CS 312 – Midterm 2 – Fall 2014

SOLUTION SOLUTION SOLUTION

Circle your TA's Name: Dory Tamara Eric

Jose Stas Aaron

Problem Number	Topic	Points Possible	Points Off
1	code trace	30	
2	scanners	15	
3	program logic	16	
4	strings	10	
5	arrays 1	10	
6	arrays 2	15	
7	simulation	15	
TOTAL POINTS OFF:			
SCORE OUT OF 111:			

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. 30 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?

```
String sa1 = "soccer";
String sa2 = sa1.substring(3);
System.out.print(sa2);
```

Output: **cer**

B. Are the two `boolean` expressions below logically equivalent? In other words given the same inputs do the two expressions always evaluate to the same `boolean` result? `xb` and `yb` are `int` variables and `p` is a `boolean` variable.

Expression 1: `!(xb <= yb && !p)`
Expression 2: `xb > yb || p`

Answer: **yes**

C. What is output by the following code?

```
int xc = 3;
double[] ac = new double[xc * 3];
System.out.print( (xc == ac.length && xc == ac[0]) + " "
          + (ac[xc] == 0 || ac[xc] == xc));
```

Output: **false true**

D. What is output by the following code?

```
String sd1 = "dice";
String sd2 = sd1.substring(0, 2);
System.out.print(sd1 + " " + sd2.length());
```

Output: **dice 2**

E. What is output by the following code?

```
String se1 = "swimming";
int i1 = se1.indexOf("m");
int i2 = se1.indexOf("ii");
System.out.print(i1 + " " + i2);
```

Output: **3 -1**

F. What is output by the following code?

```
String sf1 = "volleyball";
String sf2 = sf1.substring(6);
String sf3 = "ball";
System.out.print((sf1 == sf2) + " " + (sf2 == sf3));
```

Output: **fasle false**

G. What is output by the following code?

```
int[] ag = new int[5];
System.out.print(ag[0] + " " + ag[ag.length - 1]);
```

Output: **0 0**

H. What is output by the following code?

```
int xh = 4;
int yh = 2;
int[] ah = {5, 3, 4, 1, -1, 0, 4};
System.out.print(ah[xh] + " " + ah[yh]);
```

Output: **-1 4**

I. What is output by the following code?

```
int[] ai = {5, 3, 4, 1};
methodI(ai);
System.out.print(Arrays.toString(ai));

public static void methodI(int[] ai) {
     ai[0] += 2;
     ai[2] -= 2;
}
```

Output: **[7, 3, 2, 1]**

J. What is output by the following code?

```
int xj = 4;
int yj = 3;
String[] aj = new String[xj * yj];
System.out.print(aj[4].length() + " " + aj.length);
```

Output: **Runtime error** or **Exception**

K. What is output by the following code?

```
int[] ak = {5, 3, 4};
ak[0]++;
methodK(ak, 2);
System.out.print(" " + Arrays.toString(ak));

public static void methodK(int[] ak, int x) {
      ak[x] += x;
      ak = new int[x];
      ak[0] = x;
      System.out.print(Arrays.toString(ak));
}
```

Output: **[2, 0] [6, 3, 6]**

L. What is output by the following code?

```
int xl = 3;
int yl = 2;
methodL(xl, yl);
System.out.print(" " + xl + " " + yl);

public static void methodL(int xl, int yl) {
      xl++;
      yl = xl;
      xl = yl;
      System.out.print(xl + " " + yl);
}
```

Output: **4 4 3 2**

M. List the possible values the following code will output.

```
Random rm = new Random();
int x = (rm.nextInt(4) - 3) * 2;
System.out.print(x);
```

Output: **-6, -4, -2, 0**

N. What is output by the following code?

```
String[] an = {"OAS", "IMS", "KJS"};
int xn = 2;
System.out.print(an[1].length() < 10 || an[an.length + xn] == an[0]);
```

Output: **true**

O. What is output by the following code?

```
String on = "Jim_56_p";
System.out.print(on.toUpperCase());
```

Output: **JIM_56_P**

2. Scanners. 15 points. Write a method `processText`. The method accepts a Scanner already connected to a file. The method prints out the number of lines in the file, the average number of tokens per line, the number of tokens, and the average number of characters in each token.

For example, if the Scanner were connected to the following file:

```
line 1 has 5 tokens

this is line 3 line 2 above has 0 tokens
this_is_a_really_long_token
last line
```

then the output would be:

```
lines: 5
tokens: 18
tokens per line: 3.6
chars per token: 4.5
```

You may assume the file has at least 1 line and at least 1 token.

You may use the methods from the `Scanner` class and you may construct new `Scanners`.

You may use the `String` `length` method.

Do not use any other Java classes or methods. Do not use arrays.

Complete the method on the next page.

```
   public static void processText(Scanner sc) {
        int lines = 0;
        int tokens = 0;
        int chars = 0;
        while(sc.hasNextLine()) {
            lines++;
            Scanner line = new Scanner(sc.nextLine());
            while(line.hasNext()) {
                tokens++;
                chars += line.next().length();
            }
        }
        System.out.println("lines: " + lines);
        System.out.println("tokens: " + tokens);
        System.out.println("tokens per line: "
                + (1.0 * tokens / lines));
        System.out.println("chars per token: "
                + (1.0 * chars / tokens));
    }
```

```
local vars, 2
loop for next line, 3
get line, 1
loop for tokens in a given line, 3
update lines and tokens correctly, 2
calculations, 2
print results, 2
```

3. 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.

```
public static int assertionPractice(int x) {
     int result = -1;
     int d = 2;
     // POINT A
     if(x > 2) {
          result = 0;
          while(d < x) {
               int r = x % d;
               if(r == 0) {
                    result = result + 1;
                    // POINT B
               }
               d++;
               // POINT C
          }
     }
     return result; // POINT D
}
```

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

	result == -1	d == x	x > 2	result == 0
POINT A	A	S	S	N
POINT B	N	N	A	N
POINT C	N	S	A	S
POINT D	S	S	S	S

4. Strings 10 Points. Write a method `removeTrailingChar`. The method has two parameters: a String `str` and a char `ch`. The method creates and returns a new String that is the same as the parameter, except any characters at the end of the original String equal to the parameter `ch` are not present.

Examples:

```
removeTrailingChar("xAAxAA", 'A') -> returns "xAAx"
removeTrailingChar("xAAxAA", 'x') -> returns "xAAxAA"
removeTrailingChar("", 'A') -> returns ""
removeTrailingChar("AAAAAAA", 'A') -> returns ""
removeTrailingChar("xxBBBxBBbX", 'A') -> returns " xxBBBxBBbX"
removeTrailingChar("..X...x!.....", '.') -> returns "..X...x!"
```

You may use String concatenation and the String `charAt()`, `length()`, `indexOf()`, and `substring()` methods.

You may not use any other Java classes or methods.

Complete the following method:

```
public static String removeTrailingChar(String str, char ch)
{
    // find the index of the last occurrence of ch
    int index = str.length() - 1;
    while(index >= 0 && str.charAt(index) == ch)
        index--;
    return str.substring(0, index + 1);
}
```

```
find index of last occurrence of ch
loop 1
stop when not == ch, 4
stop when index negative, 2
create result, 2
return, 1
```

5. Arrays 10 Points. Write a method `numOutsideRange`. The method has 3 parameters: an array of `int`s, and two `int`s that represent the low and high ends of a range. The method returns the number of elements in the array that are **outside** of the range from low to high inclusive.

Examples:

```
numOutsideRange( {}, 0, 3) -> returns 0
numOutsideRange( {0, 3, 0, 3, 3, 1}, 0, 3) -> returns 0
numOutsideRange( {-1, -10, 15, 4, 3, 1, 10}, 0, 3) -> returns 5
numOutsideRange( {15, -5, 20}, -3, 10) -> returns 3
numOutsideRange( {5, -4, 4, 10}, 4, 4) -> returns 3
```

You may not use any other Java classes or methods in your answer.

Of course you may access the `length` field of the given array.

You may assume `low <= high`

Complete the following method:

```
public static int numOutsideRange(int[] data,
                                  int low, int high) {
    int count = 0;
    for(int i = 0; i < data.length; i++) {
        if(data[i] < low || data[i] > high)
            count++;
    }
    return count;
}
```

```
count var, 1
loop through array, 3
access elements correctly, 1
check if outside range correctly, 3
increment count, 1
return, 1
```

6. Arrays 15 Points. Write a method `getValuesOutsideRange`. The method has 3 parameters: an array of `int`s, and two `int`s that represent the low and high ends of a range.

The method returns a **new array** with all of the values from the original array that are outside the range from low to high inclusive.

The values are arranged with the values less than the given range at the front of the array and the values greater than the given range at the end of the array.

The relative order of the elements less than the given range is unchanged.

The relative order of the elements greater than the give range is reversed.

Examples:

```
getValuesOutsideRange ( {}, 0, 3) -> returns {}, an empty array

getValuesOutsideRange ( {0, 3, 0, 3, 3, 1}, 0, 3) -> returns {}

getValuesOutsideRange ( {-1, -10, 15, 4, 3, 1, 10}, 0, 3) ->
     returns{-1, -10, 10, 4, 15}

getValuesOutsideRange ( {15, -5, 20}, -3, 10) -> returns {-5, 20, 15}

getValuesOutsideRange ( {5, -4, 4, 10}, 4, 4) -> returns {-4, 10, 5}

getValuesOutsideRange ( {5, 3, 6, 7}, 10, 15) -> returns {5, 3, 6, 7}

getValuesOutsideRange ( {5, 3, 6, 7}, 0, 2) -> returns {7, 6, 3, 5}
```

The returned array is a new array.

Call the method `numOutsideRange` from question 5 as appropriate in your answer. Do not repeat that functionality in your answer here.

You may not use any other Java classes or methods.

Of course you may access the `length` field of given array and create a new array.

The original array is not altered by this method.

You may assume `low <= high`.

Complete the method on the next page.

```
public static int[] getValuesOutsideRange(int[] data, int low,
int high) {
    int sizeOfNewArray = numOutsideRange(data, low, high);
    int[] result = new int[sizeOfNewArray];
    int lowIndex = 0;
    int highIndex = result.length - 1;
    for(int i = 0; i < data.length; i++) {
        int val = data[i];
        if(val < low) {
            result[lowIndex] = val;
            lowIndex++;
        }
        else if(val > high) {
            result[highIndex] = val;
            highIndex--;
        }
        // else nothing to do
    }
    return result;
}
```

8. Simulation 15 Points. Write a method that implements a strategy for the Pig dice game from assignment 7. The strategy is called ***score base, keep pace, and end race*** and it has three parts.

Assume we have parameters for:

- the active player's score at the start of the turn
- the other player's score at the start of the turn
- a goal point value for the turn
- a threshold point value

During a turn a player rolls again if **any** of the following conditions are met:

1. The point total for the turn is less than the goal point value for the turn. This is similar to the point goal for turn strategy from the assignment.

2. The active player's score at the start of the turn plus the score for the turn thus far is still less than the other player's score. For example if the active player's score at the start of the turn was 40 and the score for the turn is 20, the player would roll again if the other player's score was 65 regardless of the goal for the turn. (40 + 20 < 65)

3. The active player's score at the start of the turn plus the score for the turn thus far OR the other player's score are within the threshold value of the winning score. For example if the other players score is 92 and the threshold point value is 10 the active player will keep rolling until they win or roll a pig.

Of course if the active player reaches 100 they don't roll again. They hold and win.

Recall if a player rolls a 1 (pig), the turn is over and the score is 0.

Complete a method that returns a player's score for the turn when following the above strategy.

Unlike the assignment your method does not produce any output.

The only method you may use is the Random class `nextInt` method.

Complete the method on the next page.

```
public static int getTurnScore(Random r, int activeScore,
        int otherScore, int goalForTurn, int thresholdValue) {
     int score = 0;
     boolean rollAgain = true;
     int closeScore = 100 - thresholdValue;
     while(rollAgain) {
         int roll = r.nextInt(6) + 1;
         if(roll == 1) {
             rollAgain = false; // PIG!
             score = 0;
         }
         else if(score + activeScore >= 100) {
             rollAgain = false; // we win!
         }
         else {
             score += roll;
             boolean goalNotMet = score < goalForTurn;
             boolean lessThanOther
                          = score + activeScore < otherScore;

             boolean withinThreshold
                     = otherScore >= closeScore
                     || (score + activeScore >= closeScore);

                 rollAgain = goalNotMet || lessThanOther
                     || withinThreshold;
         }
     }
    return score;
}
```

score, 1
while loop with correct condition, 2
roll, 1
check if pig, 2
check if win, 2
condition for goal met, 2
condition for keeping pace, 2
condition for ending race, 2
return result, 1