CS 312 – Midterm 2 – Fall 2013

SOLUTION SOLUTION SOLUTION SOLUTION SOLUTION SOLUTION

Problem Number	Topic	Points Possible	Points Off
1	code trace	28	
2	arrays	14	
3	strings	16	
4	program logic	16	
5	scanner	23	
6	arrays and strings	15	
7	simulation	30	
	TOTAL POINTS OFF:		
	SCORE OUT OF 142:		

Grading Acronyms:

NAP = No Answer Provided

LE = Logic Error

GCE = Gross Conceptual Error, based on answer did not understand question.

OBOE = Off By One Error

BOD = Benefit Of the Doubt, can't be certain it works, but can't find anything wrong either

GACK = poor style or approach, but no points off

NN = Not Necessary. Generally no points off.

1. Evaluating Code. 28 points, 2 points each. Answer each question below. If the snippet contains a syntax error answer **syntax error**. If the snippet results in a runtime error or exception answer **runtime error**. If the code results in an infinite loop answer **infinite loop**. Assume all necessary imports have been made.

A. What is output by the following code?

```
int xa = 3;
int ya = 2;
boolean pa = xa > ya;
boolean qa = xa % ya == 1;
System.out.print( (pa && qa) + " " + (pa || qa));
```

Answer: **true true**

B. Are the two `boolean` expressions below equivalent? In other words do the two expressions always evaluate to the same value given the `boolean` variables `p` and `q` store the same values?

Expression 1: `!(p && q)`
Expression 2: `!p && !q`

Answer: **NO**

C. What is output by the following code?

```
String strc = "PORTER";
String subc = strc.substring(2, 5); // inclusive first index, exclusive second
System.out.print(strc + " " + subc);
```

OUTPUT: **PORTER RTE**

D. What is output by the following code?

```
String strd = "KLIVANS";
String subd = strd.substring(3);
System.out.print(subd);
```

Answer: **VANS**

E. What is output by the following code?

```
int xe = 3;
int ye = 2;
int[] datae = new int[xe * ye];
xe += 2;
System.out.print(datae.length);
```

Answer: **6**

F. What is output by the following code?

```
int[] dataf = new int[10];
dataf[1] += dataf[3] * dataf[6];
System.out.print(dataf[1]);
```

Answer: **0**

G. What is output by the following code?

```
int[] data1g = {1, 2, 3, 4};
int[] data2g = {1, 2, 3, 3};
data2g[3] = 4;
System.out.print(data1g == data2g);
```

Answer: **false**

H. What is output by the following code?

```
int[] data1h = {2, 4, -5};
int[] data2h = {1, 2, 1};
data2h = data1h; // 2 pointers or references, one array
data1h[1] += 2;
data2h[2] -= 2;
System.out.println(Arrays.toString(data2h));
```

Answer: **[2, 6, -7]**

I. What is output by the following code?

```
int x = 5;
int[] datai = new int[x / 2 - 4];
System.out.print(datai.length);
```

Answer: **RUNTIME ERROR or EXCEPTION or NegativeArraySizeException**

J. What is output by the following code?

```
int[] dataj = {-5, 2, 1, 2};
dataj[2] += dataj[1];
dataj[3] -= dataj[2];
System.out.print(Arrays.toString(dataj));
```

Answer: **[-5, 2, 3, -1]**

K. What is output by the following code?

```
int[] datak = {0, 2, 4};
datak[1] += datak[datak[1]];
datak[2] -= datak.length;
System.out.print(Arrays.toString(datak));
```

Answer: **[0, 6, 1]**

L. What is output by the following code?

```
int[][] mat = new int[5][7];
System.out.print(mat.length + " " + mat[0].length);
```

Answer: **5 7**

M. What is output by the following code?

```
int[] datam = {-5, 2, 7};
methodM(datam);
System.out.print(Arrays.toString(datam));

public static void methodM(int[] values) {
      values[2] += 3;
      values[1] = 12;
      values = new int[4];
      values[1] = 5;
}
```

Answer: **[-5, 12, 10]**

N. What is output by the following code?

```
int[] datan = new int[10];
datan.length += 10;
int xn = 5;
System.out.print(datan[xn - 10]);
```

Answer: **SYNTAX ERROR -> cannot alter the length field. Code never runs.**

2. Arrays. 14 points. Write a method `sumArrays`. The method has two parameters. Each parameter is an array of `int`s. You may assume neither of the array variables equals `null`. The method returns an array equal in length to the smaller of the two parameters. The elements of the returned array equal the sum of the corresponding elements in the two parameters.

Neither of the parameters is altered as a result of this method.

Examples of calls to `sumArray`:
`sumArrays({1, 2}, {2, 3})` returns the array `{3, 5}`
`sumArrays({}, {2, 3})` returns the array `{}`
`sumArrays({1, 2, 2, 4}, {4, 6})` returns the array `{5, 8}`
`sumArrays({}, {})` returns the array `{}`

You may not use any other methods or classes in your answer other than Java's built in, native arrays.
You may not use the static methods from the Arrays class.

```
public static int[] sumArrays(int[] a1, int[] a2) {
    // find min length of a1 and a2
    int smaller = a1.length;
    if(a2.length < a1.length)
        smaller = a2.length;
    int[] result = new int[smaller];
    for(int i = 0; i < result.length; i++)
        result[i] = a1[i] + a2[i];
    return result;
}
```

Criteria:

find length of smaller array: 4 points (-2 if use Math.min)

create resulting array: 2 points (-1 if no size)

loop with correct bounds: 4 points (not all or nothing: can take off 1, 2, or 3 as appropriate)

add corresponding elements of parameters to result correctly: 3 points (not all or nothing, can take off 1 or 2)

return result: 1 point

other:
altering either parameter: - 3

Common problems:
1. Altering one of the parameters
2. Always returning array of length 2, with sum of 2 arrays. (Not what question asked.)

3. Strings 16 points. (Based on a question from codingbat.com)Write a method `repeatedReversedEnd`. The method has 3 parameters, a `String str`, an `int n`, and an `int num`. The method creates and returns a new `String` that consists of the last `n` characters of `str` in reverse order, repeated `num` times.

Examples of calls to `repeatedReversedEnd(String str, int n, int num)`:
`repeatedReversedEnd("ABCD", 2, 3)` returns `"DCDCDC"`
`repeatedReversedEnd("ABCD", 0, 3)` returns `""`
`repeatedReversedEnd("ABCD", 3, 0)` returns `""`
`repeatedReversedEnd("ABCD", 4, 1)` returns `"DCBA"`
The `String` in the next example contains a single space
`repeatedReversedEnd("A *D", 4, 3)` returns `"D* AD* AD* A"`
`repeatedReversedEnd("AAABB", 4, 3)` returns `"BBAABBAABBAA"`

Assume `0 <= n <= str.length(), num >= 0,` and `str != null`

The only methods you may use from the String class are the `length`, `charAt`, and `substring` methods plus `String` concatenation. You may not use any other Java classes or methods in your answer.

```
    public static String repeatedReversedEnd(String str, int n, int num) {
        String temp = str.substring(str.length() - n);
        String revEnd = "";
        for(int i = temp.length() - 1; i >= 0; i--)
            revEnd += temp.charAt(i);
// alternate approach for creating reversed end
//      for(int i = str.length() - n; i < str.length(); i++)
//          revEnd = str.charAt(i) + revEnd;
        String result = "";
        for(int i = 0; i < num; i++)
            result += revEnd;
        return result;
    }
```

Criteria:
Correctly create a String that is the correct length (n characters) and is the reverse of the n characters at the end of str. (or use indices correctly in a nested loop): 7 points
Create resulting String as empty String: 2 points
loop correct number of times for copy (outer loop in nested loop solution): 3 points (not all or nothing)
concatenate end to result correctly (could be char by char build up): 3 points
return result: 1 point

Other:
disallowed methods: -4 per occurrence up to -8

Common problems:
1. Not creating reverse of end of String (or incorrect indices in nested loop)
2. Wrong bounds for the substring
3. Most common error: result = result + result; This creates a result that is 2^{num} or $2^{(num + 1)}$ characters in length. (Doubles length of result each time.) The result must be n * num characters long. For most values of n and num, n * num != 2^{num} or $2^{(num + 1)}$

4. Program Logic 16 Points. Consider the following method. For each of the four points labeled by comments and each of the four assertions in the table, write whether the assertion is ***always*** true, ***sometimes*** true, or ***never*** true at that point in the code. Abbreviate ***always*** with an A, ***sometimes*** with an S and ***never*** with an N.

```
// this is an algorithm to find the Greatest Common Divisor of two
// positive ints
public static int assertionPractice(int a, int b) {
    int result = -1;
    // POINT A
    if(a > 0 && b > 0) {
        while(a != b) {
            // POINT B
            if(a > b) {
                a -= b;
            }
            else {
                b -= a;
            }
        }
        result = a;
        // POINT C. Here we KNOW a == b && b == result && result == -1
        // it also turns out a must be positive, thus result cannot
        // be negative
    }
    // POINT D
    return result;
}
```

Abbreviate ***always*** with an A, ***sometimes*** with an S and ***never*** with an N. **1point each**

	result == b	b > a	result == -1	a == b
POINT A	S	S	A	S
POINT B	N	S	A	N
POINT C	A	N	N	A
POINT D	S	S	S	S

5. Scanner 21 Points. Write a method `tokensPerLine`. The method has one parameter, a `Scanner` object. The `Scanner` is already connected to a file. The file contains at least one line. The method returns the average number of tokens (based on whitespace) per line.

For example, if the `Scanner` object was connected to the following file,

```
10    cat    20
30

50   20cat20
cat
dog  32
```

then the method would return `1.5 (9 / 6)`.

The given file contains 9 tokens (10, cat, 20, 30, 50, 20cat20, cat, dog, and 32) and has 6 lines. There is one blank line with no tokens between the line that contains 30 and the line that contains 50 20cat20.

You may only use `Scanner` objects and the constructors and methods from the `Scanner` class. You may not use any other Java classes or methods in your answer. Write the complete method, including the header, below:

```
public static double tokensPerLine(Scanner sc) {
    int lines = 0;
    int tokens = 0;
    while(sc.hasNextLine()) {
        lines++;
        Scanner lineScanner = new Scanner(sc.nextLine());
        while(lineScanner.hasNext()) {
            tokens++;
            lineScanner.next();
        }
    }
    return 1.0 * tokens / lines;
}
```

Criteria:

header correct, 2 points (not all or nothing, can take off 1 if appropriate) Other:

variables for number of lines and number of tokens: 2 points, 1 each

loop for file scanner has next line: 3 points (not all or nothing)

increment number of lines correctly: 1 point

Scanner for line, reading in line: 4 points (not all or nothing)

inner loop for number of tokens in line: 3 points (not all or nothing

increment number of tokens correctly: 1 point

move to next token correctly: 2 points

calculate correct result: 2 points (-1 if not fp division)

return result: 1 point

6. Arrays and Strings 15 Points. Write a method `getNGrams`. The method takes 2 parameters, a `String str` and an `int n`. The method returns an array of Strings that are the *n-grams* of the `String str`.
An n-gram *is a contiguous sequence of n characters from a given String*.

For example if n = 2 and the String is "dogs" the n grams (in this case *bigrams*) are "do", "og", and "gs".

If n = 3 and the String is "computer" the n grams are (in this case *trigrams*) are "com", "omp", "mpu", "put", "ute", and "ter".

If n = 4 and the String is "***" there are no *quadgrams* and the method shall return an empty array.

You may `String`s including the `substring` and `charAt` methods and Java's built in, native arrays.
You may not use any other methods or classes.

You may assume `str != null` and `n > 0`.

```
public static String[] getNGrams(String str, int n) {

    int numGrams = str.length() - n + 1; // 3
    // nasty special case
    if(numGrams < 0) // 3
        numGrams = 0;
    String[] result = new String[numGrams]; // 2
    for(int i = 0; i < numGrams; i++)  // 3
        result[i] = str.substring(i, i + n); // 3
    return result; // 1
}
```

Criteria:
Calculate number of ngrams correctly: 3 points (not all or nothing, can take off 1 or 2)

handle special case of no n grams correctly: 3 points (not all or nothing) Using Math.max or min -2

create resulting array: 2 points

loop with correct bounds to create correct number of N grams: 3 points (not all or nothing)

create N gram correctly using substring or charAt(inner loop) and add to array: 3 points (not all or nothing)

return result: 1 point

Other:
disallowed methods: -3 first, -2 second, -2 third up to -7

Common problems:
1. trying to create array with negative length
2. Not creating the correct nGrams

7. Simulation 30 Points. Write a method `rollUntilSame`. The method has one parameter, `int num`.

The method rolls `num` 6-sided dice per round. The method prints out the results of each round.

The method repeats rounds of rolling the `num` 6-sided dice until a round occurs where all `num` dice have the same value. All dice are rerolled every round.

Here is one sample run with `num` dice equal to 5. Your method's output must match the format below.

```
4 4 5 6 6
4 5 4 2 5
1 6 1 5 6
1 3 6 6 3
3 3 3 6 6
4 5 6 5 6
5 1 6 3 2
4 4 4 4 4
Number of rounds before all dice the same: 8
```

Recall how to create an object of type `Random`:

```
Random r = new Random();
```

and the method from the `Random` class, `nextInt(n)` that returns an `int` from 0 to n - 1.

You may not use any other Java classes or methods except Java's built in arrays and the `Random` class.
You may not use the static methods from the Arrays class.

Assume the parameter `num` is greater than or equal to 2.

```
public static void rollUntilSame(int num) {
```

Complete this method on the next page.

```
public static void rollUntilSame(int num) {
    int[] dice = new int[num];
    Random r = new Random();
    boolean done = false;
    int rounds = 0;
    while(!done) {
        rounds++;
       // roll dice and print out
       for(int i = 0; i < dice.length; i++) {
           dice[i] = r.nextInt(6) + 1;
           System.out.print(dice[i] + " ");
       }
       System.out.println();
       // check if all the same, gacky because it does more work than
       // is necessary. Better to stop when one not equal.
       // Should use while loop.
       done = true;
       for(int i = 1; i < dice.length; i++)
           if(dice[i] != dice[0])
               done = false;
    }
    System.out.println("Number of rounds before all dice the same: "
                + rounds);
}
```

Criteria:
create array of dice: (okay to simply track last roll or first roll of round and compare to that) 2 points
create object of type Random: 2 points
variable to track number of rounds: 2 points
loop until all dice the same: 5 points
increment number of rounds correctly: 1 point
loop to roll the dice: 3 points
roll dice correctly using Random object: 3 points
print out dice rolls for round correctly: 3 points
loop to check if dice for round are all the same (can handle this while rolling): 6 points
print number of rounds necessary to get all dice the same: 3 points
return result: 1 point

Students did fairly well on this question. Some students hard coded number of dice or number of rounds. Most common problem: not checking all dice the same correctly. A check such as this simply sets the boolean to true or false based on if the last pair of dice in the round are the same or not, ignoring all the other dice.)

```
for(int i = 1; i < dice.length; i++)
     if(dice[i - 1] == dice[i])
          allSame = true;
     else
          allSame = false;
```

Some people tried to compare the sum of the dice rolls for the round to the first roll time num. In other words if the there are 5 dice and the first roll is 2 and the sum of the dice is 10 then I rolled all 2's. Not necessarily so: 2, 2, 1, 3, 2 -> 2 + 2 + 1 + 3 + 2 = 10, but those are not all 2's.