York University EECS 4101/5101 March 26, 2019

Homework Assignment #8
Due: April 2, 2019 at 11:30 a.m.

1. In class, we looked at a linearizable implementation of a counter object that stored an integer value and provides two operations: Read and Inc. Now, consider a counter object that stores an integer and provides *three* operations:

- Read returns the value stored,
- Inc adds one to the value stored (and returns Ack), and
- Dec subtracts one from the value stored (and returns Ack).

(a) Show that the following implementation is *not* linearizable. It uses an array $A[1..n]$, where n is the number of processes allowed to access the counter. The following code is executed by process i.

```
Inc
    $x \leftarrow$ read $A[i]$
    write $x + 1$ into $A[i]$
    return Ack
end Inc

Dec
    $x \leftarrow$ read $A[i]$
    write $x - 1$ into $A[i]$
    return Ack
end Dec

Read
    $total \leftarrow 0$
    for $j \leftarrow 1..n$
        $x \leftarrow$ read $A[j]$
        $total \leftarrow total + x$
    end for
    return total
end Read
```

(b) Show that it is possible to implement a non-blocking, linearizable counter that supports Inc, Dec and Read operations using only reads and writes of shared memory.

Hint: your answer can be quite short.

(c) **Bonus question:** The (incorrect) implementation in part (a) uses the fact that all processes are assigned unique labels $1..n$, so that process i can write its contributions to the counter's value in location $A[i]$. Your algorithm in part (b) likely uses this fact too. An implementation of a counter is called *anonymous* if processes do not have unique labels, and for each of the three operations, all processes have identical programme code.

Is there an anonymous implementation of a counter that is non-blocking and linearizable? Show your answer is correct.

Hint: Think carefully about what happens when two processes trying to do the same operation run at exactly the same speed.