CSE 374 - Week 6 (Wed)

Testing

Instructor: Andrew Hu

TAs: Jim Limprasert, Kaelin Laundry, Keith Jiang, Nick Durand, Simon McFarlane, Yitian Hao
Plan for the Week

● Software engineering week!
● Problems we will look at
  ○ Monday: What does it mean for our program to be correct?
  ○ Wednesday: How do we test our program's correctness?
  ○ Friday: How do we keep track of the changes we make to our code?
● Solutions
  ○ Monday: pre and post conditions
  ○ Wednesday: testing
  ○ Friday: git
Testing

- We can write tests to check if our functions generate **correct** output for **valid** input

- Writing a test:
  - Start with input which satisfies the **pre-condition**
  - Run the function on that input
  - Check whether the **post-condition** is satisfied
Tests are Good Enough

- Tests can never 100% prove that your program is correct
  - Even if you test every single input! (e.g. randomness)
- However, tests are usually good enough to give you confidence
- Ideally, your tests should cover all the different "ways" your program can be used
- Together, these tests form a test suite
Testing Isn't Perfect

Program testing can be used to show the presence of bugs, but never to show their absence!

-Edsger Dijkstra, 1970

(1972 Turing Award Winner)
Kinds of Tests

- Unit tests
- Integration tests
- Black box tests
- Clear box tests
Unit vs. Integration Testing

- **Unit testing** = testing one module/function by itself
  - i.e. Testing only one thing at a time
  - If a test fails, we can pinpoint exactly what input it fails on
  - Hard to catch bugs created by a cascade of smaller errors
  - Often done immediately after (or before!) implementing

- **Integration testing** = testing many modules/functions together
  - i.e. Testing how functions will interact when called in sequence
  - Often more realistic simulation of how code will be used
Black vs. Clear Box Testing

- **Black box testing** = tests designed using only information from the specification
  - These can be written even before implementing the spec
- **Clear box testing** = tests influenced by the implementation
  - Author may know about potential weaknesses that need to be tested
  - What are the edge cases?
  - These tests should still pass, even if tested on a different implementation
How Much Testing?

- Writing tests is expensive in time & money
- How much should we test before we're satisfied?
- Many heuristics exist to answer this question
  - One common one: code coverage
- **Code coverage** = was each line of code executed?
  - Having code coverage isn't enough, but not having code coverage is a problem
  - Code coverage is part of clear box testing!
Poll Question: PollEv.com/andrewhu
Poll Question (PollEv.com/andrewhu)

I have the header files (.h) object files (.o) of some program and I want to test many functions working together.

What kind of test should I use in this situation?

A. Black-box unit tests
B. Clear-box unit tests
C. Black-box integration tests
D. Clear-box integration tests
Regression Tests

- As you develop larger projects, you may edit code that you wrote earlier
- You want tests to make sure that any fixes/edits don't introduce further bugs (i.e. a regression)
- **Regression testing** is where you test against old output to make sure that the behavior has not changed
- Good practice is to create at least one regression test each time you fix a bug, to make sure it doesn't come back
Testing Frameworks

● Various programming languages have various frameworks that make testing easier

● For CSE 374, we will give you a testing framework, safe_assert.h
  ○ You must #include "safe_assert.h" in your code

● Testing framework has its own syntax, which is different from C's syntax
safe_assert.h

- Instead of a main() function, there is a suite() block
  - The suite block takes a string literal which is the name of the suite
- Inside of the suite() are similar test() blocks

```c
suite("My test suite") {
    test("Test case 1") {
        int res = pow(10, 2);
        safe_assert(res == 100);
    }
}
```
The `safe_assert()` Macro

- `safe_assert()` is very similar to `assert()`, except that if you segfault inside of an assert, it will tell you!

  ```c
  Segfault (test.c:21): `*s == 'a'`
  ```

- The test suite process will survive the segfault
  - Then, it will tell you that there was a segfault and run the other test cases
  - You don't have to know how this works (it's magic 🪄)
Demo: Testing Framework