

# 2019 CS PROGRAMMING GURU COMPETITION

---

**Background:** Welcome to the 2019 CS / IT Programming GURU Competition. In the enclosed packet you will find custom challenges constructed by CS faculty.

Contest documentation can be 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://10.24.110.31:8080>

Contestant Login Info:

UID: SIE-1222-x (where x is the computer machine number 1 through 24)

PWD: Same as UID

Contest Problem Set: <https://compsci.csuci.edu/>

## Problem 1: Floor Plans

*Contributed by AI*

### Description

Problem 1 is a warm-up question to help ease you into the Guru 2019 competition.

William will be moving into Santa Cruz dorms for the summer. He was given the dimensions of a number of different rooms. Each room is rectangular and can be defined by its length and width.

### Requirements

For Problem 1, create a program that accepts the length and width of a dorm room. Length and Width are separate by x. Your program should output a visualized representation of that room encoded in asterisks (i.e.'\*'). More specifically, the output will consist of length-rows that are width-columns wide. Rows are separated by new lines and there are no spaces between columns. There is no new line after the last row.

### Sample Input / Output

#### Input / Output

Input: 3x4 Output: **** **** ****	Input: 1x2 Output: **
--	--------------------------------

## Problem 2: Proportional Election

*Contributed by Prof. AJ Bieszczad*

### Description

In the elections to the House of Representatives of the United States of America, the voters elect a single person representing their electoral district. That type of election is called **plurality voting**. The system used in the United States is a specific type of plurality voting called **winner-takes-all**, or - alternatively - **first-past-the-post (FPTP)**, voting. In such systems, the candidate with the highest number of votes in a district becomes the representative of all the people of the district. Similar systems are used in France, Canada, India, and in United Kingdom and its many former colonies and protectorates.

In contrast, numerous places in the world (e.g., many countries in continental Europe such as Germany, Poland, or Sweden) use the **proportional representation** system in which parties are assigned electoral seats according to their scores in the election. In such systems, each party submits a list of candidates, and the available seats are allocated to the names from the party lists according to the number of votes for a given party. Most of the allocation algorithms use the **highest averages**, or **divisor method** to allocate the seats.

The divisor method uses a number of divisors to create a **quotient matrix** from which a number of largest values equal to the number of the available seats is drawn. For each subsequent largest number, a name from the list of candidates from the corresponding party is assigned as one of the winning electoral seats. There are several methods based on the selection of the divisors; some give preference to larger parties; others, promote the smaller ones.

For example, if there are **five** parties competing for **ten** seats in an electoral district, the divisors can be specified as:

1 2.5 4 5.5 7 8.5 10 11.5 13 14.5

If the following are the results of the election in the district:

Law and Justice: 150000 Civic Platform: 70000 Liberal-Democratic: 40000  
United Peasants: 20000 Green: 10000

then the quotient matrix will be:

	Law and Justice Green	Civic Platform	Liberal-Democratic	United Peasant	
	150000.00*	70000.00*	40000.00*	20000.00*	
10000.00	60000.00*	28000.00*	16000.00	8000.00	
4000.00	37500.00*	17500.00	10000.00	5000.00	
2500.00	27272.73*	12727.27	7272.73	3636.36	
1818.18	21428.57*	10000.00	5714.29	2857.14	
1428.57	17647.06*	8235.29	4705.88	2352.94	
1176.47	15000.00	7000.00	4000.00	2000.00	
1000.00	13043.48	6086.96	3478.26	1739.13	
869.57	11538.46	5384.62	3076.92	1538.46	769.23
10344.83	4827.59	2758.62	1379.31	689.66	

The first row of the matrix is composed of the number of votes for a given party multiplied by the first divisor. In the example, the first divisor is 1, so the numbers in the first row of the matrix represent the number of votes for a given party; i.e., 150000 voters voted for the Law and Justice party, 70000 voters voted for the Civic Platform, 40000 voters voted for the Liberal-Democratic party, and so on. Each column represents a party; the first column is for Law and Justice, the second for Civic Platform, the third for the Civic Platform, and so on.

The second row of the matrix is composed of the values in the first row divided by the second quotient. Each subsequent row is constructed by dividing the first row of the matrix by the corresponding divisor. In our example, the fourth row is composed of number from the first row divided by the fourth divisor (5.5); namely,  $150000/5.5$ ,  $70/5.5$ ,  $40000/5.5$ ,  $20000/5.5$ , and  $10000/5.5$ .

Please note that the numbers in the matrix printout are rounded to two positions after the dot by the print function; internally, the elements of the matrix hold true values that depend on the computer precision.

After creating the matrix, a number of largest matrix elements corresponding to the number of contested seats in the district is selected. In the example, there are ten contested seats, so ten largest numbers in the matrix are found (they are annotated by \* in the printout of the matrix). Each subsequent \*-tagged number in a column representing a party means that the candidate from the party list at the same position as the row number has been elected for one of the contested seats. In the example, the top six candidates from the list submitted by Law and Justice are selected, the top two from Civic Platform, and the first candidates from the lists of the Liberal-Democratic party and the United Peasant party. None of the candidates from the list of the Green party is elected in the example.

Therefore, the allocation of the seats in the example is as follows:

Law and Justice won 6 seats Civic Platform won 2 seats Liberal-Democratic won 1 seat United Peasants won 1 seat

## Task

Your task is to implement the proportional voting algorithm. The program should accept the following input:

- the first line will contain two space-separated integers representing the number of parties and the number of contested seats,
- the second line will have a number (equal to the number of contested seats) of space-separated floats that represent the divisors, and
- the subsequent number of lines equal to the number of participating parties will provide the election results by stating the name of the party and the number of votes for the party separated by a colon.

The output of the program should list the parties and the corresponding allocation of the won seats. The parties with no seats won should not be listed.

## Sample input

```
5 10 1 2.5 4 5.5 7 8.5 10 11.5 13 14.5 Law and Justice: 150000 Civic  
Platform: 70000 Liberal-Democratic: 40000 United Peasant: 20000  
Green: 10000
```

## Corresponding sample output

```
The voters have allocated the 10 contested seats as follows: Law and Justice  
won 6 seats Civic Platform won 2 seats Liberal-Democratic won 1 seat  
United Peasants won 1 seat
```

## Problem 3: Sliding Windows

*Contributed by Prof. Anna Bieszczad*

### Description

You are given a **matrix** of integers of **size**  $N \times N$  and an integer **K** representing a **window size**. Your task is to find all local maxima in the matrix within the sliding window of size  $K \times K$ . A local maximum is the largest integer in a given position of the window.

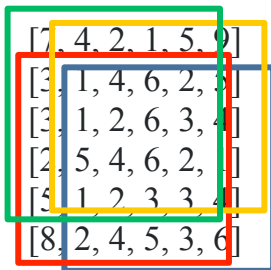
For example, let's consider the following  $6 \times 6$  matrix:

```
[7, 4, 2, 1, 5, 9]
[3, 1, 4, 6, 2, 5]
[3, 1, 2, 6, 3, 4]
[2, 5, 4, 6, 2, 1]
[5, 1, 2, 3, 3, 4]
[8, 2, 4, 5, 3, 6]
```

If **K** was set to **6** it would mean that there is only **one possible position for the sliding window** which covers the whole matrix, so the expected result would be:

```
[9]
```

If **K** was set to **5** it would mean that there are **four possible positions for the sliding window**:



```
[7, 4, 2, 1, 5, 9]
[3, 1, 4, 6, 2, 5]
[3, 1, 2, 6, 3, 4]
[2, 5, 4, 6, 2, 1]
[5, 1, 2, 3, 3, 4]
[8, 2, 4, 5, 3, 6]
```

and the expected result would be:

```
[7, 9]
[8, 6]
```

being a matrix of the local maximum for each position of the sliding window.

The following shows expected results for the remaining possible values of K for this matrix:

- for **K= 2** (which represents 2x2 sliding window), the expected result is:

```
[7, 4, 6, 6, 9]
[3, 4, 6, 6, 5]
[5, 5, 6, 6, 4]
[5, 5, 6, 6, 4]
[8, 4, 5, 5, 6]
```

- for **K = 3** (which represents 3x3 sliding window), the expected result is:

```
[7, 6, 6, 9]
[5, 6, 6, 6]
[5, 6, 6, 6]
[8, 6, 6, 6]
```

- for **K = 4** (which represents 4x4 sliding window), the expected result is:

```
[7, 6, 9]
[6, 6, 6]
[8, 6, 6]
```

## INPUT/OUTPUT

The program should get the following input:

**size of the window**, followed by **size of the matrix**, followed by **elements of the matrix** separated by spaces.

**Sample input:**

```
4 6
7 4 2 1 5 9
3 1 4 6 2 5
3 1 2 6 3 4
2 5 4 6 2 1
5 1 2 3 3 4
8 2 4 5 3 6
```

**Sample output:**

[7, 6, 9]

[6, 6, 6]

[8, 6, 6]

Note: your program should ignore all whitespace (such as spaces, tabs, end of line characters, etc.).



## Problem 4: Ultra Radish

*Contributed by Prof. Nick Stern*

Billy Bob has been tasked with being the finding of the Ultra Radish. For centuries the treasure instructions have been passed down, but now must be followed so as to summon the Ultra Radish.

Billy Bob is standing on the origin of x axis. He has a list of instructions. Your task is to output its final position after executing all the instructions.

- LEFT: move one unit left (decrease p by 1, where p is the position of the robot before moving)
- RIGHT: move one unit right (increase p by 1)
- SAME AS i: perform the same action as in the i-th instruction. It is guaranteed that i is a positive integer not greater than the number of instructions before this.

### Input

The first line contains the number of instruction sets T ( $T \leq 100$ ). Each instruction set begins with an integer n ( $1 \leq n \leq 100$ ), the number of instructions. Each of the following n lines contains an instruction.

### Output

Print the final position of the Ultra Radish. Note that after processing each test case, the coordinates should not be reset.

### Sample Input

2

3

LEFT

RIGHT

SAME AS 2

5

LEFT

SAME AS 1

SAME AS 2

SAME AS 1

SAME AS 4

## **Sample Output**

-4

## Problem 5: Who's Turn Is It Anyway?

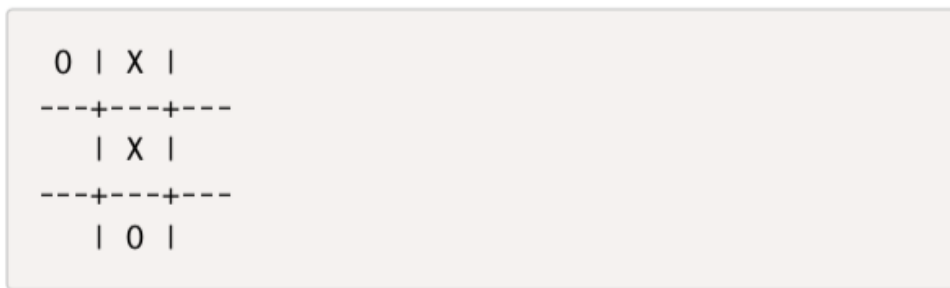
*Contributed by Prof. Kevin Scrivnor*

### Background

You're playing **Tic Tac Toe** (American for Noughts and Crosses) with a distracted but oddly intelligent child. She turns around and says, "who's turn is it?"

Having stopped paying attention 8 minutes ago you have a mild panic and your brain starts working overtime, performing the necessary calculus to exclaim, "it's your turn...still."

Let us examine your thinking. Luckily you recall that *X always goes first*.

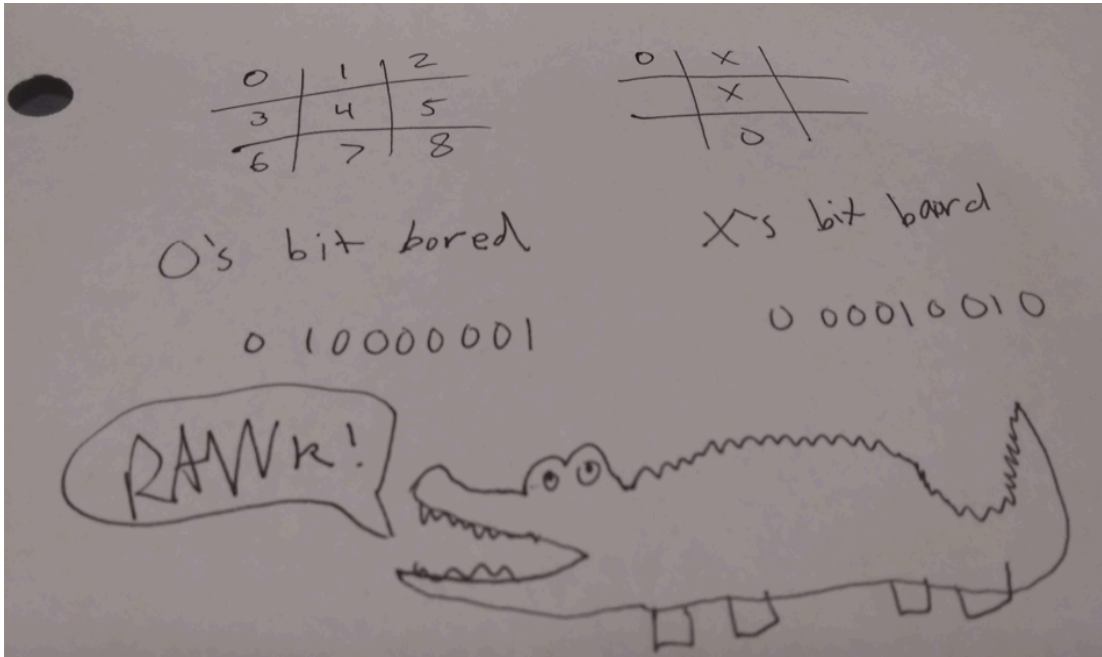


If X went first, and there are four squares taken, then clearly it must be turn 5. X always plays on odd turns therefore it is X's turn. QED.

### A Futile Game

After some arbitrarily large number of draws, your distracted but oddly intelligent child decides, correctly, that this is a futile game. She now wants to play on a playing field that is up to 8x8.

She explains to you that each players moves can be represented by their own bit boards. She crudely draws a quick example:



How it works:

From left to right and top to bottom is labeled  $0-n$ . If O has played at  $k$ , then bit  $k$  is 1, otherwise 0. Consider bit 0 the least significant binary digit and bit  $n$  the most significant binary digit.

### So, Who's Turn Is It Anyway?

Boards are  $n \times n$ , with  $0 \leq n \leq 8$ . First line of input contains  $n$ . Second line is X's bit board as a positive whole number in the format:  $[1-9][0-9]\{0,18\}$ . Third line is O's bit board as a positive whole number in the format:  $[1-9][0-9]\{0,18\}$ . Only valid input will be provided, input error checking no necessary. Output the character X if it is X's turn or O if it is O's turn.

Sample Input	Sample Output
3	X
18	
129	

## **Problem 6: Kaprekar's Constant**

*Contributed by Prof. Jason Isaacs*

### **Description**

Write a program that accepts an integer as the input. This input should meet the following conditions:

The number of digits in the input should equal 4. At least two of the digits must be unique.

Feed this validated input (call it x) to a recursive functions that performs the following operations:

1. Create a new integer by arranging the digits of x in descending order. Print the result of this step as in the Sample Output.
2. Create a new integer by arranging the digits of x in ascending order. Print the result of this step as in the Sample Output.
3. Subtract the integer created in step 3 from the integer created in step 2. Print the result of this step as in the Sample Output.
4. If the result of step 4 is equal to the input to the function (the result from step 3) then return the result and terminate the program. Else return the result and repeat.
5. Upon termination print the result.

### **Sample Input:**

5469

### **Sample Output:**

Step 1: 9654  
Step 2: 4569  
Step 3: 5085  
Step 1: 8550  
Step 2: 558

Step 3: 7992  
Step 1: 9972  
Step 2: 2799  
Step 3: 7173  
Step 1: 7731  
Step 2: 1377  
Step 3: 6354  
Step 1: 6543  
Step 2: 3456  
Step 3: 3087  
Step 1: 8730  
Step 2: 378  
Step 3: 8352  
Step 1: 8532  
Step 2: 2358  
Step 3: 6174  
Step 1: 7641  
Step 2: 1467  
Step 3: 6174  
6174

## Problem 7: It's never too early... Or too late...

Contributed by Prof. Thoms

### Background

A pension is a fund into which a sum of money is added during an employee's employment years, and from which payments are drawn to support the person's retirement from work in the form of periodic payments. Pension formulas consider multiple factors but primarily age, years of service, final salary at retirement in addition to some multiplier.

### Requirements

Your program will allow users to check their current yearly pension based on an employees total number of years of service multiplied by their final salary at retirement multiplied by the sum of assigned multipliers. Multipliers are based on the tables below.

<i>Years of Service Multipliers</i>		<i>Age Multipliers</i>	
Years of Service	Multiplier	Age at Retirement	Multiplier
< 5	0	< 40	0
5 to 10	0.5%	40-50	0.20%
10-15	1.25%	50-60	0.40%
15-20	1.75%	60-65	0.80%
20+	2.25%	65+	1%

### Input / Output

Data is passed in using comma-separated fields. The first field is the employee's age at retirement, followed by total years of service, followed by the employee's final salary. (Note, the text 'Example input x' and 'Example output x' are not a part of i/o operations. Also note that there is a space between the semicolon and result.)

Example input 1: 50,19,100000 Example output 1: Yearly Retirement Salary: 40850	Example input 2: 39,6,100000 Example output 2: Yearly Retirement Salary: 3000
Example input 3: 62,15,120000 Example output 3: Yearly Retirement Salary: 45900	Example input 4: 39,3,200000 Example output 4: Yearly Retirement Salary: 0
All values rounded to the nearest whole number.	

## Problem 8: Am I there yet...

*Contributed by Prof. Thoms*

### Background

Building on Problem 8, pension calculators often provide employees with a method for projecting target retirement dates.

### Requirements

This program should calculate the number of years of service remaining until an employee's retirement. In addition to the formula in P8, the program assumes two new factors. The first is that the standard rate of salary inflation remains consistent at an annual rate of 3%. The second is that an employee will define their desired retirement salary, which is read in as a percentage of the final salary.

### Input / Output

Data is passed in using comma-separated fields. The first field is the employee's age at retirement, followed by total years of service, followed by the employee's final salary, followed by their desired retirement salary as a percentage of their overall salary. (Note, the text 'Example input x' and 'Example output x' are not a part of i/o operations. Also note that there is a space between the semicolon and result.)

Example input 1: 50,19,100000,0.75 Example output 1: Service Years Left: 6	Example input 2: 39,6,100000,0.85 Example output 2: Service Years Left: 15
Example input 3: 69,2,150000,0.40 Example output 3: Service Years Left: 11	Example input 4: 39,3,200000,0.75 Example output 4: Service Years Left: 17
All values rounded to the nearest whole number.	