My program implements the producer-consumer problem. It creates four threads, one producer and three consumers, and uses a single linked queue as the data structure in which all of the information is stored. Each consumer thread has a color and type of number associated with it. The first is “red” paired with a prime number, the second is “green” paired with a Fibonacci number, and the third is “blue” paired with multiples of three and five.

The program uses a few mechanisms to synchronize the threads. First, I use a mutex lock which “sandwiches” the access to the queue (as well as the majority of what each thread does over all), locking immediately after the thread enters its main loop and unlocking immediately before the loop reverts to its beginning.
The second mechanism I use is a condition variable. Within the main loop of each thread there is a smaller while loop. The condition for this while loop is the condition that, when fulfilled, will release cause the thread to release its mutex to the other threads so that they can commence their part of the producer-consumer process. The order of the calls in this while loop and pthread_cond_signal() and then pthread_cond_wait(). This order is important, because if it were reversed, the executing thread would suspend itself and release the mutex before any of the other threads were ever signaled that they could begin.

I put a max length of five on my queue. The producer fills it from the input generator that we were provided with and once it is full, the inner while loop in the producer is entered and a statement prints telling the user that the producer is getting ready to pass of the mutex to the consumers. The consumers, who entered their inner while loop immediately (because the condition is when the queue is empty) then receive the mutex, pull the elements from the queue if the belong to them, and then re-enter the inner while loop when the queue is empty again and return the mutex to the producer. The producer then tells the user that the queue is empty (all five of its items have been dequeued) and that it is going to start filling again. This process will continue as long as the input generator creates input.