CSE 143: Computer Programming II — Spring 2015

Midterm Exam Solutions

Name:	**Sample Solutions**		
ID #:	**1234567**		
TA:	**The Best**	**Section:**	**A9**

INSTRUCTIONS:

- You have **50 minutes** to complete the exam.
- You *will* receive a deduction if you keep working after the instructor calls for papers.
- This exam is closed-book and closed-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.
- You may not use scratch paper on this exam. If you need extra space, use the back of a page.
- 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.
- If you get stuck on a problem, move on and come back to it later.

Problem	Points	Score	Problem	Points	Score
1	14		4	20	
2	16		5	16	
3	14		6	20	
			Σ	100	

Question #1: Scratch Work

This page left intentionally almost Adam.

Mechanical Missions.

This section tests whether you are able to trace through code of various types in the same way a computer would.

1. Recursive Tracing [14 points]

```
1  public static void mystery(int n) {
2     if (n == 1) {
3        System.out.print(n);
4     }
5     else {
6        System.out.print(n + ", ");
7        if (n % 3 == 0) {
8           mystery(n / 3);
9        }
10       else {
11          mystery(2 * n - 1);
12       }
13    }
14 }
```

For each of the following, fill in the String *printed* when the given `mystery` method call is run.

	`mystery` Call	String Printed
(a)	`mystery(1)`	**1**
(b)	`mystery(2)`	**2, 3, 1**
(c)	`mystery(9)`	**9, 3, 1**
(d)	`mystery(14)`	**14, 27, 9, 3, 1**
(e)	`mystery(15)`	**15, 5, 9, 3, 1**

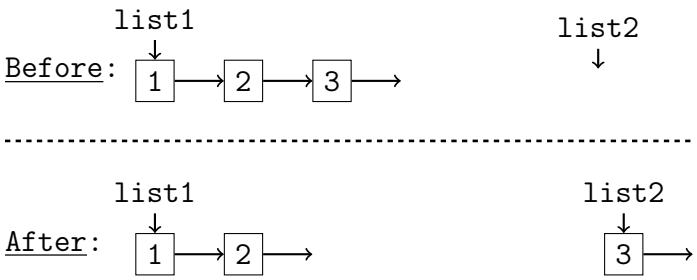
Question #2: Scratch Work (Hint: Draw Pictures)

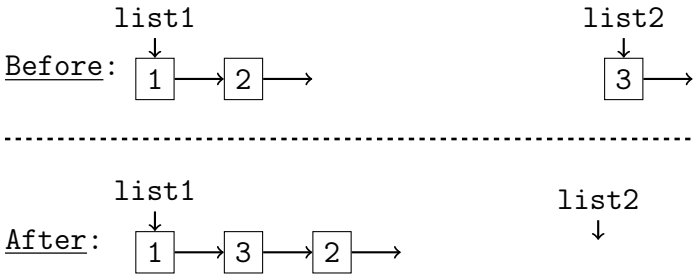
This page left intentionally almost Adam.

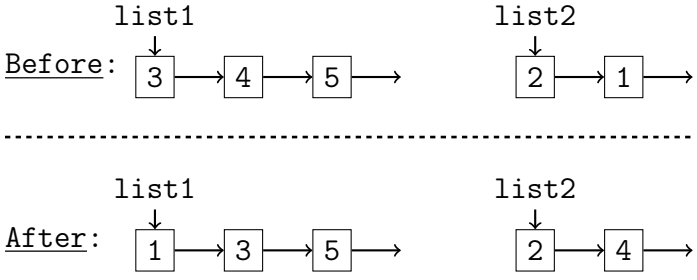
2. `ListNode` **Before & After** [16 points]

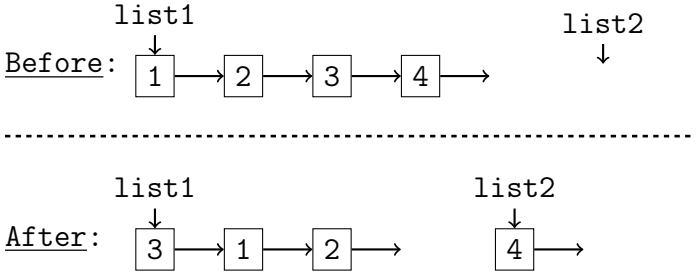
Your task in this question is to write code to turn the "before" picture into the "after" picture. You may not change any existing node's data value. You may not construct any new nodes. In each part, you may declare *up to two* new variables of type `ListNode`. Recall the `ListNode` class from lecture:

```
1 public class ListNode {
2    public final int data;
3    public ListNode next;
4
5    public ListNode(int data) { ... }
6    public ListNode(int data, ListNode next) { ... }
7 }
```

	Before & After	Your Code
(a)		`list2 = list1.next.next;` `list1.next.next = null;`

(b)		`list2.next = list1.next;`
		`list1.next = list2;`
		`list2 = null;`

(c)		`list2.next.next = list1;`
		`list1 = list2.next;`
		`list2.next = list1.next.next;`
		`list1.next.next = list1.next.next.next;`
		`list2.next.next = null;`

(d)		`LN three = list1.next.next;`
		`list2 = three.next;`
		`three.next = list1;`
		`three.next.next.next = null;`
		`list1 = three;`

Programming Pursuits.

This section tests whether you synthesized various topics well enough to write novel programs using those topics.

3. Replacing the Min-utes [14 points]

Add a method `replaceMin` to the `ArrayIntList` class that removes the *minimum* element from the list and replaces the hole with the element from the end of the list (if such an element exists). If the list contains the minimum element multiple times, you should remove *only* the *first copy*. If the list is empty, it should remain unchanged.

Example Output

Before `list.replaceMin()`	**After** `list.replaceMin()`
`[1, 3, 2, 7]`	`[7, 3, 2]`
`[-2, 5, 5, -5]`	`[-2, 5, 5]`
`[42, 5, 5, 99]`	`[42, 99, 5]`

Implementation Restrictions

- You may not call any other methods on the `ArrayIntList` object (e.g., `add`, `remove`)
- You may not use a foreach loop on the `ArrayIntList`.
- You may not use any other data structures such as arrays, lists, queues, etc.
- Your solution should run in $\mathcal{O}(n)$ time, where n is the number of elements in the list.

```
public class ArrayIntList {
   private int size;
   private int[] data;

   // Write Your Solution Here




}
```

Question #3: Extra Solution Space

Solution:

```
1  public void replaceMin() {
2     if (this.size > 0) {            // Do nothing in case where size == 0
3        // Find-Min Part
4        int minIdx = 0;              // Use first element
5        for (int i = 0; i < this.size; i++) {
6           if (this.data[minIdx] > this.data[i]) {
7              minIdx = i;
8           }
9        }
10
11       //Replace-Min Part
12       if (this.size > 1) {
13          this.data[minIdx] = this.data[size - 1];
14       }
15       this.size--;
16    }
17 }
```

This page left intentionally almost Adam.

4. [RR][ee][cc][uu][rr][ss][ii][oo][nn] [20 points]

Write a *recursive* method called `bracketXX` that takes a `String` as a parameter and returns a new string with brackets around all occurrences of identical adjacent characters.

Example Output

Method Call	Return Value
`bracketXX("odegaard")`	`"odeg[aa]rd"`
`bracketXX("mississippi")`	`"mi[ss]i[ss]i[pp]i"`
`bracketXX("who am I??")`	`"who am I[??]"`
`bracketXX(".. . .-.. .-..")`	`"[..] . .-[..] .-[..]"`
`bracketXX("oopsies")`	`"[oo]psies"`
`bracketXX("foobar")`	`"f[oo]bar"`
`bracketXX("")`	`""`

Implementation Restrictions

- You must use recursion.
- You may assume that the input `String` is not `null`.
- You may assume that no letter appears more than two times in a row in the input `String`.
- You may only use the `String` methods on the cheatsheet.

Question #4: Extra Solution Space

Solution:

```
1  public static String bracketXX(String s) {
2     // Base Cases
3     if (s.length() < 2) { return s; }
4     // Found Case
5     else if (s.charAt(0) == s.charAt(1)) {
6        return '[' + s.substring(0, 2) + ']' + bracktXX(s.substring(2));
7      }
8     // Not Found Case
9     else { return s.charAt(0) + bracketXX(s.substring(1)); }
10 }
```

This page left intentionally almost Adam.

5. I'm The Map! [16 points]

Write a method called `indexMap` that takes a `List` of `Strings` as a parameter and returns a map that associates each `String` from the `List` to a `Set` of indices that string occurs at.

The map returned by your method should be `ordered alphabetically`, and the sets that each string maps to should *be as fast as possible*.

Example Output

Input List	Return Value
`[]`	`{}`
`[to, be, or, not, to, be]`	`{be=[1, 5], not=[3], or[2], to=[0, 4]}`

Implementation Restrictions

- You may assume that the list passed is not null (and that no value in the list is null).
- Your method should not modify the list of words passed as a parameter
- You may not create and structures other than the ones described above.
- Your method should be reasonably efficient.

Question #5: Extra Solution Space

Solution:

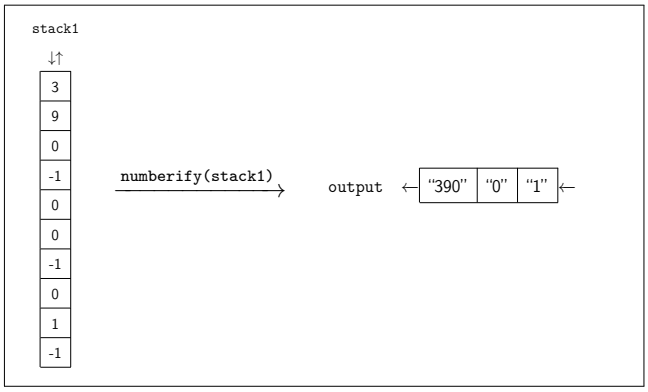
```
1  public static Map<String,Set<Integer>> indexMap(List<String> list) {
2     // Return Value
3     Map<String, Set<Integer>> map = new TreeMap<String, Set<Integer>>();
4
5     // Looping through the List
6     for (int i = 0; i < list.size(); i++) {
7        String s = list.get(i);
8        if (!map.containsKey(s)) {
9           map.put(s, new HashSet<Integer>());
10       }
11       map.get(s).add(i);
12    }
13
14    returns map;
15 }
```

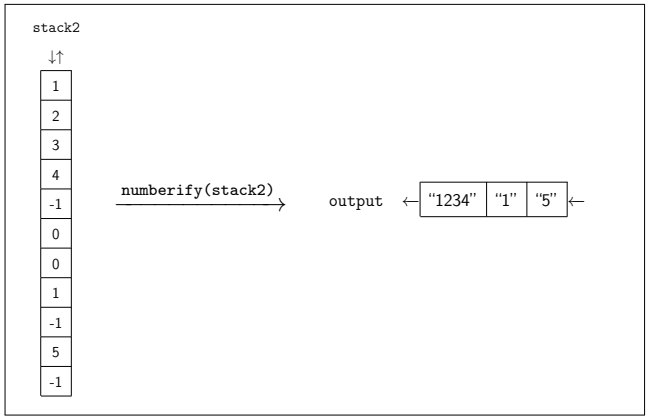
This page left intentionally almost Adam.

6. Number Me Once... [20 points]

Write a method `numberify` that takes a `Stack` of digits (`ints` 0 through 9) separated by -1's as a parameter and returns a `Queue` of `Strings` with the numbers made up of those digits with leading zeroes removed. You should remove all leading zeroes from each number. If all digits in the number are zero, you should output zero.

Example Output





Implementation Restrictions

- You may assume that the only numbers in the input stack are 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, and -1.
- You may assume the input stack has no adjacent -1's.
- You may assume the input stack is not `null` or empty.
- You may assume the input stack has a -1 at the bottom, and it does *not* have a -1 at the top.
- When your method returns, the input stack must be *unchanged*.
- You may create **at most two queues** and no other data structures.
- You may not use recursion to solve this problem.
- Your solution must run in $\mathcal{O}(n)$ time, where n is the size of the stack.

Use This Box For Scratch Work

DO NOT WRITE YOUR SOLUTION HERE

Question #6: Solution Space

Solution:

```
1  public static Queue<String> numberify(Stack<Integer> digits) {
2     Queue<String> output = new FIFOQueue<String>();
3     Queue<Integer> temp = new FIFOQueue<Integer>();
4     while (!digits.isEmpty()) {
5        while (digits.peek() == 0) {
6           temp.enqueue(digits.pop());
7        }
8        String num = "";
9        while (digits.peek() != -1) {
10          num += digits.peek();
11          temp.enqueue(digits.pop());
12       }
13       digits.pop();
14       if (num.length() == 0) {
15          output.enqueue("0");
16       }
17       else {
18          output.enqueue(num);
19       }
20    }
21    while (!temp.isEmpty()) {
22       digits.push(temp.dequeue());
23    }
24    while (!digits.isEmpty()) {
25       temp.enqueue(digits.pop());
26    }
27
28    while (!temp.isEmpty()) {
29       digits.push(temp.dequeue());
30    }
31
32    return output;
33 }
```

This page left intentionally almost Adam.