SYSC 4101

Laboratory 2

In number theory, the integer square root of a positive integer `n` is the positive integer `m` which is the greatest integer less than or equal to the square root of `n`. For example, the integer square root of integers 16, 17, 18, … 23, and 24 is 4.

Below are three different implementations (three different algorithms) of the integer square root functionality: `integer_sqrt_v1()`, `integer_sqrt_v2()`, `integer_sqrt_v3()`. (The actual code is available on the course web site.)

Your tasks, using jacoco for structural coverage measurement, are to:

1. Create a test suite (TestSuite1) for `integer_sqrt_v1()` that allows you to exercise all statements (lines) and all branches of its code.
2. Execute this test suite on `integer_sqrt_v2()`. What coverage of its statements and branches do you observe? If this coverage is not 100% for both statements and branches, add tests to Testsuite1 to obtain test suite TestSuite2 that exercises all the statements and branches of the implementation of `integer_sqrt_v2()`.
3. Similarly, execute test suites TestSuite1 and TestSuite2 on `integer_sqrt_v3()`. What coverage of its statements and branches do you observe for these two test suites? If this coverage is not 100% for both statements and branches, add tests to either TestSuite1 or Testsuite2 to obtain test suite TestSuite3 that exercises all the statements and branches of the implementation of `integer_sqrt_v3()`.
4. We will discover later in the course that exercising all the statements of a piece of code is called a criterion. Similarly, exercising all the branches of a piece of code is another criterion. Another family of criteria asks that we exercise specific paths (a.k.a. sequences of lines of code). Consider the path in the code of `integer_sqrt_v2()` made of the sequence of lines numbered 9, 10, 13, 14, 15, 16, 17, 18, 19. The criterion asks that we execute these lines in that specific order; of course, an execution will start by executing lines 1, 2, …, and will eventually terminate with line 21. So a complete execution that exercises the requested path must start with lines 1, 2, … reach line 9 and execute the path as required (the exact sequence of line numbers without missing anyone line number, without adding any line number along the way between 9 and 19) and eventually reaches line 21.
   Does any of the test suites you created earlier (TestSuite1, TestSuite2, TestSuite3) exercise this path?
   - If yes, which test case(s) of which test suites do that? Justify.
   - If not, which test input should you use to exercise this path? Justify.
5. Is path 9, 10, 13, 14, 15, 16, 17, 19 (skipping line 18 from the previous path) feasible?
   - If yes, which test input(s) make this path execute? Justify.
   - If not, justify why.

<u>One important note about jacoco</u>: When you execute a piece of code that has been instrumented by jacoco, jacoco records coverage information into a jacoco.exec file. Each time you execute the instrumented code, the jacoco.exec file gets updated. You then obtain cumulative coverage information. If you wish to start collecting coverage information from scratch, you must first delete file jacoco.exec before running the instrumented code.

```
public static int integer_sqrt_v1(int n) {
      int small_candidate, large_candidate;
      if (n<0) {
            System.out.println("Input should be positive or null.");
            return -1;
      }
      if (n<2)
            return n;
      small_candidate = integer_sqrt_v1(n >> 2) << 1;
      large_candidate = small_candidate +1;
      if (large_candidate * large_candidate > n)
            return small_candidate;
      else
            return large_candidate;
}
```

```
public static int integer_sqrt_v2(int n) {
      int shift, result, large_candidate;
      if (n<0) {
            System.out.println("Input should be positive or null.");
            return -1;
      }
      if ((n==0)||(n==1))
            return n;
      shift = 2;
      while ( (n >> shift) != 0) {
            shift = shift + 2;
      }
      result = 0;
      while ( shift >= 0) {
            result = result << 1;
            large_candidate = result + 1;
            if (large_candidate*large_candidate <= (n >> shift) )
                  result = large_candidate;
            shift = shift - 2;
      }
      return result;
}
```

```
public static int integer_sqrt_v3(int n) {
      if (n<0) {
            System.out.println("Input should be positive or null.");
            return -1;
      }
      int x0 = n >> 1;
      if (x0 >= 1) {
            int x1 = ( x0 + n / x0 ) >> 1;
            while ( x1 < x0 ) {
                  x0 = x1;
                  x1 = ( x0 + n / x0 ) >> 1;
            }
            return x0;
      } else
            return n;
}
```