CS2134 HOMEWORK 6
Due* Monday March 21, 2016 at 11:00 p.m.

Be sure to include your name at the beginning of each file! Assignment 6 includes a programming portion and a written part. The programming portion must compile and consist of a single file (hw06.cpp). The written portion should consist of a single file (hw06written.pdf). It *must* be in a .pdf format. Be sure to include your name at the beginning of each file! You must hand in the file via NYU Classes.

Programming Part:

1. Add the following methods[1] to the List class:
   (a) The copy constructor, `List( const List & rhs )`. This method must take `O(n)` time.
   (b) The method `front( )`. It performs as stated in `www.cplusplus.com/reference/forward_list/forward_list/front/` This method must take O(1) time.
   (c) The method `merge( List & alist)`. It performs as stated in `www.cplusplus.com/reference/forward_list/forward_list/merge/`
   (d) The method `remove_adjacent_duplicates( )`. The method removes any element if it is adjacent to a node containing the same item[2]. Thus if the list contained $a, a, a, b, b, c, c, c, c$ afterwards it would contain a, b, c. If the list contains a, b, a, b, a then afterwards it contains a, b, a, b, a (i.e. no change, since no items adjacent to each other were the same).
   (e) The method `insert_after( iterator pos, Object && x)` which moves x into a new node which is located after `pos`. Your method must run in `O(1)` time.
   (f) The method `remove_if( Predicate pred )` that performs as stated in `www.cplusplus.com/reference/forward_list/forward_list/remove_if/` This method must run in `O(n)` time. Your method *should* call your method `erase_after`.

   A simplified version of the `List` class is in a file `simple-list.cpp`. We will test your code by including it in a driver program that uses the methods you implemented. You must test your code by writing and executing your own driver. Remember to hand in your driver code for any programming assignment.

2. Efficiently implement[3] a queue class called `Queue` using a singly linked lists, with no header *node* or tail *node*(i.e. nodes that contain no data). Your class should have the methods: `front, back, empty, enqueue, dequeue`.

3. (Extra Credit) Suppose that a doubly linked list class, `Dlist` is implemented with both a head and a tail node. Write a method[4] to remove all nodes containing `x`.

```
template<class Object>
void DList::remove(const Object & x)
```

   A simplified version of the `DList` class is in a file `simple-doubly-linked-list.cpp`.

*10% extra credit will be given if you turn this assignment in on Sunday March 20 at 11:00 p.m.
[1]Do written question 2 before this question in order to get experience writing pseudo code.
[2]If the list was sorted, the list would remove all duplicates.
[3]Please do written question 7 first.
[4]The `erase` method is *not* included in the simplified doubly linked list class I uploaded. I did not include it so you would get to practice working with pointers for a doubly linked list class. You may write your *own* `erase` method.

Written Part:

1. Draw the conceptual representation for our implementation of a link list (include the header) containing a single item `A`.

2. For the programming questions in 1 (except 1a)

   - draw pictures showing how the links change (or don't change) for programming questions
   - provide the pseudo code for the methods

3. For each of the following, determine if the iterator is valid.

```
vector<int> A = {1,2,3,4,5};
vector<int> B;
vector<int>::iterator vItr;
list<int> C = {1, 2,3,4,5};
list<int> D;
list<int>::iterator lItr;
```

   (a)
```
B = A;
vItr = B.begin();
B.erase(B.begin()+1);
```

   (b)
```
B = A;
vItr = B.begin()+2;
B.erase(B.begin()+1);
```

   (c)
```
D = C;
lItr = C.begin();
C.erase(++C.begin());
```

   (d)
```
 D = C;
lItr = ++D.begin();
++lItr
D.erase(++D.begin());
```

4. Which of the following code snippets are valid? If the code snippet is invalid, state why.

   (a)
```
list<int> l;
list<int>::iterator lIter;
l.push_back(200);
lIter = l.begin();
for (int i = 1; i < 100; ++i)
    l.push_front(i);
for (int i = 1; i < 100; ++i)
    l.push_back(-i);
cout << *lIter << endl;
```

   (b)
```
list<int> l;
list<int>::iterator lIter1;
list<int>::iterator lIter2;
list<int>::iterator mid;
for (int i = 0; i < 100; ++i)
    l.push_back(i);
lIter1 = l.begin();
lIter2 = l.end();
mid = lIter1 + (lIter2 - lIter1)/2;
cout << *mid << endl;
```

(c)
```
vector<int> v;
vector<int>::iterator vIter1;
vector<int>::iterator vIter2;
vector<int>::iterator mid;
for (int i = 0; i < 100; ++i)
   v.push_back(i);
vIter1 = v.begin();
vIter2 = v.end();
mid = vIter1 + (vIter2 - vIter1)/2;
cout << *mid << endl;
```

5. For the List class, what if the following code for the method remove was used. Would it work correctly? Explain.

```
void remove( const Object & x )
{
    Node * prev = header->next;
    while ( prev != nullptr )
    {
        if ( prev->next->data == x )
            erase_after( iterator(prev) );
        else
            prev = prev->next;
    }
}
```

6. For the `linked list` implementation of the `ADT stack`, show the conceptual representation of the contents of the stack for each line of code below:

```
Stack<char> s;
s.push('a');
s.push('b');
s.push('d');
s.pop();
s.push('c');
s.pop();
s.pop();
```

7. For your `linked list` implementation of the `ADT queue`, show the conceptual representation of the contents of the stack for each line of code below:

```
Queue<char> q;
q.enqueue('a');
q.enqueue('b');
q.enqueue('d');
q.dequeue();
q.enqueue('c');
q.dequeue();
q.dequeue();
```