1. You may recall from an earlier lab the following method for determining whether a given number is prime:

```java
public static boolean isPrime(long candidate) {
    if (candidate % 2 == 0)
        return (candidate == 2);
    else {
        int divisor = 3;
        while (divisor * divisor <= candidate) {
            if (candidate % divisor == 0)
                return false;
            divisor += 2;
        }
        return true;
    }
}
```

But the distinction between prime and composite numbers properly applies only to natural numbers greater than or equal 2. Therefore, it is a precondition of this method that \( candidate \geq 2 \). Add an `assert`-statement to this method definition to enforce this precondition, and write a test program to confirm that, when assertions are enabled, the execution of the program is terminated if the assertion fails.

2. State the loop-exit condition for the `while`-loop in the `isPrime` method shown above.

3. State the loop invariant for that loop. (It is not necessary to write the loop invariant in Java. It is probably better to state it in English prose instead.)

4. Show that the loop invariant and the loop-exit condition, taken together, guarantee that `candidate` is a prime number.

5. Consider the problem of reading successive numbers from the keyboard until a non-positive value is read, and finding the average of the positive values. For example, if the numbers 1, 2, 3, 4, 5, and 0 are entered, then the average of 1, 2, 3, 4, and 5 is computed, and the resulting value is printed.

The file `/home/reseda/object-oriented-programming/code/LoopInvariants.java` contains five attempts to solve this problem, each of which presupposes that the user will enter at least one non-negative value.

6. Without compiling and running the program, review each of these attempts and state the loop invariant for each one, expressing the relationship between the values of the variables `sum`, `count`, and `value` and the data that the user has so far provided. Remember that a loop invariant must hold at the beginning and end of each iteration of the loop body.

7. Still without compiling and running the program, but considering the loop invariants you formulated in the preceding exercise, which of these code segments will give the correct average when run for any data set (as long as the data begin with a non-negative number)? Show that the loop invariant and the loop-exit condition, taken together, justify your answer.
8. For each of the flawed attempts, find the simplest change in the code that would make it correct by establishing the correct loop invariant for this problem.

9. Choose one of the correct methods (or write your own, if none of them is correct) and develop and run tests for it.

10. Add an `assert` statement to your method to confirm that the user has supplied at least one positive value before the division operation takes place. (Hint: In this case, the assertion should not be placed at the beginning of the method body.)

Several of the exercises in this lab are adapted from an earlier version created by my colleague Henry Walker, as part of an ongoing project to introduce the concepts of assertions and loop invariants informally in CS1 and CS2 courses. Early funding for this work came, in part, from NSF Grant CDA 9214874, “Integrating Object-Oriented Programming and Formal Methods into the Computer Science Curriculum.” Professor Walker was the Senior Investigator for the part of the project that dealt with assertions and loop invariants. I am indebted to Professor Walker for permitting me to use and adapt his work.

Copyright © 2007, 2013, 2018 John David Stone

This work is licensed under the Creative Commons Attribution-ShareAlike 4.0 International License. To view a copy of this license, visit

http://creativecommons.org/licenses/by-sa/4.0/deed.en

or send a letter to Creative Commons, 543 Howard Street, 5th Floor, San Francisco, California, 94105, USA.