Summary:

This program demonstrates the classic reader writer issue when it comes to dealing with threads and sharing data. The goal is for each thread to get the correct information of the data that they are accessing. For instance, if a thread is writing to memory – changing the data – a reader thread cannot read that data since it is being altered. Similarly, another writer thread cannot write to that data [location] if another thread is writing to that data.

In this implementation, however, there is a reader precedence where if a reader thread is reading the data then other threads can read that same data. There is no issue with this. The writer thread is only able to write the data only when the last reader thread is done reading the data.

A mutual exclusion is used to only allow one thread at a time to write to the data. A mutual exclusion is also used in the reader threads to avoid a race condition when it comes to incrementing and decrementing the number of reader threads reading the data.

Discoveries:

I learned a lot about threads and how to create them and clean up the threads resources after the threads that were created are done with the pthread\_join function. I learned more about semaphores and what they are used for and how they “postpone” the execution of a section of code and allow multiple threads to execute that specific chunk of code in an orderly and individual fashion. I learned more about the race conditions and how semaphores are used to avoid these issues as well. I also learned that print statements are printed with respect to order differently on the Mac OSX operating system (Unix) versus the Linux operating system.