1. The main function contains calls to `exit()` (line 66) and `pthread_exit()` (line 80). How will the effect of these two calls differ when they are executed?

A) exit() terminates the entire process, whereas pthread_exit() terminates only the current thread and leaves other threads and the process running. exit() ends the program itself. pthread_exit() only terminates the main thread and let other threads and the program running.

2. The main function calls `pthread_join()` (line 90) with the parameter `thread_return`. Where does the value stored in `thread_return` come from when the `consumer_thread` is joined?

A) The value of thread_return is taken from a ‘consumer_routine’ return, whose type is long, so the return can be casted to long.

3. Where does the value stored in `thread_return` come from if the joined thread terminated by calling `pthread_exit` instead of finishing normally?

A) We can call pthread_exit with argument, such as pthread_exit(arg), and the passed argument are stored in thread_return. Thus, a parent thread that created the thread that calls pthread_exit can use thread_return to execute the logic. So In normal execution and in execution of the pthread_exit call, the logic will behave consistently.

4. On the same call to `pthread_join()` (line 90), what will it do if the thread being joined (`consumer_thread`, in this case) finishes before the main thread reaches the that line of code (line 90)?

A) The consumer_thread completes its work, but its resources are not freed. It goes into a zombie state until the main thread calls pthread_join. When the main thread calls pthread_join, the resources of consumer_thread are freed and an OS cleans up the consumer_thread.

5. In this program, the main thread calls `pthread_join()` on the threads it created. Could a different thread call `pthread_join()` on those threads instead? Could a thread call `pthread_join()` on the main thread (assuming it knew the main thread's thread ID - i.e. `pthread_t`)?

A) Yes, a different thread call `pthread_join()` on the threads if they are joinable, the main thread created, but it is not recommended because it may lead an undefined behavior, deadlocks, and logical errors.

6. The `consumer_routine` function calls `sched_yield()` (line 194) when there are no items in the queue. Why does it call `sched_yield()` instead of just continuing to check the queue for an item until one arrives?

A) For efficiency and responsiveness. If consumer_routine continues to check the queue, it takes cpu resource, so it is better to call sched_yield() to let another thread run. The producer_thread can work, so an item is added into the queue.