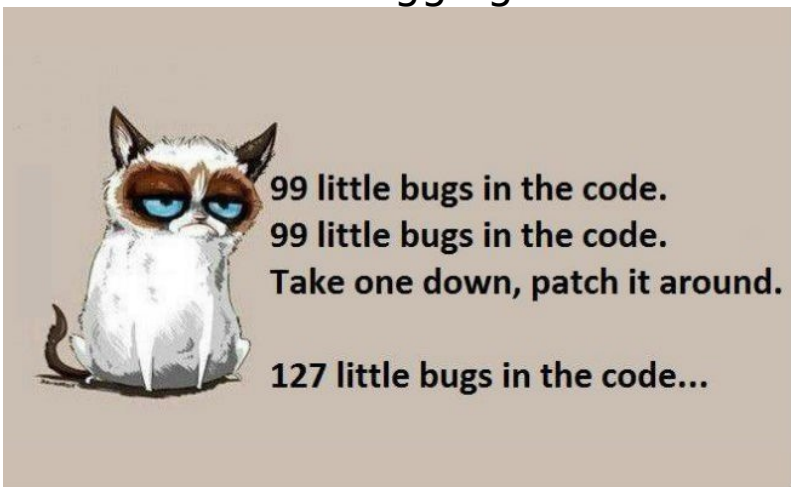
Debugging



There are several .c files in the post with this worksheet. They all have errors; some the compiler will catch and some it won't. Use the pointers below to help you dubug them.

If one of you is brave and has a nice buggy code, work through it with the TAs and those around you to track down those bugs and potentially see how you can rework code to make it more "debuggable."

Understand Compiler Errors

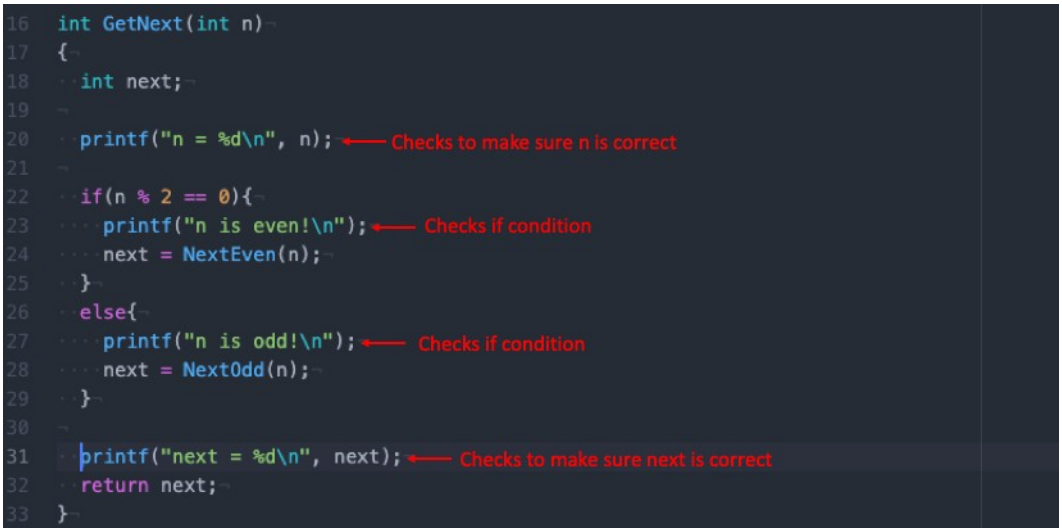
```
test.c: In function 'main':
test.c:4:21: error: expected ')' before ';' token
    4 | #define ARRAY_SIZE 3;
      |                     ^
```

- First line tells you the file and the function where the error is
- Second line: test.c – file name; 4 – line number; 21 – column number
- ERRORS CAN BE MISLEADING! In this example I put a ';' at the end of a #define, which was the mistake, however the compiler tells me that the error is an expected ')' (it also says the error is in main). Usually the error will at least get you close.
- If you have no idea what an error means, try Google.

Check Code For Common Errors

This list contains 11 of the most common errors in C. https://developerinsider.co/11-most-common-pitfalls-in-c-programming-language/ I'm sure a Google will turn up some other frequent ones.

Use Print Statements
Not sure where it's all going wrong? Add some print statements to make the code "feed back" what's going on. This can help you pinpoint the problem. Note: Adding print statements *can* occasionally change your code's behavior. So be aware.



Comment Out
No idea where an error/leak is coming from? When in doubt, comment it out!
1. Begin by commenting out pretty much everything (you may need to get clever. Eg If you comment out a function, you'll also need to comment out the function call.)
2. Re-introduce sections of code one at a time
3. When the error/leak reappears, you have found what section you need to focus on!

Rubber Duck Debug
1. Get a rubber duck (or pet, or plant, or mug of tea, or roommate who doesn't code).
2. Explain your (broken) code and its goals, generally. Don't worry about details, just set the context for your duck.
3. Line-by-line, explain what the flow of the whole function or method that's not working is. Don't skip details, ducks love details!
4. Find your solution! (Typically) in the process of talking it through, you'll find your problem.

Take a Walk
Get up and away from your desk. Get some fresh air or exercise. Don't think about code for a little while. It's amazing how often you'll find your error as soon as you sit back down.

Resources

Generally helpful: https://levelup.gitconnected.com/debugging-without-a-debugger-823dcddea3ae
Rubber duck debugging: https://www.thoughtfulcode.com/rubber-duck-debugging-psychology/