Race Condition Examples

Example 1: list_remove_front --- list * lp is a list with a single element with value 5.

Thread 0	**Thread 1**
Int num; List_element * element; assert(!is_empty(lp)); // list is NOT empty, so we can proceed	
CONTEXT SWITCH - note that T0 has NOT changed lp yet, and believes it is non-empty	

	Int num;
	
	List_element * element;
	
	assert(!is_empty(lp)) // list is NOT empty, so we can proceed
	
	Element = lp->first
	
	Num = lp->first->item = 5
	
	lp->first == lp->last // single element list
	
	lp->first = lp->last = NULL;
	
	lp->num_in_list --; // now equal to 0
	
	free(element)
	
	Return 5

CONTEXT SWITCH - note that list lp is now empty, but T0 does not know this	
Element = lp->first = NULL // since the list was emptied by T1 Num = lp->first->item **ERROR!** lp->first is NULL, we cannot dereference it!	

Example 2: list_append--- list * lp is an empty list

Thread 0 (appends 5)	**Thread 1 (appends 7)**

List_element *element = malloc ...
 	
element->item = 5
 	
assert(!is_in_list(lp, 5) // 5 is not in the list, so proceed
 	
is_empty(lp) // lp is empty!!	

CONTEXT SWITCH - note that T0 has NOT changed lp yet, and believes it is EMPTY	

	List_element *element = malloc ...
	
	element->item = 7
	
	assert(!is_in_list(lp, 7) // 7 is not in the list, so proceed
	
	is_empty(lp) // lp is empty!!
	
	lp->first = element; // lp->first->item = 7
	
	lp->last = element; // lp->last->item = 7
	// note first and last both point to the same element
	
	lp->num_in_list ++ // is now 1

CONTEXT SWITCH - note that list lp now has one element, but T0 does not know this	

lp->first = element; // lp->first->item = 5
 	
lp->last = element; // lp->last->item = 5
// note first and last both point to the same element
 	
lp->num_in_list ++ // is now 2
 	
MEMORY LEAK! Where is element with value 7? First and last both point to element 5! Also, there's only one element in the list, but the size is 2.	

Example 3: lock Acquire (Slide 13) --- lock is initially FALSE

Thread 0	**Thread 1**
While (*lock == true) // lock is FALSE, break	
CONTEXT SWITCH - note that T0 believes the lock is AVAILABLE	
	While (*lock == true) // lock is FALSE because T0 has not yet taken it, break *lock = true // lock is now taken by T1
CONTEXT SWITCH - note T1 owns the lock, but T0 believes the lock is free	
*lock = true // lock is now ALSO TAKEN by T0 !!! **NO MUTUAL EXCLUSION** !!! Two threads are now in the critical section!	