Exercises: Intro to Java

This document defines the exercises for "Java Advanced" course @ Software University. Please submit your solutions (source code) of all below described problems in Judge.

Problem 1. Rectangle Area

Write a program that reads the **sides of a rectangle** (two integers **a** and **b**), calculates and prints the rectangle's **area**. Format the result to the **second digit** after the decimal separator.

Examples

Input	Output
7 20	140.00
5 12	60.00

Problem 2. Triangle Area

Write a program that reads **3 points** in the plane (with integer **x** and **y** as coordinates), calculates and prints the **area of the triangle** composed by these 3 points. Round the result to a whole number. In case the three points do not form a triangle, print "**0**" as result.

Examples

Input	Output
-5 10 25 30 60 15	575

Input	Output
53 18 56 23 24 27	86

Input	Output
1 1 2 2 3 3	0

This resource could help you: http://www.mathopenref.com/coordtrianglearea.html.

Problem 3. Formatting Numbers

Write a program that reads **3 numbers**:

- an integer **a** (0 ≤ **a** ≤ 500)
- a floating-point **b**
- a floating-point **c**

Print them in 4 virtual columns on the console, separated with a pipe '**|**'. Each column should have a width of 10 characters.

- First, the number **a** should be printed in **hexadecimal, left aligned**
- Second, the number **a** should be printed in binary form, padded with zeroes
- Third, the number **b** should be **printed with 2 digits after the decimal point, right aligned**
- Lastly, the number **c** should be **printed with 3 digits after the decimal point, left aligned**

You will receive **a**, **b** and **c** on a **single line**, separated by **one or more white spaces**.

Examples

Input	Output
254 11.6 0.5	|FE |0011111110| 11.60|0.500 |
499 -0.5559 10000	|1F3 |0111110011| -0.56|10000.000 |

Problem 4. Calculate Expression

Write a program that reads three floating point numbers **a**, **b**, and **c** from the console and calculates the following expressions:

$$f1 = ((a^2 + b^2) / (a^2 - b^2))^{(a + b + c) / \sqrt{c}} \qquad f2 = (a^2 + b^2 - c^3)^{(a - b)}$$

Then the program calculates the **absolute value** of the **difference between** the average of the three numbers and the average of the two expressions.

$$Abs\ (Avg\ (a, b, c) - Avg\ (f1, f2))$$

You will receive **a**, **b** and **c** on a **single line**, separated by **one or more white spaces**.

Examples

Input	Output
5 2 3	F1 result: 6.45; F2 result: 8.00; Diff: 3.89
3.8 2.5 1.2	F1 result: 569.60; F2 result: 45.84; Diff: 305.22

Problem 5. *Odd and Even Pairs

You are given an **array of integers** as a single line, separated by a **space**. Write a program that checks consecutive pairs and prints if **both are odd/even or not**.

Note that the array length should also be an **even** number.

Examples

Input	Output
1 2 3 4	1, 2 -> different 3, 4 -> different
2 8 11 15 3 2	2, 8 -> both are even 11, 15 -> both are odd 3, 2 -> different
1 8 11 1 2	invalid length

Problem 6. *Hit the Target

Write a program that takes as input an **integer** – the **target** – and outputs to the console **all pairs of numbers** between **1** and **20**, which, if **added** or **subtracted**, **result in the target**.

Examples

Target	Output
5	1 + 4 = 5 2 + 3 = 5 3 + 2 = 5 … 19 - 14 = 5 20 - 15 = 5
35	15 + 20 = 35 16 + 19 = 35 17 + 18 = 35 18 + 17 = 35 19 + 16 = 35 20 + 15 = 35
0	1 - 1 = 0 2 - 2 = 0 … 19 - 19 = 0 20 - 20 = 0

Problem 7. Character Multiplier

Create a program that takes **two strings** as arguments and returns the **sum** of their **character codes multiplied** in pairs. (multiply str1.charAt (0) with str2.charAt (0) and add to the total sum, then continue with the next two characters). If one of the strings is longer than the other, **add** the remaining character codes to the total sum **without** multiplication.

Examples

Input	Output
Gosho Pesho	53253
123 522	7647
a aaaa	9700

Problem 8. Get First Odd or Even Elements

Write a program that returns the first **N** odd/even elements from a collection. Return as many as you can.

Format of the input: **Get {number of elements} {odd/even}**

Examples

Input	Output
1 2 3 4 5 Get 3 odd	1 3 5
11 6 2 8 1 0 Get 8 even	6 2 8 0

Problem 9. Byte Party

You will be given an integer number **N** and on each of the next N lines - a **positive 8-bit integer**. On the next lines you will be given a series of commands, one of the following:

- **"-1 [position]"** – Upon receiving this command you should **flip** the bits at the specified **position** in **all** numbers you received. Flipping a bit means turning its value **from 1 to 0 or the other way around**.
- **"0 [position]"** – upon receiving this command you should **unset** the bits at the specified **position** for **all** numbers, i.e. turn all bits to **0** regardless of their current value.
- **"1 [position]"** – upon receiving this command you should **set** the bits at the specified **position** for **all** numbers, i.e. turn all bits to **1** regardless of their current value.
- **"party over"** – when you receive this command **print** back the numbers after all changes have been made; each number stays on a separate line.

Input

- The input data should be read from the console.
- The first input line holds **the number N – the count of integers** you'll receive.
- On each of the next **N lines** you'll receive a positive 8-bit integer number. Input ends with **the string "party over".**
- The input data will always be valid and in the format described. There is no need to check it explicitly.

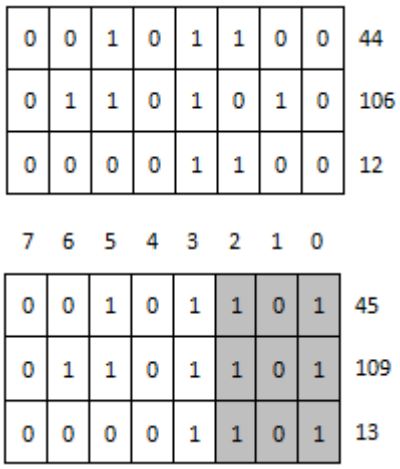
Output

- You should **print N lines, each containing a number** – the numbers **after all manipulations**.

Constraints

- All **input numbers** are in the range [0 … 255].
- **[position]** will be between [0 … 7].
- Allowed working time for your program: 0.1 seconds. Allowed memory: 16 MB.

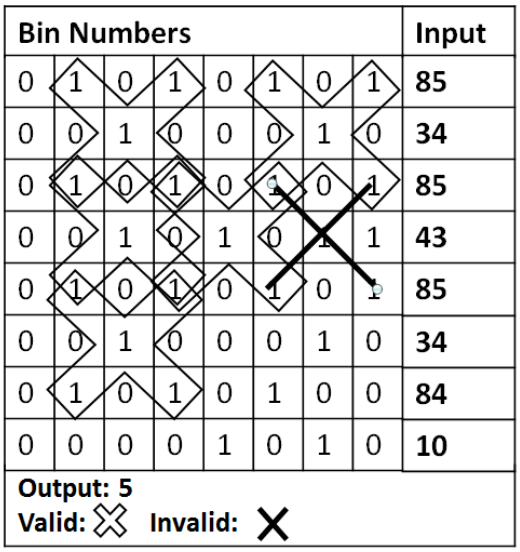
Examples

Input	Output	Comments
3 44 106 12 -1 0 0 1 1 2 party over	45 109 13	

Problem 10. X-Bits

You are given **8 positive 32-bit integer numbers**. Write a program to **count** all **X-bits**.

X-bits are groups of **9** bits (3 rows x 3 columns) forming the letter "**X**". Your task is to count all **X-bits** and print their count on the console. **Valid** X-bits consist of 3 numbers where their corresponding bit indexes are **exactly {"101", "010", "101"}.** All other combinations like: **{"111", "010", "101"}** or **{"111", "111", "111"}** are **invalid.** All valid X-bits can be part of **multiple** X-bits (with **overlapping**). Check the example **on the right** to understand your task better.



Input

The input data should be read from the console.

- On the first **8 lines**, you will be given **8 32-bit positive** integers.

The input data will always be valid and in the format described. There is no need to check it explicitly.

Output

The output should be printed on the console. It should consist of exactly **1** line:

- At the **first line** print the count of the **X-bits**.

Constraints

- The 8 input **integers** will be in the range [0 … 2 147 483 647].
- Allowed working time: 0.2 seconds. Allowed memory: 16 MB.

Examples

Input	Output	Comments
160 64 170 4 90 167 82 165	4	1 0 1 0 0 0 0 0 0 1 0 0 0 0 0 0 1 0 1 0 1 0 1 0 0 0 0 0 0 1 0 0 0 1 0 1 1 0 1 0 1 0 1 0 0 1 1 1 0 1 0 1 0 0 1 0 1 0 1 0 0 1 0 1

Input	Output
7583 1374 12345 8888 91834 1234 1852 24912	0

Input	Output
365 146 365 365 658 365 640 160	7

Problem 11. Game of Names

Write a program to **calculate points for all players** and **find who the winner is**. You will be given **the count of the players**, **their names** and **initial scores**. Score for every player depends on their name. To the ***player*** **score add** or **subtract** the **ASCII code of each letter**. If **ASCII code** is **even, add** it to the score. If is **odd** – **subtract** it from the score.

Find **the one with highest score** and print his name and score on the console. If **two or more players** are with **same points** – **the winner is the first one**.

Input

On the **first input line,** you will be given **number N - the count of players**.

On the **next 2*N lines** you will be given player name and his initial score.

Output

The output should be printed on the console and consists of **the name of the winner** and **his score** in the following format:

"`The winner is {name} - {points} points`**"**

Constraints

- **N – the count of players** will be a **positive integer** in the range **[1...100]**
- **Names** will be **strings** with **length between 3 and 30**
- **The score for each player** will be an **integer** in the range **[-100,000...100,000]**

Examples

Input	Output	Comments
3 Bojidar 123 Preslav 123 Pesho 123	The winner is Preslav - 230 points	B(66)o(111)j(106)i(105)d(100)a(97)r(114) Initial points 123 scores 123 +66 -111 +106 -105 +100 -97 +114 = 196 P(80)r(114)e(101)s(115)l(108)a(97)v(118) Initial points 123 scores 123 +80 +114 -101 -115 +108 -97 +118 = 230 P(80)e(101)s(115)h(104)o(111) Initial points 123 scores 123 +80 -101 -115 +104 -111 = -20 Preslav(230) > Bojidar(196) > Pesho(-20)

Problem 12. *Vehicle Park

You are manager on a vehicle park. Your job is to sell cars and give reports to the accounting. You will be given all vehicles that are available for selling in **format** like the example below:

c2 c4 v10 v20 b50

Each car is described by **vehicle type** (single character '**b**', '**c**' or '**v**') and **number of seats** in the vehicle (natural number).

For example, **"c4"** means **car with 4 seats**, "**b50**" means **bus with 50 seats** and "**v10**" means **van with 10 seats**.

Then you need to **process** a sequence of **incoming requests**. Each request holds **type of vehicle** and **number of seats** in the following format:

Car with 4 seats
Bus with 20 seats
…

If you have **vehicle** that **matches** the description of the **desired vehicle**, you should **sell it**, otherwise print "**No**". The **price** is calculated as a **product** of the **character ASCII code** and the **number of seats**. For example, the **price** for "**c4**" (**car with 4 seats**), will be calculated as **99('c') * 4 = 396**. If there are **2 or more matching vehicles** you should **sell the leftmost** one.

After you **run out of customers**, you need to **print the vehicles** that you **didn't sell** and the **count of sold vehicles**.

Input

The input data should be read from the console.

- On the **first input line,** you will receive **all vehicles** in the park, separated with **single whitespace**.
- On the next lines, you will receive **requests for vehicles** in the following format:
  "`{Vehicle Type} with {Number of seats} seats`"
  until you receive **"`End of customers!`"**

The input data will always be **valid** and in the format described. There is no need to check it explicitly.

Output

The output should consist of:

- For **each vehicle request** you either need to **print**:
  - "`Yes, sold for {price}$`" – if the **wanted vehicle** is **available** in the park.
  - "No" – if there is no such vehicle in the vehicle park.
- After you **stop receiving request**, you need to **print two lines**:
  - On the first line, you need to print the remaining vehicles in the format:
    "Vehicles left: `x1, x2, x3…`"
  - On the second line, you need to print the total number of vehicles sold in the following format:
    "`Vehicles sold: x1, x2, x3…`"

Constraints

- The **number of vehicles** will be in range [**0 – 10,000**].
- The **amount of request** for vehicles will be in range [**0 – 10,000**].
- The **number of seats** for **each vehicle** will be in range [**1 – 10,000**].
- The **vehicle type** can only be one of the following **Car – c**; **Bus – b**; **Van – v;**
- Allowed working time for your program: 0.1 seconds. Allowed memory: 16 MB.

Examples

Input	Output	Comments
c2 c4 v10 v20 b50 Car with 4 seats Bus with 20 seats Bus with 33 seats Van with 20 seats Bus with 50 seats End of customers!	Yes, sold for 396$ No No Yes, sold for 2360$ Yes, sold for 4900$ Vehicles left: c2, v10 Vehicles sold: 3	c4 -> 99('c') * 4 = 396$ v20 -> 118('v') * 20 = 2360$ b50 -> 98('b') * 50 = 4900$

Input	Output
c2 v1 b2 v2 c20 b150 v1 Van with 50 seats Van with 1 seats Bus with 1000 seats End of customers!	No Yes, sold for 118$ No Vehicles left: c2, b2, v2, c20, b150, v1 Vehicles sold: 1

Problem 13. **Blur Filter

Bojo is a bad photo editor, but he wants to do some amazing pictures for his Facebook page. He can't do it alone, so he needs your help. For each picture, you will be given a **matrix** with pixels. Each pixel has **weight**. The **blur filter** is applied to a certain cell (pixel) and **all cells around** it. **The blur** has **amount**, which needs to be **added to the weight of the pixel** that it blurs. Print the matrix after the blur applied as output.

Example: on the picture on the left **apply blur** with amount **2** over the **pixel** at position **[2, 2]**.

1	2	3	4
5	6	7	8
9	10	11	-12
-13	14	15	16

→

1	2	3	4
5	8	9	10
9	12	13	-10
-13	16	17	18

Input

The input data should be read from the console.

- The **first line** holds the **blur amount**.
- The **second line** holds the number of rows **r** and columns **c** separated by a space.
- The **next r lines** hold the matrix numbers. Each line holds **c** integers, separated by space.
- The **last line** holds the **coordinates of the blur** – **row** and **column**, separated by space.

The input data will always be valid and in the format described. There is no need to check it explicitly.

Output

The output should consist of the matrix after the blur filter is applied.

Constraints

- The **blur amount** will be an integer number in the range [-2,147,483,648...2,147,483,647].
- The **pixel weight** will be an integer number in range [-2,147,483,648...2,147,483,647].
- The number of **rows** and **columns** will be an integer number in the range [1...20].

Examples

Input	Output	Comments
9 3 3 1 1 1 1 1 1 1 1 1 1 1	10 10 10 10 10 10 10 10 10	**Blur amount = 9** Target = **[1, 1]** **[0, 0] = 1+9; [0, 1] = 1+9; [0, 2] = 1+9;** **[1, 0] = 1+9; [1, 1] = 1+9; [1, 2] = 1+9;**

		[2, 0] = 1+9; [2, 1] = 1+9; [2, 2] = 1+9;

Input	Output
3 3 4 0 -5 4 20 0 20 4 -5 20 4 -5 0 1 2	0 -2 7 23 0 23 7 -2 20 7 -2 3