Things to do:

1. Level 0.A test
   1. Implement the Pseudo Random Number generator from Visual Basic. The magic numbers are (2^24, 1140671485, 12820163).
      1. This is a 24-bit random number generator. So it loses it randomness if you run a sequence that is longer than 2^22.
      2. Make a test for the random number generator.
      3. Added to the test infrastructure library.
   2. Create an integer square root function and add it to the test infrastructure library.
      1. Newton’s method.
      2. Use an initial guess that is half the number of bits.
   3. Fix the example so that each delay is random.
      1. Random distribution that is Pareto.
      2. Power = 2,min= 100
      3. Compute the uniform random number and apply the inverse CDF.
      4. This will require the square root function.
   4. Have the PC send the initial seed over the serial line.
   5. Make this run n times on each timer, in order.
      1. Typically we’ll only run it 100 times now instead of 1000 times.
      2. Load the number of timer events from the PC.
2. Create a new test. Level 0.B
   1. Implement a log function.
   2. Use a Weilbull distribution in the region where the time interval is too short.
   3. Do this about 5,000 times.
   4. On the PC side see if the function is as expected.
3. Create a new test. Level 1.A
   1. Switch timers randomly.
   2. So for each timer event we will pick a random delay and random timer to do the job.
   3. Use a uniform random number generator for which timer to pick.
4. Create a new test. Level 1.B
   1. This is the level 0.A test, but with sleep.
   2. Go in and out of sleep mode.
5. Create a new test. Level 2
   1. Interleave all 8 timers.
   2. Start all timers at the same instance and when fire, set the next random number in the sequence.
   3. Use the Pareto distribution.
   4. What happens when two timers fire too close together?
   5. On the PC side the test passed unless the two timers fired too close together at least once.
   6. Go in and out of sleep mode.