Problem 1: Broken Lock

```
void SpinLock(volatile unsigned int *lock) {
    while (*lock == 1) // TEST (lock)
        ; // spin
    *lock = 1;         // SET (lock)
}

void SpinUnlock(volatile unsigned int *lock) {
    *lock = 0;
}
```

Fill in the rest of the timeline with an example showing the lock is broken:

Thread 1	Thread 2
while(*lock == 1)	

Problem 2: Protecting Data Structures

```
typedef struct __node_t {
     int key;
     struct __node_t *next;
} node_t;

typedef struct __list_t {
     node_t *head;
} list_t;

void List_Init(list_t *L) {
     L->head = NULL;
}

void List_Insert(list_t *L, int key) {
     node_t *new = malloc(sizeof(node_t));
     assert(new);
     new->key  = key;
     new->next = L->head;
     L->head   = new;
}

int List_Lookup(list_t *L, int key) {
    node_t *tmp = L->head;
    while (tmp) {
       if (tmp->key == key) return 1;
       tmp = tmp->next;
    }
    return 0;
}
```

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

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

Problem 3: Building Your Own Lock

TEMPLATE: FILL THIS IN TO MAKE YOUR OWN LOCK

```
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
}
```

(a) using test-and-set (aka atomic-exchange)

```
// given ptr, sets *ptr to new value; returns the old value at *ptr
int Exchange(int *lock, int new) {
    int old = *lock;
    *lock = new;
    return old;
}
```

ANSWER:

```
typedef struct __lock_t { int flag; } lock_t;
void init(lock_t *lock) { lock->flag = 0; }
void acquire(lock_t *lock) {
    while(Exchange(&lock->flag, 1) == 1)
        ; // spin-wait (do nothing)
}
void release(lock_t *lock) { lock->flag = 0; }
```

(b) using compare-and-swap

```
int CompareAndSwap(int *ptr, int expected, int new) {
    int actual = *ptr;
    if (actual == expected)
        *ptr = new;
    return actual;
}
```

(c) using load-linked and store-conditional

```
int LoadLinked(int *ptr) {
    return *ptr;
}
int StoreConditional(int *ptr, int value) {
    if (no one has updated *ptr since LoadLinked to this address) {
        *ptr = value;
        return 1; // success
    } else {
        return 0; // fail (does not do the store)
    }
}
```