WHAT ARE YOU WORKING ON?

TRYING TO FIX THE PROBLEMS I CREATED WHEN I TRIED TO FIX THE PROBLEMS I CREATED WHEN I TRIED TO FIX THE PROBLEMS I CREATED WHEN...
Announcements

- Lunch with Professors – sign up, including today!
- Prelim 2 Regrade Requests due tonight at 11:59PM
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](https://www.ioccc.org)
"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...

...despite your best efforts, your code will not work

Now what?
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.
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
Debugging 101

**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...

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

- Print statements
- Use a debugger!
How to start the Eclipse debugger

1. Click on the green arrow with the exclamation mark icon. This starts the debugger in the active project.

OR

2. Go to the Run menu, select "Debug As...", and choose the appropriate option for your project.
This kind of launch is configured to open the Debug perspective when it suspends.

This Debug perspective is designed to support application debugging. It incorporates views for displaying the debug stack, variables and breakpoint management.

Do you want to switch to this perspective now?

Remember my decision

No

Switch
Under the Run Menu

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

Lines or variables of interest
What are break points and how are they used?

I had an interview and they asked about break points.
Breakpoints, Watchpoints, etc.

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
Breakpoints & Watchpoints: