

University of Illinois at Urbana-Champaign
Dept. of Electrical and Computer Engineering

ECE 120: Introduction to Computing

Learning to Test Your Programs

ECE 120: Introduction to Computing

© 2016 Steven S. Lametta. All rights reserved.

slide 1

A Necessary Skill: Testing Code

How do you know that your program works?

There's only one correct answer: test it!*

Brooks' Rule of Thumb

- 1/3 planning and design
- 1/6 writing the program
- **1/2 testing**

Just because your program compiles
does not mean your program works!

*Becoming a good tester will take years.
Don't worry if it seems tough.

ECE 120: Introduction to Computing

© 2016 Steven S. Lametta. All rights reserved.

slide 2

Our Next Program Calculates the Roots of a Quadratic

Remember the equation?

$$F(x) = Ax^2 + Bx + C$$

has roots ($F(x) = 0$) at

$$x = \frac{-B \pm \sqrt{B^2 - 4AC}}{2A}$$

where \sqrt{N} is the square root of N .

ECE 120: Introduction to Computing

© 2016 Steven S. Lametta. All rights reserved.

slide 3

Every Statement Must be Executed

How can we test our program?

Let's start with something simple.

Let's say that we have **a statement that is never executed by tests.**

Does the statement work correctly?

How can we know? We have no tests!

So, no, it **does not work correctly.**

At a minimum, we **must execute every statement** (called **full code coverage**).

ECE 120: Introduction to Computing

© 2016 Steven S. Lametta. All rights reserved.

slide 4

What Happens When We Run the Program?

Imagine that we compile and run the program.

Take a look at the code.

The **first statement is a printf**.

The **printf** always executes, so

- we can **check whether the printf works**
- by simply **looking at the output**.

Choose a Line of Text as Our First Test

The program then **waits for input with scanf**.

What input should we give?

Let's **just choose something** concrete.

Say **"0 0 0"** (and then **<Enter>** to start).

What are the values of variables **a**, **b**, and **c**?

0, 0, and 0

What does **scanf** return? **3**

What happens next? **Skip the "then" block!**

Continue Analyzing Until the End

With input **"0 0 0"** our program next

- prints the equation to be solved, and
- calculates the discriminant **D**.

What is the value of D? 0

(Remember that **a**, **b**, and **c** are all 0.)

So **which of the three if-else blocks is executed** (first, second, or third)? **second**

And what is x1? 0 / 0 → NaN

Was that a Bug?

I think so.

The equation is not quadratic when **a** is 0.

The person who wrote the code perhaps didn't think of that case.

And neither did I when I edited the code to present to you.

Bugs can be subtle, and testing can be hard!

We won't fix the bug.

Remember: We Want Full Code Coverage

Let's try again with input "1 0 0".

The same parts of the code execute.

And x_1 is? 0

So the **single root is at 0**, and the **program ends successfully**.

Our equation was $F(x) = x^2 (+0x + 0)$, so plugging in $x = 0$ does produce $F(x) = 0$.

But our test does not execute all code!

Adjust the Inputs to Change the if-else Results

What statements did not execute?

- "then" block of `scanf` check
- first case of **if-else** solution computation
- third case of **if-else** solution computation

Let's adjust our inputs

to execute the other solution cases.

"1 0 0" gave the second case because **D was not positive** and **D was 0**.

Use "1 0 1" to Test the Third if-else Case

To get D negative, change c to 1
(then D is $-4 == 0 * 0 - 4 * 1 * 1$).

For the next test,

- we type "1 0 1",
- and the program tells us
- **there are no real roots.**

Our equation was $F(x) = x^2 (+0x) + 1$, so in fact no value of x can produce $F(x) = 0$.

Use "1 1 0" to Test the First if-else Case

For the first if-else case, we need D positive.

To get D positive, change b to 1 and c to 0
(then D is $1 == 1 * 1 - 4 * 1 * 0$).

For the next test,

- we type "1 1 0",
- and the program gives **roots at 0 and -1.**

Our equation was $F(x) = x^2 + x (+0)$, so $F(x) = 0$ at both $x = 0$ and at $x = -1$.

We Need to Execute the “then” Block of `scanf`

So far, we have four tests:

“0 0 0” (known bug), “1 0 0”, “1 0 1”, “1 1 0”

But we still need a test to execute the “then” block of the `scanf` check!

Anything that stops `scanf` from finding three numbers will do. Let’s type “hello”.

So **five tests** (and **verifying the output by hand!**) gives full code coverage for this program.

Good Testing Must Consider Both Purpose and Structure

Full code coverage is **just a starting point**.

In fact, you should notice that

- one of our tests (“0 0 0”)
- exposes a bug
- in a statement that was already covered
- by another test (“1 0 0”).

In general, good testing requires that one **think carefully about the purpose** of the code **as well as the structure** of the code.

***** So Easy that a Computer Can Do It

Full code coverage is easy to explain.

Finding tests to cover more statements means solving some equations.

Computers are good at that (well ... pretty good).

The automatic programming feedback tool uses this approach to try to find bugs in your code:

- generate tests to cover everything (if possible),
- then compare your program’s results with a “gold” program (written by a professor or TA).