Your Name_____

Your UTEID _____

Problem Number	Topic	Points Possible	Points Off
1	expressions	10	
2	code trace	20	
3	critters	15	
4	Scanners	15	
5	arrays and Strings	15	
6	arrays	15	
7	ArrayLists	15	
8	2d arrays	15	
	TOTAL POINTS OFF:		
	RAW SCORE OUT OF 120:		

Instructions:

1. Please silence your mobile devices.
2. You have 3 hours to complete the exam.
3. You must complete this exam by yourself. You may not have ANY outside assistance including, but not limited to, 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. `112 % 100 + 2 % 10` _____

B. `"E_ALLEN_EMERSON".substring(3, 7)` _____

C. `(double) 1 / 2 + 3 / 2` _____

D. `"GDC" == "AT_GDC".substring(3)` _____

E. `3 + 4 + "CS" + 2 + 12` _____

F. `"ML".length() > 9 && "SIMULA".charAt(20) == 'a'` _____

G. `"ALGOL".indexOf("go")` _____

H. `1.5 * 2 + 7 / 3` _____

I. `(char) ('A' + 3)` _____

J. `(int) (2.99 * 2)` _____

2. Code Tracing - 1 point each, 20 points total. For each code snippet state the exact output to the screen. **Placed your answers to the right of the snippet in the provided box.**
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.
```
String sa = "SOOP";
String[] sa1 = {"CC", sa.substring(1)};
String[] sa2 = {"OOP", "LANG", "ALGOL"};
System.out.print(sa2[1].equals(sa1[1]));
```

B.
```
int b = 5;
int c = 3;
System.out.print( methodB(b, c) + " " + methodB(b, b));

public static int methodB(int x, int y) {
     x += 2;
     y--;
     System.out.print(x + " ");
     return x - y;
}
```

C.
```
int[] dataC = {2, -5, 10};
methodC(dataC);
System.out.print(Arrays.toString(dataC));

public static void methodC(int[] nums) {
     nums[nums.length - 1] += nums[1];
     nums[2] *= 3;
     nums[0]--;
     nums[1] = nums[1] + 3;
}
```

D.
```
ArrayList<String> list = new ArrayList<>();
list.add("K");
list.add("X");
list.add("C");
list.add(1, "GG");
list.add(2, "OO");
list.set(3, list.remove(1));
System.out.print(list);
```

E. Given the following expression, how many of the 8 combinations of values for `p`, `q`, and `r` (all `boolean` variables) result in the expression evaluating to `true`?

`!(p && !q) || !(r && p)`

F.

```
int f = 12;
methodF(f);
System.out.print(f + " ");
f = methodF(-f);
System.out.print(f);

public static int methodF(int x) {
     x = x / 2 + 3;
     x++;
     return x;
}
```

G.

```
String[] dataG = {"ADA", "ML"};
methodG(dataG);
System.out.print(Arrays.toString(dataG));


public static void methodG(String[] data) {
     data[1] = "C#";
     data = new String[] {"C++", "JAVA", "SED"};
     System.out.print(data.length + " ");
}
```

H.

```
double[] hArray = new double[5];
double s = 2.0;
for (double a : hArray) {
     s += a;
}
System.out.print(s);
```

I.

```
int[] dataI = {2, 1, 3};
methodI(dataI, dataI);
System.out.print(Arrays.toString(dataI));
```

```
public static void methodI(int[] ar, int[] nums) {
     ar[1] = ar[1] + nums[1];
     nums[2] = ar[0] + nums[1];
     ar[0]++;
}
```

J.

```
ArrayList<Integer> list2;
list2 = new ArrayList<>();
list2.add(list2.size());
methodJ(list2);
System.out.print(list2);
```

```
public static void methodJ(ArrayList<Integer> list) {
     list.add(0, 7);
     list.add(new Integer(5));
     list.add(list.get(1));
}
```

For the short answer questions K through T consider these classes:

```
public class Dwelling {
     private int rooms;
     private String street;

     public Dwelling () { street = "Elm"; }

     public Dwelling (int r) {
          this();
          rooms = r;
     }

     public void setName(String s) { street = s; }
     public String toString() { return street + rooms; }
     public void addRooms() { rooms += 2; }
     public int rooms() { return rooms; }
}

public class Apt extends Dwelling {

     public Apt(int r) { super(r + 2);  }

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

public class House extends Dwelling {

     private boolean garage;

     public House(boolean g, String str) {
          garage = g;
          setName(str);
     }

     public boolean hasGarage() { return garage; }

     public int rooms() { return 8; };
}

public class CityHouse extends House {
     public CityHouse() { super(false, "1st"); }
}
```

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

```
Dwelling d1 = new CityHouse();     // legal       syntax error

Apt a1 = new Object();             // 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.

L.
```
House h1 = new House(true, "MAIN");
System.out.print(h1);
```

M.
```
CityHouse h2 = new CityHouse();
System.out.print(h2);
```

N.
```
Apt a3 = new Apt(5);
System.out.print(a3 + " " + a3.rooms());
```

O.
```
Dwelling d3 = new House(true, "OAK");
System.out.print(d3.rooms() + " " + d3);
```

P.
```
Dwelling d4 = new CityHouse();
System.out.print(d4.hasGarage()
          + " " + d4.rooms());
```

Q.
```
Dwelling d5 = new Dwelling(2);
Dwelling d6 = new Dwelling();
d6.addRooms();
System.out.print(d6.equals(d5));
```

R.
```
House hs = new House(true, "WALK");
hs.addRooms();
hs.setName("PARK");
System.out.print(hs.hasGarage() + " " + hs);
```

S. The following class does not compile. Why not?

```
public class RandomHouse extends Random, CityHouse {}
```

T. What class does the **`Dwelling`** class inherit from?

3. Critters - 15 Points. Implement a **Sloth** 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.

As the name suggests. **Sloth**s move rather slowly. The first 10 times they are asked to move a **Sloth** responds with the direction **CENTER**. The 11th time they are asked to move they respond with **WEST**. The **Sloth** then repeats this pattern of 10 "moves" of **CENTER** followed by one move to the **WEST**.

When fighting a **Sloth** respond with **SCRATCH** if it would return **CENTER** the next time it is asked to move, otherwise it responds with **ROAR**.

When it is born a **Sloth** is given a positive integer value. This value is used to determine if the **Sloth** eats or not when it is asked if it wants to eat. If the number of times the **Sloth** has responded with **WEST** when asked to move plus the number of times the **Sloth** has had its fight method call is less than the given integer, then the **Sloth** does not eat, otherwise it does.

```
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 `Critter` class, and the `Attack` and `Direction` enumerated types, but no Java classes or methods.

Write the complete `Sloth` class on this page. Following good object oriented practices when completing the class.

4. Data Processing 15 points. Write a method that determines which is larger, the sum of the **`int`**s in a file, the sum of the **`double`**s in the file, or the sum of the lengths of all other tokens in the file.

The sum for doubles shall only consist of those tokens that could be interpreted as **`double`**s, but not **`int`**s.

Consider this example file:

```
12    first words  -6    0.05   .100

more words  -10

2.0 3.5

last line here
```

The sum of the **`int`**s in the sample file is -4 (12 + -6 + -10).

The sum of the **`double`**s in the sample file is 5.65 (0.05 + .100 + 2.0 + 3.5)

The sum of the lengths of all other tokens in the file is 31 (5 + 5 + 4 + 5 + 4 + 4 + 4)

The method returns a **`String`** indicating which of the three sums had the largest total. The method returns **`"I"`** if the **`int`**s has the largest sum, **`"D"`** if the **`double`**s has the largest sum, or **`"T"`** if the sum of the lengths of the other tokens has the largest sum.

If there is a tie among one or more of the sums, the method can return the corresponding **`String`** for any one of the highest sums.

You may use the **`hasNext, hasNextInt, hasNextDouble, nextInt, nextDouble, and next`** methods from the **`Scanner`** class and the **`length`** method from the **`String`** class.

Do not use any other Java classes or methods.

Complete the method on the next page.

```
// sc != null.
public static String processFile(Scanner sc) {
```

5. Arrays and Strings - 15 Points. Write a method that given a **`String`** creates and returns the ***suffix array*** for that **`String`**.

A suffix array is a sorted array (you can use the **`Arrays.sort`** method) of all the suffixes of a **`String`**.

Consider the simple String **`"good"`**. This String has 4 non-empty suffixes: `d, od, ood,` and `good`

Note, in the following examples the doubles quote for **`String`**s are not shown.
Before sorting the suffix array could be **`{good, ood, od, d}`**. After sorting the suffix array would be **`{d, good, od, ood}`**.

To ensure that no suffix is the prefix to another suffix a special sentinel or anchor character is added to the end of each suffix, including the empty **`String`**. The sentinel character is typically represented with a **`"$"`** when looking at examples. This character must be less than (smaller when doing comparisons) than all other characters that can appear in the String we are forming suffixes of.

With the addition of the sentinel character the unsorted suffix array
starts as **`{good$, ood$, od$, d$, $}`**.

After sorting the suffix array becomes **`{$, d$, good$, od$, ood$}.`**

Use the character with the ASCII code (value) of 5 as the sentinel character for this question.
Not the character **`'5'`**, rather the character with the assigned ASCII code 5.

Example 2. The suffix array of the string **`"queue"`** after sorting (shown vertically) would be:

```
$
e$
eue$
queue$
ue$
ueue$
```

Even though these examples only show lower case letters, the **`String`** parameter can contain any characters except the character with the assigned ASCII code of 5.

You may create and use an array of `String` variables. You may of course use the length field from the array you create.

You may use the `charAt(int index)` and `length()` methods from the `String` class and `String` concatenation.

You may call the `Arrays.sort` method.

Do not use any other Java classes or methods.

Complete the method on the next page.

```
/* s != null. No characes of s are the character with the assigned
ASCII code 5. */
public static String[] getSuffixArray(String s) {
```

6. Arrays - 15 points. (Based on a question from UW CSE 142)

Write a method named **`lastIndexOf`** that is given an array of **`int`**s and an **`int`** N.

The method determines the last index of each value between 0 and N inclusive in the given array of **`int`**s.

These values are stored in an array of length N + 1. The method returns this array as a result.

If a value does not appear in the given array, then the element at that index is set to -1.

If the given array is **`[5, 0, 3, 12, 3, 7, 0, -3, 36]`** and N = 3 then the returned array would be:

```
 0   1   2  3
[6, -1, -1, 4]
```

Note in the given array the last 0 is at index 6. Therefore, in the resulting array the element at index 0 stores 6.

Neither 1 or 2 appear in the given array. The elements at those indices
in the resulting array are -1.

The last 3 in the given array is at index 4 and so the element at index 3 in the resulting array stores a 4.

You may create and use one array of `int`s that is returned by the method.

You may of course use the length field for arrays.

Do not use any other Java classes or methods.

Your method shall avoid doing unnecessary computations.

Complete the method on the next page.

```
// data != null, n >= 0
public static int[] lastIndexOf(int[] data, int n) {
```

7. ArrayLists and Objects, 15 points. This question uses a new class called **`BikeRaceRecord`**. This class stores information about bicycle races, specifically the name of the race and the fastest winning time ever for the race. The public methods for the **`BikeRaceRecord`** class are:

```
public class BikeRaceRecord {
     /*   Each BikeRaceRecord object contains the name of the race,
          such as "Tour de France", and the fastest winning time ever
          recorded for the race expressed simply as seconds. For example,
          18656 for 5 hours, 10 minutes, and 56 seconds. */

     public String getName() // returns the name  of this race

     public int getTime() // returns record time for this race in seconds

     /* returns true if obj is a BikeRaceObject with the same name as
          the calling object regardless of the times */
     public boolean equals(Object obj)
}
```

Write a method **`consolidateRecords`** that given an **`ArrayList`** of **`BikeRaceRecord`** objects removes any **`BikeRaceRecord`** objects that have the same name. If there are multiple **`BikeRaceRecord`** objects retain the one with the minimum time.

The relative order of the remaining items in the **`ArrayList`** is unchanged.

For example, the **`ArrayList`** initially held these values: (The objects that are retained are bolded.)

```
[(Paris-Roubaix, 16200), (Giro, 36010), (Paris-Roubaix, 16000),
     (Paris-Tours, 12600), (Paris-Roubaix, 18000), (Paris-Tours, 12000)]
```

then the ArrayList would hold these values in the order shown after the method had completed:

```
[(Giro, 36010), (Paris-Roubaix, 16000), (Paris-Tours, 12000)]
```

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.

indexOf(E val) Return the index of the first occurrence of the given
object in the list or -1 if it not present.
```

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, the given `BikeRaceRecord` class including all its methods, and the `equals` method from the `String` class.

```
// assume list != null, no elements of list = null
public static void consolidateRecords(ArrayList<BikeRaceRecord> list) {
```

9. 2D arrays - 15 Points. Write a method **`isDistinctColumnSum`** that given a rectangular 2d array (all rows have the same number of columns) of **`int`**s and a column index, returns **`true`** if the sum of the values in the given column is distinct among the sums of the columns in the given 2d array.

In other words, return **`true`** if no other column in the 2d array sums to the same total as the given column

Consider the following 2d array.

8	7	4	3	3	4	-7
6	1	2	3	2	0	3
8	11	0	5	2	7	3
8	9	1	2	7	2	-8
4	2	5	8	1	2	-5

```
34   30   12   21   15   15   -14
```

The sum of each column is show under the column.

If the given column index was 1, the method would return **`true`**. No other column sums to 30.

If the given column index was 4, the method would return **`false`**. Columns 4 and 5 both sum to 15. The column sum of 15 is not distinct in the sample 2d array.

Do not use any other Java classes or methods.

Of course you can use the array length field.

You method shall avoid doing unnecessary computation.

Complete the method on the next page.

```
/* mat != null. mat is a rectangular matrix.
     0 <= index < mat[0].length*/
public static boolean isDistinctColumnSum (int[][] mat, int index) {
```