

C Sc 127B Practice Quiz 4 Section Leader _____ Name _____

From each class of Recursion problems, complete the "a" question before your section this week. These and the other questions constitute the practice quiz. All answers will be linked Friday afternoon. Writing many recursive solutions will help you understand recursion. Note: Some of these exercises may be a repeat of what you have seen or done.

Before section, complete these 5 problems with pencil and paper and turn in at the beginning of your section for a 3pt homework: 1a, 2a, 3a, 4a, and 5a. Before the quiz, complete all.

1. Write the output from mystery functions

1a) Write the output generated by the method call `mystery("X", 6)`;

```
public void mystery(String s, int digit) {
    if(digit <= 1)
        System.out.println(digit);
    else {
        s = s + "<";
        mystery(s, digit - 2);
        System.out.println(s + digit);
    }
}
```

1b) Write the return values from each call to the method named `mystery`.

____ `mystery(0)` ____ `mystery(1)` ____ `mystery(2)` ____ `mystery(3)` ____ `mystery(4)`

```
public int mystery(int n) {
    if (n <= 0)
        return 1;
    else
        return 3 + mystery(n - 1);
}
```

1c) Write the return values from each call to the method named `mysteryTwo`.

_____ `mysteryTwo("T")` _____ `mysteryTwo("ab")` _____ `mysteryTwo("123")`

```
public String mysteryTwo(String s) {
    if (s.length() == 0)
        return "";
    else
        return mysteryTwo(s.substring(1, s.length())) + "/" + s.charAt(0);
}
```

1d. Write the output of `stars` with arguments 1, 2, and 3

```
public static void stars(int n) {
    if (n > 1)
        stars(n-1);
    for (int i = 0; i < n; i++)
        System.out.print("*");
    System.out.println();
}
```

2) Use a recursive definition to write a recursive method

2a) Implement the `tribonacci` recursively.

$$\text{trib}(n) \begin{cases} 1 & \text{if } n \leq 3 \\ \text{trib}(n-1) + \text{trib}(n-2) + \text{trib}(n-3) & \text{if } n \geq 4 \end{cases}$$

2b) Implement the `power` function recursively.

$$\text{power}(x,n) \begin{cases} 1 & \text{if } n = 0 \\ x \cdot \text{power}(x, n-1) & \text{if } n \geq 1 \end{cases}$$

2c) Implement the factorial function recursively as `fact`.

$$\text{fact}(n) \begin{cases} 1 & \text{if } n \leq 1 \\ n \cdot \text{fact}(n-1) & \text{if } n > 1 \end{cases}$$

2d) Implement the Greatest Common Divisor algorithm as recursive method `GCD`. Use recursion. Do NOT use a loop.

$$\text{GCD}(m,n) \begin{cases} m & \text{if } n == 0 \\ \text{GCD}(n, m\%n) & \text{if } n > 0 \end{cases}$$

3) Write a method to do an abstract thing

3a) Given a string, compute recursively (no loops) a new string where all the lowercase 'x' chars have been changed to 'y' chars.

```
changeXY("codex") → "codey"  
changeXY("xxhixx") → "yyhiyy"  
changeXY("xhixhix") → "yhiyhiy"
```

3b). Write recursive method `isPalindrome` that returns true if the string argument is a palindrome. Do not use a loop.

3c) Write recursive method `printInt` to print an integer with commas in the correct places. `printInt(12004)` should print 12,004 and `printInt(123456780)` should print 123,456,780

3d) Write recursive method `printIntInNewBase` to print an integer in any base from 2 through decimal (base 10) without using an array. `printIntInNewBase(17, 10)` should print 17, `printIntInNewBase(17, 2)` should print 10001, and `printIntInNewBase(17, 8)` should print 21.

4) Recursive solutions to array processing

4a) Write recursive method `getRange` that returns the range in a filled array of `int` referenced by `x`. The range is defined as the largest minus the smallest. Do not use a loop. You may use a helper method.

4b) Write recursive method `sum` that returns the sum of all the `int` elements in a filled array referenced by `y`. Do not use a loop. You must have a recursive call somewhere in your answer.

4c) Write recursive method `printArrayInReverse` that prints all array elements in a filled array of ints referenced by `x` in their reverse order. Do not change the array. Do not use a loop. You may use a helper method.

4d) Implement method `compare` that returns 0 if all elements in two arrays of ints are in the same exact order and have the same length. Return the difference between the first two non-equal integers—negative if the element in the first array

argument is less than the element at the same index of the other array argument. If both arrays have the same elements up to the end of the shorter array, return the difference in lengths, negative if the first array is shorter or positive otherwise.

```
compare({ 1, 2, 3 }, {1, 2, 3}) → 0
compare({ 1, 7, 3 }, {1, 2, 3}) → 5
compare({ 1, 2, 3 }, {1, 5, 3, 4, 12, 99}) → -3
compare({ 9, 8, 3 }, {9, 8, 3, 4, 5}) → -2
compare({ 9, 8, 3, 4, 5 }, {9, 8, 3}) → +2
```

5) Recursive solutions to linked structure processing

Add methods to this collection class that uses a singly linked data structure

```
public class LinkedList {

    private class Node {
        private int data;
        private Node next;
        public Node(int element) {
            data = element;
            next = null;
        }
    }

    private Node front;
    private int n;

    public LinkedList() {
        front = null;
        n = 0;
    }

    public void addFirst(int el) {
        front = new Node(el, front);
        n++;
    }
}
```

5a) Write recursive method **numberOfOdds** to return the number of odd integers in a `LinkedList` object.

5b) Write method **occurrencesOf** that return the number of times an int element occurs in a `LinkedList` object.

5c). Write method **addLast** that recursively adds the int argument to the end of the linked structure

5d). Write method **accumulateAll** that changes the `int` value of all nodes (except the last) in a `LinkedList` object . The first node must have the sum of all integers, the second node has the sum of all integers from index 1 through the last node, and so on. The assertions must pass.

```
@Test public void testAccumulateSums() {
    LinkedListOfInts list = new LinkedListOfInts();
    list.addLast(1);
    list.addLast(2);
    list.addLast(3);
    list.addLast(4);
    list.accumulateSums();
    assertEquals(10, list.getElementAt(0));
    assertEquals(9, list.getElementAt(1));
    assertEquals(7, list.getElementAt(2));
    assertEquals(4, list.getElementAt(3));
}
```