CSE143X Section #13 Problems

1. Recursive Tracing, 15 points. Consider the following method:

```
public void mystery(int n) {
    System.out.print("+");
    if (n >= 10) {
        mystery(n / 10);
    }

    if (n % 2 == 0) {
        System.out.print("-");
    } else {
        System.out.print("*");
    }
}
```

For each call below, indicate what output is produced:

Method Call	Output Produced
mystery(5);	_____
mystery(15);	_____
mystery(304);	_____
mystery(9247);	_____
mystery(43269);	_____

2. Recursive Programming, 15 points. Write a recursive method called doubleDigit that takes an integer n and a digit d as parameters and that returns the integer obtained by replacing all occurrences of d in n with two of that digit. For example, doubleDigit(3797, 7) would return 377977 because the two occurrences of the digit 7 are replaced with two of that digit. The table below includes more examples.

Method Call	Value Returned	Method Call	Value Returned
doubleDigit(2, 2)	22	doubleDigit(2001, 0)	200001
doubleDigit(0, 0)	0	doubleDigit(12345, 6)	12345
doubleDigit(8, 6)	8	doubleDigit(72773, 7)	77277773
doubleDigit(55, 2)	55	doubleDigit(3445, 5)	34455
doubleDigit(33, 3)	3333	doubleDigit(54224, 4)	5442244
doubleDigit(-101, 1)	-11011	doubleDigit(-624243, 4)	-62442443
doubleDigit(323, 3)	33233	doubleDigit(5340909, 0)	534009009

Notice that the number can be negative. Your method should throw an IllegalArgumentException if the value of d is not a 1-digit number (i.e., not between 0 and 9 inclusive). You are not allowed to construct any structured objects to solve this problem (no string, array, ArrayList, StringBuilder, Scanner, etc) and you may not use a while loop, for loop or do/while loop to solve this problem; you must use recursion.

3. Details of inheritance, 20 points. Assuming that the following classes have been defined:

```
public class Fork extends Pot {
    public void method2() {
        System.out.println("Fork 2");
        super.method2();
    }
}

public class Pot {
    public void method2() {
        System.out.println("Pot 2");
    }

    public void method3() {
        System.out.println("Pot 3");
        method2();
    }
}

public class Bowl extends Fork {
    public void method1() {
        System.out.println("Bowl 1");
    }

    public void method2() {
        System.out.println("Bowl 2");
    }
}

public class Spoon extends Pot {
    public void method1() {
        System.out.println("Spoon 1");
    }

    public void method2() {
        System.out.println("Spoon 2");
    }
}
```

And assuming the following variables have been defined:

```
Pot var1 = new Spoon();
Bowl var2 = new Bowl();
Pot var3 = new Bowl();
Pot var4 = new Pot();
Object var5 = new Bowl();
Pot var6 = new Fork();
```

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.

Statement	Output
`var1.method2();`	_____
`var2.method2();`	_____
`var3.method2();`	_____
`var4.method2();`	_____
`var5.method2();`	_____
`var6.method2();`	_____
`var1.method1();`	_____
`var2.method1();`	_____
`var3.method1();`	_____
`var1.method3();`	_____
`var2.method3();`	_____
`var3.method3();`	_____
`var4.method3();`	_____
`((Spoon)var1).method1();`	_____
`((Bowl)var3).method1();`	_____
`((Fork)var3).method3();`	_____
`((Fork)var5).method1();`	_____
`((Spoon)var5).method1();`	_____
`((Fork)var6).method2();`	_____
`((Bowl)var6).method3();`	_____

4. Linked Lists, 15 points. Fill in the "code" column in the following table providing a solution that will turn the "before" picture into the "after" picture by modifying links between the nodes shown. You are not allowed to change any existing node's data field value and you are not allowed to construct any new nodes, but you are allowed to declare and use variables of type ListNode (often called "temp" variables). You are limited to at most two variables of type ListNode for each of the four subproblems below.

You are writing code for the ListNode 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>
}
```

As in the lecture examples, all lists are terminated by null and the variables p and q have the value null when they do not point to anything.

before	after	code
p q->[1]->[2]->[3]	p->[3] q->[1]->[2]	
p->[1] q->[2]->[3]	p->[1] q->[3]->[2]	
p->[1]->[2] q->[3]->[4]	p->[4]->[2] q->[1]->[3]	
p->[1]->[2]->[3] q->[4]->[5]	p->[2]->[4] q->[5]->[3]->[1]	

5. ArrayIntList, 10 points. Write a method called extractOddIndexes that constructs and returns a new ArrayIntList of values that contains the sequence formed by removing the values at odd indexes in an existing ArrayIntList of values. For example, suppose that an ArrayIntList called list stores the following sequence of values:

```
[13, 5, 7, 12, 42, 8, 23, 31]
```

If we make the following call on the method:

```
ArrayIntList result = list.extractOddIndexes();
```

After the call, list and result would store the following:

```
list  : [13, 7, 42, 23]
result: [5, 12, 8, 31]
```

Notice that the values that were at odd indexes have been moved to the new ArrayIntList in the same order as in the original list and that the list now stores just values that were at even indexes also in the same order as in the original list. The list might have an odd number of values, as in:

```
[14, -64, 16, 88, 21, 17, -93, 81, 17]
```

in which case the lists would store the following after the call:

```
list  : [14, 16, 21, -93, 17]
result: [-64, 88, 17, 81]
```

Notice that list stores five values while result stores only four. 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 may use the zero-argument constructor for ArrayIntList and you may assume that it will construct an array of sufficient capacity to store the result. If the original list is empty, the result should be an empty list.

You may call the ArrayIntList constructor, but otherwise you may not call any other methods of the ArrayIntList class to solve this problem. You are not allowed to define any auxiliary data structures other than the new ArrayIntList you are constructing (no array, String, ArrayList, etc). Your solution must run in O(n) time where n is the original list.

6. Stacks/Queues, 25 points. Write a method called mirrorSplit that takes a stack of integers as a parameter and that splits each value into two halves, adding new values to the stack in a mirror position. For example, suppose that a stack s stores the following values:

```
bottom [14, 20, 8, 12] top
```

If we make the following call:

```
mirrorSplit(s);
```

Then after the call, the stack should store the following values:

```
bottom [7, 10, 4, 6, 6, 4, 10, 7] top
        ^   ^  ^  ^  ^  ^   ^  ^
        |   |  |  +--+  |   |  |
        |   |  +--------+   |  |
        |   +---------------+  |
        +----------------------+
            mirror positions
```

The first value 14 has been split in half into two 7s which appear in mirror positions (first and last). The second value 20 has been split in half into two 10s which appear in mirror positions (second and second-to-last). And so on. This example included just even numbers in which case you get a true mirror image. If the stack contains odd numbers, they should be split so as to add up to the original with the larger value appearing closer to the bottom of the stack. For example, if the stack stores these values:

```
bottom [13, 5, 12] top
```

After the call, it would store the following values:

```
bottom [7, 3, 6, 6, 2, 6] top
        ^  ^  ^  ^  ^  ^
        |  |  +--+  |  |
        |  +--------+  |
        +--------------+
        mirror positions
```

The first value 13 has been split into 7 and 6 with the 7 included as the first value and 6 included as the last value. The value 5 has been split into 3 and 2 with 3 appearing as the second value and 2 appearing as the second-to-last value. And so on.

You are to use one queue as auxiliary storage to solve this problem. You may not use any other auxiliary data structures to solve this problem, although you can have as many simple variables as you like. You also may not solve the problem recursively. Your solution must run in O(n) time where n is the size of the stack. Use the Stack and Queue structures described in the cheat sheet and obey the restrictions described there (recall that you can't use the peek method or a foreach loop or iterator).