## CSCI 2720

### Project 1: Templated DLL classes

### Project Due: July first before 11:59pm Points: 100

You will implement a templated list class, called DLL, in which the nodes are dynamically allocated linked structures. For the class, it should be possible to traverse the list in O(n) time, in either the forward or backward direction. That is, the "next" operation should take O(1) time, as should the "prev" operation.

* In the DLL class, each node should contain three fields: a "next" field, a "value" field of the templated type, and a "prev" field.

The public methods (member functions declared to have public access) that your list class needs to provide are:

1. a default constructor (creating an empty list);
2. a member function int IsEmpty( ) const, returning 1 if the list is empty and 0 if not;
3. a member function int Length( ) const, returning the number of elements in the list; this should run in O(1) not O(n), where n is the number of elements in the list
4. member functions void Push\_front(const Dtype & X) which adds X at the front of the list, and void Push\_back(const Dtype & X) which adds X at the back of the list;
5. member function void Pop\_front( ) which removes the front element of the list; and void Pop\_back( ) which removes the back element of the list;
6. member functions Dtype & Front( ) const which returns the front element of the list, and Dtype & Back( ) const which returns the back element of the list;
7. member function void Print( ) const which writes the elements of the list from front to back, to cout, without altering the list.
8. member function void PrintBack( ) const which writes the elements of the list from back to front, to cout, without altering the list.
9. member function void Iterate(void vist(Dtype & item)) which visits each element of the list and invokes the function pointed to by visit on the "value" field of the node.
10. member function void Reverse( ) which reverses the entire list without copying any data (use the "link inversion" method)
11. default destructor (deallocates all nodes using delete)

Your specification for the list should be in a file called list.h and your implementation code should be in a file called list.cc.

### Other useful files attached

Test program, [test1.cc](http://www.cs.uga.edu/%7Eeileen/2720/Projects/p1/test1.cc) and sample output [test1.out](http://www.cs.uga.edu/%7Eeileen/2720/Projects/p1/test1.out)

Another test program, [test2.cc](http://www.cs.uga.edu/%7Eeileen/2720/Projects/p1/test2.cc) and sample output [test2.out](http://www.cs.uga.edu/%7Eeileen/2720/Projects/p1/test2.out)

[Makefile](http://www.cs.uga.edu/%7Eeileen/2720/Projects/p1/Makefile) for test programs.

A [node.h](http://www.cs.uga.edu/%7Eeileen/2720/Projects/p1/node.h) if you wish to use it.

A [list1.h](http://www.cs.uga.edu/%7Eeileen/2720/Projects/p1/list1.h) if you wish to use it.

[Grading sheet](http://www.cs.uga.edu/%7Eeileen/2720/Projects/p1/p1grade.html)

**Testing**

You must test that your program compiles and runs correctly on nike using /usr/bin/g++. You are responsible for testing your program with our test cases and additional test cases that you create on your own. Thoroughly and rigorously test, test, and re-test your program before submitting it to ensure that it works correctly on any test cases that we may use for grading.