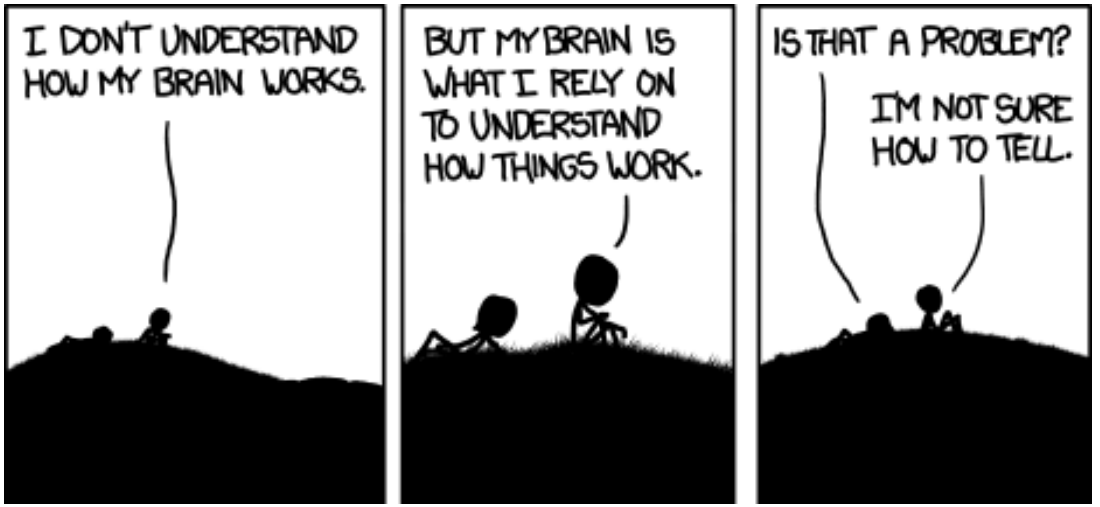
CS 261
Fall 2016

Mike Lam, Professor



Debugging

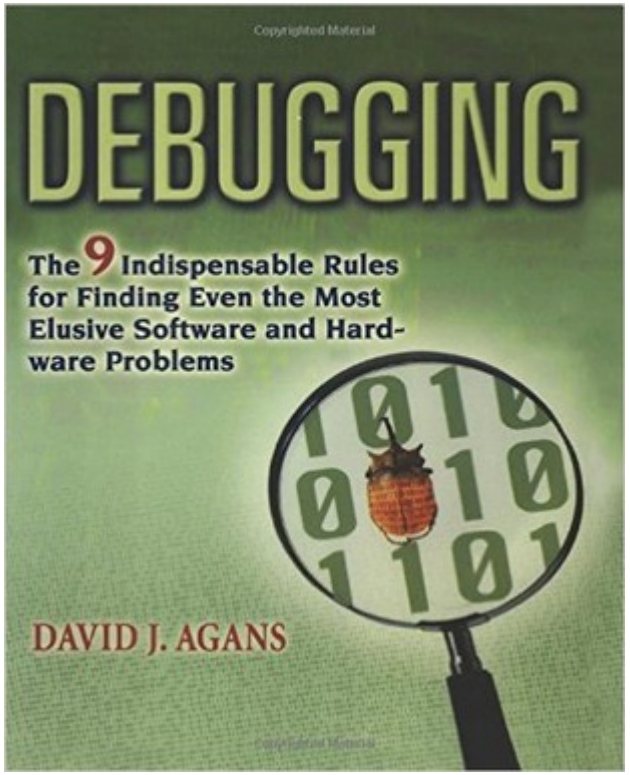
Debugging

- “It’s 2am and I just wrote 500 lines of code!”
  - “All the functions are there.”
  - “I’m done now, right?
- “I should probably run some tests”
  - “Just to be sure...”
- “@#$%, it’s not working!”
  - “But it **looks** like it should work...”

Debugging

- A software defect is an error in code that produces incorrect or undesired behavior
  - Colloquially called “bugs”
  - Many types: syntax, logic, integration, concurrency
  - Many causes: typos, incorrect code, design flaws, ambiguous spec
- Fundamental issue: mismatches between user’s expectations and machine’s behavior
  - Proximate cause (symptom) vs. root cause (defect)
  - Debugging is the process of starting from the former and working towards discovering the latter
  - Basically: the process of continually asking “why is this happening?”
  - One of the most important practical skills in programming

9 Rules of Debugging



Recommended book
ISBN-13: 978-0814474570

1) **Understand the system**
2) **Make it fail**
3) **Quit guessing and look**
4) **Divide and conquer**
5) **Change one thing at a time**
6) **Keep an audit trail**
7) **Check the obvious**
8) **Get a fresh view**
9) **If you didn't fix it, it ain't fixed**

Author: David Agans

www.debuggingrules.com

1) Understand the system

- Read The F!@#'ing Manual! (**RTFM**)
  - All of it! (or at least all of the non-reference parts)
- Become familiar with the system
  - What does normal operation look like?
  - What tools are available?
  - Where can you find more details if you need them?

2) Make it fail

- Reasons to induce failures:
  - To examine the system more closely
  - To allow you to focus on debugging
  - To know when you've fixed the bug
- Reproducible reports: small, self-contained examples that illustrate the errant behavior
- Watch for intermittent / non-deterministic errors
- Watch for "impossible" errors
  - "When you have eliminated the impossible, whatever remains (however improbable) must be the truth." - Sherlock Holmes
- Never throw away a debugging tool

3) Quit guessing and look

- Make sure you are seeing the actual failure
  - Including all the details—dig deep!
  - Know your tools: printf, debuggers, profilers, etc.
- Add instrumentation to the system
  - Even better: design the instrumentation into the system
  - Remember that your instrumentation affects the system
  - Beware of “heisenbugs” that disappear when you try to examine them
- Guess at the root cause if you must, but make sure you check your guesses

4) Divide and conquer

- Iteratively narrow the search space
  - Do an experiment that will eliminate a large number of causes, then do another experiment to narrow the field even further
  - Use comments or (in C) #ifdefs to selectively disable code
  - Sometimes version control can help (“git bisect”)
- Fix the bugs you’re aware of
  - Don’t let them mask other problems or interfere with debugging
  - If there is “noise” in the system, fix that first

5) Change one thing at a time

- Change things carefully and deliberately
  - Don’t just randomly start changing things!
  - Weapon analogy: use a sniper rifle, not a shotgun
  - Change something; test it before changing something else
  - Compare “bad” behavior with existing “good” behavior
  - Think about what has changed since the last time it worked

6) Keep an audit trail

- Keep notes on what you did and what happened
  - Some of the details are important, but not all of them; learn what to keep and what to ignore
  - Leave the important details in comments
- Try to correlate debugging information with observations or other information
  - Does the problem only reproduce under certain circumstances?
- Considering incorporating debugging test cases into your regular test suite
  - For when you break the same thing again in the future

7) Check the obvious

- Question your assumptions
  - Ask your colleagues if you need a sanity check
  - *In this class: ask on Piazza!*
- Start over from the beginning
  - Make sure initialization is happening the way you think it is
  - Check every step along the way
- Test and calibrate your tools
  - You can’t use them to find errors if they’re errant themselves

8) Get a fresh view

- Don’t be too proud to ask for help
- Report the symptoms, not your theories
- Don’t insist it’s not your fault
- Admit your uncertainties
- Refrain from complaining about well-known unresolved issues until you’ve fixed your bug
  - You may find they’re unrelated

9) If you didn’t fix it, it ain’t fixed

- Make sure you fixed it
  - Corollary: make sure that **your fix** is what fixed it!
  - Be **very suspicious** when a problem “just goes away” by itself—it is probably still there but is now hidden
- Once you find the problem, fix the root cause
  - If you’ve exposed a system design flaw, fix that too (or alert someone who can)
  - Test, test, and test again
  - Remember that your “fix” could have broken other things

Debuggers

- A debugger is a program that allows you to examine another program while it is running
  - Execute the program step-by-step
  - Examine the contents of memory at any point
  - Add breakpoints and watchpoints
  - Reverse execution to find the root cause
- Debuggers are more useful with extra information from the compiler
  - In gcc, compile with the "`-g`" option to enable this
  - It's also useful to disable optimization ("`-O0`")

GDB (GNU debugger)

- Basic commands (launch with "`gdb <exename>`")
  - `run <args>` - begin execution
  - `start` - run and stop at beginning of main()
  - `break <file:lineno>` - stop at given location
  - `print` / `p <expr>` - print the current value of an expression
  - `watch <var>` - stop the next time the given variable changes
  - `next` / `n` - continue to next line of code
  - `continue` / `c` - continue to next breakpoint
  - `step` / `s` - step into function
  - `finish` - continue to end of function
  - `backtrace` / `bt` - show all functions on the call stack
  - `up` / `down` - navigate through functions on the stack
  - `quit` - exit the debugger

GDB’s Text User Interface (TUI)

- Combined source/debug interface
  - Use `CTRL-X 1` to enter TUI mode with source only
  - Use `CTRL-X 2` to enter TUI mode with source and assembly
  - Use `CTRL-L` to refresh the screen if it glitches
  - Use `CTRL-X a` to exit

GDB quick reference guide

GNU Debugger

Basic GDB commands

Starting GDB

`gdb`
Starting GDB with no debugging files

`gdb [program]`
Begin debugging program

`gdb --args [program] [argument(s)]`
Begin debugging program and pass argument(s)

`gdb --pid [program] [process]`
Begin debugging program and attach to process

`gdb [program] [core]`
Debug coredump core produced by program

`gdb --help`
Describe command line options

Working Files

`file [file]`
Use file for both symbols and executables; with no arg, discard both

`core [file]`
Read file as coredump; with no arg, discard

`exec [file]`
Use file as executable only; with no arg, discard

`symbol [file]`
Use symbol table from file; with no arg, discard

`load [file]`
Dynamically link file and add its symbols

Stopping GDB

`quit / q`
Exit GDB

Breakpoints & Watchpoints

`break`
`b`
Set breakpoint at next instruction

`break [file:][line]`
Set breakpoint at line number in file

`break [file:][func]`
Set breakpoint at func in file

`break [+offset]`
`break [-offset]`
Set break at offset lines from current stop

`watch [expr]`
Set a watchpoint for expression expr

`catch [event]`
Break at event, which may be catch, throw, exec, fork, vfork, load or unload

`clear`
Delete breakpoints at next instruction

`delete [n]`
Delete breakpoints; or breakpoint n

Getting Help

`help`
List classes of commands

`help [class]`
One-line description for commands in class

`help [command]`
Describe command

Program Stack

`backtrace [n]`
`bt [n]`
Print trace of all frames in stack; or of n frames

`frame [n]`
Select frame number n; or frame at address n

`up [n]`
Select frame n frames up

`down [n]`
Select frame n frames down

`info frames [addr]`
Describe selected frame; or frame at addr

Miscellaneous

`print [expr]`
Show value of expr

`show copying`
Display GNU general public license

Executing Your Program

`run`
Start your program with current argument list

`run [arglist]`
Start your program with arglist

`kill`
Kill running program

Execution Control

`continue [count]`
`c [count]`
Continue running; if count specified, ignore this breakpoint next count times

`step [count]`
`s [count]`
Execute until another line reached; repeat count times if specified

`next [count]`
`n [count]`
Execute next line, including any function calls

`jump [line]`
Resume execution at specified line number

http://users.ece.utexas.edu/~adnan/gdb-refcard.pdf