Assignment 3

# Problem 7.1

// Use Euclid's algorithm to calculate the GCD.

provate long GCD( long a, long b )

{

// Get the absolute value of a and b

a = Math.abs( a );

b = Math.abs( b );

//Repeat until we're done

for( ; ; )

{

// Set remainder to the remainder of a / b

long remainder = a % b;

// If remainder is 0, we're done. Return b.

If( remainder == 0 ) return b;

// Set a = b and b = remainder.

a = b;

b = remainder;

};

}

The problem with these comments is they don’t provide any additional information. The code they are describing is self-documenting. This is how I’d fix it

// Use Euclid's algorithm to calculate the GCD.

provate long GCD( long a, long b )

{

a = Math.abs( a );

b = Math.abs( b );

//Repeat until we're done

while(true)

{

long remainder = a % b;

if( remainder == 0 ) {

return b;

}

a = b;

b = remainder;

};

}

# Problem 7.2

The person could have “commented first”, laying out their entire function in comments, but they forgot to delete the comments afterword. They, also, could have written non-self-documenting code, fixed it later, but didn’t remove the comments

# Problem 7.4

The algorithm requires a be larger b, so you could throw an exception if it’s not.

# Problem 7.5

Any time you throw exceptions, you need to have error handling. Not having elegant error handling leads to programs that just crash and the user doesn’t know what happened.

# Problem 7.7

# Problem 8.1

// Return true if it works

function relativelyPrimeTest () {

return IsRelativelyPrime(1, 2) &&

IsRelativelyPrime(3, 4) &&

!IsRelativelyPrime(9, 3) &&

!IsRelativelyPrime(9, 6) &&

IsRelativelyPrime(127, 419) &&

!IsRelativelyPrime(400, 300) &&

IsRelativelyPrime(1, -2) &&

IsRelativelyPrime(-1, 2) &&

!IsRelativelyPrime(-1, -2) &&

!IsRelativelyPrime(9, -3) &&

!IsRelativelyPrime(9, -6);

}

# Problem 8.3

This is definitely black box testing. I have no idea how IsRelativelyPrime works. I cannot do Exhaustive testing without implementing my own version of the program that I’m SURE works, and doing that to write a test for a function I’m pretty sure works makes no sense. Black box testing is really the only option here, provided we don’t have access to the code of IsRelativelyPrime.

# Problem 8.5

# Problem 8.9

Exhaustive testing is basically black-box testing. Since you don’t know how the program works, you can try every possible input rather than random ones. The problem is you can’t do exhaustive testing dynamically (not by hand) unless you have a way to know what the expected result is. If you’re just checking to see if it breaks or errors, it will work.

# Problem 8.11

They found 5, 4, and 5 bugs respectively. The only one in common is 2. I’m not entirely sure how this works with 3 people, but I think the answer is 100. That seems like a lot, though.

# Problem 8.12

If the testers don’t get any bugs in common, you can’t use the Lincoln method explicitly because you cannot divide by zero. We know, however, that as the number of bugs in common decreases, the number of actual bugs increases. Since 1 is the lowest integer of bugs in common, you could set S equal to 1 and use that as the estimate.