CSE 331 Software Design and Implementation

Handout C8: *How to Debug*

Contents:

Debugging is the process of understanding why your code is erroneous, then fixing the code. You typically begin debugging after you have written tests and one of the tests fails, or after a user reports a bug to you.

You should think of debugging as several separate steps:

a. determine what part of your code is defective
b. determine why that code is defective
c. change the code to fix the defect

This document addresses each of these three steps in turn.

Localizing the failure

To "localize" a failure is to determine which parts of your code might be defective and which are correct, so that you can focus your attention just on the places the defect might be.

1. Your testing framework, such as JUnit, will generally inform you what test case is failing. Since you know what code is executed by that test case, you have narrowed down the parts of your code that might be buggy.

You might want to narrow down the problem more. For example, a given test case might execute multiple methods in your code, and you might want to know which one of them is buggy. A less common reason is that the testing framework does not indicate which test fails, which could happen when your code suffers an infinite loop. (See below for more about this situation.)

A good way to further isolate the problem is by writing more tests. If you have a large test that fails for a reason you do not understand, then create new tests, each of which exercises just part of the functionality of the original test. Then, run those tests to see which part is problematic. An advantage of this is that you can re-run your new tests as often as you like, and they will be of value in the future.

Another way to isolate the problem is to execute the test step-by-step, manually observe the values it computes, and decide whether those results are correct. This can be a quick way to reduce the scope of your investigation, because it doesn't require you to write new tests. On the other hand, it can be tedious, you can easily make errors, and you may have to repeat that manual effort many times rather than being able to easily re-run a test. It's typically a good idea to start with this "manually observe the execution" strategy - then, if it doesn't yield insight quickly, you should create tests.

Executing the test step-by-step can be done in several ways.

- Play computer and manually execute your code with paper and pencil.
- Use the debugger to step through your program one statement at a time, examining the values computed as you go.
- Add logging (print statements) to your program that output key values, run the program, and then examine the console log with all the output. Common places to add logging are the beginning and end of each procedure.

Any of these approaches is also useful for debugging an infinite loop. If you use logging, the console output will indicate which functions or tests started but never completed.

Understanding the failure

To determine why the test case is failing, your best approach is manual reasoning about all the parts of the code that are executed.

For example, suppose that you are debugging an infinite loop. What is the decrementing function for the loop? Can you show that it is reduced every time through the loop and that the loop exits when it is minimal? What is the loop invariant? Can you show it is maintained?

If you are debugging an incorrect computed value, then similar reasoning approaches apply.

If you have trouble with reasoning, you might find it helpful to execute the test case manually and examine the values computed. This may give you the insight to return to manual reasoning. To execute the test case manually, you can play computer, simulating the execution of the test case with paper and pencil. You can also use the debugger to help with this: the debugger can execute your code one line at a time, which lets you observe your code's execution in a way similar to you playing computer. Playing computer or running the debugger will yield the same result as the actual execution that is freezing up your computer.

Fixing the bug

To correct a defect, you should follow the same process as you do when writing code in the first place. In fact, in many cases the best approach, when faced with confusing, buggy code, is to throw it out and start over in a disciplined way with a better design.

Hints for debugging

One good way to go about debugging is to construct an argument that states that the output that you saw is impossible. Oftentimes, just stating this argument will make clear where it is wrong. Or, you may find that you can't make the argument, in which case you can determine an input that does produce that output. If you can make the argument, then you can examine it for mistaken assumptions or incorrect logic. If you can make the argument but can't determine why it is incorrect, then you can present it to someone who can help you (such as the course staff).

In any event, please don't blindly write a lot of tests - this is a terrible use of your time! Instead, follow the scientific method. First form a hypothesis that is grounded in what you know so far, then perform an experiment to test the hypothesis. The experiment might involve writing a test, but don't write a test if you don't have a concrete reason for believing why it might fail or otherwise yield insight.

Before you start debugging, be sure to fix all of the style, documentation, etc. errors that your TA identified when grading your assignment. You may find that doing so fixes errors; even if it does not, it is likely to help you to understand your design better and may give insight into its flaws. Cleaning up your code will also make it easier for other people to help you.

Back to Conceptual Info
Back to the CSE 331 home page.