CSE 143, Winter 2011
Midterm Exam
Friday, February 11, 2011

Personal Information:

Name: _____

Section: _____ **TA:** _____

Student ID #: _____

- You have 50 minutes to complete this exam.
  You may receive a deduction if you keep working after the instructor calls for papers.
- This exam is open-book/notes. You may not use any computing devices including calculators.
- Code will be graded on proper behavior/output and not on style, unless otherwise indicated.
- Do not abbreviate code, such as "ditto" marks or dot-dot-dot ... marks.

  The only abbreviations that *are* allowed for this exam are:
  - `S.o.p` for `System.out.print, and`
  - `S.o.pln` for `System.out.println.`
- You do not need to write `import` statements in your code.
- If you write your answer on scratch paper, please clearly write your name on every sheet and write a note on the original sheet directing the grader to the scratch paper. We are not responsible for lost scratch paper or for answers on scratch paper that are not seen by the grader due to poor marking.
- If you enter the room, you must turn in an exam before leaving the room.
- You must show your Student ID to a TA or instructor for your exam to be accepted.

Good luck!

Score summary: (for grader only)

Problem	Description	Earned	Max
1	ArrayList Mystery		20
2	Recursive Tracing		20
3	Stacks and Queues		15
4	Collections		15
5	Linked Lists		15
6	Recursive Programming		15
TOTAL	**Total Points**		**100**

1. `ArrayList` Mystery

Write the output produced by the following method when passed each of the following `ArrayLists`:

```
public static void mystery(ArrayList<Integer> list) {
    for (int i = 0; i < list.size(); i++) {
        int n = list.get(i);
        if (n % 10 == 0) {
            list.remove(i);
            list.add(n);
        }
    }
    System.out.println(list);
}
```

List	Output
a) `[1, 20, 3, 40]`	
b) `[80, 3, 40, 20, 7]`	
c) `[40, 20, 60, 1, 80, 30]`	

2. Recursive Tracing

Write the output produced by the following recursive method when passed each of the following integer parameters:

```
public static int mystery(int n) {
    if (n <= 1) {
        System.out.print(": ");
    } else {
        System.out.print((n % 2) + " ");
        mystery(n / 2);
        System.out.print(n + " ");
    }
}
```

Call	Returns
a) `mystery(8);`	
b) `mystery(25);`	
c) `mystery(46);`	

3. Stacks and Queues

Write a method **`reverseFirstK`** that accepts an integer *k* and a queue of integers as parameters and reverses the order of the first *k* elements of the queue, leaving the other elements in the same relative order. For example, suppose a variable `q` stores the following elements:

```
front [10, 20, 30, 40, 50, 60, 70, 80, 90] back
```

The call of `reverseFirstK(4, q);` should change the queue to store the following elements in this order:

```
front [40, 30, 20, 10, 50, 60, 70, 80, 90] back
```

If *k* is 0 or negative, no change should be made to the queue. If the queue passed is `null` or does not contain at least *k* elements, your method should throw an `IllegalArgumentException`.

For full credit, obey the following restrictions in your solution. A solution that disobeys them can get partial credit.

- You may use **one queue or stack** (but not both) as auxiliary storage.
  You may not use other structures (arrays, lists, etc.), but you can have as many simple variables as you like.
- Use the `Queue` interface and `Stack`/`LinkedList` classes discussed in lecture.
- Use stacks/queues in stack/queue-like ways only. Do not call index-based methods such as `get`, `search`, or `set` (or for-each) on a stack/queue. You may call only `add`, `remove`, `push`, `pop`, `peek`, `isEmpty`, and `size`.

You have access to the following two methods and may call them as needed to help you solve the problem:

```
public static void s2q(Stack<Integer> s, Queue<Integer> q) { ... }
public static void q2s(Queue<Integer> q, Stack<Integer> s) { ... }
```

4. Collections

Write a method **`duplicateValues`** that accepts a `Map` from strings to strings as a parameter and returns a count of how many values in the map are duplicates of another value in the map. For example, if a variable named `map` contains the following key/value pairs:

```
{Marty=Stepp, Jessica=Miller, Mike=Stepp, Stuart=Reges, James=Hale, Amanda=Camp,
 Hal=Perkins, Biz=Camp, Kendrick=Perkins, Sam=Perkins, Eve=Riskin, Jane=Perkins}
```

In the above map, there are two occurrences of the value Stepp, so one (1) of them is a duplicate; and there are four occurrences of the value Perkins, so three (3) of them are duplicates. Therefore the call of `duplicateValues(map)` would return `4`. (One way of thinking of it is that we could hypothetically remove 1 occurrence of Stepp and 3 occurrences of Perkins and those values would still be represented in the map.) If the map contains no duplicate values or contains no values at all, your method should return `0`. Note that we are asking about duplicate *values*, not keys. You should not make any assumptions about the order in which the keys or values appear in the map. You may assume that the map is not `null` and that none of its keys or values are `null` strings.

For full credit, obey the following restrictions in your solution. A solution that disobeys them can get partial credit.

- You may create only **up to two (2) new data structures of your choice** (list, stack, queue, set, map, array, etc.) as auxiliary storage. (You can have as many simple variables as you like.)
- Your solution should run in strictly less than $O(N^2)$ time, where N is the number of pairs in the map.
- You should not modify the contents of the map passed to your method.

---

5. Linked Lists

Write a method **`frontToBack`** that could be added to the `LinkedIntList` class that moves the first element of the list to the back end of the list. Suppose a `LinkedIntList` variable named `list` stores the following elements from front to back:

```
[18, 4, 27, 9, 54, 5, 63]
```

If you made the call of `list.frontToBack();`, the list would then store the elements in this order:

```
[4, 27, 9, 54, 5, 63, 18]
```

If the list is empty or has just one element, its contents should not be modified.

For full credit, obey the following restrictions in your solution. A solution that disobeys them can get partial credit.

- **Do not call any other methods** on the `LinkedIntList` object, such as `add`, `remove`, or `size`.
- **Do not create new `ListNode` objects** (though you may have as many `ListNode` variables as you like).
- Do not use other data structures such as arrays, lists, queues, etc.
- Your solution should run in O(*N*) time, where *N* is the number of elements of the linked list.

Assume that you are adding this method to the `LinkedIntList` class (that uses the `ListNode` class) below.

```
public class LinkedIntList {
    private ListNode front;
    ...
```

```
public class ListNode {
    public int data;
    public ListNode next;
    ...
```

6. Recursion Programming

Write a recursive method **`largestDigit`** that accepts an integer parameter and returns the largest digit value that appears in that integer. Your method should work for both positive and negative numbers. If a number contains only a single digit, that digit's value is by definition the largest. The following table shows several example calls:

Call	Returns
`largestDigit(1426**6**3203)`	6
`largestDigit(**8**45)`	8
`largestDigit(5264**9**)`	9
`largestDigit(**3**)`	3
`largestDigit(**0**)`	0
`largestDigit(-5**7**3026)`	7
`largestDigit(-**2**)`	2

For full credit, obey the following restrictions in your solution. A solution that disobeys them may get partial credit.

- You may not use a `String`, `Scanner`, array, or any data structure (list, stack, map, etc.).
- Your method must be recursive and not use any loops (`for`, `while`, etc.).
- Your solution should run in no worse than O(*N*) time, where *N* is the number of digits in the number.

Scratch Paper

(If you write an answer to a test question here, please also go back to that question's original page and leave a note telling the grader to refer to this page. Otherwise the grader might not notice all of your work.)

Scratch Paper

(If you write an answer to a test question here, please also go back to that question's original page and leave a note telling the grader to refer to this page. Otherwise the grader might not notice all of your work.)