This handout describes the implementation of a shared-memory queue that supports a single producer thread and multiple consumer threads. For simplicity, we assume the queue has infinite space. The queue uses the atomic compare-and-swap (CAS) instruction, defined as follows:

\[
\text{CAS} \text{ old, new, Imm(base) atomically loads the value at the effective memory address and compares it with the value stored in register old. If both values are equal, it updates the memory location with the value stored in register new. If both values are not equal, it updates the value in old with the value loaded from memory.}
\]

The queue stores single-word messages. The code for producer and consumers are shown below, with memory operations highlighted in bold.

**Code for producer to enqueue a message:**

# R1 - contains message to enqueue
# R2 - contains address of the tail pointer of the queue

\[
P1: \text{LD R3, 0(R2)} \quad \text{# get tail pointer}
P2: \text{ST R1, 0(R3)} \quad \text{# write message to tail}
P3: \text{ADD R3, R3, 4} \quad \text{# update tail pointer}
P4: \text{ST R3, 0(R2)}
\]

**Code for consumer to dequeue a message:**

# R1 - contains dequeued message after code finishes
# R2 - contains address of the head pointer of the queue
# R3 - contains address of the tail pointer of the queue
# R4 - contains address of the head pointer write lock
# R5 - contains value 1

\[
C1: \text{SpinLock: MOV R6, R0} \quad \text{# set R6 to 0}
C2: \text{CAS R6, R5, 0(R4)} \quad \text{# try to acquire lock}
C3: \text{BNEZ R6, SpinLock}
C4: \text{LD R7, 0(R2)} \quad \text{# get head pointer}
C5: \text{LD R8, 0(R3)} \quad \text{# get tail pointer}
C6: \text{BEQ R7, R8, Retry} \quad \text{# is there a message?}
C7: \text{LD R1, 0(R7)} \quad \text{# read message from queue}
C8: \text{ADD R7, R7, 4} \quad \text{# update head pointer}
C9: \text{ST R7, 0(R2)}
C10: \text{ST R0, 0(R4)} \quad \text{# release lock}
\]