CSE101 Linked List Test Questions: Keeping your data connected

- All code must be written in C/C++.
- Please be careful about using built-in libraries or data structures. The instructions will tell you what is acceptable, and what is not. If you have any doubts, please ask the instructors or TAs.

1 Setting

You are already provided with a linked list code, that handles basic operations. So far, that is "insert", "delete", "print", and "length".

The I/O format has been fixed. The input and output files are given as command line arguments. Each line of the input file is an operation on the linked list. The "print" operation leads to printing of output in the output file.

For the linked list section test, you will have to implement one of the "advanced" functions described below. (These are based on actual interview questions.) *You cannot store the list in an array or some other data structure. You must manipulate the linked list directly to perform these functions.*

1.1 The test questions

There are five advanced operations that you need to study for the test. For your test, you will be given one of these to code up.

- `void dedup()`. This is a *deduplication* function, meaning that all duplicate values are deleted from the list. Specifically, for every value, you need to delete all *subsequent* occurrences from the list. For example, suppose the list was (in order) 10 20 10 50 20. On deduplication, the list becomes 10 20 50. It is important that you do not shuffle the values. This function just performs this operation, so there is no argument.
- `bool palindrome`. The function returns true if the list is a palindrome, and returns false otherwise. A palindrome is a list where the first and last elements are the same, the second and second last elements are the same, etc. Thus, the list 10 20 10 is a palindrome, while 10 10 20 is not. By default, the empty list and the list with one element are always palindromes.

• `Node* deletelast(int)`. Given an int argument, this function deletes the *last* occurrence of the element, and returns a pointer to the deleted node. It does not affect the order of anything else. Thus, on `deletelast(42)` in the list 7 42 101 42 50 3, we get 7 42 101 50 3. (The usual delete would result in 7 101 42 50 3.)

• `void rotate(int)`. Given a non-negative int argument, this function "rotates" the list by this value. Rotation by x means that all elements "move" their position by x, with the end "wrapping around" to the head. For example, suppose our list is 10 20 30 40. The function `rotate(1)` would lead to 40 10 20 30, while `rotate(2)` would lead to 30 40 10 20. Note that we can rotate by more than the length of the list, so `rotate(6)` would lead to 30 40 10 20. (You do not need to worry about negative inputs.)

• `void reverse(int)`: Given a non-negative int argument (say, val), this function reverses the order of the first val elements of the list, and leaves the others unchanged. Suppose our list is 10 20 30 40. The function `reverse(2)` would lead to 20 10 30 40, while the function `reverse(3)` leads to 30 20 10 40. If the argument is more than the length of the list, we reverse the entire list.

Other than `palindrome`, all other functions change the list. They do not return a new list, but perform the operation on the existing list. The `palindrome` operation will lead to printing "palindrome" or "not palindrome" in the console output and in the output file. You do not need to worry about implementating any of this; this is all taken care of in the wrapper file. But it will help you debug during the test.

1.2 How it works

The linkedlist.h file (that you will get in the test) will already have the right function declarations. Your job is to simply code up the function (that you will only get one as a test question) in the file linkedlist.cpp. Note that if you design more functions to help with your code, you may need to declare them as well in linkedlist.h.

The I/O will be taken care of, so there will be specific operations (in the input file) to perform these functions. So just like we already have "i <INT>" and "d <INT>", I will define more such syntax for all these functions. These will all be explained in a README file. You don't need to deal with any of this, but it might help you debug (if you run on your own inputs).

You will also get a small test input and test output. To make life easier, I will actually give you access to my grading scripts (that you can run directly through a shell script). If it catches a bug in your code, it will give you a test input where your code file. So you will get to see your score *before* you submit.

The test is completely closed book. You cannot bring any written material, but you can get blank scratch paper. You will get and can only open the specific Codio box for the test. You cannot refer to any other codes, or even open any other Codio box. If you do so, it will be considered cheating and you will get -10 points.

1.3 What should you do

Umm...how about coding these functions in the linked list Codio box I provided (after I teach linked lists)?

1.4 Other challenging questions

This is a collection of other questions, that might be good practice for exercising your linked list skills. (And great interview practice.) For these, you don't need to write code, but you might want to figure out the logic.

- Convert a linked list into a circular linked list.
- Implement the function `delete(val,k)`, where you delete the kth occurrence of val.
- Sort a linked list of integers in increasing order. (You can do this without changing pointers.)
- Insert an element at the end of the linked list (instead of at the head, as usual).
- Suppose you have a linked list in sorted order. Insert a new element at the correct position in sorted order. (Good coding exercise.)
- A more complicated version of `reverse(val)` (as defined earlier) is to reverse as follows: reverse the first val elements, reverse the second val elements, reverse the third val elements, so on and so forth.
- Find the middle element of the linked list, making only one pass over the linked list. (It's quite easy if you make two passes. For one pass, there's a cute trick.)
- A classic. Determine if a linked list has a loop. Closely related to previous question, if you get the trick.