CS312 Fall 2019 Exam 2 Solution and Grading Criteria.

Grading acronyms:
AIOBE - Array Index out of Bounds Exception may occur
MCE 0 Major Conceptual Error, Student had serious misunderstanding of question, answered different question, serious errors in code and / algorithm.
BOD - Benefit of the Doubt. Not certain code works, but, can't prove otherwise
Gacky or Gack - Code very hard to understand even though it works. (Solution is not elegant.)
LE - Logic error in code.
NAP - No answer provided. No answer given on test
NN - Not necessary. Code is unneeded. Generally no points off
NPE - Null Pointer Exception may occur
OBOE - Off by one error. Calculation is off by one.
RTQ - Read the question. Violated restrictions or made incorrect assumption.

1. Code Trace: (2 points each) first 3 instances of " on output wrong.

A. `8`
B. `no`
C. `true false`
D `2 10`
E. `RUNTIME ERROR (Exception okay, trying to read non-int as an int.)`
F. `3.5`
G. `5 0.0`
H. `[5, 3, 8, 9]  (differences in brackets and separators okay)`
I. `[4, 5, 6]`
J. `RUNTIME ERROR (Exception okay, ArrayIndexOutOfBounds)`
K. `[0, 12] [4, 1]   (differences in brackets and separators okay)`

2. Array Trace (1 point each)
A. `0`
B. `0`
C. `10`
D `4`
E. **120**

3. Program Logic (0.5 each)

	x == 1	x % 2 == 1	y == 0
POINT A	S	S	A
POINT B	N	S	S
POINT C	S	S	N
POINT D	N	N	S
POINT E	A	A	S

4. Scanners and Strings - 18 Points. The question did not address what to do in case of tie so no however student handled the tie was acceptable. It was also not clear what to do if all the lines with actually ints were negative. Does a line with no ints have a higher sum? Most people wrote code that made this assumption and that was fine.

```java
public static int[] lineWithMax(Scanner sc) {
    int max = Integer.MIN_VALUE;
    int lineWithMax = -1;
    int line = -1;
    while (sc.hasNextLine()) {
        line++;
        Scanner lineScanner = new Scanner(sc.nextLine());
        int lineTotal = 0;
        while (lineScanner.hasNext()) {
            if (lineScanner.hasNextInt()) {
                numValues++;
                lineTotal += lineScanner.nextInt();
            } else {
                lineScanner.next();
            }
        }
        if (lineTotal > max) {
            max = lineTotal;
            lineWithMax = line;
        }
    }
    int[] result = {lineWithMax, max};
    return result;
}
```

- variables for max sum, line with max sum, and line number (can use array for the first 2), 1 point
- max initialized correctly, 1 point (Must be first line total assuming first line has an int or Integer.MIN_VALUE.
- outer while loop correct, 2 points
- create new Scanner for line, 1 point
- correctly increment line number, 1 point
- variable for total on ints in line, 1 point
- correctly loop through tokens in line, 2 points
- correctly check if next token is an int and if so add to running total, 3 points
- skip non int tokens, 3 points
- check if current line total greater than max and if so update variables correctly, 2 points (Okay to assume at least one line will have positive values.) (lose if check in inner loop. What idfstarts with positive ints but then line has many negative ints bring down actual line total.)
- correctly return array with proper results, 1 point

OTHER:

- infinite loop, - 6
- inner loop simply checks hassNextInt -5 (need to read all tokens in line)
- reading extra lines which skips them, -5

5. Scanners - 15 points By far the hardest question on the exam. Many answers only checked pairs of tokens and did not correctly check the runs. Many answers did not track the previous token or char correctly. Likewise, many answers did not set the run length correctly. If a token don't match, the second token is the start of a run of length 1.

```
    public static int numTokensToGetAlliterationRun(Scanner sc,
                                                              int runLength) {
        int tokensRead = 1;
        int currentRun = 1;
        String previous = sc.next();
        while (sc.hasNext() && currentRun < runLength) {
            tokensRead++;
            String current = sc.next();
            if (current.charAt(0) == previous.charAt(0)) {
                currentRun++;
            } else {
                currentRun = 1;
            }
            previous = current;
        }
        int result = -1;
        if (currentRun == runLength) {
            result = tokensRead;
        }
        return result;
    }
```

- variable to track number of tokens and updated correctly, 1 point
- initialize first String to first token or char to first char or ' ', 2 points
- loop while tokens left, 1 points
- stops as soon as run found with the desired length, 2 points
- increment tokens read, 1 point
- check current String starts with same char as previous String correctly, (check run continues correctly), 3 points
- reset run length to 1 on mismatch, 2 points
- correctly update previous token or char when doesn't match, 2 points
- return -1 if run not found, 1 point

OTHER:

- calling next without checking hasNext, -4
- String methods besides charAt, -3 (can vary)
- infinite loop, -5
- skipping tokens, -4
- reading pairs of tokens instead of runs of tokens, -5
- out of scope variables, -2
- using arrays, -5
- inner loop that is a for loop, this is a Major Conceptual Error, MCE. Assumes the next runLength tokens exists and start with the same character.

6. Arrays 1 - 16 points

```
public static boolean containsAll(int[] ar1, int[] ar2) {
    for (int i = 0; i < ar1.length; i++) {
        boolean found = false;
        int j = 0;
        while (!found && j < ar2.length) {
            found = (ar1[i] == ar2[j]);
            j++;
        }
        if (!found) {
            return false;
        }
    }
    return true;
}
```

- outer loop for elements of ar1 correct, 3 points
- inner loop with correct bounds, 3 points
- correctly access length field of arrays, 1 point
- stop inner loop as soon as we find a match for current element, 2 point
- correctly compare elements, 2 points (-1 if .equals OR -1 wrong indices)
- correctly access elements from arrays, 2 points
- stop outer loop as soon as we don't find a match for an element from ar1, 2 points
- return true correctly, 1 point

OTHER:

- switch arrays, -3
- only depends on last element -5
- failure to reset inner check (boolean, int), -4
- early return of true, -5
- off by one errors, -1
- array index out of bounds, -4
- infinite loops -5
- any new arrays, -5
- altering arrays, -5

7. Arrays 2. 16 points.

```java
public static int[] doubleArray(int[] data) {
    int[] result = new int[data.length * 2];
    for (int i = 0; i < data.length; i++) {
        if (data[i] % 2 == 0) {
            result[i * 2] = data[i] * 2;
            result[i * 2 + 1] = -data[i];;
        } else {
            result[i * 2] = -data[i];;
            result[i * 2 + 1] = data[i] * 2;
        }
    }
    return result;
}
```

- create result correctly with proper size, 2 points
- loop through elements of data correctly, 3 points
- correctly calculate new elements, 2 points
- check if current element odd or even correctly, 2 points
- handle indices in the resulting array correctly, 4 points
- correctly place elements in resulting array, 2 points
- return result correctly, 1point

OTHER:

- nested loops, MCE, -5
  not using *, -1
- off by one errors, -2
- array index out of bounds, -4
- infinite loops -5
- altering original array, -5
- disallowed methods, varies