Homework 3

John Hardy

19 March 2019

Professor Johnson

7.1)

// Use Euclid's algorithm to calculate the Greatest Common Divisor of two numbers.

private long GCD( long a, long b )

{

// First, ensure that a and b are positive numbers

a = Math.abs( a );

b = Math.abs( b );

//Next, use repeated Euclidean division until we find a zero-remainder.

//The last non-zero remainder is the GCD.

for( ; ; )

{

long remainder = a % b;

If ( remainder == 0 ) return b;

a = b;

b = remainder;

};

}

7.2) Bad code comments come about when you are either making comments for the sake of having them or if you are making them after the code has been written already. If a programmer wants to make good comments, they should write them before writing the code.

7.4) I would add an assert statement that would throw an exception when the input provided is bad. I would also delete the code the takes the absolute value of a and b since the assert statement would make this code useless. This makes the GCD function offensive rather than defensive because it would throw a tantrum rather than fix the mistake. Here is the code:

private long GCD( long a, long b )

{

// First, ensure that a and b are positive numbers

Debug.Assert(a > 0 && b > 0);

//Next, use repeated Euclidean division until we find a zero-remainder.

//The last non-zero remainder is the GCD.

for( ; ; )

{

long remainder = a % b;

If ( remainder == 0 ) return b;

a = b;

b = remainder;

};

}

7.5) No, error handling should be done by the code that called the function in the first place. This helps the developer understand and fix problems in their code sooner before the whole project become big and complex.

7.6) DriveToMarket()

1. Manipulate the vehicle in order to travel to the supermarket by identifying hazards, traffic signage, and landmarks as necessary along the way and adjusting appropriately.

8.1)

def test\_is\_rel\_prime():

try:

is\_relatively\_prime(-1000001, 1)

print("Test 1 failed")

except:

print("Test 1 passed")

try:

is\_relatively\_prime(1000001, 1)

print("Test 2 failed")

except:

print("Test 2 passed")

if (is\_relatively\_prime(0, 0) == False):

print("Test 3 passed")

else:

print("Test 3 failed")

if (is\_relatively\_prime(1, 0) == True):

print("Test 4 passed")

else:

print("Test 4 failed")

if (is\_relatively\_prime(0, 1) == True):

print("Test 5 passed")

else:

print("Test 5 failed")

if (is\_relatively\_prime(1, 1) == True):

print("Test 6 passed")

else:

print("Test 6 failed")

if (is\_relatively\_prime(-1, 1) == True):

print("Test 7 passed")

else:

print("Test 7 failed")

if (is\_relatively\_prime(21, 35) == False):

print("Test 8 passed")

else:

print("Test 8 failed")

8.3) When I made the unit tests above, I used gray-box testing. I knew some of the functionality of the code already since I had written up parts of it for exception raising. Thus I made code that was designed to fail but maybe was not as elaborate as it could have been. The rest of the tests were written with no idea how the code would execute since I used a built-in python library for my gcd function.

Exhaustive unit testing would not be applicable for this example since there are over 2 million possible values for each of the inputs, which means that there are 4,000,000,000,000 possible inputs that I would have to test. This would take too long to execute on most computers, so it would not be a viable option. This only includes *valid* inputs. There are infinitely many *invalid* inputs, but there is no point testing all of them.

8.5)

#Return true if a and b are relatively prime.

def is\_relatively\_prime(a, b):

#Only 1 and -1 are relatively prime to 0.

if( a == 0 ): return ((b == 1) or (b == -1))

if( b == 0 ): return ((a == 1) or (a == -1))

gcd = GCD( a, b )

return ((gcd == 1) or (gcd == -1))

#Use Euclid's algorithm to calculate the

#greatest common divisor (GCD) of two numbers.

#See https://en.wikipedia.org/wiki/Euclidean\_algorighm

def GCD( a, b ):

a = abs( a )

b = abs( b )

#if a or b is 0, return the other value.

if( a == 0 ): return b

if( b == 0 ): return a

while(True):

remainder = a % b

if( remainder == 0 ): return b

a = b

b = remainder

Above is the code that I produced from the C# code given. When I ran my tests, all but the first two passed. It seems that I made the assumption that the AreRelativelyPrime function would raise exceptions if the inputs provided were out of bounds. It seems that this was not implemented at all. Thus I found a bug in my unit tests rather than my actual implementation in Python. This was helpful because it helped me understand that the code I was testing was not adequately set up to throw exceptions on bad input.

8.9) Exhaustive testing falls into the category of black-box testing. This is because an exhaustive test runs through all of the possible inputs for a given method without regard for how the method works. In other words, it does not care what weaknesses the code may or may not have, it just wants to test all of the possible things.

8.11) Alice found 5 errors, Bob found 4 and Carmen found 5. Of these, bugs 1, 2 and 5 were commonly found between them. So our Lincoln index is (5 \* 4 \* 5) / 3 = 33.33 bugs roughly speaking within the program at large.

8.12) Obviously, if the testers did not find any bugs in common, then the equation would not hold since you cannot divide by zero. My “lower bound” solution would be to assign some minimum epsilon value for S so that we can still calculate the Lincoln index if no bugs in common were found. I would set S = 0.0001 or some really small number.

If no testers found a bug in common, this means that there are more bugs present then the multiplication of all bugs found. In other words, you have only scratched the surface of all the bugs present in the system.