

Practice 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	

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 int mystery(int n, int m) {
2     if (n == 0 || m == 0) {
3         return 0;
4     }
5     else if (n % 10 == m % 10) {
6         return 1 + mystery(n / 10, m / 10);
7     }
8     else {
9         return mystery(n / 10, m / 10);
10    }
11 }
```

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

	mystery Call	Value Returned
(a)	mystery(18, 0)	0
(b)	mystery(8, 18)	1
(c)	mystery(25, 21)	1
(d)	mystery(305, 315)	2
(e)	mystery(20734, 1724)	2

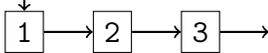
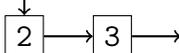
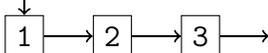
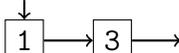
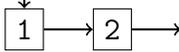
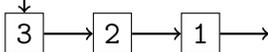
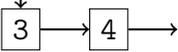
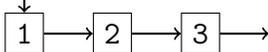
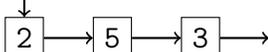
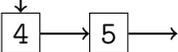
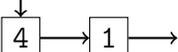
2. ListNode Before & After [16 points]

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)	<p>list1</p> <p>Before: </p> <hr/> <p>list1</p> <p>After: </p> <p>list2</p> <p>Before: </p> <hr/> <p>list2</p> <p>After: </p>	<pre> list2 = list1; list1 = list1.next; list2.next = null; </pre>
(b)	<p>list1</p> <p>Before: </p> <hr/> <p>list1</p> <p>After: </p> <p>list2</p> <p>Before: </p> <hr/> <p>list2</p> <p>After: </p>	<pre> list2 = list1.next; list1.next = list1.next.next; list2.next = null; </pre>
(c)	<p>list1</p> <p>Before: </p> <hr/> <p>list1</p> <p>After: </p> <p>list2</p> <p>Before: </p> <hr/> <p>list2</p> <p>After: </p>	<pre> ListNode temp = list2; list2 = list2.next; temp.next = list1.next; temp.next.next = list1; list1 = temp; list1.next.next.next = null; </pre>
(d)	<p>list1</p> <p>Before: </p> <hr/> <p>list1</p> <p>After: </p> <p>list2</p> <p>Before: </p> <hr/> <p>list2</p> <p>After: </p>	<pre> list2.next.next = list1.next.next; list1.next.next = list2.next; list2.next = list1; list1 = list1.next; list2.next.next = null; </pre>

Programming Pursuits.

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

3. Seven is the Most Random Number! [14 points]

Add a method `moreSevens` to the `ArrayIntList` class that takes an integer as an argument and replaces all occurrences of that integer with the number 7. If there are none of the provided integer in the list, it should remain unchanged.

Example Output

Before <code>list.moreSevens(3)</code>	After <code>list.moreSevens(3)</code>
[1, 3, 2, 7]	[1, <u>7</u> , 2, 7]
[2, 2, 2, 2]	[2, 2, 2, 2]
[3, 1, 3, 1]	[<u>7</u> , 1, <u>7</u> , 1]

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
```

Solution:

```
1  public void moreSevens(int n) {
2      for (int i = 0; i < this.size; i++) {
3          if (this.data[i] == n) {
4              this.data[i] = 7;
5          }
6      }
7  }

}
```

4. r-e-c-u-r-s-i-o-n [20 points]

Write a *recursive* method called `separate` that takes a `String` as a parameter and returns a new string in which all the characters are lowercase and separated by dashes.

Example Output

Method Call	Return Value
<code>separate("hEllo")</code>	<code>"h-e-l-l-o"</code>
<code>separate("OHNOES")</code>	<code>"o-h-n-o-e-s"</code>
<code>separate("Computer")</code>	<code>"c-o-m-p-u-t-e-r"</code>

Implementation Restrictions

- You must use recursion.
- You may only use the `String` methods on the cheatsheet.

Solution:

```
1 public static String separate(String s) {
2     if (s.length() < 2) {
3         return s.toLowerCase();
4     }
5     else {
6         return Character.toLowerCase(s.charAt(0)) + "-" + separate(s.substring(1));
7     }
8 }
```

5. Come Forward and Address Me! [16 points]

Write a method called `countOfAddressesByPerson` that takes a `Map` of addresses (`Strings`) to names (`Strings`) as a parameter and returns another map that associates each name with the number of addresses recorded for that person. The map returned by your method should be *ordered alphabetically*.

Example Output

Input Map	Return Value
<code>{}</code>	<code>{}</code>
<code>{"123 No St"="Allison", "CSE444"="Adam", "Tahiti"="Adam"}</code>	<code>{"Adam"=2, "Allison"=1}</code>

Implementation Restrictions

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

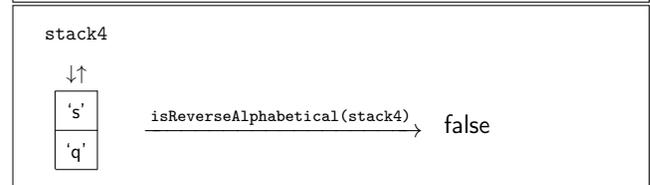
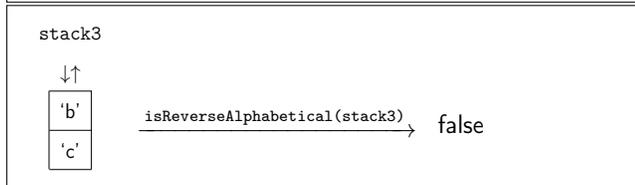
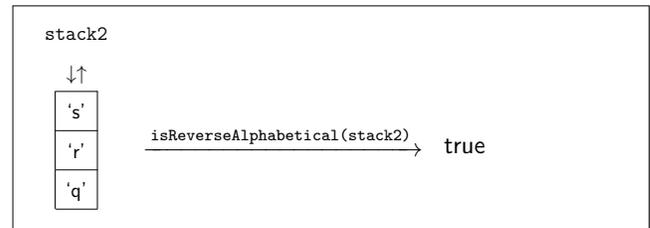
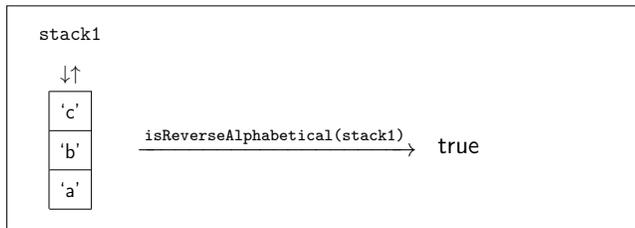
Solution:

```
1 public static Map<String, Integer> countOfAddressesByPerson(Map<String, String> addresses) {
2     // Count the addresses for each person
3     Map<String, Integer> m = new TreeMap<String, Integer>();
4     for (String addr : addresses.keySet()) {
5         String name = addresses.get(addr);
6         if (!m.containsKey(name)) {
7             m.put(name, 0);
8         }
9         m.put(name, m.get(name) + 1);
10    }
11    return m;
12 }
```

6. ZYXWVUT, Now I Know My ABCs... [20 points]

Write a method `isReverseAlphabetical` that takes a Stack of chars as a parameter and returns `true` when the stack is in reverse alphabetical order (and `false` otherwise). Reverse alphabetical order means that no characters are missing (and there are no repeats) between the top and bottom.

Example Output



Implementation Restrictions

- You may assume any input stack with fewer than two letters is reverse alphabetical.
- You may assume the input stack only contains lowercase alphabetic letters.
- When your method returns, the input stack must be *unchanged*.
- You may create **at most one queue** 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.

Write Your Answer On The Next Page

Solution:

```
1 public static boolean isReverseAlphabetical(Stack<Character> s) {
2     if (s.size() < 2) {
3         return true;
4     }
5
6     boolean sorted = true;
7     char prev = s.pop();
8     Queue<Character> backup = new FIFOQueue<Character>();
9     backup.enqueue(prev);
10    while (!s.isEmpty()) {
11        char curr = s.pop();
12        backup.enqueue(curr);
13        if (prev - 1 != curr) {
14            sorted = false;
15        }
16        prev = curr;
17    }
18
19    while (!backup.isEmpty()) {
20        s.push(backup.dequeue());
21    }
22
23    while (!s.isEmpty()) {
24        backup.enqueue(s.pop());
25    }
26
27    while (!backup.isEmpty()) {
28        s.push(backup.dequeue());
29    }
30
31    return sorted;
32 }
```

This page left intentionally almost Adam.