1 Problem Description

This lab will continue the study of the new topic: linked list. The design of lab 6 is for a doubly linked list, but we have only worked on the forward direction. In this lab, we will complete the backward or reverse direction and manipulate a doubly linked list. In fact, in the standard library of C++ there is a list class (a template class, as in vector class) and its underlying structure is a doubly linked list.

By modifying the code of lab 6 (mainly augmenting in the main program, with minor changes to the show function), a doubly linked list has been built for us in the main.cpp file and you may want to study the code of building and showing a doubly linked list. We are asked to add some code to manipulate the doubly linked list.

2 Purpose

Understand the concept of linked list and in particular doubly linked list, link or pointer, and how to modify a doubly linked list.

3 Design

A very simple design of a node structure and the associated doubly linked list is provided for us as well as some code to build and to show the doubly linked list (see main.cpp file). Here is the declaration:

```c
typedef int element_type;
struct Node
{
    element_type elem;
    Node * next;
    Node * prev;
};
Node * head;
Node * tail;
```

Note `struct` keyword is equivalent to `class` keyword where all members are public.

If head is a pointer that points to an object of Node type, then `head->elem` (or `(head).elem`) refers to the elem field. Similarly `head->next` refers to the next field, the value of which is a pointer and may point to another node.
Note that in our program the head->next should always points to the first node of the doubly linked list, and tail->prev should always points to the last node of the doubly linked list. The two pointer variables head and tail point to the head node and tail node respectively, which are not the “data” nodes in the doubly linked list. Having the two extra nodes makes the implementation of a doubly linked list simpler. For a doubly linked list that has no element (empty list), the value of head->next is equal to tail and the value of tail->prev is equal to head. Draw diagrams of linked lists and understand the code provided before writing our code.

4 Implementation

After each modification, we will always call the show function and the reverse_show function, which will be implemented by us in the lab, to display the doubly linked list in both directions. This action will not be listed each time in the following to do list.

1. Write a ”void reverse_show(Node* head, Node* tail)” function by modifying the code of ”show” function. The only difference between the two functions is that reverse_show will output the node in reverse order, that is the first node is actually the last one, and the second node is the second to the last, and so on. (Hint: start with tail instead of head, follow the prev link instead of next link, testing against head instead of tail to determine the list is traversed).
2. Insert a node with elem value of 100 to the front of the list
3. Insert a node with elem value of -99 to the back of the list
4. Remove the first node from the front
5. Remove the first node from the back
6. Find the third node from the back and have a pointer variable, called tmp, to point to it (use a loop to do it, so we may easily change the third to the kth for any k)
7. Delete the node pointed by tmp from the doubly linked list, where tmp may point to any node in general and we should not use loop.

5 Test and evaluation

Testing should have been done during the implementation. You may want to think about if your solutions are still correct if say the list is empty during any of the operations.
6 Report and documentation

A short report about things observed and things learned and understood about the doubly linked list. Program should always be properly documented.

7 Lab submission

Get instructions from the Lab instructor.