Homework Assignment #9
Due: November 28, 2025 at 11:59 p.m.

1. Consider the following proposed implementation of a stack. It uses a shared array $A[1..n]$ and a shared integer variable Top. You may assume that no process ever tries to perform a push that would cause the stack to contain more than n elements. Initially, the stack is empty, all entries of A are nil and $Top = 1$.

The elements of the stack are stored in a prefix of A, and Top is approximately the last index of such an element. It is not possible to keep Top perfectly accurate, but the idea is that Top should either index the last element on the stack or the nil entry just beyond the last stack element.

```
1: PUSH(element x)
2:     while true do
3:         t ← Top
4:         if A[t] ≠ nil then then
5:             CAS(Top, t, t + 1)
6:         else if CAS(A[t], nil, x) succeeds then
7:             return ack
8: POP
9:     while true do
10:        t ← Top
11:        if A[t] = nil then
12:            if t = 1 then return empty
13:            else CAS(Top, t, t − 1)
14:        else
15:            r ← A[t]
16:            if CAS(A[t], r, nil) succeeds then return r
```

[2] **(a)** Is the implementation lock-free? Explain why your answer is correct.

[1] **(b)** Is this implementation linearizable?

[5] **(c)** If your answer to (b) is yes, define the linearization points of each operation and describe the relationship between the abstract stack and what is stored in memory. (Recall that in class, we did something similar for the list-based stack, where we said the contents of the abstract stack are exactly the sequence of elements reachable from the Top pointer, in reverse order.) You do not have to provide a detailed proof of linearizability.

If your answer to (b) is no, give a detailed description of an execution that is not linearizable. You should say which exactly lines of code each process performs, and how those lines are scheduled to take place in the execution. Also, show the state of the shared memory at key points in your execution.