Q1. 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?

A1. exit() would result in termination of entire program, whereas pthread\_exit() would only terminate the calling thread.

Q2. The main function calls pthread\_join() (line 77) with the parameter thread\_return. Where does the value stored in thread\_return come from when the consumer\_thread is joined?

A2. The stored in thread\_return is equal to the value returned by consumer\_rouite(). thread\_return is of type “void \*” which it the type returned by consumer\_routine(). Reference to the pointer thread\_return is passed to thread\_join(), the value returned by consumer\_routine is copied to the location pointed by this reference.

Q3. Where does the value stored in thread\_return come from if the joined thread terminated by calling pthread\_exit instead of finishing normally?

A3. pthread\_exit() takes a “void \*” as a parameter. thread\_return gets that value when a thread terminates calling pthread\_exit().

Q4. On the same call to pthread\_join() (line 77), 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 77)?

A4. If the thread being joined finishes before a call to pthread\_join(), pthread\_join() returns immediately. But, it would successfully return the value returned by the thread.

Q5. 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)?

A5. Yes and Yes. It is to call pthread\_join() on any other thread.

Q6. The consumer\_routine function calls sched\_yield() (line 180) 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?

A6. sched\_yield() relinquishes control of CPU and moves the calling thread to the schedular queue. It is a good practice to free up the CPU for something “more” useful than just busy waiting. It helps in improving the overall performance of the system.