The University of Chicago
Department of Computer Science

CMSC 15200 – Introduction to Computer Science 2
Summer Quarter 2006
Homework #5 (08/09/2006)
Due: 08/11/2006 @ 5pm

Name:	
Student ID:	
Instructor:	Borja Sotomayor

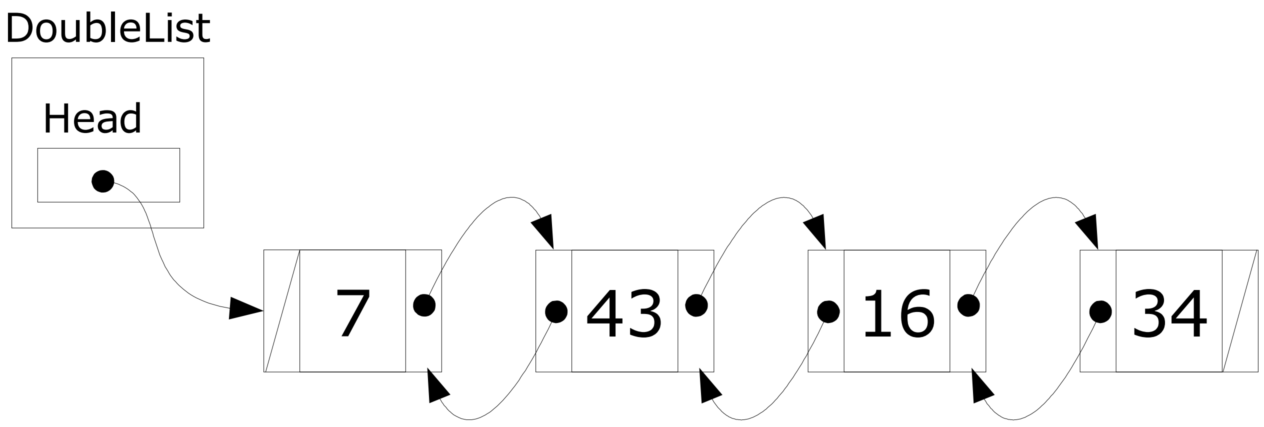
Do not write in this area

1	2	**TOTAL**

Maximum possible points: 45

Exercise 1 <<35 points>>

You will implement a doubly linked list (as described in class).



The structure and function declarations are the following (dlist.h in the homework files):

```
struct ListNode {
       int data;
       ListNode *next;
       ListNode *prev;
};

struct DList {
       ListNode *head;
};

void createList(DList &l);
ListNode* first(DList &l);
void insertHead(DList &l, int data);
void insertAfter(ListNode* node, int data);
void printData(DList &l);
bool find(DList &l, int data);
void deleteHead(DList &l);
void deleteAfter(ListNode* node);
void deleteData(DList &l, int data);
void deleteList(DList &l);

// New functions
/* Inserts a new node with provided data before the specified node */
void insertBefore(DList &l, ListNode* node, int data);
/* Deletes the specified node */
void deleteNode(DList &l, ListNode* node);
```

Don't reinvent the wheel. Reuse as much code as possible from the list implementation seen in class! In fact, you should only tweak the existing functions to make sure that the "prev" pointer always has a valid value, and then implement the new "insertBefore" and "deleteNode" functions.

To test your list implementation, a main_double.cpp is provided in the homework files. Running this program with a correct doubly linked list implementation should yield the following:

```
5 4 3 2 1 5 4 3 2 1
5 1 4 3 2 1 5 4 3 2 1
1
0
1 4 3 2 1 5 4 3 2 1
1 3 2 1 5 4 3 2 1
1 2 1 5 4 3 2 1
42 1 2 1 5 4 3 2 1
42 37 1 2 1 5 4 3 2 1
37 1 2 1 5 4 3 2 1
37 1 2 1 4 3 2 1
37 2 4 3 2
List is empty!
```

Note: Debugging programs that use data structures is no easy matter, and you are likely to encounter run-time errors due to dangling pointers. One good strategy is to draw the data structure on paper, and for each operation see how the "next" and "prev" pointers are affected. Don't forget to consider special cases (e.g. "What happens if I try to delete the *first* node?") Also, if your program crashes unexpectedly (the hallmark of a dangling pointer), the Eclipse debugger can come in handy to pinpoint the dangling pointer (the debugger will pause execution at the exact line that caused the crash).

Note 2: In your code, make sure you explicitly point out (with comments) what code you had to add/modify to turn the list into a doubly linked list.

Exercise 2 <<10 points>>

Given the list implementation seen in class, finding the length of a list can be a time-consuming operation, as we have to traverse the entire list to find out the length. Modify the list implementation seen in class so that finding the length of the list *never* requires that we traverse the list. Don't overthink this problem: it has a very easy solution. You will need to make some very simple modifications to the code. Also, you will need to add the following function:

```
int length(List &l);
```

To test your list implementation, a main_length.cpp is provided in the homework files. Running this program with a correct list implementation should yield the following:

```
5 4 3 2 1 5 4 3 2 1
Length: 10
5 1 4 3 2 1 5 4 3 2 1
Length: 11
1
0
1 4 3 2 1 5 4 3 2 1
Length: 10
1 3 2 1 5 4 3 2 1
Length: 9
1 3 2 1 4 3 2 1
Length: 8
3 2 4 3 2
Length: 5
List is empty!
Length: 0
```