Class 32: Detour: Experimenting, Testing, and Debugging

Held: Monday, 14 November 2011

Summary: We take a bit of time away from the main topic of this course to consider an issue that some of you have found difficult - How do you make sure that your code is correct and how do you fix it if you find that it is not?

Related Pages:
- EBoard.

Notes:
- We will devote a few minutes at the start of class to questions on the current assignment.
- Today’s class topic is inspired by questions I’ve gotten via email and in person.
- Thursday’s CS extra: A casual chat with Jillian Goetz - Transitioning to grad school in an interdisciplinary field.

Overview:
- Experiments and Testing.
- Things to Test.
- Example: Tests for the Symbol Table Library.
- Debugging.
- An Exercise in Testing and Debugging.

Experiments and Testing

Background

- How do we know that the code we write is correct?
- A formal method: We use annotations, invariants, and such to “prove” that our code is correct.
- But programmers are lazy and formal proofs are hard.
- Programmers are also falliable and can get those proofs wrong.
- So we also like to check our running programs to see how well they work.

Experiments

- Our first way of checking our code is what I would deem an experiment. We run the program, give it some input, and look at the output.
- If the output is wrong, we know that the program is wrong.
- If the output is right, we have some confidence that the program is right.
- More successful experiments increase our confidence.
Testing

- But reading output can also be time consuming.
- And humans are noticeably bad at matching; an output that is wrong may look right.
- Computers, on the other hand, are very good at matching.
- So why not formalize our experiments - Instead of running the program and looking at the results, why not have the computer look at the results?
- I call experiments in which the computer checks the results tests.
- Automatic checking of results also lets us significantly increase the number of experiments/tests we do.

Unit Testing

- In unit testing, we test each component (object, library, etc.) separately.
- Unit testing is often done assuming that the code is a black box: You know what the code is supposed to do (interface) but now how it achieves the ends (implementation).
- It is also possible to do clear box (or white box) testing in which you check to see that internals have the right state.
  - But clear box testing makes it much harder to change the implementation.
  - I tend to avoid it.
  - If you use white box testing, you should separate your white box tests (which need to be changed if you change your implementation) from black box tests (which do not need to be changed).

Things to Test

- Okay, we know that we want to test our program. What do we test?
- Clearly, we test all of the procedures to ensure that they work as expected.
  - We should think about special cases and surprising uses.
- In any code that uses memory, we might also want to check memory usage.
  - Are there leaks (easy to check).
  - Are there other errors (harder to check; probably not a unit test).
- In some cases, we might also check the run time of procedures - Do our procedures have the asymptotic efficiency we would expect.
  - This kind of testing is much harder, and often requires us to annotate our code.
- I generally focus on basic black box testing: Does the code work as advertised.

An Example: Designing Tests for Our Symbol Table

A class exercise.

- We start with some simple tests that we might do by hand.
- We then progress to more complex tests that we probably would not do by hand.
Debugging

- So, what do you do when your tests fail?
- You look at the code that the tests call. Sometimes, just looking at it helps.
- More frequently, you’re not sure what went wrong. So you must debug.
- Clear box testing also serves as an alternative, but can add lots of tests that are only useful for small sections of code.
- How do you debug?
  - Lots of printf statements can be useful if you know that a procedure fails rarely.
  - But most of the time, a debugger is useful.

An Exercise in Testing and Debugging

*As a class, we will test and debug symbol table code written by one of your peers.*