



Lab2 – Lists, Queues, & Stacks.

Important! Read the document “Information and rules about the labs” . Read through the entire lab specs before starting to write code so that you get an idea of what to do, the extent and can plan your work and time. Read each assignment carefully: it is important that you follow the instructions to get everything done correctly. If something is unclear, contact your lab assistant: Gabriele Capannini (gabriele.capannini@mdh.se) or Abu Naser Masud (masud.abunaser@mdh.se). You need to complete all the lab assignments to be eligible to attend the final lab examination.

1. Lists

Task 1.1 Compile the skeleton

Download and extract the file `Lab2.zip` . Create a new project with a proper name and add the `*list.*` files (i.e., `list.c`, `list.h`, and `test_list.c`) from the skeleton to your project. Make sure you can compile the project without any error messages (there may be warnings: just ignore them at the moment).

Task 1.2 Get acquainted with Lists

Now it's time to choose if you want to implement your linked list as single- or double-linked. Once you have decided, if your choice is to implement a double-linked list, go to `list.h` and “uncomment” the line `#define DOUBLE_LINKED_LIST`. Otherwise, if you have opted for the single-linked version, nothing needs to be done.

Read carefully `list.h` and `list.c` and think about how the different functions work. All functions specified in the interface and the auxiliary function named `createListNode` must be implemented. The functions in `list.h` aim to:

1. Create a new empty list.
2. Check if a list is empty or not.
3. Create a new list node by allocating memory and data (auxiliary function).
4. Add elements (of type `const Data`) first in the list.
5. Add elements (of type `const Data`) last in the list.
6. Remove the first item in the list (and release the memory).
7. Remove the last item in the list (and release the memory).
8. Remove a given element (of type `const Data`) from the list (and free the memory).
9. Determine if a given element (of type `const Data`) is in the list or not.
10. Count the number of nodes in the list.
11. Delete the entire list (free memory for all nodes)

12. Print the list (here you can assume that the data is an integer, int)

(*) In the part “ADTs” of this lab assignment, you will also implement functions that return the first and last node in the list (data in the node).

Remind that:

- The word `const` in front of an argument means that the function can not change the content of the argument. This is useful in conjunction with pointers to ensure that the function does not change data that the pointer points to. The reason for using the type of `const Data` as parameter is that the function is not allowed to change data in the implementation (regardless of whether there is a pointer).
- A function that does not make any change in the list, gets a `const list` as argument. A function that changes the list takes the list in the form of a `List*`. Keep in mind that `List` is defined as a pointer to a `Node` type.
- Consider pre- and post-conditions for your functions and use `assert` to verify them.
- Consider which special cases needed to handle, for example that ones when the list is empty.
- Your implementation must be able to fill the list of nodes, delete all nodes, and then add new nodes **without crashing the program!**

You can use the menu in `test_list.c` to test your linked list as you complete the features. If you want, you can of course write your own menu function. Last but not the least, check the test function found in `test_list.c` too.

Task 1.3 Implementation

Now is the time to implement. The code should be written in `list.c`. You cannot change in any way `list.h`. Consider the order in which the functions are implemented and test them as they are completed. Use the debugger when you have problems. Hint: use paper & pencil to draw a sketch of what you want to happen and what actually happens when the code is running. This is a way to get a better understanding of the code and functionality as well as finding logical problems.

Task 1.4 Testing

When all functions are implemented and working as intended, it is mandatory to run the test function in `test_list.c`. If `assert` is activated, you need to find out what errors are in your implementation (comments in `test_list.c` should help). If you do not get stuck in any asserts, the linked list works as the lab specs and you can proceed to task 2 on the lab.

2. ADTs

In this part, it is required that you write the implementations for the ADTs Stack, Queue, and Set by means of the linked lists you have implemented before. Use the `stack.h`, `stack.c`, `queue.h`, `queue.c`, `set.h`, `set.c`, and `test_queue_stack_set.c` files found in `Lab2.zip`. The files `list.h` and `list.c` must be included in the project. However, you can NOT change anything in the list files.

The file `test_queue_stack_set.c` contains test functions for each ADTs. These may not be changed. There are also menu functions for each ADTs.

If an assertion in the test function is activated, you need to find out which errors are present in your implementation. If you do not get stuck in any asserts, the tested ADT will work

Here it is important to reuse code, NOT copy or (re)write. Each function in each of the data structures (Stack, Queue, and Set) will be very short, only a few lines.

Good luck!