Homework Assignment #7
Due: March 27, 2023 at 10:00 p.m.

1. Recall the Michael-Scott lock-free queue discussed in class. Elements are stored in a singly-linked list with one dummy node at the head of the list. It uses two shared variables $Head$ and $Tail$ that point to nodes in shared memory. The other variables in the code below are local to each process.

```
1:  function ENQUEUE(x)
2:      node ← new node containing x
3:      loop
4:          t ← Tail                          ▷ read Tail and store value in local variable t
5:          n ← t.next                        ▷ read t.next to see if Tail needs updating
6:          if t = Tail then                  ▷ reread Tail to see if it is still equal to t
7:              if n = null then              ▷ if, at line 5, Tail pointed to the last node t in list
8:                  exit when CAS(t.next, n, node) ▷ try to append my node
9:              else
10:                 CAS(Tail, t, n)           ▷ help complete another enqueue before trying again
11:             end if
12:         end if
13:     end loop
14:     CAS(Tail, t, n)                       ▷ finish my enqueue by advancing Tail pointer
15: end function
16:
17: function DEQUEUE
18:     loop
19:         h ← Head                          ▷ read Head and store value in local variable h
20:         t ← Tail                          ▷ read Tail and store value in local variable t
21:         n ← h.next                        ▷ read second element in list
22:         if h = Head then                  ▷ reread Head to see if it is still equal to h
23:             if h = t then                 ▷ check whether, at line 20, Head was equal to Tail
24:                 if n = null then          ▷ check whether, at line 20, list contained only dummy node
25:                     return null           ▷ returning null indicates queue is empty
26:                 end if
27:                 CAS(Tail, t, n)           ▷ help complete an enqueue before trying again
28:             else
29:                 if CAS(Head, h, n) then   ▷ try to advance Head to accomplish dequeue
30:                     return n.value        ▷ return value in node that I advanced Head to
31:                 end if
32:             end if
33:         end if
34:     end loop
35: end function
```

We argued in class that we could linearize

- an ENQUEUE operation when it performs a successful CAS on line 8,
- a DEQUEUE operation that returns null at its last execution of line 20, and
- a DEQUEUE operation that performs a successful CAS on line 29 at that CAS.

Linearizing in this way guaranteed the invariant that, at any time, the sequence of elements that should be in the queue (if the operations linearized so far were done sequentially in their linearization order) is exactly the same as the sequence of elements reachable from the node $Head$ by following $next$ pointers (excluding $Head$ itself). We used this invariant to argue that each operation's response is consistent with the linearization ordering.

(a) If line 14 were removed from the code, would the invariant above still be true? Would the linearization still be valid? Briefly justify your answer.

(b) Now consider the code above (including line 14). Suppose we modify the linearization so that an ENQUEUE operation is linearized when $Tail$ is changed to point to the node created by the ENQUEUE. DEQUEUE operations are linearized as before. Is this linearization still correct?

If your answer is yes, explain why the linearization point of each operation is between the invocation and response of that operation, give an invariant, and use the invariant to explain why each operation's response is consistent with the linearization ordering.

If your answer is no, give an execution for which this modified linearization is incorrect.

(c) Finally, consider the implementation with line 14 removed. Repeat part (b) for this implementation. (Is the modified linearization given in part (b) correct for the implementation with line 14 removed?)