Programming Exercise 1 - Linked List

Objective

This exercise is designed to reinforce your understanding of C, in particular C pointers, and to implement a data structure that will be critical to your future success in this course. You are to write this implementation from scratch. In order to allocate memory and deallocate memory, you must use *malloc* and *free* from stdlib.h. In order to print to the console, you may use *printf* from stdio.h. It is also meant to force you to set up a C development, and debugging environment.

Assignment

Implement a stack based linked list in a C file named ***list.h*** with the following functions:

- struct linked_list* ll_create(void)
- int ll_destroy(struct linked_list *ll)
- void ll_add(struct linked_list *ll, int value)
- int ll_length(struct linked_list *ll)
- bool ll_remove_first(struct linked_list *ll)
- int ll_contains(struct linked_list *ll, int value)

- ll_create allocates and returns a pointer to a linked list. If unable to allocate the list, ll_create returns NULL. You must always check if malloc returns NULL.
- ll_destroy deallocates a linked list, only if it is empty. Return 1 if the linked list is destroyed, and 0 if it couldn't be destroyed.
- ll_add inserts a value at the head of the linked list.
- ll_length returns the total number of values in the linked list.
- ll_remove_first removes the value at the head of the linked list and returns true. If the list is empty, ll_remove_first returns false.
- ll_contains searches the linked list from head to tail and returns the first position at which the value is found. In a list with n values, the head is position 1 and the tail is position n; therefore, if the value is in the list, ll_contains returns a logical true, the offset into the linked list. If the value is not found in the list, 0 is returned; therefore, if the value is not in the list, ll_contains returns a logical false. Most often, this will be used to determine if a node is in the linked list, and not where it is in the list, thus the optimization to have it return 0 (logical false), and a 1-indexed offset otherwise (logical true).

Your solution should compile with no errors or warnings with *gcc -I. -Wall -Wextra -Werror main.c,* and should include no print statements in the header file. We have provided a ***Makefile*** for this assignment as an example to get you going. We will not provide this in future assignments. Regardless, you must always provide one. You should be able to compile with the command: *make*. *make* must properly compile your program, and generate a test binary. You can clean the directory with *make clean*.

You will receive no credit if your solution doesn't compile, if doesn't adhere exactly to the function prototypes provided. You will receive significant penalties if your solution compiles with warnings.

Testing

You will need to test your list.h. Do this by creating a separate file containing a main function, called **main.c**. You will need to test each of the functions that you have defined in varying ways. Consider that each function may behave differently when the list is empty and when the list is non-empty. Too simple a test may give the false impression that the list works properly in all cases when it contains a bug that is only exposed after the list has transitioned through several states. For example, testing the list by only adding values and then removing them fails to consider what happens when a list is emptied and then repopulated.

Additionally, well-structured test data may give the false impression that a function works properly. Real-world data is rarely well-structured. Therefore, do not rely on a simple test that uses a set like {1, 2, 3, 4, 5} as the results may be misleading. Your implementation will be evaluated against randomized data. We will test your program by simply *#include <list.h>* your implementation in our test harness. Any part of the specification that you don't test and have working, will likely lead to points off.

Submission

File names should be as per the instruction. Create *<anyname>.tar.gz* file containing your list.h, Makefile, main.c, and submit it to the blackboard. Use piazza for any clarification.