Many threads can read from a data structure—e.g., a list or tree—in parallel as long as the data structure is not being updated. Such parallelism can be safely enabled using reader/writer locks, as we discussed in class (also see Section 31.5 of OSTEP). We discussed an implementation of reader/writer locks using semaphores.

You should implement each of the reader/writer lock functions whose prototypes are shown below using only mutexes—i.e., you may NOT use condition variables or semaphores. You should also provide a definition for the rwlock struct.

```c
typedef struct rwlock rwlock_t;

// Called by a thread to initialize a reader/writer lock
void init(rwlock_t *rw);

// Called by a thread before reading
void read_lock(rwlock_t *rw);

// Called by a thread after it is done reading
void read_unlock(rwlock_t *rw);

// Called by a thread before writing
void write_lock(rwlock_t *rw);

// Called by a thread after it is done writing
void write_unlock(rwlock_t *rw);

struct rwlock {

};

void init(rwlock_t *rw) {

}
```
void read_lock(rwlock_t *rw) {
}

void read_unlock(rwlock_t *rw) {
}

void write_lock(rwlock_t *rw) {
}

void write_unlock(rwlock_t *rw) {