Concurrent Programming - Sheet 2

3.
The buffer will internally use a Queue of type [T]. The main loop of the buffer will be a serve on both of the provided channels. The buffer will listen to the channel `in` and will enqueue anything coming from it to the queue and try to send a dequeued element on the `out` channel (with the added boolean guard that makes sure that the queue isn't empty). This behaviour can be seen in the controller process in programming problem 4 (integration).

7.
a) A process that has already been checked to be passive can receive a message from a process further down the ring (making it active again). The process down the ring may then itself turn passive, so the token finds no active processes even though some are always present.
b) We need more tokens, specifically in at least one moment in time, we need every process to have one token. This can be done by making the processes buffer a token, and sending it forward exactly when they receive a second one. After N tokens are sent, every process contains one. These N tokens contain the information needed.
Now process 0 will send another N tokens, so that all the tokens containing information are pushed back to process 0. We can then terminate if all the N processes are true. As processes will have tokens left over, we must also arrange for either their disposal or maybe make process 0 ignore first N tokens it receives every check.
c) This change solves the problem from a). If a process wants to send a message to a process that has already been checked by the token, it needs to go through process 0 or through a process that currently has the token. The latter will be caught by the token, the former must be checked manually by process 0. Therefore if process 0 receives a message between sending and receiving the token, the system cannot yet terminate. Other that that, the system can function just like in a).