CSE143 Summer 2008 Midterm Exam
July 25, 2008

Name : _____

Section (eg. AA) : _____ TA : _____

This is an open-book/open-note exam. Space is provided for your answers. Use the backs of pages if necessary. The exam is divided into six questions with the following points:

Question	Points	Score
Recursive Tracing	15	
Recursion	15	
Linked Lists	15	
More Linked Lists	20	
Stacks and Queues	20	
Array Manipulation	15	
Total:	100	

Do not begin work on this exam until instructed to do so. Any student who starts early or who continues to work after time is called will receive a 10 point penalty.

The exam is not, in general, graded on style and you do not need to include comments. For the stack/queue question, however, you are expected to use generics properly and to declare variables using interfaces when possible.

Please turn off an cell phones or other devices that might disturb others during the exam. You are NOT to use any electronic devices while taking the test, including calculators. Anyone caught using an electronic device will receive a 10 point penalty.

If you finish the exam early, please hand your exam to the instructor and exit quietly through the front door.

1. (15 points) **Recursive Tracing**
   Write what the method returns, given the specified inputs. If the method will enter an infinite recursion, say "infinite recursion".

```
public static String mystery(int x, int y) {
    if(x == y)
        return "!";
    if(x == y+1)
        return "**";
    return mystery(x+1,y) + mystery(x+2,y);
}
```

`mystery(6,5)` : _____

Solution: **

`mystery(-5,-4)` : _____

Solution: !**

`mystery(3,1)` : _____

Solution: infinite recursion

`mystery(4,6)` : _____

Solution: !**!

`mystery(3,6)` : _____

Solution: !**!!**

2. (15 points) **Recursion**

Write a recursive method isReversal that takes two strings as parameters. The method should return true if the second string is the first string reversed. Your method should throw an IllegalArgumentException if the input strings are not the same length.

You are not allowed to construct any structured objects other than Strings (no array, StringBuilder, ArrayList, etc) and you may not use a while loop, for loop or do/while loop to solve this problem; you must use recursion.

Four methods of String you may find helpful are:

substring(int begin, int end) - returns a String starting at begin and going to end - 1
equals(String otherString) - returns true if the two strings are the same
length() - returns the length of the string
charAt(index) - returns the character at the specified index

Examples:

isReversal("foo","oof") returns true
isReversal("foaob","boqof") returns false

Solution:

```
public static boolean isReversal(String first, String second)
{
    int length = first.length();
    if(length != second.length())
        throw new IllegalArgumentException("different lengths!");

    if(length == 0)
        //empty strings are always the reverse of each other
        return true;

    if(first.charAt(0) != second.charAt(length - 1))
        return false;

    return isReversal(first.substring(1,length),
                      second.substring(0,length - 1));
}
```

3. (15 points) **Linked Lists**

Write a method hasDuplicates for the LinkedIntList we discussed in class. This method returns true if there is at least one element duplcated in the list. A list of 0 or 1 elements has no duplicates by definition.

You are writing a method for the LinkedIntList class discussed in lecture:

```
public class ListNode {
    public int data;       // data stored in this node
    public ListNode next;  // link to next node in the list

    <constructors>
}

public class LinkedIntList {
    private ListNode front;

    <methods>
}
```

You may not call any other methods of the LinkedIntList class to solve this problem

Examples:

For a list containing [1, 2, 1], hasDuplicates returns true
For a list containing [1, 2, 3], hasDuplicates returns false
For a list containing [3, 2, 5, 4, 2], hasDuplicates returns true

Solution:

```
public boolean hasDuplicates() {
    ListNode current = front;
    while(current != null) {
        ListNode comparison = current.next;
        while(comparison != null) {
            if(comparison.data == current.data)
                return true;
            else
                comparison = comparison.next;
        }
        current = current.next;
    }
    return false;
}
```

4. (20 points) **More Linked Lists**

Write a method addListAt for the LinkedIntList class (see previous page for a specification). This method takes a ListNode, which is the head of another linked list, and a integer index.

The method should modify the LinkedIntList so that the passed in list is inserted into the existing list at the index specified. If index is 0 it should be inserted at the head of the list. If index is 1 it should be inserted after the 1st element, etc. Your method should throw an IllegalArgumentException if the index is negative. Your method should throw an IllegalArgumentException if the index is greater than the length of the LinkedIntList.

You should modify the existing ListNode objects and should not need to create any new ones or use any auxilary structures. You should not call any other methods on the LinkedIntList object.

Examples:

If a LinkedIntList containing [7, 11, 13, 19] has the list [8, 20] added to it at index 2, the original LinkedIntList will become [7, 11, 8, 20, 13, 19].

If a LinkedIntList containing [8, 20] has the list [7, 44, -6] added to it at index 0, the orignal LinkedIntList will become [7, 44, -6, 8, 20].

If a LinkedIntList containing [1, 1] has the list [2, 2, 2] added to it at index 2, the original LinkedIntList will become [1, 1, 2, 2, 2].

Solution:

```
public void addListAt(ListNode list, int index) {
    if(index < 0) throw newIllegalArgumentException("index negative");
    if(list == null) return; //nothing to add
    ListNode oldListAfterIndex = front;
    ListNode oldListBeforeIndex = null;

    for(int i = 0; i < index; i++) {
        if(oldListAfterIndex == null)
            throw new IllegalArgumentException("index beyond list");
        oldListBeforeIndex = oldListAfterIndex;
        oldListAfterIndex = oldListAfterIndex.next;
    }

    ListNode newListLast = list;
    while(newListLast.next != null)
        newListLast = newListLast.next;

    if(oldListBeforeIndex == null)
```

```
        front = list;
    else
        oldListBeforeIndex.next = list;

    newListLast.next = oldListAfterIndex;
}
```

5. (20 points) **Stacks and Queues**

Write a function getMax that takes a stack and a queue as parameters and returns the largest integer stored in either the stack or the queue. Your method should be sure to restore both the stack and the queue to their orginal state.

You may use one additional stack to help you solve this problem, but no other auxiliary structures.

Examples:

For input stack [7, 11, 9] and queue [2, 777, 1] getMax returns 777.

For input stack [ ] and queue [-2, -13, -6] getMax returns -2.

Solution:

```
public int getMax(Stack<Integer> stack, Queue<Integer> queue) {
    int max = 100;
    boolean maxValid = false;

    for (int i = 0; i < queue.size(); i++) {
        int currentVal = queue.dequeue();
        if(!maxValid || max < currentVal) {
            maxValid = true;
            max = currentVal;
        }
        queue.enqueue(currentVal);
    }

    Stack<Integer> auxStack = new ArrayStack<Integer>();
    while(!stack.isEmpty()) {
        int currentVal = stack.pop();
        if(!maxValid || max < currentVal) {
            maxValid = true;
            max = currentVal;
        }
        auxStack.push(currentVal);
    }
    while(!auxStack.isEmpty())
        stack.push(auxStack.pop());
    if(!maxValid)
        throw new IllegalArgumentException("stack and queue empty");
    return max;
}
```

6. (15 points) **Array Manipulation**
Write a method copyMultiple that takes two parameters: an int array and integer timesToCopy. The method should return a new int array with the original array duplicated timesToCopy times. Your method should throw an IllegalArgumentException if timesToCopy is less than 1.

Examples:

copyMultiple([7,17],2) returns [7, 17, 7, 17]
copyMultiple([45],4) returns [45, 45, 45, 45]
copyMultiple([66,-3,11],2) returns [66,-3,11,66,-3,11]

Solution:

```
public int[] copyMultiple(int[] array, int timesToCopy) {
    if(timesToCopy < 1)
        throw new IllegalArgumentException("timesToCopy < 1");
    int[] result = new int[array.length*timesToCopy];
    for(int sourceIndex = 0; sourceIndex < array.length; sourceIndex++)
    {
        for(int copyNum = 0; copyNum < timesToCopy; copyNum++)
        {
            int val = array[sourceIndex];
            result[copyNum*array.length + sourceIndex] = val;
        }
    }
    return result;
}
```