Homework 3

1.

a) no because sum_stat_a is not atomic and therefore modifying it could lead to a race condition if the program runs in parallel.

b) initialize a global variable mutex, and lock and unlock it at the beginning and end of the sum_stat function.

c)

2.

Since semaphores lock and unlock based on a counter, it can function as a mutex if implemented as a binary semaphore. Its value of either 0 or 1 can mimick a mutex's lock and unlock.

Semaphore are also able to function as condition variables because of its functions: up and down. These act similar to cv's wait and signal functions. Semantically, these functions are similar but the 'condition' that determines whether or not a semaphore is locked is its count.

3.

```
std::mutex sample_mutex;
std::mutex sum_mutex;
std::vector<double> samples;
double sample_sum;

// need to make sure the sum and sample vectors match, so they need to correspond to one another and be added / computed in the same order the instructions arrive.
// else, the average could be off.
void addSample(const double sample) {
    sample_mutex.lock(); // because of ^^ the two mutices need to be locked and unlocked together.
    sum_mutex.lock()
    if (std::isnan(sample)) {
        return;
    }
    samples.push_back(sample);
    sample_sum += sample;
    sample_mutex.unlock();
```

```
    sum_mutex.unlock()

    return;
}
double computeAverage() {
 sum_mutex.lock(); // while reading the average, no further
samples should be able to be added (else race condition)
 return sample_sum / samples.size();
 sum_mutex.unlock();
}
```

4.

n = 2 threads:

With n=2 threads, there would be a one ‘level’ buffer. Both threads enter the algorithm and one will enter the the buffer. Since this thread has reached the n-1 level, it obtains the lock. The other thread will continue to wait until the thread with the lock leaves its level and therefore releases the lock.

The peterson’s alg works as using bool vector and a victim variable. If another thread’s corresponding vector index is true but the victim is set to the current thread, it causes the current thread to busy wait until the flag is set to false. With n=2 threads, they two algorithms function similarly.

n threads:

With n threads, there would be n-1 levels buffer. The first level can hold n-1 threads which decrements as the levels progress. The singular thread at the end of the buffer obtains the lock. A thread can only advance to the next level if there are no threads in the next level or if another thread tries to enter the level that the thread wants to go into.