EECE.3220 Fall 2019: Exam 2
Class Definitions and ADT Descriptions

Linked `Node` and `Stack` definition *(Questions 1a, 1b, 1c, 3a, 5)*

Notes:

- *No stack function definitions are shown because (1) I assume you understand what each function does, and (2) the ones you haven't seen before are functions you have to write!*
- *The array-based stack definition (for the extra credit problem) is on the second sheet of this document.*

```
template <class T>
class Node
{
public:
   Node(T v, Node<T> *p) : val(v), next(p) {}     // Constructor

   T getVal() { return val; }                     // Accessor for data

   Node<T> *getNext() { return next; }            // Accessor for next ptr
private:
   T val;              // Actual data stored in node
   Node<T> *next;      // Pointer to next node
};

template <class T>
class Stack {
public:
   Stack();                        // Constructor
   ~Stack();                       // Destructor
   bool empty() const;             // Checks if stack is empty
   void push(const T &val);        // Pushes data on top of stack
   void pop();                     // Removes top item from stack
   T getTop();                     // Accessor for data in top node
   unsigned size();                // QUESTION 1B—TO BE WRITTEN
   T getBottom();                  // QUESTION 1C—TO BE WRITTEN

   Stack<T> &operator =(const Stack<T> &rhs);     // Assignment
   bool operator ==(const Stack<T> &rhs);         // Equality test

   template <class T>
   friend ostream &operator <<(ostream &out, Stack<T> &aStack);

   Stack<T> &operator +=(const Stack<T> &rhs);    // QUESTION 3A
                                                  // TO BE WRITTEN
private:
   Node<T> *top;       // Node at top of stack
};
```

EECE.3220 Fall 2019: Exam 2
Class Definitions and ADT Descriptions

Array-based `Queue` definition *(Questions 2a, 2b, 3b)*
Note: no queue function definitions are shown because (1) I assume you understand what each function does, and (2) the ones you haven't seen before are functions you have to write!

```
template <class T>
class Queue {
public:
   Queue(unsigned maxSize = 1024);  // Constructor
   ~Queue();                        // Destructor
   bool empty() const;              // Checks if queue is empty
   void enqueue(const T &val);      // Adds data to back of queue
   void dequeue();                  // Removes data from front of queue
   T getFront();                    // Returns data at front of queue

   // We didn't write this operator in class, but assume it prints
   //   queue data from front to back, all on one line, with 1 space
   //   between each item and a newline at the end of the line
   // For example: 10 20 30 40
   template <class T>
   friend ostream &operator <<(ostream &out, Queue<T> &aQueue);

   bool operator ==(const Queue<T> &rhs);   // QUESTION 3B
                                            // TO BE WRITTEN
private:
   T* list;           // Dynamically allocated array to hold queue data
   int front, back;   // Front and back indexes
   unsigned cap;      // Capacity of dynamically allocated array
};
```

Linked list definition *(Questions 4b, 4c)*

```
class LList {
public:
   LList();                 // Default constructor
   ~LList();                // Destructor
   bool empty();            // True if list is empty
   void insert(int v);      // Add new value to list
   void remove(int v);      // Remove node with v

   void display(ostream &out);   // Print contents of list
private:
class Node {
   public:
      int val;              // Value in each node
      Node *next;           // Pointer to next node
   };

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

EECE.3220 Fall 2019: Exam 2
Class Definitions and ADT Descriptions

Array-based stack definition (`AStack`) *(Question 5)*

Notes:

- *This class has been changed as follows from the implementation shown in class:*
  - *The name is changed to `AStack` to allow linked stacks and array-based stacks to be used together in the same function.*
  - *The class is rewritten as a class template.*
  - *A prototype for the extra-credit question (!= operator) has been added.*
- *The constructor, push, and pop definitions are also shown below to remind you how this class is managed.*

```
template <class T>
class AStack {
public:
   AStack(unsigned maxSize = 1024);    // Constructor
   ~AStack();                          // Destructor
   bool empty() const;                 // Returns true if stack empty
   void push(const T &val);            // Push val to top of stack
   void pop();                         // Remove top of stack
   T top();                            // Read contents of top of stack

   // EXTRA CREDIT QUESTION--COMPARE ARRAY-BASED STACK TO LINKED STACK
   // RETURNS TRUE IF THEY DON'T MATCH, FALSE OTHERWISE
   bool operator !=(const Stack<T> &rhs);
private:
   T *list;      // The actual data stored on the stack
   int tos;      // Index for top of stack
   unsigned cap; // Capacity (max size) of stack
};

template <class T>
AStack<T>::AStack(unsigned maxSize) : tos(-1), cap(maxSize) {
   list = new T[cap];
}

template <class T>
void AStack<T>::push(const T &val) {
   if (++tos < cap)
      list[tos] = val;
   else
      tos--;
}

template <class T>
void AStack<T>::pop() {
   if (!empty())
      tos--;
}
```