Debugging

Prof. Ivan Lanese

Di solito i programmi non funzionano

```
int main()
{
int n, prod = 1;
do
     {
      printf("Inserire un numero (0 per uscire):");
      scanf(“%d”,&n);
      prod = prod * n;
     }
while (n != 0);
printf("Il prodotto è %d\n",prod);
return 0;
}
```

Che cosa vuol dire che un programma funziona?

- Vuol dire che in ogni condizione restituisce il valore atteso
- Un programma che esegue all'infinito senza interagire con l'utente non funziona
- Un programma che si pianta se l'utente inserisce valori sbagliati non funziona
  - In questo caso il programma deve restituire un opportuno messaggio di errore
- Un programma che "ogni tanto" si pianta o restituisce dati sbagliati non funziona
  - La frase "L'avevo provato e funzionava tutto" non è una giustificazione valida

Testing e verifica

- Quando fate un programma dovete fare testing
- Testing: attività per valutare la qualità del software
- Può portare a individuare (e poi correggere) bug
- Non può assicurare la correttezza del software
  - Non è possibile provare il programma in tutte le condizioni
  - Sono tipicamente un numero molto alto
- Per assicurare la correttezza è necessario fare verifica
  - Molto complessa e costosa
  - Viene effettuata per sistemi safety-critical

3 categorie di errori

- Ci concentriamo nella ricerca di errori funzionali
  - Il programma non fa quello che deve fare
  - Problemi non funzionali (performance, scalabilità, usabilità, ...) richiedono competenze che ancora non avete
- Esistono 3 categorie di errori funzionali
  - Errori di compilazione
  - Errori a runtime
  - Errori logici

Errori di compilazione

- Esempi: manca un punto e virgola, variabile non dichiarata, …
- Notificati dal compilatore
  - Leggete il messaggio di errore!
- Semplice rendersi conto dell'esistenza dell'errore
- Non sempre semplici da correggere
  - A volte il messaggio di errore non è preciso
  - A volte il messaggio di errore si riferisce alla riga sbagliata
  - Non possiamo eseguire il programma per avere ulteriori informazioni

Errori a runtime

- Esempio: divisione per 0, segmentation fault
- Si verificano durante l'esecuzione e ne causano l'interruzione
- Non necessariamente in tutte le esecuzioni
- Difficili da individuare
  - In quali circostanze si verifica l'errore?
  - Quale riga del programma ha causato l'errore?
- Il segmentation fault si verifica quando il programma va a leggere un'area di memoria non sua
  - Controllate se accedete a un puntatore NULL

Errori logici

- Il programma esegue normalmente ma il risultato non è quello atteso
- Esempio: il programma restituisce sempre 0
- Non semplice rendersi conto dell'esistenza dell'errore
  - Il risultato potrebbe essere simile a quello atteso
  - Il risultato atteso potrebbe non essere noto
  - L'errore potrebbe presentarsi solo in certe esecuzioni
- Molto difficili da individuare
  - In quali circostanze si verifica l'errore?
  - Quali risultati parziali sono giusti e quali sbagliati?

Tecniche di debugging: debugging “a mano”

- Si inseriscono delle stampe ausiliarie per capire
  - il flusso dell'esecuzione
    - stampando “sono passato di qui”
  - gli stati intermedi
    - stampando il valore delle variabili
- Non richiede supporto dai tool
- Richiede attenzione a non introdurre ulteriori errori
  - Potrebbe essere necessario aggiungere parentesi
- Un po' artigianale

Tecniche di debugging: debugger

- Richiede l'uso di un tool chiamato debugger
  - Tipicamente integrato nell'ambiente di sviluppo
- Consente di verificare
  - il flusso dell'esecuzione: tracing, breakpoints
  - gli stati intermedi
- Molto potente
- Noi vedremo GDB

GDB

- The GNU Project Debugger
- Si può usare da riga di comando
  - … ma non è comodissimo
- Integrato sia in Eclipse che in Code::Blocks
  - Accessibile da appositi menu/bottoni
  - Le interfacce fornite dai 2 ambienti sono leggermente diverse
  - Ci concentreremo su quella Eclipse, trovate sulle slide informazioni anche su quella Code::Blocks

Debugging in Eclipse: tracing

- Lanciate il programma con Debug dal menu Run
  - Passate alla prospettiva Debug
- A destra potete vedere i valori delle variabili e i breakpoints definiti
  - Potete modificare i valori delle variabili per vedere cosa succede
- Eseguite il programma passo-passo con Step into del menu Run
- Potete far completare l'esecuzione con Resume del menu Run

Debugging in Code::Blocks: tracing

- Assicuratevi di aver compilato la versione di debug
  - Da Project, Build options selezionate Produce debugging symbols
- Eseguite il programma passo-passo con Step into dal menu Debug
  - Passate alla prospettiva Debug
- Potete aprire la finestra per vedere i valori delle espressioni
  - Da Debug -> Debugging windows -> Watches
- Potete far completare l'esecuzione con Start/Continue del menu Debug

Debugging in Eclipse: breakpoints

- Potete definire un breakpoint con Toggle breakpoint (cliccando col tasto destro sulla barra a sinistra della linea desiderata)
  - Sospende l'esecuzione quando viene raggiunta la riga in cui si trova
  - Se il programma è eseguito in modalità Debug (tramite Resume)
- Potete gestire i vostri breakpoint nella finestra a destra
  - Abilitarli/disabilitarli
  - Eliminarli
  - Aggiungere informazioni di log
    - Da Breakpoint properties…
    - Create azioni
    - Fate Attach e poi Apply
    - Informazioni visibili nella console

Debugging in Code::Blocks: breakpoints

- Potete definire un breakpoint con Toggle breakpoint
  - Sospende l'esecuzione quando viene raggiunta la riga in cui si trova
  - Se il programma è eseguito in modalità Debug
- Potete gestire i vostri breakpoint aprendo la finestra Debug -> Debugging windows -> breakpoints
  - Abilitarli/disabilitarli
  - Cambiarne le proprietà

Errori classici

- Alcuni errori classici vengono individuati dal compilatore e segnalati come warning
  - Assegnamento al posto di una condizione
  - Variabili non usate (non in tutte le configurazioni)
  - ...
- Obi-Wan (off by one)
  - Quando un ciclo viene eseguito una volta in più o in meno di quanto serve
- Errore nei casi limite
  - Nei programmi che lavorano su numeri controllate sempre cosa succede se un numero è 0
- Un buon modo per evitare errori è scrivere codice "pulito"

Debugging di funzioni

- I programmi reali sono composti da funzioni
- Per effettuarne il debugging conviene prima verificare il funzionamento delle singole funzioni (divide et impera)
  - Chiamato unit testing
  - Definite un semplice main che consente di testarle con vari parametri
  - Testate anche i casi "inaspettati"
- Quando siete confidenti nella correttezza delle singole funzioni potete testare il programma nel suo complesso
  - Chiamato integration testing
- Potete eseguire le funzioni in un passo unico con Step over (Step out in Code::Blocks)
  - Si ferma comunque in presenza di breakpoints

Bonus stage: debugging reversibile in Eclipse

- Eclipse consente di usare il debugging reversibile
  - Code::Blocks no
- Si registra un'esecuzione e poi ci si può spostare avanti e indietro lungo di essa
- Utile se si va troppo avanti in un'esecuzione di debugging
  - Ad esempio, se un breakpoint è troppo avanti
- Bisogna abilitare il debugging reversibile
  - Andare sul menu Window e selezionare Customize perspective
  - Scegliere il tab Action Set Availability e selezionare Reverse debugging
  - Compare un bottone per abilitare/disabilitare il reversible debugging
  - Quando il bottone è premuto eseguendo registrate la storia dell'esecuzione
    - L'esecuzione è molto più lenta
- Ora avete a disposizione i comandi di tracing anche all'indietro