CSE 374: Programming Concepts and Tools

Eric Mullen
Spring 2017
Lecture 19: testing and specifications

Administrivia

- Homework 5 due last night
    - Was a tough one, lots of late day use expected
- Homework 6 out tonight, part 1 due next Thurs
    - Everyone got a partner OK? Talk to me after class if not

Where we are

- You’ve built a large-ish program in C
- You’re about to build a bigger, more complicated one, with a partner
- Today is a bit about Software Engineering, and how to make that process better

Is my code right?

YES

NO

Testing

“Test your software, or your users will”

-Hunt & Thomas, *The Pragmatic Programmer*

Design

“There are two ways of constructing a software design:

- One way is to make it so simple that there are no obvious deficiencies
- The other way is to make it so complicated that there are no obvious deficiencies

The first method is far more difficult”

-Sir C.A.R. Hoare

Debugging

"Debugging is twice as hard as writing the code in the first place. Therefore, if you write the code as cleverly as possible, you are, by definition, not smart enough to debug it."

-Brian Kernighan

Testing

"Program testing can be a very effective way to show the presence of bugs, but is hopelessly inadequate for showing their absence."

-Edsger Dijkstra (1972 Turing Award Lecture)

Fixing Bugs

- Think before you code: easiest debugging is before you write the bug
- Use tools/languages that eliminate classes of bugs when you can
    - Java eliminates large classes of memory bugs
- Make defects visible/fail early
- Debugging as a last resort

Fail Early

- C has one more great tool called `assert`
- `assert(x)` takes any boolean expression x, and crashes the program if it's not true
    - Note: there are compiler flags to turn it into an empty statement instead, used for deployment
- Amazingly useful in debugging, helps enforce invariants
- If you want to use, `#include <assert.h>`

Testing Theory

- Testing is limited and difficult
  - Small number of inputs
  - Small number of calling contexts, compilers, environments, etc…
  - Small amount of observable input
  - If test fails, was test broken, or was code?
- There are some standard coverage metrics, but they only emphasize how limited testing is

How much is enough?

```
int my_or(int a, int b) {
    int ans = 0;
    if (a) {
        ans += a;
    }
    if (b) {
        ans += b;
    }
    return ans;
}
```

my_or(1,-1)

my_or(1,1) my_or(1,0)

my_or(0,0) my_or(0,1)

Statement Coverage:
% of statements executed

Branch Coverage:
% of branches executed

Path Coverage:
% of paths executed

Colored Boxes

- **Black Box** testing: don't look at implementation, just consider what it should do
  - Pros: don't make same mistakes twice, think in terms of abstraction
  - Basics: try negative numbers, 0, NULL, empty list, etc…
- **White Box** testing: look at the implementation
  - Pros: can be more efficient, harder to miss corner cases
  - Basics: try loop boundaries, any special constants, max values, empty/full data structure

Stubs

- **Unit Testing**: testing a small unit
- **Integration Testing**: testing the combination of units
- **System Testing**: run the whole thing
- Each has different benefits
    - How do you perform them? Stub out functions
    - Stubs are just small, fake implementations

Stubbing Techniques

- It’s an art, not a science
  - Hardcode a few cases
  - Use a slower, simpler algorithm
  - Don’t do non-essential things (logging, printing, etc)
  - Return wrong answers that won’t mess up testing
  - Use fixed size implementation (array instead of list)
  - Other ideas?

Is my code right?

- What does it mean to have correct code?
- How do we know what we want code to do?
- Can we write that down?

Full Specification

- A full specification is a fully formal description of all constraints on the code
  - While tractable for trivial examples, can be hard for larger systems
  - Exercise: What's the right specification for sorting a list?

Partial Specification

- You can only specify some things, and that's better than nothing

- E.g. can I pass in NULL? are arguments allowed to alias? What if I pass in a negative size?

- Writing these specs in comments can help guide testing

    - Writing these specs as `asserts` can help debugging

Specifications

- It's a very quick transition from easy to very hard
  - Easy: pointer is not NULL
  - Hard: list is not cyclic
  - Impossible (in C): Caller has no other pointers to this object

Homework 6

- You're building a memory allocator (with a partner)
  - Formal writeup will be up later today
  - Gitlab repos for each group will go out today
  - Method stubs are due next Thurs, complete implementation following Thurs
  - Anyone without partners come see me now