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

```
gcc -g -O0 program.c
```

Metodi tradizionali

Un sitema semplice è quello di stampare il valore di una variabile (o anche solo il numero di linea) in vari punti del programma con

printf(“formato” ,variabile);

Il tutto si può rendere più sistematico definendo la macro TRACE

#define TRACE {printf(“__LINE__”);}

- è come dire “sono arrivato fin qui”
- l’istruzione però non viene eseguita subito
- si può forzare l’istruzione con fflush(stdout)

Definisco allora meglio

#define TRACE {printf(“__LINE__”); fflush(stdout);}

Si tratta comunque di metodi piuttosto rudimentali

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 → 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 allocato 100 posti per numeri in doppia precisione per double a 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