2018 CS PROGRAMMING GURU COMPETITION

Background: Welcome to the 2018 CS / IT Programming GURU Competition. In the enclosed packet you will find custom challenges constructed by CS faculty.

Prior to beginning, please read the contestant documentation found here:
https://www.domjudge.org/docs/team-manual.pdf

Languages

Language	Extension
Java	.java
C	.cc
C++	.cpp
Python 2	.py2
Python 3	.py3

Contest URL: http://gitlord.cs.csuci.edu/domjudge

Contestant Login Info:

UserId: userx (where x is the computer machine number 1 through 24)

Pwd: same as UserId

1. Square Shuffle (Prof. Soltys)

Problem description: Write a program (Python 3) that checks whether a given input string is a square shuffle. That is, you should write a program that runs as follows:

Where x is a string (over any alphabet, it does not matter), and the output will be True or False depending on whether x is a square shuffle. We say that a string w is a shuffle of string x and y if there is a way to express w as a shuffle of x and y, meaning that w is an interleaving of x and y.

For example, if w=0101 then it is a shuffle of x=00 and y=11 since w=x1y1x2y2. Note that the xi's must appear in the same order in w as they do in x, and all the xi must be present in w - in particular |w|=|x|+|y|. Same for the yi's. On the other hand, w=abca is not a shuffle of x=aa and y=cb because the only way to compose it would be as x1y2y1x2 and so the yi's would be out of order.

Finally, we say that w is a square shuffle if there exists an x such that w is a shuffle of x with itself.

Sample computer code attached:

Sample I/O examples:

Input	Output
lattelatte	True
nn whuffleshuffle nn whuffle	True
never gonna never gonna	False
a lion a day a lion a day	False
nanananananaaaannn	False
1101100010010011111001101000110110010001	False

2. Error Detecting Decoder (Prof. AJ)

Error correcting codes are commonly used in data communication. The fundamental idea is to inject some extra information depicting the data into the data stream, so the integrity of the data can be later verified by the receiver. Similar schemes exist for data storage.

A function computing bit parity is the simplest form of an error correcting code. For example, if the data is `11011` an extra bit `0` is appended to form the final data shape as `110110`, because the parity of `11011` is `0`. Hence, the data size N=5 is sent as a binary string of length N=6.

The extra bit can be used to verify the integrity of the data. For example, if `110110` is received, then the data integrity has been preserved, since the parity computed for the received data matches the parity computed when the data was sent. If however, the data received is - say - `100110` then an error is detected, since the parity of `10011` is `1` and therefore it does not match the parity of `0`that was computed when the data was sent.

In the described scheme, one-bit errors can be detected, but cannot be corrected as the position of the erroneous bit cannot be determined. In an enhanced scheme, additional information is injected into the data stream every M*N bits. The extra information takes a form of N bits that represent the parities computed for the M bits at the same position K within each of the M groups of N bits including the parity bits.

The easiest way to look at the scheme is to think of the data as a sequence of two-dimensional MxN arrays. Each of the arrays is extended to an (M+1)x(N+1) array by adding parity bits computed for each of the M rows as the (N+1)-th column and for each of the N columns as the (M+1)-th row.

The transmitter of the data sends the values of N and M at the beginning of each transmission. Each of the numbers is encoded with 4 bits, but the bits are repeated to allow the receiver to verify that they have been received correctly, so N and M occupy the first sixteen bits of each message. For example, for N=4 and M=3, the first sixteen bits of each message would be `0100010000110011`; the actual data encoded as described earlier would follow.

Your task is to implement a decoder that unpacks received data.

Assume that the input always provides valid data; i.e., there is a sufficient number of bits in the message for the given N and M.

For example, for the input:

```
0100010000110011111011010101100001001010101001000101111011111010100111000011100
```

(that verifies correctly) the output should be:

```
11010101110001011001001011100100110 0
```

Please note that the input can be viewed in two dimensions as:

```
0100 << N = 4 (four "columns")
0100
0011 << M = 3 (three "rows")
0011

1101 1 << a "row" of 4 bits of data and their parity
0101 0
1100 0

0100 1 << the parity row of the "columns"

0101 0
1001 0
0010 1

1110 1

1110 1
0100 1
1100 0

0110 0
```

For messages that are not verified positively, the decoder should print an error message. For example, for the input:

```
0100010000110011111010010101100001001010101001000101111011110101001110000 1100
```

the output should be:

```
MESSAGE VERIFICATION FAILURE
```

Note that in the latter case, the parity bit for the first 4 bits of the actual message - i.e., bit number 20 counting from 0 - is `0`, and does not match the computed value of the parity for bits 16, 17, 18, and 19 that is equal to `1`.

That is better visible in a two-0dimensional view of the message:

```
0100
0100
0011
0011

1101 0 <<< ERROR in parity
0101 0
1100 0

0100 1

0101 0
1001 0
0010 1

1110 1

1110 1
0100 1
1100 0

0110 0
```

3. Smith Numbers --- Part II (Prof. Claveau)

A Smith number is a composite number for which, in a given base (in base 10 by default), the sum of its digits is equal to the sum of the digits in its prime factorization. For example, 378 = 2 × 3 × 3 × 3 × 7 is a Smith number since 3 + 7 + 8 = 2 + 3 + 3 + 3 + 7.

In this definition the factors are treated as digits: for example, 22 factors to 2 × 11 and yields three digits: 2, 1, 1.

Therefore 22 is a Smith number because 2 + 2 = 2 + 1 + 1.

Two consecutive Smith numbers are called Smith brothers.

Write a program that finds the first two Smith brothers and return the larger. Your output will be as follows, where the blanks represents Smith number 1 and Smith number 2 as defined above.

Brother 1 = _____
Brother 2 = _____

4. Domino Completeness Problem (Prof. Pilarczyk)

Domino is a set of rectangular tiles with two labels at the ends. (It is possible that two different tiles have the same labels.) Two tiles can be put in a line next to each other if the labels at the meeting ends agree. For example, the tiles {(1,1), (1,2), (1,2), (1,3), (1,4)} can be put in the line as follows: (3,1) (1,1) (1,2) (2,1) (1,4); note that a tile can be rotated to have the labels in the opposite order. We say that a set of domino tiles is complete if it is possible to arrange all the tiles along a circle (a closed loop) in such a way that the meeting ends have the same labels. For example, the previous example is not complete, but if we add the tile (3,4) then we get a complete set. Another example of a set that is not complete is the following: (11,12) (12,11) (30,40) (40,50) (50,30). Note that the set might have just one element; then (1,1) is an example of a complete set, while (1,2) is not.

The problem to solve is the following: Given a set of domino tiles, please, decide if it is complete or not. On the input, the first line contains the number of sets to check, followed by the sets, each defined by the number of its elements followed by the pairs of labels (integers in the range 1-255), without any additional symbols except for spaces and line endings (in order to simplify the reading process). The output should contain answers for all the sets in the order they are defined: a line containing "yes" if the set is complete or a line containing "no" if it is not. Sample input / output is as follows:

Input	Output
5 5 1 1 1 2 1 2 1 3 1 4 6 1 1 2 1 1 2 1 3 4 1 4 3 5 11 12 12 11 30 40 40 50 50 30 1 1 1 1 1 2	no yes no yes no

5. Moving Average Filters (Prof. Dalali)

This problem challenges you to create a moving average filter for a given average window size.

Based on a given sequence of numbers, write a program to apply a filter with the given size on the given input sequence and return an output.

For example, the window size of 3 and the input sequence of:

[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1]

will produce the averaged sequence :

[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.3333333333333333, 0.6666666666666666, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 0.6666666666666666, 0.3333333333333333, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.3333333333333333, 0.6666666666666666, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0]

Sample Output for Window 3:

The input sequence is: [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]

The averaged sequence with window size 3 is [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.3333333333333333, 0.6666666666666666, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 0.6666666666666666, 0.3333333333333333, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.3333333333333333, 0.6666666666666666, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0]

The input sequence to solve for with window size 3:

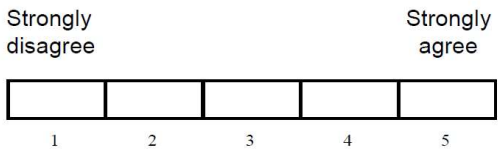
[1, 0, 1, 0, 0, 0, 1, 0, 0, 1, 1, 1, 0, 0, 1, 0, 1, 0, 1, 1, 1, 0, 1, 1, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 0, 1, 0, 1, 1, 1]

6. Scoring the System Usability Scale (Prof. Thoms)

SUS Background

The System Usability Scale (SUS) is a simple, ten-item scale (1 being strongly disagree and 5 being strongly agree) that provides a global view of subjective assessments of technology usability. The questions SUS ask are detailed in Table 1 below.

Table 1 - SUS Questionnaire

1. Use the system frequently. 2. System is unnecessarily complex 3. System is easy to use. 4. Need technical person to help use this system. 5. Various functions were well integrated. 6. Inconsistency in this system. 7. Most people would learn this system quickly. 8. System is cumbersome to use. 9. Confident using this system. 10. Needed to learn a lot of things before using this system.	

SUS Scoring

SUS yields a single number representing a composite measure of the overall usability of the system being studied. Note that scores for individual items are not meaningful on their own. To calculate the SUS score, first sum the score contributions from each item. Each item's score contribution will range from 0 to 4. For items 1, 3, 5, 7, and 9 the score contribution is the scale position minus 1. For items 2, 4, 6, 8 and 10, the contribution is 5 minus the scale position. Multiply the sum of the scores by 2.5 to obtain the overall value of SU.

SUS scores have a range of 0 to 100. All numbers are rounded to the highest decimal.

Program Input / Output

Your program will accept series of raw SUS survey data. The first item on each line will be the survey number (shown in bold below in the input and corresponding output), followed by 10 responses for each survey item. Sample input/output is shown below:

Input:

1,5,1,5,1,5,1,5,1,5,1
2,1,5,1,5,1,5,1,5,1,5
3,4,2,5,2,4,1,5,1,4,2

Output:

Survey **1**:100
Survey **2**:0
Survey **3**:85
Average SUS:62

7. Closest Point in 2D Plane (Prof. Isaacs)

When working with wheeled-mobile robots it is common to need to evaluate the straight-line distance between the robot and one of several target points. It may be that the robot needs to choose which target point to visit and one approach is to always visit the closest point. Write a program to calculated the closest point. This program should take as input two objects:

1) The current (x,y) location of the robot.
2) A list of target locations.

and calculates two outputs:

1) The target location that is closest to the current location of the robot.
2) The distance between the closest target location and the current location of the robot.

Note: There may be more than one target location at the minimum distance. In this case please return the first closest point from the input list.

Example:

The Input Text File will contain an x,y pair in each row of the file with the first row corresponding to the robot location and all subsequent rows corresponding to the locations of the target points.

Example Input File:

1 1

0 1

1 0

-1 0

0 -1

Corresponding Inputs:

Robot Location = (1, 1)

Target Points: (0, 1) (1, 0) (-1, 0) (0, -1)

Outputs:

Closest Target Point: (0.0, 1.0)

Distance to Closest Target Point: 1.0

8. Possible Combinations (Prof. Anna)

Write a program that given two integers `n` and `k` as input, outputs all possible different **combinations** of `k` numbers out of the range from `1` to `n`.

Unlike permutations, two combinations are considered to be the same if they contain the same elements, but may be in a different order, for example `[1, 2]` and `[2, 1]` would be considered the same.

The program expects two inputs as integers: the value of `n` followed by the value of `k`.

For example, for the following input:

```
4 2
```

Your program should generate the following output:

```
[[1, 4], [2, 4], [3, 4], [1, 3], [2, 3], [1, 2]]
```

For the input:

```
5 5
```

Your program should generate the following output:

```
[[1, 2, 3, 4, 5]]
```

For the input:

```
5 6
```

Your program should generate the following output:

```
Empty set
```

9. Camarillo Superstore (Prof. Stern)

Overview:

The Camarillo Superstore is reopening after years of sitting abandoned. The new owners have encountered a problem with the storefront they have inherited. All of the inventory management and accounting was done on typewriters and calculators! They need a new and simple way to keep track of what people are buying as well as what the store has in stock. Help get the Camarillo Superstore back on track!

The Input:

The input will consist of three portions:

1. An initial list of items
2. A list of item transactions
3. A Quit command "Q"

1. Initial Item List Input:

The input will begin with a list of every item the store currently sells, its price, and its initial quantity. Each item's respective information is contained on its own line. The list begins with the number of items.

Example Item List Input:

3
Apple 2.50 5
Banana 1.25 7
Carrot 0.50 3

2. List of Item Transactions:

Following the list of sold items, the input will contain a list of item transactions. There are two types of item transactions, with the information of a transaction always contained on its own line.

1. Customer Transactions:

A customer transaction consists of the following information:

- Begins with a "C "
- The item name
- Quantity of that item being purchased.

Errors to Look Out For:

Check that the item in the request is in the inventory otherwise print a line saying:

Error: item not found

Check that the item is in sufficient quantity to satisfy the request, otherwise print

Error: insufficient item quantity

Example Customer Transaction:

C Apple 2

2. Restock Transactions:

A restock transaction consists of the name of the item being restocked and the quantity to be added. It begins with "R ".

Errors to Look Out For:

Check that the item in the restock transaction was listed in the initial item list otherwise print a line saying:

Error: item not found

Example Restock Transaction:

R Apple 3

Example Transaction Section:

C Apple 2
C Banana 3
C Dog 1
R Carrot 4
C Apple 4
R Potato 4

3. Quit Command:

The list of transactions will be followed by a quit command which is simply the letter Q on a line by itself. This will cause the program to output the information detailed in the following section before exiting.

The Output

In addition to the error messages listed in the input section, once all of the transactions have been processed the following information should be printed:

1. Ending Balance Line:

The line indicating the ending cash balance of the store will consist of "Store Balance: ", followed by the amount of money properly formatted in a standard dollar amount.

Example Ending Balance Line:

Store Balance: $40.50

2. Current Inventory Lines:

For each item in the initial inventory list the name of the item and its ending quantity should be printed, separated by a colon and a space.

Example Current Inventory Lines:

Apple: 3
Banana: 0
Carrot: 7

Completed Sample INPUT / OUTPUT:

INPUT	*OUTPUT*
3 *Apple 2.50 5* *Banana 1.25 7* *Carrot 0.50 3* *C Apple 2* *C Banana 3* *C Dog 1* *R Carrot 4* *C Apple 4* *R Potato 4* *Q*	*Error: item not found* *Error: insufficient item quantity* *Error: item not found* *Store Balance: $8.75* *Apple: 3* *Banana: 4* *Carrot: 7*