EECE.3220 Spring 2017: Exam 2
Class Definitions and ADT Descriptions

ADT descriptions for Question 2

The `Stack` and `Queue` data types used in Question 2 hold integer values. Both data types support the following operation, defined as a member function:

- `bool empty()`: returns true if the stack or queue is empty
    - You should assume that a newly declared `Stack` or `Queue` starts as empty

The `Stack` type supports the additional operations below, defined as member functions:

- `void push(int v)`: Push the value `v` on the top of the stack
- `void pop()`: Pop the top item off of the stack, but do not return its value
- `int top()`: Return the value of the top item on the stack without removing anything from the stack

The `Queue` type supports the additional operations below, defined as member functions:

- `void enqueue(int v)`: Add the value `v` to the back of the queue
- `void dequeue()`: Remove a single item from the front of the queue
- `int front()`: Return the value of the item at the front of the queue without removing anything from the queue

Class definitions for Question 3

Static array-based list (used in Question 3a):

```
#define CAPACITY 1024

class AList {
public:
   AList();                           // Default constructor
   bool empty() const;                // Returns true if List empty
   void insert(int item, int pos);    // Add item at position pos
   void remove(int pos);              // Remove item from position
                                      //   pos

   // Operator to be written for Question 3a
   AList operator -(const AList &rhs)
private:
   int mySize;                  // Current size of list in myArray
   int myArray[CAPACITY];       // Array to store list elements
};
```

Class definitions for Question 3 (continued)

Dynamic array-based list (used in Question 3b):

```
class DList {
public:
   DList(int maxSize = 1024);                // Constructor
   ~DList();                                 // Destructor
   DList(const DList & origList);            // Copy constructor
   DList & operator=(const DList & origList); // Assignment
   bool empty() const;                       // Checks if empty
   void insert(int item, int pos);  // Add item at position pos
   void remove(int pos);            // Remove item from position
                                    //   pos

   // Operator to be written for Question 3b
   bool operator ==(const DList &rhs);
private:
   int mySize;                 // Current size of list in array
   int myCapacity;             // Capacity (max size) of array
   int *myArrayPtr;            // Pointer to dynamically-
                               //   allocated array
};
```

Linked list (used in Question 3c):

```
class LList {
public:
   LList();                    // Default constructor
   LList(LList &orig);         // Copy constructor
   ~LList();                   // Destructor
   LList & operator=(const LList &rhs);  // Assignment operator
   bool isEmpty();             // Returns true if list is empty
   void insert(int v);         // Add new value to list
   void remove(int v);         // Remove node with matching value

   // Operator to be written for Question 3c
   LList & operator~();
private:
   class Node {
   public:
      int val;  // Value in each node
      Node *next;    // Pointer to next node
   };

   Node *first; // Pointer to first node
};
```