1. Recall that Fischer’s mutual exclusion algorithm for processes $p_0, \ldots, p_{N-1}$ works as follows: $turn$ is a MRMW register with range $\{-1, 0, \ldots, N-1\}$; initially it has value $-1$. A process $p_i$ that wants to enter its critical section, spins on $turn$ until it is $-1$. Within one time unit of this read, $p_i$ sets the value of $turn$ to $i$. $p_i$ waits for more than one time unit, and then reads $turn$ again. If it still has the value $i$, then $p_i$ enters its critical section. Else $p_i$ returns to spinning on $turn$ until it is $-1$. When a process exits its critical section, it sets the value of $turn$ to $-1$.

(a) Explain why this algorithm provides mutual exclusion and deadlock-freeness. (12 pts)

(b) Give an example to show that mutual exclusion could be violated if $p_i$ would wait less than one time unit before reading $turn$ again. (5 pts)

2. Consider the construction of a MRSW register from SRSW registers. Give (in detail) an execution in which one write by writer $B$ and three reads by different readers $A_i, A_j, A_k$ occur concurrently. Let the read by $A_i$ return the old value, while the reads by $A_j$ and $A_k$ return the new value; moreover, let $A_k$ read the old value on position $a_{\text{table}}[k][k]$ of the array. Also discuss linearization points of your execution. (12 pts)

3. Give implementations of an isLocked() method that tests whether a thread is holding a lock (but does not acquire that lock) for:

(a) the testAndSet() spin lock; (3 pts)

(b) the CLH queue lock; and (4 pts)

(c) the MCS queue lock. (4 pts)
4. A savings account object holds a nonnegative balance, and provides `deposit(k)` and `withdraw(k)` methods:

- `deposit(k)` adds `k` to the balance.
- `withdraw(k)` subtracts `k` if the balance is at least `k`, and otherwise blocks until the balance becomes `k` or greater.

Give a (Java pseudocode) implementation of this savings account using locks and conditions. (12 pts)

5. Suppose that the `contains()` method in the optimistic version of list-based sets does not lock nodes (and performs no validation), but instead simply returns `true` if it observes the value it is looking for, or `false` otherwise.

Either give a counterexample to show that this version of `contains()` is not linearizable, or give for each possible call of `contains()` a linearization point (and consider the interplay with `add()` and `remove()` calls to argue that your linearization is correct). (12 pts)

6. Describe a variant of the termination detection barrier for the work-stealing bounded queue in which a thief communicates with the barrier not before it tries to steal a task, but after it has successfully stolen a task. Let thieves write a special marker at the place where they have stolen a task. (12 pts)

7. Give a transactional specification, using atomic blocks, of the readers-writers lock. That is, when a writer is in its critical section, no concurrent readers and writers are in their critical section; but there can be concurrent readers in their critical section. Make sure that writers cannot be kept waiting indefinitely by a continuous stream of readers. (15 pts)