Instructions: Answer the questions below in the space provided. You may write on this test. You are allowed one sheet of paper, front and back, for notes. No other aids are allowed. You must use a pencil. Please write clearly and legibly. Pencils and/or additional paper will be provided upon request. You have one hour and 30 minutes to take this test.

ArrayList Mystery

Consider the following method:

```
public static void arrayListMystery(ArrayList<Integer> list) {
    for (int i = 0; i < list.size(); i++) {
        int element = list.get(i);
        list.remove(i);
        list.add(0, element + 1);
    }
    System.out.println(list);
}
```

What is the output produced by the method when passed each of the following ArrayLists?

`[10, 20, 30]` ______________________

`[8, 2, 9, 7, 4]` ______________________

`[-1, 3, 28, 17, 9, 33]` ______________________

Map Mystery

Write the output that is printed when the method below is passed each of the following maps as its parameter. Your answer should display the right values in the right order. You can assume that a for-each loop over the map will produce the elements in the order they are shown in the question.

```
public static void mapMystery(HashMap<String, String> map) {
    ArrayList<String> list = new ArrayList<String>();
    for (String key : map.keySet()) {
        if (map.get(key).length() > key.length()) {
            list.add(map.get(key));
        } else {
            list.add(0, key);
            list.remove(map.get(key));
        }
    }
    System.out.println(list);
}
```

Map: `{horse=cow, cow=horse, dog=cat, ok=yo}`
Output: ______________________

Map: `{bye=hello, bird=dog, hi=hello, hyena=apple, fruit=meat}`
Output: ______________________

Map: `{a=b, c=d, e=a, ff=a, gg=c, hhh=ff}`
Output: ______________________

Recursive Tracing

Consider the following method:

```
public static void recursionMystery(int n) {
    if (n % 2 == 1) {
        System.out.print(n);
    } else {
        System.out.print(n + ", ");
        recursionMystery(n / 2);
    }
}
```

For each call below, write the output that is produced.

`recursionMystery(13)` ______________________________

`recursionMystery(42)` ______________________________

`recursionMystery(40)` ______________________________

`recursionMystery(60)` ______________________________

`recursionMystery(48)` ______________________________

Inheritance Mystery

Assume that the following classes have been defined:

```
public class Blue extends Green {
    public void one() {
        System.out.println("Blue 1");
        super.one();
    }
}

public class Red extends Yellow {
    public void one() {
        super.one();
        System.out.println("Red 1");
    }
    public void two() {
        System.out.println("Red 2");
        super.two();
    }
}
```

```
public class Yellow extends Blue {
    public void two() {
        System.out.println("Yellow 2");
    }

    public void three() {
        two();
        System.out.println("Yellow 3");
    }
}

public class Green {
    public void one() {
        System.out.println("Green 1");
    }

    public void three() {
        System.out.println("Green 3");
    }
}
```

and the following variables have been defined:

```
Green var1 = new Blue();
Green var2 = new Red();
Blue var3 = new Yellow();
Object var4 = new Green();
```

In the table below, indicate in the right–hand column the output produced by the statement in the left–hand column. If the statement produces more than one line of output, indicate the line breaks with slashes as in "a/b/c" to indicate three lines of output with "a" followed by "b" followed by "c". If the statement causes an error, fill in the right–hand column with either the phrase "compiler error" or "runtime error" to indicate when the error would be detected.

Method call	**Output**
`var1.one();`	
`var1.two();`	
`var1.three();`	
`var2.one();`	
`var2.two();`	
`var2.three();`	
`var3.one();`	
`var3.two();`	
`var3.three();`	
`var4.one();`	
`((Blue)var1).one();`	
`((Yellow)var1).two();`	
`((Red)var2).three();`	
`((Object)var3).one();`	
`((Yellow)var3).two();`	
`((Green)var4).three();`	
`((Red)var4).one();`	

Programming 1

Write a method `collapse` that collapses a list of integers by replacing each successive pair of integers with the sum of the pair. For example, if a variable called `list` stores this sequence of values:

```
[7, 2, 8, 9, 4, 13, 7, 1, 9, 10]
```

and the following call is made:

```
list.collapse();
```

The first pair should be collapsed into 9 i.e 7 + 2, the second pair should be collapsed into 17 i.e. 8 + 9, the third pair should be collapsed into 17 i.e. 4 + 13 and so on to yield:

```
[9, 17, 17, 8, 19]
```

If the list stores an odd number of elements, the final element is not collapsed. For example, the sequence:

```
[1, 2, 3, 4, 5]
```

would collapse into:

```
[3, 7, 5]
```

with the 5 at the end of the list unchanged.

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

```
public class ArrayIntList {
    private int[] elementData; // list of integers
    private int size;          // current # of elements in the list

    <methods>
}
```

You are not to call any other `ArrayIntList` methods to solve this problem, you are not allowed to define any auxiliary data structures (no array, ArrayList, etc), and your solution must run in $\mathcal{O}(n)$ time.

Write your implementation to `collapse` on the next page.

Write your implementation to `sollapse` in the space below.

Programming 2

Write a recursive method `isReverse` that accepts two strings as a parameter and returns `true` if the two strings contain the same sequence of characters as each other but in the opposite order (ignoring capitalization), and `false` otherwise. For example, the string `"hello"` backwards is `"olleh"`, so a call of `isReverse("hello", "olleh")` would return `true`. Since the method is case–insensitive, you would also get a true result from a call of `isReverse("Hello", "oLLEh")`. The empty string, as well as any one-letter string, is considered to be its own reverse. The string could contain characters other than letters, such as numbers, spaces, or other punctuation; you should treat these like any other character. The key aspect is that the first string has the same sequence of characters as the second string, but in the opposite order, ignoring case. The table below shows more examples:

Call	Value Returned
`isReverse("CSE143", "341esc")`	`true`
`isReverse("Madam", "MaDAm")`	`true`
`isReverse("Q", "Q")`	`true`
`isReverse("", "")`	`true`
`isReverse("e via n", "N aIv E")`	`true`
`isReverse("Go!  Go", "OG !OG")`	`true`
`isReverse("Obama", "McCain")`	`false`
`isReverse("banana", "nanaba")`	`false`
`isReverse("hello!!", "olleh")`	`false`
`isReverse("", "x")`	`false`
`isReverse("madam I", "i m adam")`	`false`
`isReverse("ok", "oko")`	`false`

You may assume that the strings passed are not `null`. You are not allowed to construct any structured objects other than Strings (no array, List, Scanner, etc.) and you may not use any loops to solve this problem; you must use recursion. However, you allowed to use `String` methods such as `length()`, `charAt()`, `substring()`, `equals()`, `equalsIgnoreCase()`, and `toLowerCase()`. If you like, you may declare other methods to help you solve this problem, subject to the previous rules.

Write your implementation to `isReverse` on the next page.

Write your implementation to `isReverse` in the space below.