WIC Beginner's Programming Competition

Fall 2013

Problem Set

DO NOT OPEN UNTIL TOLD TO DO SO

Problem 1: Adding Numbers Divisible by 2, 3, & 5

Given an array of integers, find the following three sums:

1. Sum of all numbers divisible by 2.
2. Sum of all numbers divisible by 3. Don't count numbers already included in the previous sum.
3. Sum of all numbers divisible by 5. Don't count numbers already included in either previous sum.

Return these 3 sums in an array of length 3.

Example:

```
[2, 3, 12, 5] => [14, 3, 5]

[1, 32, 43, 5, 6, 56, 65, 21, 0, 2, 6, 3, 12] => [114, 24, 70]
```

Sample File Format:

sample-in: One input per line, containing a series of numbers separated by spaces
sample-out: One line per input, containing the three sums separated by spaces.

Provided Method Signature:

```
//Parameters: int[] nums - numbers to sum
//Returns:    int[] - array of length 3 with the 3 sums
public static int[] sums(int[] nums) {
    //TODO
}
```

Problem 2: First Unique Char

Given a String, find the first unique character in that String. A *unique* character is one that only appears in the String once. The *first* unique character is the first of the unique characters to appear in the String.

Return the first unique character, or '0' if there are no unique characters in the String.

Example:

```
"hdjoqddohca" => 'j'

"bnbnbnmmcidlaia" => 'c'
```

Sample File Format:

sample-in: One input per line, containing a series of characters representing the input String.
sample-out: One line per input, containing a single character representing the first unique character in the input.

Provided Method Signature:

```
//Parameters: String in - String to find first unique char in
//Returns:    char - First unique char in the input String, or '0'
public static char firstUniqueChar(String in) {
    //TODO
}
```

Problem 3: String Translation

Given a String that is comprised of a series of pairs of letters and numbers, return a String consisting of each letter repeated the number of times specified the number following it, concatenated together.

Example:

```
"C3F5G10" => "CCCFFFFFGGGGGGGGGG"

"A0B3" => "BBB"
```

Sample File Format:

sample-in: One input per line, containing a series of letters and numbers representing the input String.

sample-out: One line per input, containing the translated String.

Provided Method Signature:

```
//Parameters: String formatStr - the format string
//Returns:    String - the translated string
public static String translate(String formatStr) {
    //TODO
}
```

Problem 4: Make Me a Sandwich

There is a sandwich shop in which the customer does not need to know exactly what she wants. The customer simply specifies what she absolutely does want, does not want, or what she is okay with or without, and the number of possible combinations will be displayed.

The following ingredients are available:

Bread	Protein Source	Condiments	Produce
Whole Wheat	Tofu	Ketchup	Spinach
Rye	Salmon	Mustard	Avocado

There must be exactly 1 bread in a sandwich. There can be 0-2 protein sources, 0-2 condiments, and 0-2 produce options in a sandwich.

The customer's sandwich preferences will be represented as follows:

- Must contain: 1
- Must not contain: -1
- May or may not contain: 0

The customer's preferences will be provided in an array of length 8 in the following order: [Whole Wheat, Rye, Tofu, Salmon, Ketchup, Mustard, Spinach, Avocado]

Return the number of possible sandwich combinations meeting both the customer's preferences and the constraints above. If making a sandwich according to the customer's preferences is not possible given the constraints above, return 0.

Example:

```
[1, -1, 0, 0, 1, 1, 0, -1] => 8

[1, 1, 0, 0, 1, 1, 0, -1] => 0
```

Sample File Format:

sample-in: One input per line, containing 8 integers representing the customer's sandwich preferences, separated by spaces.
sample-out: One line per input, containing the number of possible sandwich combinations.

Provided Method Signature:

```
//Parameters: int[] prefs - array of customer's sandwich preferences
//Returns:    int - number of possible sandwich combinations
public static int sandwichChoices(int[] prefs) {
    //TODO
}
```

Problem 5: Calculating Track Scores

Five students have participated in five races at a track meet and need their final scores calculated. 1st, 2nd, and 3rd place earn 10, 5, and 1 point respectively. Places below 3rd are not awarded points. Higher scores are better. A tie earns all students in the tie the number of points for that place. The number of students in the tie determines how many places are taken. For example, if two students tie in first, each of the two students gets 10 points and the next points awarded are for third place.

Given a 5x5 array, with each array representing a student's scores in the five races, return an array that contains the final calculated score for each student.

Example:

```
scores[] = [[35, 145, 12, 53, 245], // s0: [ 5, 10,  5,  0,  0]
            [29, 140, 12, 56, 260], // s1: [ 0,  5,  5,  1,  1]
            [40, 122, 10, 55, 300], // s2: [10,  0,  0,  0, 10]
            [28, 128, 11, 60, 266], // s3: [ 0,  1,  0,  5,  5]
            [30, 120, 14, 67, 246]] // s4: [ 1,  0, 10, 10,  1]
return [20, 12, 20, 11, 22]
```

Sample File Format:

sample-in: Five lines per input, with each of the five lines containing five integers representing the student's scores in the five races, separated by spaces.
sample-out: One line per input, containing five integers representing each of the five students' final scores, separated by spaces.

Provided Method Signature:

```
//Parameters: int[][] scores – array of five arrays, each with five
//                             integers representing that student's
//                             score for the five races.
//Returns:    int[] – array of length 5 representing each student's
//                    calculated scores
public static int[] calculateScores(int[][] scores) {
    //TODO
}
```

Problem 6: Exponential Occurrences

Given a String containing only numbers, replace each number in the String with that number to the power of the number of times that it appears in a row in the String. No exponents will overflow a 32-bit signed integer.

Example:

"112344425" => "112364646425" // $1^2 1^2 2^1 3^1 4^3 4^3 4^3 2^1 5^1$

"222" => "888" // $2^3 2^3 2^3$

Sample File Format:

sample-in: One input per line, containing a series of numbers representing the input String.
sample-out: One line per input, containing a series of number representing the output String.

Provided Method Signature:

```
//Parameters: String nums - the input string to convert
//Returns:    String - the converted string
public static String convert(String nums) {
    //TODO
}
```

Problem 7: Split at the Vowels

Given a String, divide it into an array of Strings, splitting immediately after each vowel. Vowels are a, e, i, o, u. A String will be comprised of upper case letters, lower case letters, and spaces.

Example:

```
"hdioqddohca" => ["hdi", "o", "qddo", "hca"]

"I eat an apple" => ["I", " e", "a", "t a", "n a", "pple"]
```

Sample File Format:

sample-in: One input per line, containing a series of characters representing the input String.
sample-out: One line per input, containing a series of characters, with each substring separated by commas.

Provided Method Signature:

```
//Parameters: String str - input String
//Returns:    String[] split - input String, split into substrings
//                             around vowels.
public static String splitString(String str) {
    //TODO
}
```

Problem 8: Match Number

Given two arrays of integers that contain the same numbers in different orders, return an array containing the positions of the numbers of the first array in the second array. Positions are 0-indexed. If a number appears more than once, the first occurrence should contain the first index of that number in the second array, the second occurrence should contain the second index, etc.

Example:

```
a = [1, 4, 2, 7, 3, 9, 6, 2, 8]
b = [6, 9, 2, 4, 1, 5, 7, 8, 3]
return [4, 3, 2, 6, 8, 1, 0, 2, 7]

a = [0, 5, 2, 8, 5]
b = [2, 5, 5, 0, 8]
return [3, 1, 0, 4, 2]
```

Sample File Format:

sample-in: Two lines per input. First line contains a series of space-separated integers representing the first integer array. Second line contains a series of space-separate integers representing the second integer array.

sample-out: One line per input, containing a series of space-separated integers representing the appropriate positions.

Provided Method Signature:

```
//Parameters: int[] a - first array
//            int[] b - second array
//Returns:    int[] - the array of positions
public static int[] numPositions(int[] a, int[] b) {
    //TODO
}
```

Problem 9: Cupcakes

The head chef at a bakery has baked and frosted a batch of her famous cupcakes, and has to now package them to be sent out. The chef has **N** cupcakes, and needs to decide how many cupcakes to place in each package. All packages must contain the same number of cupcakes. The chef will choose an integer **A** between 1 and **N**, inclusive, and place exactly **A** cupcakes into each package. The chef gets to eat leftover cupcakes (when she doesn't have enough to fill a final package). She really enjoys eating cupcakes! Help the chef choose the package size **A** that will let her eat as many cupcakes as possible.

Example:

```
4 => 3

5 => 3
```

Sample File Format:

sample-in: One input per line, containing a single integer representing the number of cupcakes the chef has baked.

sample-out: One line per input, containing the number of cupcakes the chef should place in each box.

Provided Method Signature:

```
//Parameters: int n - the number of cupcakes the chef has baked
//Returns:    int - the number of cupcakes the chef should place into
//                  each package
public static int cupcakes(int n) {
    //TODO
}
```

Problem 10: Encoding Words Using Binary

For a given string, its encoded result will take each letter's numeric position in the alphabet (a = 1, b = 2, c = 3, etc.) and convert it to its binary representation, such that each 0 = '#' and each 1 = '$'. Each binary number should contain 5 digits, filling in leading #'s as needed.

Given a String, return its encoded result as an array of Strings with each letter encoded in binary. The given String will contain only lowercase letters a through z.

Example:

```
"cab" => ["###$$", "####$", "###$#"]

"wic" => ["$#$$$", "#$##$", "###$$"]
```

Sample File Format:

sample-in: One input per line, containing a series of lower case letters representing the input String.

sample-out: One line per input, containing a series of encoded letters, separated by spaces.

Provided Method Signature:

```
//Parameters: String word - word to encode
//Returns:    String[] - string array with encoded letters
public static String[] encode(String word) {
    //TODO
}
```

Problem 11: Number Possibility

Given an array of 5 different positive integer numbers, list all of the numbers that can be calculated by adding some or all of them together. At least one of the numbers must be used in each sum. Return an array of all potential sums sorted in ascending order, without any duplicates.

Example:

```
[1, 2, 3, 4, 5] => [3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15]

[2, 6, 9, 10, 23] => [8, 11, 12, 15, 16, 17, 18, 19, 21, 25, 27, 29,
32, 33, 34, 35, 38, 39, 40, 41, 42, 44, 48, 50]
```

Sample File Format:

sample-in: One input per line, containing a series of integers separated by spaces representing the input array.

sample-out: One line per input, containing a series of integers separated by spaces representing the sorted possible sums.

Provided Method Signature:

```
//Parameters: int[] nums - array of 5 different positive integers
//Returns:    int[] - array of all possible sums, sorted in ascending
//                    order
public static int[] numberSums(int[] nums) {
    //TODO
}
```

Problem 12: Decimals and Arithmetic

Given a double value, first find the integer sum of the digits to the left of the decimal, then divide by the integer after the decimal. Perform an integer division (round down to the nearest whole number).

Example:

```
123.2 => 3 // (1 + 2 + 3)/2 = 3

9989.12 => 2 // (9 + 9 + 8 + 9)/12 = 2
```

Sample File Format:

sample-in: One input per line, containing a single double value.
sample-out: One line per input, containing a single integer.

Provided Method Signature:

```
//Parameters: double num - input number
//Return:     int - sum of digits to the left of the decimal divided
//                  by the integer to the right of the deicimal
public static int sumDivideDecimal(double num) {
    //TODO
}
```

Problem 13: Repetition Free Numbers

A repetition-free number is one in which each digit {1, 2, … , 9} appears at most once, and the digit 0 does not appear. A repetition-free number can have at most nine digits, but it may also have fewer than nine digits.

For a given number N, find the smallest number greater than N that is repetition free. If it does not exist, return 0.

Example:

```
2 => 3

489 => 491
```

Sample File Format:

sample-in: One input per line, containing a single integer value N.
sample-out: One line per input, containing a single integer value that is the next-highest repetition-free number.

Provided Method Signature:

```
//Parameters: int num - the given number N
//Returns:    int - the smallest number greater than num that is
//                  repetition free.
public static int findRepetitionFree(int num) {
    //TODO
}
```

Problem 14: Time

Given a starting time as a String in the form "YYYYMMDDHHMM" (year, month, day, hours, minutes) and an integer representing the number of minutes that have passed since that time, return a String in the same format as the input representing the current time.

Note: The given time is in 24-hour format.

Hint: Leap years are calculated as following: If year is divisible by 400 then it is a leap year. If year is not divisible by 400 but is divisible by 100, then it is not leap year. If year is not divisible by 400 or 100 but is divisible by 4, then it is a leap year. If year is not divisible by 400, 100, or 4, then it is not a leap year.

Example:

```
startTime = "201310311548"
minutes = "1000"
return "201311010828"

startTime = "200002281739"
minutes = "2138"
return "200003010517"
```

Sample File Format:

sample-in: Two lines per input. First line is a series of numbers representing the start time. Second line is an integer representing the number of minutes that have elapsed since the start time.

sample-out: One line per input, containing a series of numbers representing the current time.

Provided Method Signature:

```
//Parameters: String startTime - start time in YYYYMMDDHHMM format
//            int minutes - number of minutes elapsed since startTime
//Return:     String - current time in YYYYMMDDHHMM format
public static String time(String startTime, int minutes) {
    //TODO
}
```