GMU Spring 2023 – CS 211 – Project 1

Due Date: Thursday, February 16th, 11:59pm

Changelog

- Nothing yet!

Description

The purpose of this assignment is to get practice with strings, arrays, loops and dynamic memory allocation.

Instructions

- Honor Code: This assignment is **individual work** of each student, you're not allowed to collaborate in any form. Copying code from other sources (peers, websites, etc.) is a serious violation of the University's Honor Code. Your code will be examined for similarities with other sources.
- Documentation: Comments are not required in this assignment but it's a good practice to add comments to any piece of code that is not obvious what it does.
- Imports: You may **not** import any classes or packages.
- You are **not** allowed to use any language construct that hasn't been covered in class yet.

Submission

Submission instructions are as follows:

1. Name your file **`P1.java`** and upload it to Gradescope using the following link

   https://www.gradescope.com/courses/498183/assignments/2626464

2. Download the file you just uploaded to Gradescope and compile/run it to verify that the upload was correct
3. Make a backup of your file on OneDrive using your GMU account

   If you skip steps 2 and 3 and your submission is missing the proper file(s), there won't be a way to verify your work and you will get zero points.

Grading

- Grading will be primarily automated.
- Manual grading will be used only for checking hard-coding, violations, comments, etc.
- When you upload your code to Gradescope, the built-in script will run some basic compliance checks for validation purposes only. **These are not logic tests**. The actual grading will take place afterwards and you can't see the tests that will be used.
- If your code doesn't pass all the compliance checks in Gradescope, it means that something fundamental is wrong and we won't be able to grade it unless you fix it. You will get **zero points** if you don't fix and reupload your code.

Testing

We won't be using testers in CS211. The goal is to help you put more focus on writing logically correct programs instead of trying to pass certain tests only. As a programmer, you must learn how to test your code on your own and you must also develop the ability to discover the corner cases that need to be tested.

Task

You will implement the following methods in Java. Every method is self-contained; there are no dependencies among the methods. You're free to create helper methods if you want though. Make sure you do not modify the method signatures that are provided, otherwise your code will not pass the compliance checks and we won't be able to grade it.

```
public static int stringValue(String word)
```

It returns the value of a string which is the sum of the values of its characters. The value of each character is the ASCII code of the character (e.g. **A** has a value of 65, **B** has a value of 66, **a** has a value of 97, etc.) with a few exceptions:

- Double-letters (e.g. ee, mm, ll, tt, etc.) are valued once, not twice.
- The space character has a value of zero
- The value of each number-character is multiplied by the largest value of the non-number-characters in the string.

To simplify things a bit, assume that the word will not include, and your program will not be tested with, any characters that have an ASCII code lower than 40 or higher than 123.

Examples:

word	return	solution
Mason	510	77+97+115+111+110
Ma son	510	77+97+0+115+111+110
mass	321	109+97+115
50 Cent	12110	53*116+48*116+0+67+101+110+116

```
public static double expValue(int x, double precision)
```

The calculation of the e^x function appears a lot in applications and it's considered quite an expensive computation. One way to calculate it is with the following formula:

$$e^x = \sum_{k=0}^{\infty} \frac{x^k}{k!} = 1 + x + \frac{x^2}{2} + \frac{x^3}{6} + \frac{x^4}{24} + \ldots$$

As you can tell, this series can go for ever. After some point though, the precision of the result doesn't improve much. Therefore, we can stop adding terms when the improvement is less than the **precision** parameter. In other words, if adding a certain term changes the result by less than the value of **precision**, then we do not add this term and we stop the series. The method returns the sum of the series.

```
public static int mirrorNum(int num)
```

The method accepts an integer and returns the mirrored integer. In case of a negative number, it retains the sign.

You may not use any classes from any package, not even the `Math` and `String` classes, to implement the **`mirrorNum`** method

Examples:

num	return
4	4
98014	41089
-593	-395
12300	321

```
public static boolean raisedNum(long num)
```

There are some numbers whose value is the sum of $x^y + y^x$ where x and y are integers greater than 1. Let's call these numbers raised numbers. Write a method that accepts a `long` and returns `true` if the number is raised, and `false` otherwise.

Examples:

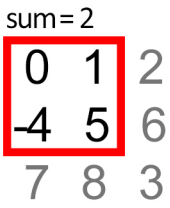
input	return	solution
6	`false`	
17	`true`	$2^3 + 3^2$
593	`true`	$9^2 + 2^9$
1125	`false`	

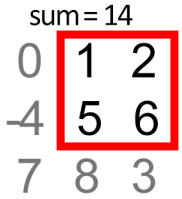
```
public static int[][] smallestSubarray(int[][] array, int sum)
```

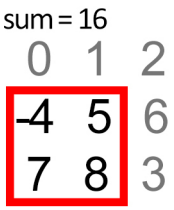
It takes a rectangular array and returns a square-size subarray of it whose values have a sum that is equal or larger to **sum**. A subarray can't be smaller than 2x2. If there are more than one square-size subarrays that satisfy the requirements, the method must return the square that has the smallest size. If two or more equally-sized squares satisfy the requirements, it must return the square whose values have the highest sum. Assume that the **sum** has such a value that always warrants a solution (i.e. no need to check or worry about this).

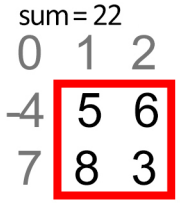
Examples:

array	sum	return
{{0,1,2},{-4,5,6},{7,8,3}}	5	{{5,6},{8,3}}
{{0,1,2},{-4,5,6},{7,8,3}}	23	{{0,1,2},{-4,5,6},{7,8,3}}










```
public static void replaceElement(int[][] array, int elem, int[] newElem)
```

It takes a two-dimensional `array` and modifies it *in-place* (i.e. the method doesn't return anything). Every occurrence of **elem** is replaced by the items contained in **newElem**. The modification must happen *in-place*, i.e. the memory address of the whole array must remain the same; but the memory addresses of its rows can change of course.

Examples:

array before method invocation	elem	newElem	array after method invocation
{{1,2,3,4,5},{6,7,8}}	2	{0}	{{1,0,3,4,5},{6,7,8}}
{{1,2,3,4,5},{5,4,3,2}}	5	{-5,5}	{{1,2,3,4,-5,5},{-5,5,4,3,2}}
{{0,1,2,3,4,5},{5,4,3,4}}	4	{1,2,3}	{{0,1,2,3,1,2,3,5},{5,1,2,3,3,1,2,3}}

```
public static int[][] removeDuplicates(int[][] array)
```

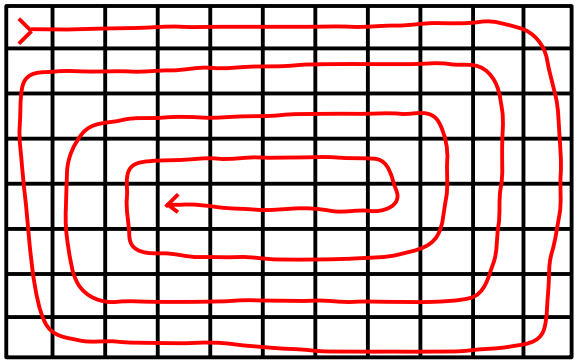
It takes as input a two-dimensional `array` and returns a new two-dimensional array where every sequence of duplicate numbers is replaced by a single occurrence of the respective number. Keep in mind that duplicate numbers can extend in more than one rows though.

Examples:

input array	**returned array**
{{1,2,3,4,5},{6,7,8,9}}	{{1,2,3,4,5},{6,7,8,9}}
{{1,2,2,2,3,4,5,5},{6,7,8,8,8,9}}	{{1,2,3,4,5},{6,7,8,9}}
{{1,2,2,2,3,4,5,5},{5,5,5,5},{5,5,9}}	{{1,2,3,4,5},{9}}

```
public static int[] vortex(int[][] array)
```

It takes as input a two-dimensional rectangular `array`, traverses its contents in a clockwise spiral order starting from index (0,0), and returns the result as a single-dimensional array.



Examples:

input array	**returned array**
{{1,2,3},{4,5,6},{7,8,9}}	{1,2,3,6,9,8,7,4,5}
{{1,2},{3,4},{5,6},{7,8},{9,10}}	{1,2,4,6,8,10,9,7,5,3}
{{1,2,3,4,5},{6,7,8,9,10},{11,12,13,14,15}, {16,17,18,19,20}}	{1,2,3,4,5,10,15,20,19,18,17,16,11,6,7,8,9,14,13,12}