Problem 1: Concurrent Hash Table

```c
#define HASH_BUCKETS (1024)

typedef struct _hash_t {
  list_t hlists[HASH_BUCKETS];
} hash_t;

void Hash_Init(hash_t **H) {
  for (int i = 0; i < HASH_BUCKETS; i++) {
    List_Init(&H->hlists[i]);
  }
}

void Hash_Insert(hash_t **H, int key) {
  int b = key % HASH_BUCKETS;
  List_Insert(&H->hlists[b], key);
}

int Hash_Lookup(hash_t **H, int key) {
  int b = key % HASH_BUCKETS;
  return List_Lookup(&H->hlists[b], key);
}

How do you modify the above code to use locks? API:

lock_init(int *mutex);
lock(int *mutex);
unlock(int *mutex);
```

Problem 2: Building a Ticket Lock

**TEMPLATE: FILL THIS IN TO MAKE YOUR OWN LOCK**

```c
typedef struct _lock_t {
  // whatever data structs you need goes here
} lock_t;

void init(lock_t *lock) {
  // init code goes here
}

void acquire(lock_t *lock) {
  // lock acquire code goes here
}

void release(lock_t *lock) {
  // lock release code goes here
}

You may use yield() and FetchAndAdd(), defined as this:

```c
int FetchAndAdd(int *ptr) {
  int old = *ptr;
  *ptr = old + 1;
  return old;
}
```
Problem 3: Queue Lock

```
typedef struct __lock_t {
  int   flag; // state of lock: 1=held, 0=free
  queue_t *q; // queue of waiters
  int    guard; // use to protect flag, queue
};

void lock_init(lock_t *lock) {
  lock->flag = lock->guard = 0;
  lock->q   = queue_init();
}

void lock(lock_t *lock) {
  while (xchg(&lock->guard, 1) == 1) // spin
    if (lock->flag == 0) { // lock is free: grab it!
      lock->flag = 1;
      lock->guard = 0;
    } else { // lock not free: sleep
      queue_push(lock->q, gettid());
      lock->guard = 0;
      park(); // put self to sleep
    }
}

void unlock(lock_t *lock) {
  while (xchg(&lock->guard, 1) == 1) // spin
    if (queue_empty(lock->q))
      lock->flag = 0;
    else
      unpark(queue_pop(lock->q));
  lock->guard = 0;
}
```

(a) This spins on guard — why? (what is protected? what is not protected?)
(b) This still spins. Why is it better than a simple spin lock?
(c) In unlock, there is no setting of flag=0 when we unpark. Why?
(d) What is the race-condition bug in this code?

Reference: Condition Variables

```
mutex_t   lock; // declare a lock
cond_t    cv;  // declare a condition variable

A condition variable (CV) is a queue of waiting threads.
A single lock is associated with each CV (see below for usage).
There are two main operations that are important for CVs:

wait(cond_t *cv, mutex_t *lock)
  - assumes the lock is held when wait() is called
  - puts caller to sleep + releases the lock (atomically)
  - when awoken, reacquires lock before returning

signal(cond_t *cv)
  - wake a single waiting thread (if >= 1 thread is waiting)
  - if there is no waiting thread, just return w/o doing anything

A CV is usually PAIRED with some kind state variable
  - e.g., integer (which indicates the state of the system that we're interested in)
int    state; // related "state" variable (could be an int)
```