CSC161 2011S *Imperative Problem Solving*

Class 25: Unit Testing

Held: Monday, 7 March 2011

Summary: We consider the related processes of unit testing and test-driven development.

Related Pages:

- EBoard.
- Reading: Unit Testing .

Notes:

- Short answers to your questions on the reading are now available.
- Missing reading questions from: AA, AG, MH, JJ, EL, EO, JY.
- Why ask reading questions? Let's do the math.
- I will reserve time at the start of class for questions on Assignment 6.
- Today will be a collaborative lecture + discussion + recitation + design session + whatever.
- The code we develop and use today can be found under Examples/Testing.
- There is no reading for Tuesday. You may want to start Wednesday's reading on Design by Contract.
- Exam 2 will be distributed tomorrow.

Overview:

- Unit Testing.
- Unit Testing Frameworks.
- Example: Average.
- Example: Binary Search.

Unit Testing Basics

- Unit Testing: An approach to testing your program.
- Follows the scientific method (more or less): Make a hypothesis about what your program should produce and check the result.
  - Done multiple times.
- We test each "logical unit" of the program separately.
- If the units don't work correctly, the whole won't work correctly.
- Write tests *before* you write the code.
  - Tests remind you what you want your code to do.
  - You're more likely to write tests if you write them first.
- Each time you modify your code, rerun the tests to make sure you haven't broken anything.
- Two kinds of testing that we normally do:
  - Black-box testing: We do not have access to the source code.

    - White-box testing: We have access to the source code, and so might think about tests that stress particular parts of the soruce code.

Unit Testing Frameworks

- Unit testing frameworks make it easier to do unit testing.
- Essentially, you mix tests in with your code and the framework can find those tests.
- As importantly, frameworks let you combine different unit tests.
    - You can check whether a change made in one unit breaks another unit.
    - If the program is well designed, the change should not affect another unit, but it might.

Exercise: Average

Let’s test a simple routine used to average values.

- We’ll start with an interactive tester.
- We’ll continue with two hard-coded testers.
- We’ll find some errors and attempt to correct our code.
- We’ll write new code and continue to test.
- We’ll improve our hard-coded tester to give more useful results.

Exercise: Binary Search

We’ll also try testing binary search, but we may not get it it cmpletely.

- We’ll start with an interactive tester.
- We’ll continue with a hard-coded tester.
- We’ll look for ways to improve the hard-coded tester.
- We’ll improve our hard-coded tester to give more useful results.
