CS 2261 Lab 11: Linked Lists

Provided Files

- `list.c`
- `list.h`
- `test.c`

Files to Edit/Add

- `list.c`

Compilation

This assignment is to be written in C, but is not for the Game Boy Advance. Thus, you will need to compile it from the command line with the following command:

```
gcc *.c -o test
```

You can then run the tests in the provided `test.c` file (which you also compiled with the previous command) with the following command:

```
./test
```

This should print out the results of the tests. If it does not, alert a TA.

Instructions

In this lab, you will be implementing a doubly-linked list of integers in C. For the list we are creating, only positive integers are valid (otherwise either our `pop_front` case would break, or `scale_up` would break). After each TODO item, the code should compile and hopefully pass more of the tests in the provided test file. You do not need to complete an entire TODO block first. All of the code you write will be in `list.c`.

- **TODO 1 – Adding to the List**
  - Since the code to create the list has already been created for you, the first code you need to write is to add items to the list. Pushing to the front has already been written for you.
    - TODO 1.0: Find the `push_back` function and complete it.
  - Compile your code and test it. If the output of any of the `push_back` tests isn't what is expected, fix it before continuing.

- **TODO 2 – Removing from the List**
  - Now that we can add things to the list, we need to be able to take them out.
    - TODO 2.0: Find the `pop_front` function and complete it, then make sure it passes the tests.
      - Note: For our implementation, we return -1 if popping from an empty list (since we consider only positive integers to be valid, returning -1 means the list couldn't be popped).
    - TODO 2.1: Now that the `pop_front` function is working, we can use it to empty the whole list. Using `pop_front`, complete the `empty_list` function. If you do it correctly, the tests for both `empty_list` and `pop_front() again` will pass.
    - TODO 2.2: Finish out the removal code by completing `pop_back`. Again, return -1 if popping from an empty list.
  - Compile run the tests. You should pass all of the tests before `size`. **Important Note:** The tests do not check that you are using `free` correctly. You need to triple-check your code to make sure you are not leaking memory. Passing all tests does not guarantee a 100% on this lab.

- **TODO 3 – Getting Size**
  - Since we haven't been keeping up with the list's size as a variable in the other function, we need a function to calculate it.
    - TODO 3.0: Complete the `size` function. It should run in O(n) time; that is, it should take one step longer for every node in the list.
      - Note: a more efficient way, in the real world, is to keep up with size as a variable in the list struct. For this lab, we aren't doing that, since we want you to get practice traversing the list before TODO 4.
  - Compile and run the tests. You should be passing all but the last block.

- **TODO 4 – Writing a Traversal Function**
  - Just adding and removing isn't all that makes linked lists useful. We need to be able to run a function on each element in the list. The `traversal` function has already been written for you. It takes in the list and a pointer to a function that's run on each node.
  - TODO 4.0: Complete the `scale_up` function. Call the `traversal` function and pass in a helper function that will scale up the value in a node. You'll write that helper function next.
  - TODO 4.1: Write the previously mentioned helper function in `list.c`. It should take in an integer (the data in a node) and scale it up by a factor of 1024 (2^10) using bit shifting. Since we only consider positive integers to be valid in our list, you can safely assume that only positive integers are passed into this function.
    - **Important note:** If shifting the number by 10 would cause the number to overflow (go past INT_MAX, the maximum value for an integer), don't let it; instead, set it to INT_MAX.
    - **Hint:** You can't just shift the number up by 10 and check if it's greater than INT_MAX. It's not. No integer can be greater than INT_MAX. Find another way.
  - Compile and run the tests. For every one of them, the actual output should match the expected output. If so, zip up your files and submit.

Submission Instructions

For this assignment, all you have to do is submit your `list.c` file, but it won't hurt to submit other ones. In either case, zip up your files and submit the zip on Canvas. Name your submission Lab11_FirstameLastname, for example: "Lab11_RahasiaSandral.zip".