Your Name_____

Your UTEID _____

Problem Number	Topic	Points Possible	Points Off
1	expressions	10	
2	code trace	22	
3	critters	16	
4	programming	17	
5	arrays 1	16	
6	arrays and objects	17	
7	ArrayLists	16	
8	2d arrays	16	
	TOTAL POINTS OFF:		
	SCORE OUT OF 130:		

Instructions:

1. Please silence your mobile devices.
2. You have 3 hours to complete the exam.
3. Complete the exam without any outside assistance including electronic devices.
4. When code is required, write Java code.
5. Ensure you follow the restrictions of the questions.
6. You **are** allowed to add your own helper methods.
7. When you finish, show the proctor your UTID, turn in the exam and all scratch paper.

1. Short Answer 1 - Expressions. 1 point each, 10 points total.

For each Java expression in the left hand column, indicate the resulting value in the right hand column.

You must show a value of the appropriate type. For example, 7.0 rather than 7 for a double, "7" instead of 7 for a String, and '7' for a char.

Answers that do not indicate the data type correctly are wrong.

A. `(int) (5 / 2 + 1.9 / .5)` _____

B. `12 % 15 + 15 % 12 + 291 / 100` _____

C. `"GDC".equals(new char[] {'G', 'D', 'C'})` _____

D. `"CS_PODS".substring(3).charAt(2)` _____

E. `"ECLIPSE".substring(2, 6).substring(3, 3)` _____

F. `(13 % 5 > 2) || (16 % (4 - 2 * 2) == 3)` _____

G. `Math.random() < 0.0 && Math.random() > 1.1` _____

H. `10 / 3 + 1.3 * 2 + 1 / 2` _____

I. `3 + 2 * 3 + "RING" + 10 / 2 + 5` _____

J. `"ABBA_BAD_CO".indexOf("BA", 4)` _____

2. Code Tracing - 1 point each, 22 points total. For each code snippet state the exact output to the screen. **Placed your answers to the right of the snippet and put a box around each answer.**
Assume all necessary imports have been made.
If the snippet contains a syntax error or other compile error, answer COMPILE ERROR.
If the snippet results in a runtime error or exception, answer RUNTIME ERROR.
If the snippet results in an infinite loop, answer INFINITE LOOP.

A.
```
int[] data1 = {3, 1, 5};
a(data1, 3);
System.out.print(Arrays.toString(data1));

public static void a(int[] arr, int lim) {
     for (int i = 0; i < lim; i++) {
          int j = 0;
          while (j < i) {
               if (arr[j] % 2 == 1) {
                    arr[i] += arr[j];
               }
               j++;
          }
     }
}
```

B.
```
// Uses method a from part A.
int[] data2 = {4, 0, 2};
a(data2, 4);
System.out.print(Arrays.toString(data2));
```

C.
```
// Uses method a from part A.
int[] data3 = new int[4];
a(data3, 2);
System.out.print(Arrays.toString(data3));
```

D.
```
// Uses method a from part A.
int[] data4 = {1, 2, 1, 3, 0};
a(data4, 5);
System.out.print(Arrays.toString(data4));
```

E.

```
ArrayList<String> listE = new ArrayList<>();
System.out.print(listE.size() + " " );
listE.add("E");
listE.add("GC");
listE.add(0, "F");
System.out.print(listE+ " " );
```

F.

```
int xf = 13;
double[][] dataF = new double[xf / 3][xf / 2 + 2];
System.out.print(dataF.length + " " + dataF[1].length
          + " " + dataF[1][2]);
```

G.

```
int[] dataG = {6, 3, 6, 10, 5};
int xg = dataG[dataG[1]];
System.out.println( (xg < dataG.length) && (dataG[xg] == 0));
```

H.

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

public static void h(int[] a) {
     a[1] *= a[2] + 1;
     a = new int[0];
}
```

I.

```
ArrayList<Integer> list2 = new ArrayList<>();
list2.add(list2.size());
list2.add(list2.size());
list2.add(7);
list2.add(list2.size());
list2.add(5 + list2.size() + list2.get(3));
list2.set(2, list2.size());
list2.remove(3);
list2.add(0);
System.out.print(list2);
```

J.

```
int xj = 5;
int[] dataj = new int[xj];
System.out.print(dataj[0] + " " + j(dataj));

public static int j(int[] data) {
     return 12 / data[2];
}
```

K.

```
int[] dataK = {0, 2};
System.out.print(Arrays.toString(dataK) + " " + k(dataK));

public static int k(int[] data) {
     data[0] += 2;
     data[1] += 3;
     System.out.print(data[1] + " ");
     return data[0] + data[1];
}
```

L.

```
int x5 = 10;
int y = 12;
int z = 20;
int t = 0;
while (x5 != y || y != z) {
     x5 += 4;
     t++;
}
System.out.print(t);
```

For the short answer questions M through V consider these classes:

```
public class Computer {
     private int memory;

     public Computer() { memory = 8; }
     public Computer(int m) { memory = m; }

     public void upgrade() { memory *= 2; }
     public String toString() { return "" + getMemory(); }
     public int getMemory() { return memory; }
}

public class PC extends Computer {

     public int getMemory() { return 10; }
}

public class Mac extends Computer {
     private int colors;

     public Mac(int c, int m) {
          super(m);
          colors = c;
     }

     public int getColors() { return colors; }
}
```

M. Consider the following statements. For each, circle if it is legal or if it causes a syntax error.

```
PC p1 = new PC(16);           // legal       syntax error

Computer c1 = new PC();        // legal       syntax error
```

N. Consider the following statements. For each, answer circle it is legal or if it causes a syntax error.

```
Mac m1 = new Computer(12);    // legal       syntax error

PC m2 = new Mac(10, 8);       // legal       syntax error
```

For each code snippet what is output?
If the snippet contains a syntax error or other compile error, answer COMPILE ERROR.
If the snippet results in a runtime error or exception, answer RUNTIME ERROR.

O.

```
Mac nm = new Mac(8, 4);
System.out.print(nm.getColors() + " " + nm.getMemory());
```

P.
```
Computer oc = new PC();
System.out.print(oc.getMemory());
```

Q.
```
Computer cp = new Mac(4, 8);
System.out.print(cp + " " + cp.getColors());
```

R.
```
PC pq = new PC();
System.out.print(pq);
```

S.
```
Mac mr1 = new Mac(8, 4);
Mac mr2 = new Mac(8, 4);
System.out.print( (mr1 == mr2) + " " + mr2.equals(mr1));
```

T.
```
Computer cs = new Computer(4);
t(cs);
System.out.print(cs.toString());

public static void t(Computer c) {
     c.upgrade();
     c.upgrade();
}
```

U.
```
PC pt1 = new PC();
Computer ct1 = pt1;
u(pt1);
System.out.print(pt1 == ct1);

public static void u(PC p) {
     p = new PC();
     p.upgrade();
}
```

V. If we try to add the following method to the `Mac` class, it won't compile. Explain in one sentence why not.

```
public void upgrade(int x) { memory += x; }
```

3. Critters - 16 Points. Implement a `Yak` class that extends the `Critter` class. The only methods you must override from the `Critter` class are the **`fight, eat`** and **`getMove`** methods. Do not override any other methods. Add a constructor if you believe it is necessary.

A Yak moves in the following way. It picks a random direction NORTH, EAST, SOUTH, or WEST with each direction being equally likely. It then moves 1 step in that direction. It then picks another random direction, again with all four directions being equally likely, and then moves 2 steps in that direction. At the completion of the 2 step leg it picks another random direction, again with all four directions being equally likely, and then moves 3 steps in the direction. The `Yak` continues this pattern of random directions with growing leg / segment lengths.

When fighting, a Yak ROARs if it would move NORTH or SOUTH the next time asked otherwise a Yak POUNCEs.

A Yak always returns `true` when asked to eat.

```
public abstract class Critter {

     public boolean eat() {  return false; }

     public Attack fight(String opponent) {
          return Attack.FORFEIT;
     }

     public Color getColor() { return Color.BLACK; }

     public Direction getMove() { return Direction.CENTER; }

     public String toString() { return "?"; }

     // constants for directions
     public static enum Direction {
          NORTH, SOUTH, EAST, WEST, CENTER
     };

     // constants for fighting
     public static enum Attack {
          ROAR, POUNCE, SCRATCH, FORFEIT
     };
```

Include the class header and instance variables. Assume any necessary imports are already made.

You may use the `Math.random()` method, the `Critter` class, and the `Attack` and `Direction` enumerated types, including the `static Direction[] values()` method from the `Direction` enumerated type, but no Java classes or methods.

Write the complete Yak class on this page:

4. Data Processing and Arrays 17 points. Write a method that determines if a file contains at least the required number of each English capital letter, A to Z.

The required number of capital letters are specified via an array of `int`s. The length of the array shall be 26 and each element shall be greater than or equal to 0.

The first element in the array specifies the required number of 'A's, the second element in the array specifies the required number of 'B's, and so forth. The last element in the array specifies the minimum required number of 'Z's.

Consider this example file:

```
The Texas Longhorns women's volleyball team
represents The University of Texas at Austin in NCAA
Division I intercollegiate women's volleyball
competition. The Longhorns currently compete in the
Big 12 Conference.

Texas has won three volleyball national
championships – one AIAW championship in 1981 and
two NCAA championships in 1988 and 2012.
```

There are 6 capital T's in the sample file. If the required number of T's were **6 or less**, then the file has the required number of T's. If the required number of T's is 7 or more, the file does not have the required number of T's.

Your method must determine if the data source the `Scanner` is connected to has the minimum required number of each capital letter. The number of occurrences in the data source does not have to equal the specified value, but can be greater than or equal to the minimum required number.

You may use the `hasNext` and `next` methods from the `Scanner` class and the `length` and `charAt` methods from the `String` class.

Do not use any other Java classes or methods.

Do not create any new arrays.

You may alter the array passed as a parameter and use its `length` field.

The method returns a `boolean`. `true` if the data source contains the required number (or more) of each capital letter, `false` otherwise.

```
/* required.length = 26. All elements of required >= 0. */
public static boolean capitalLettersPresent(Scanner sc,
                                            int[] required) {
```

5. Arrays - 16 Points. Write a method `copyWithoutRange`.

The method accepts 3 parameters, an array of `int`s, and two `int`s, `start` and `stop`, that indicate the starting index and stopping index of the range to omit from the resulting array.

The method creates and returns a new array with the elements from the original array, except elements from the `start` parameter inclusive to the `stop` parameter exclusive `[start, stop)`, are not present.

Examples of calls and results of various calls to `copyWithoutRange`: (Elements not present in the returned array are shown in bold.)

```
copyWithoutRange({0, 1, 2, 3, 4, 5}, 2, 4) -> returns {0, 1, 4, 5}

copyWithoutRange({0, 1, 2, 3, 4, 5}, 2, 3) -> returns {0, 1, 3, 4, 5}

copyWithoutRange({0, 1, 2, 3, 4, 5}, 2, 2) -> returns {0, 1, 2, 3, 4, 5}

copyWithoutRange({0, 1, 2, 3, 4, 5}, 6, 6) -> returns {0, 1, 2, 3, 4, 5}

copyWithoutRange({0, 1, 2, 3, 4, 5}, 0, 6) -> returns {}
```

You may create one additional array.

The original array is not altered by the method.

You may not use any other Java classes or methods other than the array length field.

Assume 0 <= startIndex <= array length, 0 <= stopIndex <= array length,
and startIndex <= stopIndex

```
public static int[] copyWithoutRange(int[] vals, int start, int stop)
{
```

6. Arrays and Objects - 17 points. Write a method that given an array of `Point` objects, returns the minimum distance between the two closest `Point` objects in the array.

For example, if we had the following array with 5 `Point` objects:

```
{(0, 2), (-5, -3), (1, 3), (2, 4), (0, 0)}
```

The two `Point` objects closest to each other are the `(1, 3) and (2, 4)` with a distance of approximately 1.414 between them.

Assume the Point class has the following methods:

```
int getX() Returns the x coordinate of this Point.

int getY() Returns the y coordinate of this Point.

double distance(Point p2) Returns the distance between this Point and
p2.
```

Assume the array has at least two elements and none of the elements in the array store `null`.

Do not create or use any other data structures such as arrays or `ArrayLists`.

Do not use any methods besides the given methods from the `Point` class.

Of course you can use the array length field.

```
/* Assume pts.length >= 2 and none of the elements store null. */
public static double minDistance(Point[] pts) {
```

7. ArrayLists and Data Processing 16 points. Write a method that alters a given `ArrayList` of `Strings` by removing from the `ArrayList`, values that appear in a file connected to a `Scanner`. The method returns the number of values removed from the `ArrayList`.

For example, if the `ArrayList` initially contains these `String`s (quotes not shown):

```
[Gold, Silver, King, TAs, Silver, Fatima, Carla]
```

And the Scanner is connected the following file:

```
493 King 1 guys 12 * 2465 Dayanny -19
20 13 Silver
Kate 37 Mallory 5      Anthony  -5.0 111   1011
Daniel       78        73
13 Aish 31  Larry19Rebecca-13Fatima   Carla    0
silver  King
```

Then the bolded elements would be removed from the `ArrayList`.

```
[Gold, Silver, King, TAs, Silver, Fatima, Carla]
```

And the `ArrayList` would become:

```
[Gold, TAs, Silver, Fatima]
```

If the `ArrayList` contains duplicate values and the value occurs in the file, the value closest to the beginning of the `ArrayList` is removed. The `ArrayList` in the example contains two `String`s equal to `"Silver"`. The file only contains a single `"Silver"`. Thus the first `"Silver"` (closest to the beginning) is removed. The second occurrence of `"Silver"` in the `ArrayList` is not removed. The file would have to contain a second `"Silver"` to remove the second `"Silver"` from the `ArrayList`.

Recall the following methods from the `ArrayList` class. **These are the only `ArrayList` methods you may use in your answer.**

```
int size() Number of elements in the list.

get(int pos)  Returns the element at the given position.

remove(int pos) Remove and return the value at the given position.
```

Do not create or use any other objects such as arrays or other `ArrayList`s.

Do not use any methods besides the given `ArrayList` methods, instance methods from the Scanner class, and the String equals method. Do not use any other Java classes or methods.

```
/* Assume list.size() >= 1. */
public static int removeStrings(Scanner sc, ArrayList<String> list) {
```

8. 2D arrays - 16 Points. Write a method that given a 2d array of `ints` and the row and column of the lower right corner of a rectangular region, ensures all the elements in the specified area are greater than or equal to a given target value. This is referred to as clamping values.

The method specifies the rectangular area with a width and height and the row and column of the lower right corner of the rectangular area. **Note, the area specified may include cells that are out of bounds. Your method shall not suffer an out of bounds error if this occurs. It shall check and if necessary change all the elements in the specified region that are in bounds.**

The parameters include a target value. Any values in the specified region that are less the target value are set to that target value.

Consider the following example. The initial 2d array is on the left.

The lower right corner is row 3, column 6.

If the height of the specified region is 2, the width of the specified region is 4 and the target value is 10, then the array is altered to what is show on the right.

6	10	4	15	3	7	-6
-1	1	2	3	2	0	23
5	17	30	15	2	37	8
2	1	10	2	10	13	**2**
2	2	5	8	1	12	54

6	10	4	15	3	7	-6
-1	1	2	3	2	0	23
5	17	30	**15**	**10**	**37**	**10**
2	1	10	**10**	**10**	**13**	**10**
2	2	5	8	1	12	54

Here is another example with the same starting matrix, but with the lower right corner equal to row 2, column 3. The height of the specified region is 1 and the width is 6.

The target value is 25. Some of the specified cells are out of bounds and those are simply ignored.

6	10	4	15	3	7	-6
-1	1	2	3	2	0	23
5	17	30	**15**	2	37	8
2	1	10	2	10	13	2
2	2	5	8	1	12	54

6	10	4	15	3	7	-6
-1	1	2	3	2	0	23
25	**25**	**30**	**25**	2	37	8
2	1	10	2	10	13	2
2	2	5	8	1	12	54

Assume the row and column that specify the lower right corner of the region to alter will be in bounds.

Complete the method on the next page. **Do not create any new arrays. Do not use any other Java classes or methods. You may alter the given array.**

Of course you can use the array length field.

```
/* mat is a rectangular matrix. r and c specify the lower right corner
of the region to alter. w and h indicate the width and height of the
region to alter. Complete the method per the problem description. */
public void clampValues(int[][] mat, int r, int c, int w, int h,
                                int tgt) {
```