DEBUGGING

CS2110

Announcements

2

- Lunch with Professors – sign up, including today!
- Prelim 2 Regrade Requests due tonight at 11:59PM

Context

3

- 2110 teaches you how to write code with care. Use:
  - meaningful comments
  - meaningful variable names
  - loop invariants
  - preconditions
  - asserts
  - testing (**lots and lots** of testing!)
  - clean style
  - a structure that is easy to reason about

What *not* to do: https://www.ioccc.org

Correctness first, *then Performance*

4

"Programmers waste enormous amounts of time thinking about, or worrying about, the speed of noncritical parts of their programs, and these attempts at efficiency actually have a strong negative impact when debugging and maintenance are considered. We should forget about small efficiencies, say about 97% of the time: **premature optimization is the root of all evil**. Yet we should not pass up our opportunities in that critical 3%."

—Donald Knuth

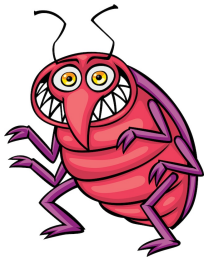
Correctness first, then speed.

If speed *really* matters, use a profiler.

Sometimes…

5

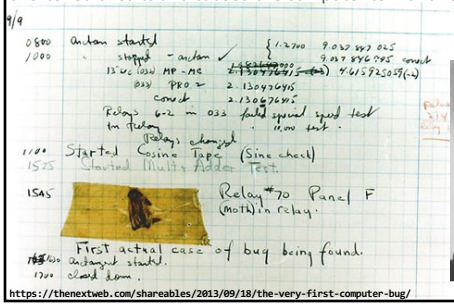
…despite your best efforts, your code will not work



Now what?

The 1st Bug

On September 9, 1947, U.S. Navy officers found a moth between the relays on the Harvard Mark II computer they were working on. In those days computers filled rooms and the warmth of the internal components attracted moths, flies and other flying creatures. Those creatures then shortened circuits and caused the computer to malfunction.



https://thenextweb.com/shareables/2013/09/18/the-very-first-computer-bug/

Grace Hopper

Debugging is twice as hard as writing the code in the first place. Therefore, if you write the code as cleverly as possible, you are, by definition, not smart enough to debug it. —BRIAN KERNIGHAN

If debugging is the process of removing software bugs, then programming must be the process of putting them in. —E. W. DIJKSTRA

Deleted code is debugged code. —JEFF SICKEL

7

Debugging 101

8

Step 1: know what the correct behavior is

Step 2: find a single, reproducible* case where your code is incorrect

Step 3: figure out **what your code is doing**

NOT: why your code *isn't* doing what it should

(your code is doing exactly what you told it to)

*if your code (or code you call) generates a random # anywhere you need to stop (or seed) that

Inspecting your code…

9

Several approaches. Look at:

+ the last thing you touched
+ the part you feel the least sure of
+ the code most associated with the error you're observing

– every single line of code

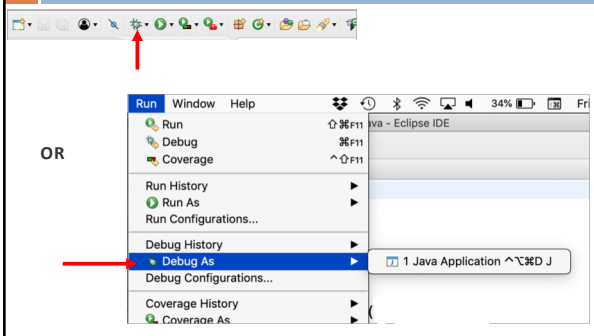
Step through your code

10

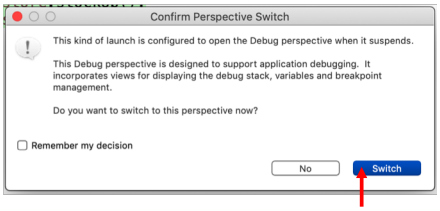
- Print statements
- Use a debugger!

How to start the Eclipse debugger

11



12

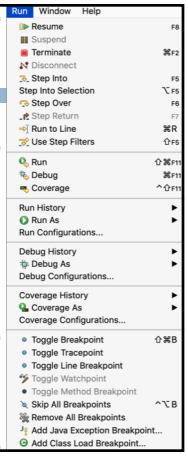


Under the Run Menu

13

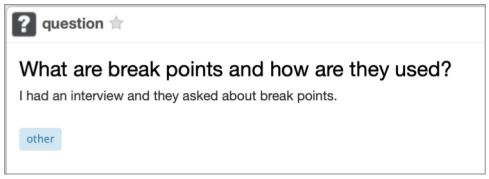
The controls of your debugger
How you navigate through the execution of your code in debug mode.

Lines or variables of interest



2 days ago on piazza

14



Breakpoints, Watchpoints, etc.

15

Some basic functionality common to all debuggers

- Breakpoint
  - a line you want to see get executed
- Conditional Breakpoint
  - a line you *sometimes* want to see get executed
  - ***Warning:*** can make your code super slow
- Watchpoint
  - a global variable you want to track
    - When read
    - When written to

16

Breakpoints & Watchpoints:

https://help.eclipse.org/photon/index.jsp?topic=%2Forg.eclipse.cdt.doc.user%2Ftasks%2Fcdt_o_brkpnts_watch.htm