Allen Vartanian

Dr. Johnson

CMSI 402-01

19 March 2018

Homework 3

7.1) The greatest common divisor (GCD) of two integers is the largest integer that evenly divides them both. Knowing that background, what's wrong with the comments in the following code? Rewrite the comments so that they are more effective.

- The comments were not descriptive enough while describing unnecessary details. They also interrupted the flow of the code.

```
// Use Euclid's algorithm to calculate the GCD.
// Assuming a > b, repeatedly divide a by b until the GCD has been
found.
private long GCD(long a, long b) {
  a = Math.Abs(a);
  b = Math.Abs(b);

  for (; ; ) {
    long remainder = a % b;
    if (remainder == 0) return b;
    a = b;
    b = remainder;
  };
}
```

7.2) Why might you end up with the bad comments shown in the previous code?

- The programmer assumes the knowledge of the reader and only wrote comments for their own reference, not future readers.

7.4) How could you apply offensive programming to the modified code you wrote for Exercise 3?

- Break out from the code and return and error if `a < b`

7.5) Should you add error handling to the modified code you wrote for Exercise 4?

- Assert that `a > b`

7.7) Using top-down design, write the highest level of instructions that you would use to tell someone how to drive your car to the nearest supermarket. (Keep it very high level.) List any assumptions you make.

- Find out which supermarket is the closest on a map and use the car to drive there.
- Assumptions: Has a map, can legally drive a car, knows how to operate the car.

8.1) Two integers are *relatively prime* (or *coprime*) if they have no common factors other than 1. Suppose you've written an efficient `IsRelativelyPrime` method that takes two integers between -1 million and 1 million as parameters and returns `true` if they are relatively prime. Write a method that tests the `IsRelativelyPrime` method. (Hint: You may find it useful to write another method that also tests two integers to see if they are relatively prime.)

-
```
private bool areRelativelyPrime( int a, int b ) {
  a = Math.abs(a);
  b = Math.abs(b);

  if ((a == 1) || (b == 1)) return true;
  if ((a == 0) || (b == 0)) return false;

  int min = Math.min(a, b);
  for (int i = 2; i <= min; i += 1) {
    if ((a % i == 0) && (b % i == 0)) return false;
  }

  return true;
}

private bool checkIsRelativelyPrime( int a, int b ) {
  return areRelativelyPrime(a, b) == isRelativelyPrime(a, b);
}
```

8.3) What testing techniques did you use to write the test method in Exercise 1? (Exhaustive, black-box, white-box, or gray-box?) Which ones *could* you use and under what circumstances.

- I used black-box testing because I don't exactly know how the method works, just what it does.
- I could use white-box or gray-box if I knew more about how the method goes about determining its answer, allowing me to test edge cases that would strain the method.

8.5) The following code shows a C# version of the \`AreRelativelyPrime\` method and the \`GCD\` method it calls.

```
// Return true if a and b are relatively prime.
private bool AreRelativelyPrime( int a, int b )
{
  // Only 1 and -1 are relatively prime to 0.
  if( a == 0 ) return ((b == 1) || (b == -1));
  if( b == 0 ) return ((a == 1) || (a == -1));

  int gcd = GCD( a, b );
  return ((gcd == 1) || (gcd == -1));
}

// Use Euclid's algorithm to calculate the
// greatest common divisor (GCD) of two numbers.
// See https://en.wikipedia.org/wiki/Euclidean_algorighm
private int GCD( int a, int b )
{
  a = Math.abs( a );
  b = Math.abs( b );

  // if a or b is 0, return the other value.
  if( a == 0 ) return b;
  if( b == 0 ) return a;

  for( ; ; )
  {
    int remainder = a % b;
    if( remainder == 0 ) return b;
      a = b;
      b = remainder;
    };
  };
}
```

The `AreRelativelyPrime` method checks whether either value is 0. Only -1 and 1 are relatively prime to 0, so if `a` or `b` is 0, the method returns `true` only if the other value is -1 or 1.

The code then calls the `GCD` method to get the greatest common divisor of `a` and `b`. If the greatest common divisor is -1 or 1, the values are relatively prime, so the method returns `true`. Otherwise, the method returns `false`.

Now that you know how the method works, implement it and your testing code in your favorite programming language. Did you find any bugs in your initial version of the method or the testing code? Did you get any benefit from testing the code?

- Both have no checks for max/min values of `int`.

8.9) Exhaustive testing actually falls into one of the categories black-box, white-box, or gray-box. Which one is it and why?

- White-box testing because not only can you use a bunch of different values like black-box testing, you can use specific cases to test the edge-cases of your code.

8.11) Suppose you have three testers: Alice, Bob, and Carmen. You assign numbers to the bugs so the testers find the sets of bugs `{1, 2, 3, 4, 5}`, `{2, 5, 6, 7}`, and `{1, 2, 8, 9, 10}`. How can you use the Lincoln index to estimate the total number of bugs? How many bugs are still at large?

- Need three cases to compare each possible pair:
  - Alice & Bob: (5 x 4) / 2 = 10
  - Alice & Carmen: (5 x 5) / 2 = 12.5
  - Bob & Carmen: (4 x 5) / 1 = 20
- Average = (10 + 12.5 + 20) / 3 ≈ 14

- So there are approximately 14 bugs in the code, but further development and testing may reveal more.

8.12) What happens to the Lincoln estimate if two testers don't find any bugs in common? What does it mean? Can you get a "lower bound" estimate of the number of bugs?

- If there are no bugs in common, then the Lincoln index equation fails because we have to divide by 0. In this case, assume the testers found 1 bug in common.