Homework Exercise #6
Due: November 10, 2009

1. The model of computation for this question is an asynchronous shared-memory system of n processes. The shared memory consists of linearizable registers. Processes may experience crash failures.

A PROD object stores a positive number (initially 1) and provides two types of operations:

- MULT(x) multiplies the value stored in the object by x and returns ACK (where x is a positive integer).
- PRODUCT returns the current state of the object without changing it, and

(a) Show that the following implementation of a PROD object is *not* linearizable. It uses n shared registers, denoted $R_1, \ldots, R_n$ and the code below is for process P_i. Each register initially contains 1.

```
MULT(x)
    v ← read(R_i)
    write(v * x) into R_i
    return ACK
end MULT

PROD
    result ← 1
    for i = 1 to n
        result ← result * read(R_i)
    end for
    return result
end PROD
```

(b) Is the implementation in part (a) linearizable if the only argument allowed for MULT operations is 2?

(c) Describe how the PROD object can be implemented in a linearizable and lock-free way. The implementation should work for all possible arguments to MULT operations, not just 2. ("Lock-free" will be defined in the Nov 5 lecture, but it means that, even if some processes crash, whenever there are pending operations on the PROD object by non-faulty processes, one of those operations will eventually finish.)