Senior Project 2: Assignment 03

Problem 7.1

```
// Uses Euclid's algorithm to calculate the GCD of two numbers.
//See en.wikipedia.org/wiki/Euclidean_algorithm.

 provate 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;
          };
        }
```

Problem 7.2

1. The programmer might have taken a top-down design of the code giving too many details
2. The programmer added the comments after they wrote the code

Problem 7.4

Throw an error if the input is invalid. In this case, neither a nor b can be 0. To apply offensive programming, check that the input values are not zero and throw an error if this is the case. Modified code is below.

```
provate long GCD( long a, long b ) {
        if (a == 0 || b == 0) {
         ThrowInvalidArgumentError();
       }
          a = Math.abs( a );
          b = Math.abs( b );

          for( ; ; ) {
             long remainder = a % b;
             If( remainder == 0 ) return b;

             a = b;
             b = remainder;
          };
       }
```

Problem 7.5

Yes! The input value of a and cannot be 0! We must check that the input values of a and b are not zero and throw an error if either of them is.

Problem 7.7

1. Find a car
2. Open the door
3. Get in
4. Start the Car
5. Drive out of the parking space
6. Turn left
7. Drive straight
8. Turn right
9. Find a parking spot

10. Pack the car
11. Walk inside the store

Problem 8.1

```
//Validates whether two numbers are relatively prime

//by considering all possible factors between the two numbers other than 1.

private bool ValidateAreRelativelyPrime(int a, int b) {

        // Use positive values.

        int a = Math.Abs(a);
        int b = Math.Abs(b);

        // If either value is 1, return true.

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

        // If either value is 0, return false.
        // (Only 1 and -1 are relatively prime to 0.)

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

        // Loop from 2 to the smaller of a and b looking for factors.

        int min = Math.Min(a, b);
        for (int factor = 2; factor <= min; factor++) {

            if ((a % factor == 0) && (b % factor == 0)) return false;

       }

        return true;

}
```

Tests

For 100 loops:

Int a = pickRandomNumber()

Int b = pickRandomNumber()

Assert AreRelativelyPrime(a, b) = ValidateAreRelativelyPrime(a, b)

For 100 loops:

Int a = pickRandomNumber()

Assert AreRelativelyPrime(a, a) = ValidateAreRelativelyPrime(a, a)

For 100 loops:

Int a = pickRandomNumber()
Assert AreRelativelyPrime(a, 1) == true

Assert AreRelativelyPrime(a, -1) == true

Assert AreRelativelyPrime(1, a) == true

Assert AreRelativelyPrime(-1, a) == true

For 100 loops:

Int a = pickRandomNumber() except 1 and -1

Assert AreRelativelyPrime(a, 0) == false

Assert AreRelativelyPrime(0, a) == false

For 100 loops:

Int a = pickRandomNumber()
Assert AreRelativelyPrime(a, -1,000,000) = ValidateAreRelativelyPrime(a, -1,000,000)

Assert AreRelativelyPrime(a, 1,000,000) = ValidateAreRelativelyPrime(a, 1,000,000)

Assert AreRelativelyPrime(-1,000,000, a) = ValidateAreRelativelyPrime(-1,000,000, a) Assert

AreRelativelyPrime(1,000,000, a) = ValidateAreRelativelyPrime(1,000,000, a)

Assert AreRelativelyPrime(-1,000,000, -1,000,000) = ValidateAreRelativelyPrime(-1,000,000, -1,000,000)

Assert AreRelativelyPrime(1,000,000, 1,000,000) = ValidateAreRelativelyPrime(1,000,000, 1,000,000)

Assert AreRelativelyPrime(-1,000,000, 1,000,000) = ValidateAreRelativelyPrime(-1,000,000, 1,000,000)

Assert AreRelativelyPrime(1,000,000, -1,000,000) = ValidateAreRelativelyPrime(1,000,000, -1,000,000)

Problem 8.3

Black-box tests because we do not know how the method works.

Problem 8.5

Did not find any bugs. Testing taught me how to do unit tests with Javascript. Code is submitted with homework.

Problem 8.9

Black-box testing because they do not need knowledge of how the functions they are testing work.

Problem 8.11

Alice/Bob: 5 × 4 ÷ 2 = 10

Alice/Carmen: 5 × 5 ÷ 2 = 12.5

Bob/Carmen: 4×5÷1=20

Bugs at large = (10 + 12.5 + 20) ÷ 3 ≈ 14

Problem 8.12

The Lincoln index will divide by 0, giving a result of infinity. This means we cannot know for certain how many bugs exist. You can get a lower bound estimate for the index by pretending the testers found 1 bug in common.