# Week 3 Quiz

1. In the robust library's add_to_queue function, the fields head and count of the queue head are not checked for consistency in that function. What problem might this cause?
    - The count field may have an inaccurate count of elements in the queue
    - None; the checking is done in readref
    - If head is negative, the element to be added may overwrite another element in the queue
    - The index where the new element is to be put might exceed the length of the queue

*Correct If the checking were not done, an internal inconsistency could cause the element to be added to overwrite another element of the queue or the count field may have an inaccurate count of the elements in the queue. The checking is done in readref. Even if no checking were done the index where the new element is to be put cannot exceed the length of the queue because the mod in the assignment of the new value ensures it will never be put beyond the end of the array.*

2. Which of the following are things to do in robust programming? (Select all that apply.)
    - Make parameters that can be checked for validity

*Correct Making parameters that can be checked for validity, checking for references to outdated data and checking that parameters are valid when the function is called are all required for robust programming.*

    - Once debugged, remove the checks for internal inconsistencies as they are no longer necessary
    - Check that parameters are valid when the function is called

*Correct Making parameters that can be checked for validity, checking for references to outdated data and checking that parameters are valid when the function is called are all required for robust programming.*

    - Check for references to outdated data

*Correct Making parameters that can be checked for validity, checking for references to outdated data and checking that parameters are valid when the function is called are all required for robust programming.*

3. In the robust library, what happens if the queue array is full and the function to create a new queue is called?
    - The function to create a new queue silently ignores the failure and returns a bogus token
    - The library crashes as the index used is beyond the end of the queue array
    - A new queue element is created after the end of the queue array, and the corresponding ticket is returned
    - The function to create a new queue returns an error code

*Correct The first check in create_queue will detect the queue is full. That check will return the error.*

4. In the robust library, the token generation function qtktref does not check that the nonce is non-negative. Why not?
    - The qtktref function ensures the nonce value cannot overflow
    - Nonces are always positive
    - It is only important that the nonce be unique
    - A negative nonce indicates a library function has been called incorrectly

*Correct A nonce can take on any value; the definition of a nonce is simply a unique number. The nonce value can overflow, but it will be caught and if the nonce overflows, it is due to internal conditions and not an external call.*

5. Which of the following is *not* a problem with the take_off_queue() routine in the fragile library?
    - The mod of the head field with the size field needs to occur before the head field is used to obtain an element, because incrementing it may produce an index 1 more than the index of the last element of the queue.
    - Queue underflow may occur.
    - Queue overflow may occur.
    - Its arguments are not checked for validity.

*Correct There is no check on the number of elements in the queue before one is removed but as elements are never added to the queue, overflow cannot occur.*

6. Why does the token include a nonce?
    - To encode a checksum so the library can determine whether the token is valid
    - To make a token harder to forge
    - To distinguish between a current queue and a queue that has been deleted
    - To make it unlikely that a random number passed as a token will in fact be a valid token

*Correct The nonce starts at an arbitrary positive number and is incremented by 1 each time a queue is created, so it is not a checksum. Forgery would be used to access a queue that is normally not protected, which does not arise here. To make it unlikely that a random number passed as a token will in fact be a valid token is a side effect of the token and nonce starting from arbitrary positive numbers, but is not the reason for the nonce.*

7. Which of the following is a property of fragile programming?
    - Preventing abnormal termination
    - Enabling unexpected actions
    - Handling bad input gracefully
    - Handling internal errors gracefully

*Correct Enabling unexpected actions and handling bad input and errors gracefully are all properties of robust programming while enabling unexpected actions is a property of fragile programming.*

8. Complete the sentence: a function with high cohesion
    - is more likely robust than one with low cohesion
    - performs several different independent functions
    - is more complex than one with low cohesion
    - has no parameters

*Correct. Cohesion is a measure of how well parts of a function work towards one goal; the higher the cohesion, the more the parts of a function work towards a single goal. Performing several different independent functions is the anthesis of this. Functions with higher cohesion are generally simpler than ones with low cohesion, because there is no need to add code to determine which goal the function will perform.*

9. What is the relationship between programming robustly and programming defensively?
    - They are opposites; a program written defensively can never be robust, and vice versa
    - Robust programming refers to the program, while defensive programming refers to how the user interacts with the program.
    - They are the same; defensive programming means to program as though your program will execute in a hostile environment, and robust programming is effectively the same thing
    - They are different; defensive programming means to program as if your program will be criticized, and robust programming means to program so that your program works as expected and handles errors gracefully.

*Correct The other answers all misstate the meaning of "defensive" programming.*