Data Structures: 505 22240 / ESOE 2012

Computer Assignment 3

Doubly-Linked Lists

Due: the week after next by 9:00 p.m.

*Hand in online via **NTU COOL***

Total score: 300

This computer homework assignment is designed to give you practice working with writing doubly-linked lists and using subclasses. In this homework, you will implement a doubly-linked list ADT that allows an application to hold list nodes and hop from node to node quickly.

Part I (150)

DList.h contains a skeleton of a doubly-linked list class. Fill in the method implementations.

Your *DList* should be circularly-linked, and its head should be a sentinel node (which holds no item) as described in lecture. An empty *DList* is signified by a sentinel node that points to itself. Some *DList* methods return *DListNode*s; they should NEVER return the sentinel under any circumstances. Your *DList* should satisfy the following invariants.

1) For any *DList** d, d->head != NULL.
2) For any *DListNode** x in a *DList*, x->next != NULL.
3) For any *DListNode** x in a *DList*, x->prev != NULL.
4) For any *DListNode** x in a *DList*, if x->next == y, then y->prev == x.
5) For any *DListNode** x in a *DList*, if x->prev == y, then y->next == x.
6) For any *DList** d, the field d->size is the number of *DListNode*s, NOT COUNTING the sentinel, that can be accessed from the sentinel (d->head) by a sequence of "next" references.

The *DList* class includes a newNode() method whose sole purpose is to call the *DListNode* constructor. All of your methods that insert a new node should call this method; they should not call the *DListNode* constructor directly. This will help

minimize the number of methods you need to override in Part II.

Do not change any of the method prototypes; as usual, our test code expects you to adhere to the interface we provide. Do not change the fields of *DList* or *DListNode*. You may add helper methods as you please.

You are welcome to create a main() method program with test code. However, it will not be graded. We'll be testing your *DList* class, so you should too.

Part II (150)

Implement a "lockable" doubly-linked list ADT: a list in which any node can be "locked." A locked node can never be removed from its list. Any attempt to remove a locked node has no effect (not even an error message).

First, define a *LockDListNode* class that extends *DListNode* and carries information about whether it has been locked. *LockDListNode*s are not locked when they are first created. Your *LockDListNode* constructor(s) should call a *DListNode* constructor to avoid code duplication.

Second, define a *LockDList* class that extends *DList* and includes an additional method

public void lockNode(*DListNode** node) { ... }

that permanently locks "node".

DO NOT CHANGE THE SIGNATURE OF lockNode(). The parameter really is supposed to be of *static* type *DListNode*, not *LockDListNode*. I chose this signature for the convenience of the users of your *LockDList*. It saves them the nuisance of having to cast every node they want to lock. Instead, it's your job to take care of that cast (from *DListNode* to *LockDListNode*).

Your *LockDList* class should override just enough methods to ensure that

1) *LockDListNode*s are always used in *LockDList*s (instead of *DListNode*s), and
2) locked nodes cannot be removed from a list.

WARNING: To override a method, you must write a new method in the subclass with EXACTLY the same prototype. You can't change a parameter's type to a subclass. Overriding won't work if you do that.

Your overriding methods should include calls to the overridden superclass methods whenever it makes sense to do so. Unnecessary code duplication will be penalized.

Again, I recommend you test your code.

FINAL REMARK: The item of both lists of this assignment is designed to be a generic type. Make sure your extended classes in Part II follow this rule. I strongly suggest you test your lists with items stored in different types such as *string*, *int*, *double*, *char*, or even more complicated structures, which you may need to create more helper methods.

You are welcome to hand in the test code program associated with the results. It can help us to evaluate your code once your code fails in our test.