a.out

È utile conoscere alcune opzioni di g_{++}

-o nome	stabilisce il nome del file eseguibile (a.out se non è specificato)
-c	compila senza linkare
-Wall	avverte per qualunque errore vero o presunto nel programma
-O1, -O2, -O3	ottimizza sempre di più
-O0	non ottimizza affatto (utile per il debugging)
-lm	linka con la libreria libm per linkare con libgsl si usa -lgsl
-g	attiva il debugger.
-p	attiva il profiler.

Esempio:

$g_{++}\ -o\ prog\ -O2\ -Wall\ -lm\ main.c\ integra.c$

Ambienti di sviluppo integrato

- Nello sviluppare un programma come pure nel correggerlo, è abbastanza scomodo ripetere molte volte l'operazione di editare il programma, compilarlo, eseguirlo e fare nuove correzioni.

- Inoltre, se il programma è grande e il file eseguibile è compilato a partire da più file sorgente in cartelle diverse, diventa estremamente complicato tener traccia di tutto.

- Il secondo problema non riguarda questo corso, ma il primo sì.

Esistono programmi che hanno la capacità di editare, compilare ed eseguire un programma premendo pochi tasti: i più noti in Linux sono Kdevelop e Anjuta, tuttavia, dato che sono piuttosto complicati da usare, preferisco illustrare Geany

Geany

Geany ha il vantaggio di poter essere usato per compilare un singolo file senza creare un progetto, anche se questo pone delle limitazioni

- Se $prog.c$ è il programma che vogliamo modificare lanciamo il comando $geany\ prog.c$ oppure semplicemente $geany$ scegliendo poi $File- > Open$ selezionando quindi $prog.c$

- A questo punto avremo una finestra con l'editor nella parte alta, mentre in basso comparirà l'output del compilatore

- Prima di compilare devo configurare compilatore e linker scegliendo
  $Build- > Set\ build\ Commands$

- Qui bisogna segnalare al programma i comandi per compilare soltanto (*compile*), per creare un file eseguibile (*build*) e per eseguirlo (*execute*);

- Dove è scritto "%f " il geany sostituirà il nome del file che stiamo editando, dove è scritto "%e " il nome del file senza estensione, di solito usato come nome per l'eseguibile

- Si possono aggiungere, nei primi due casi, opzioni per il compilatore e per il linker

- Una opzione utile è $-Wall$, che aggiunge molte warning agli avvisi della compilazione. Un'altra è $-On$ con $n = 1, 2, 3$ che ottimizza sempre di più

- Se tutto è stato fatto a dovere basta ora premere F8 per compilare, F9 per compilare e linkare, F5 per eseguire

Debugging

- Se un programma compila non è detto che funzioni
- se un programma funziona non è detto che faccia quello che avete in mente;
- se un programma può essere eseguito, anche terminando prematuramente, si può usare un debugger;
- si possono monitorare le variabili e vedere il contenuto degli array;
- i più bravi possono guardare il contenuto dei registri;
- l'eseguibile deve contenere le informazioni sul sorgente;
- l'eseguibile deve poter andare in modo da potersi fermare in corrispondenza ad ogni riga del sorgente;
- Per accedere al debugging si compila con

$$g++ \ -g \ -O0 \text{ program.c}$$

Data Display Debugger ddd

- Scegliere File → Open Program e selezionare l'eseguibile;
- compare una finestra con il sorgente (Source Window) ed un'altra con i comandi del debugger (Gdb Window);
- Run fa andare il programma dall'inizio alla fine;
- il programma si ferma ai breakpoint;
- Step va alla prossima linea del programma entrando in ogni funzione;
- Next fa lo stesso ma non entra nelle funzioni;
- Finish esce dalla funzione dove siete;
- Cont continua fino alla fine o fino al prossimo breakpoint;
- Until va alla linea del sorgente che sta più in basso di quella attuale (utile per uscire dai cicli)

Vedere le variabili

- Le variabili sono mostrate in un'appostita finestra (Data Window)
- Scegliere Data $\rightarrow$ Display Local Variables;
- puntare il mouse su di una variabile e tenercelo ne mostra il valore;
- facendo click col tasto destro del mouse su di una variabile si può vederne il valore in permanenza in una finestra scegliendo display variabile;
- nella finestra del debugger sotto al sorgente scrivere l'istruzione p(rint) i oppure p(rint) a[20] etc.
- per vedere vari elementi dell'array v bisogna invece portarsi nella finestra di gdb e scrivere, ad esempio graph display v[11..21] per vedere gli elementi che vanno da 11 a 21;

Profiling

- Serve a misurare la velocità di esecuzione delle singole funzioni del programma;

- si deve compilare in questo modo

  gcc -pg sorgente.c

- il programma deve essere poi eseguito e quindi si può dare il comando

  gprof [-b] eseguibile (| less)

- si ottiene una statistica del tempo speso in ogni funzione, e il numero di volte che ogni funzione è stata chiamata;

- il profiler non corregge i programmi, serve solo a renderli più veloci;

- guardare in quali funzioni un programma passa più tempo;

- concentrarsi su quelle chiamate più volte: anche un piccolo guadagno di tempo verrà moltiplicato per un fattore grande;

- è sempre buona norma avere inizialmente un programma funzionante che produca risultati esatti; in seguito ogni modifica deve portare a risultati compatibili.

Output di gprof

La parte importante dell'output di gprof è costituita dalla prima tabella: nelle colonne ci sono, nell'ordine:

- Percentuale di tempo impiegato da una funzione;
- tempo totale impiegato, comprese le chiamate a sottofunzioni;
- tempo totale, senza le chiamate a sottofunzioni;
- numero di volte che la funzione è stata chiamata;
- tempo impiegato per ogni chiamata, senza chiamate alle sottofunzioni;
- tempo totale impiegato, incluse le chiamate a sottofunzioni.

Valgrind

Valgrind è un programma che permette di verificare il corretto uso della memoria da parte di un programma

- può segnalare errori di accesso agli array dinamici: ad esempio, se abbiamo definito un array a = nev double [100] e poi cerchiamo di accedere ad a[100] facciamo un errore che il compilatore di solito non rivela;
- devo compilare il programma da correggere con le opzioni di debugging, e cioè -g -O0
- nell' eseguire il programma a.out lancio il programma valgrind a.out e questo mi segnalerà accessi scorretti alla memoria.
- posso usare valgrind anche per studiare gli accessi alla cache cercando di migliorare i tempi di esecuzione. In questo caso il comando da usare è valgrind –tool=cachegrind a.out