Debugging

UVic SEng 265

Daniel M. German

Department of Computer Science

University of Victoria

November 20, 2002 Version: 1.00

dmgerman@uvic.ca

Debugging

- ”If debugging is the art of removing bugs, then programming must be the art of inserting them.” Anonymous
- Bug: b A defect or fault in a machine, plan, or the like. orig. U.S. 1889 Pall Mall Gaz. 11 Mar. 1/1 Mr. Edison, I was informed, had been up the two previous nights discovering ‘a bug’ in his phonograph—an expression for solving a difficulty, and implying that some imaginary insect has secreted itself inside and is causing all the trouble. OED

dmgerman@uvic.ca

Coding & Debugging

- Errors are unavoidable
- Good programmers know that they spend as much time debugging as writing code
- They tend to learn from mistakes
- And they try to **avoid** mistakes
- Debugging is hard, takes time, and we hate it

dmgerman@uvic.ca

Techniques to Reduce Debugging

- Good Design
- Good Style
- Boundary Condition Tests
- Assertions
- Sanity Checks in the Code
- Defensive Programming
- Limited Global Data
- Checking tools

dmgerman@uvic.ca

The Role of the Language

- Some languages try to prevent bugs:
  - Range checking on subscripts
  - No pointers at all
  - Garbage Collection
  - Strong type checking
- No language prevents you from creating errors
- Programmers should know the risks of their language
- Enable all warnings

dmgerman@uvic.ca

The Debugger

- Very useful tool
- Lets you run a program line by line
- We can inspect the value of variables and the stack at any point
- I am tired of teaching programming/OS details, so let's talk about SEng instead: we will talk about how to debug, without a debugger

dmgerman@uvic.ca

Good Clues, Easy Bugs

- Don’t blame the compiler, library or OS!
- Bugs are our fault (or worse, somebody else’s when working in teams)
- Look at the debugging output before the crash
- Get a stack trace of the crash (so you know where it crashed)
- Debugging is like solving a murder
  - Something improbable happened
  - We have the evidence
  - Solve it **backwards**
- Once we know what happened, it is easy to fix

dmgerman@uvic.ca

Good Clues, Easy Bugs...

- Some strategies:
  - Look for familiar patterns: ask yourself if you have seen this pattern before.
  - Examine the most recent change: What was the last change? If you test frequently, you can catch the change that prompted the bug. Make frequent backups/commits so you can make diffs.
  - Don’t make the same mistake twice: learn from your mistakes! That is why experienced programmers earn more.
  - Debug it now, not later. Don’t wait for another bug to make your debugging more miserable.

dmgerman@uvic.ca

Good Clues, Easy Bugs...

- More strategies:
  - Get a stack trace. Use the debugger to examine the location of the error and the contents of the stack at the time of failure.
  - Read Before typing. Understand the code before you make the changes.
  - Explain it to somebody else (get a personal *friend*, e.g. Kernighan's teddy bear).

dmgerman@uvic.ca

No clues, hard bugs

- Sometimes you have no idea what is going on
- Strategies:
  - Make the bug reproducible. Try to make it appear when you want and not randomly. Not always possible, but worth trying.
  - Divide and conquer. Can you make the input that creates the problem smaller or more focused? Use binary search.
  - *Numerology* of failures. Are there any numeric patterns in the error?
  - Add debugging messages to help you track the behavior

dmgerman@uvic.ca

No clues, hard bugs

- More strategies:
  - Write self checking code. Add code that verifies that the internal state of your variables has not been corrupted (assertions).
  - Write a log file. Print output to a log file that you can inspect after the program crashes.
  - Draw a picture.
  - Use tools: diff, grep, shell scripts...
  - Keep records: what have you tried?

The Debugger

- A powerful tool but often misused.
- Sometimes your mental model is just plain wrong
  - You don’t quite understand the intricacies of the language you are using
  - Or the error is a subtle typo
  - Sometimes you passed the parameters in the wrong order
- The debugger might help you to find these errors by stepping through your code
- But remember, the debugger does not replace your brain

dmgerman@uvic.ca

Non-reproducible Bugs

- Most difficult and challenging (I call them “unreliable bugs”)
- The fact they are non-deterministic is already information about them
- Strategies
  - Check for uninitialized memory: you might be picking a random value.
  - If your code disappears when you are using debugging code (or the debugger) your problem is most likely memory allocation
  - If the crash (symptom) is far away from the potential cause then it might be a dangling pointer (using memory that you don’t own any more) or using an array beyond its boundaries

Non-reproducible Bugs...

- More strategies
  - Sometimes a program fails with one person, and not with other. Works in one computer, not in another: check the environment (Environment variables, permissions, OS version, etc)
  - Some tools available to keep track of memory allocation (dmalloc library, for example)

Other People's Bugs

- You will have to fix somebody else's bugs
- Everything we have said still applies
- First you should understand the code you are debugging (discovery)
- Use tools for understanding code (cross referencing tools, code comprehension utilities)
- Use the debugger to understand the programmer's solution
- Analyze the log of revisions
- If the problem is in code you can't fix, prepare a good report of your bug diagnostic. Remember, your reputation is on the line

Conclusions

- Debugging can be fun: solving puzzles
- It is an art that we will practice regularly
- AVOID inserting bugs
- Hard thinking is the best approach
- Add debugging code (printing output, defensive code)
- Explain it to somebody else (K' teddy bear)
- Use a debugger
- Step through your code
- Know yourself

Conclusions

- Use the scientific method:
  - Gather information
  - Formulate a hypothesis
  - Create an experiment to verify your hypothesis
  - Test it (modify code and run it)
  - Did it not pass? Repeat.
- Remember, it is important that you celebrate when you finally squash it!

dmgerman@uvic.ca