CS 312 – Exam 2 – Fall 2019

Your Name___________________________________

Your UTEID __________________________________

Circle your TA's Name:	Andrew	Anthony	Carla	Fatima	Kate
	Larry	Rebecca	Tim	Varshinee	

Problem Number	Topic	Points Possible	Points Off
1	code trace	22	
2	array code trace	5	
3	program logic	8	
4	scanners & strings	18	
5	scanners	15	
6	arrays	16	
7	arrays	16	
		TOTAL POINTS OFF:	

Instructions:

1. You have 2 hours to complete the exam.
2. You must use a pencil to write your answers on the exam.
3. You may not use any electronic devices, notes, or other resources.
4. When code is required, write Java code. Limit yourself to the features from chapters 1 - 7 of the book and topics 1 - 25 in class. Do not use 2d arrays in your answers.
5. Ensure you follow the restrictions of the question.
6. You may write and call your own helper methods.
7. The exam proctors will not answer questions. If you believe there is an error or a question is ambiguous, state your assumptions and answer based on those assumptions.
8. When you finish, show the proctor your UTID, turn in the exam and all scratch paper.

1. Evaluating Code. 22 points, 2 points each. Assume all necessary imports have been made.
If the snippet contains a syntax error or compiler error, answer **compile 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**.

A. What is the largest possible value the following code could print out?

```
double a1 = Math.random();
double b1 = Math.random();
int x1 = (int) (a1 * 10) - 5;
int y1 = (int) (b1 * 10) - 5;
System.out.print(x1 + y1);
```

`Answer A:` ______________________________

B. Are the two `boolean` expressions logically equivalent? `p`, `q` and `r` are `boolean` variables.

Expression 1: `p || !(!q && r)`

Expression 1: `!q || !p || !r`

`Answer B:` ______________________________

C. What is output by the following code?

```
String c1 = "JAVA";
String c2 = "jaVa";
c2 = c2.toUpperCase();
System.out.print(c1.equals(c2) + " " + (c1 == c2));
```

`Output C:` ______________________________

D. What is output by the following code?

```
int xd = 80;
int yd = 0;
while (xd > 2) {
     xd = xd / 2;
     yd += 2;
}
System.out.print(xd + " " + yd);
```

`Output D:` ______________________________

E. What is output by the following code?

```
Scanner sc = new Scanner("12 BLUE  3  RED");
System.out.print(sc.nextInt() + sc.nextInt());
```

`Output E:` ______________________________

F. What is output by the following code?

```
Scanner sc2 = new Scanner("0.5  0.0  1.5  1.5  3.5  0.5");
double a2 = 0.0;
for (int i = 0; i < 4; i++) {
     a2 += sc2.nextDouble();
}
System.out.print(a2);

Output E: __________________________
```

G. What is output by the following code?

```
double[] ga = new double[5];
System.out.print(ga.length + " " + ga[2]);


Output G: ______________________________________
```

H. What is output by the following code?

```
int[] h1 = {5, 2, 4, 1};
int[] h2 = {h1[1], 3, 1};
h1[2] += h2[2] + h2[1];
h2 = h1;
h1[3] += h1[2];
h1[1]++;
System.out.print(Arrays.toString(h2));


Output H: ______________________________________
```

I. What is output by the following code?

```
int[] i1 = {4, 2, 5};
methodI(i1);
System.out.print(Arrays.toString(i1));

public static void methodI(int[] data) {
     data[1] += data[2] - 2;
     data[2]++;
}

Output I: ______________________________________
```

J. What is output by the following code?

```
int[] j1 = {4, 2, 5, 12, 13};
j1[12] += j1[13];
System.out.print(Arrays.toString(j1));

Output J: ______________________________________
```

K. What is output by the following code?

```
int[] k1 = {4, 2};
methodK(k1);
System.out.print(Arrays.toString(k1));

public static void methodK(int[] data) {
     data[data.length - 1]--;
     data = new int[2];
     data[1] = 12;
     System.out.print(Arrays.toString(data) + " ");
}

Output K: ___________________________________
```

2. Array Tracing - 5 Points
Consider the following method. For each of the given arrays indicate what the method `num2` returns if the array shown is passed as the argument to the method.

```
public static int num2(int[] data) {
     int t = 0;
     for (int i = 1; i < data.length; i++) {
          if (data[i - 1] <= data[i]) {
               t++;
          } else {
               t *= 10;
          }
     }
     return t;
}

A.   argument: []                   num2 returns: _______________

B.   argument: [10]                 num2 returns: _______________

C.   argument: [-5, 5, 0]           num2 returns: _______________

D.   argument: [4, 10, 20, 20, 30]       num2 returns: _______________

E.   argument: [5, 9, 0, 10, 28, 3]      num2 returns: _______________
```

2. Program Logic - 8 Points. Consider the following method. For each of the five points labeled by comments and each of the three assertions in the table, write whether the assertion is ***always*** true, ***sometimes*** true and sometimes false, or ***never*** true at that point in the code. Abbreviate ***always*** with an A, ***sometimes*** with an S and ***never*** with an N. (based on a question from the UW CSE 142, Spring 2019 course)

```
public static int mystery(int x) {
     if (x <= 0) {
          x = 12;
     }
     int y = 0;
     // Point A
     while (x != 1) {
          // Point B
          if (x % 2 == 0) {
               y++;
               x = x / 2;
               // Point C
          } else {
               x = 3 * x + 1;
               // Point D
          }
     }
     // Point E
     return y;
}
```

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

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

4. Scanners 18 points. Write a method `lineWithMax` that given a `Scanner` already connected to a file, returns an array of **`int`**s of length 2.

The first element in the array is the line number, using zero-based indexing, of the line in the file with the maximum sum of integers. The second element in the array is the sum of the integers from the line with the maximum sum.

The file the `Scanner` is connected to is guaranteed to have at least one line with at least one token that can be read as an `int`.

You may create new `Scanner` objects connected to `String`s by calling the `Scanner` constructor.

You may use the `hasNextLine, hasNextInt, hasNextDouble, hasNext, nextLine, nextInt, nextDouble,` and `next` methods from the `Scanner` class.

You may create and use an array of `int`s with a length of 2.

You may use the `Integer.MIN_VALUE` constant.

Do not use any other Java classes or methods.

Example file:

```
12 Varshinee 37 NotAnInt123456

45.61     -1000     -500 No TAs here 100

100 Tim Outstanding!!!! 1000 -500
No ints on this line are there???  12.5
Andrew doing great work!! Num students = 25
Line 1 and line 3 in this file are blank.
```

Given the example above, the method would return an array equal to `[4, 600]`.

Line 4 (using zero-based) indexing has the maximum sum of `int`s, `600 = 100 + 1000 + -500`.

Complete the method on the next page.

```
/* sc is already connected to a file with at least one line with at least
one token that can be read as an int.*/
public static int[] lineWithMax(Scanner sc) {
```

5. Scanners and Strings - 15 Points. The `numTokensToGetAlliterationRun` method accepts 2 parameters:

1. A `Scanner (sc)` already connected to a file. The file is guaranteed to have at least one token.
2. An `int (runLength)` that specifies the number of `String`s (tokens) in a row necessary for a successful outcome. This parameter shall be >= 2.

The method finds a run of `String`s (tokens) in the file that the `Scanner sc` is connected to that start with the same character equal in length to `runLength`. If such a run exists, the method returns the number of tokens that occur in the file up to and including the run of the required length.

Consider this example file:

```
Alliteration means words that start with the same letter or letters or
sounds. For example, two tons both start with t.

Example of a longer alliteration:

slim small sleds skidding

Alliteration in literature and writing don't require the words to
always be consecutive:
"let us go forth to lead the land we love" - JFK
```

For this question we define alliteration to be tokens that start with the same character. Note, based on this definition we could have tokens that alliterate that don't start with letters. For example, the tokens **`8sweep`** and **`8cheer`** alliterate based on our definition for this question because they start with the same character, **`8`**. Also, our definition is case sensitive, so the tokens **`Steep`** and **`swab`** do not alliterate.

If the `runLength` parameter was **`2`** in the example above, the first run of two words that alliterate are **`two tons`** and the method would return 17 because there are 17 tokens from the start of the file until the end of the run of **`two tons`**.

If the `runLength` parameter was **`3`** in the example above, the first run of three words that alliterate are **`slim small sleds`** and the method would return 29 because there are 29 tokens from the start of the file until the end of the run of **`slim small sleds`**.

If the `runLength` parameter was **`4`** in the example above, the first run of three words that alliterate are **`slim small sleds skidding`** and the method would return 30 because there are 30 tokens from the start of the file until the end of the run of **`slim small sleds skidding`**.

If the file does not contain a run of the required length, the method returns -1.

The only methods you may use from the `Scanner` class are the `hasNext()` and `next()` methods.

The only method you may use from the `String` class is the `charAt(int index)` method.
Do not use any other Java classes or methods.

Your method shall avoid doing unnecessary computations.

```
/* sc is already connected to a file. runLength >= 2 */
public static int numTokensToGetAlliterationRun(Scanner sc,
                                                    int runLength) {
```

6. Arrays 1. 16 points. Write a method **`containsAll.`** The method accepts two arrays of **`int`**s as parameters. The method returns true if every element in the first array appears as least once in the second array.

Examples of calls to the method:

```
ar1: [-5, 0, -5, 2, -5]
ar2: [2, 0, -5, 12]
returns true
```

Note, for this question a given value in the second array, **`ar2`**, can be used to match multiple values in the first array. The single **`-5`** in **`ar2`** can be matched to each of the **`-5`**'s in **`ar1`**. The second array does **not** have to have three copies of **`-5`** for **`containsAll`** for this method to return **`true`**. One instance of **`-5`** is sufficient for this question.

```
ar1: []
ar2: [2]
returns true

ar1: [0, 5, 1, 17]
ar2: [1, 0, 17, 17, -5, 15]
returns false (no 5 in ar2)

ar1: []
ar2: []
returns true

ar1: [0, 5, 1, 17]
ar2: [1, 0, 17, 17, -5, 15, 0, 5, 1]
returns true

ar1: [4, 12, 19, 4, 19]
ar2: [-19, 12, 0, 3, 8, -4, 0]
returns false (no 4 or 19 in ar2)
```

Neither array parameter is altered as a result of this method.

Your method shall avoid doing unnecessary computations.

You may not use any other Java classes or methods. You may not create any new arrays.

You may, of course, use the **`length`** field of the given arrays.

Complete the method on the next page.

```
// We expect ar1 != null and ar2 != null.
public static boolean containsAll(int[] ar1, int[] ar2) {
```

7. Arrays 2. 16 points. Write a static method named **`doubleArray`** that accepts an array of **`int`**s as a parameter and returns a new array two times as large as the original. Each element in the original array has 2 corresponding elements in the new array.

Each element in the original array is converted to a pair of elements in the new array. One of the elements is double the original value. The other value is the negative of the original value. If the original value is even, the element that is double comes first in the pair, otherwise the negative of the original value comes first.

Examples of call to **`doubleArray`**:

```
doubleArray ]([]) returns []

doubleArray([21]) returns [-21, 42]

doubleArray([0]) returns [0, 0]

doubleArray([-2]) returns [-4, 2]

doubleArray([21, 2]) returns [-21, 42, 4, -2]

doubleArray([1, 3, 6]) returns [-1, 2, -3, 6, 12, -6]

doubleArray([10, 5, 20]) returns [20, -10, -5, 10, 40, -20]

doubleArray([12, 13, 7]) returns [24, -12, -13, 26, -7, 14]
```

The array given as parameter is not altered as a result of this method.

You may not use any other Java classes or methods. You may create and use a single array of **`int`**s.

You may, of course, use the **`length`** field of the given array.

Complete the method on the next page.

```
// We expect data != null.
public static int[] doubleArray(int[] data) {
```