The queue is implemented as a **ring buffer** with indices front and rear that wrap with % QUEUE_SIZE.

The current code:

```
uint32_t Kernel_msgQ_count(KernelMsgQ_t Qname)

{

  return sMsgQ[Qname].rear - sMsgQ[Qname].front;

}
```

works only while rear >= front.
It breaks after the indices **wrap around** and rear becomes **less than** front.

1. Put 6 items → front = 0, rear = 6 → count = 6 - 0 = 6
2. Get 5 items → front = 5, rear = 6 → count = 6 - 5 = 1
3. Put 3 more items (wrap around) → indices updated with modulo:

   rear goes: 6→7→0→1

   Now: front = 5, rear = 1

Real number of items = 3 ([5],[6],[7] positions logically).
But the function returns:

rear - front = 1 - 5 = -4 → huge unsigned number

So whenever rear has wrapped and rear < front, the subtraction is wrong.

Corrected code

Use the queue size (capacity) and handle wrap-around:

```
#define MSGQ_SIZE   8u   // example capacity; use your real size


uint32_t Kernel_msgQ_count(KernelMsgQ_t Qname)

{

  uint32_t front = sMsgQ[Qname].front;

  uint32_t rear  = sMsgQ[Qname].rear;
```

```
    if (rear >= front)
    {
        // normal case: no wrap
        return rear - front;
    }
    else
    {
        // wrapped case: elements are from front..end and 0..rear-1
        return (MSGQ_SIZE - front) + rear;
        // equivalently: (rear + MSGQ_SIZE - front) % MSGQ_SIZE;
    }
}
```

In words:

- If rear is ahead of front, just subtract.
- If rear has wrapped, the count is “elements from front to end of buffer” plus “elements from 0 to rear - 1”.