Telerik Software Academy 2012/2013 – C# Part 2 – Practical Exam – Variant 3

Exam terms – Read carefully!

During the exam you **are allowed** to use any teaching materials, lectures, books, existing source code, and other paper or Internet resources.
Direct or indirect communication with anyone in the class or outside the room is absolutely **forbidden**.
The address of the judge system is: http://bgcoder.com. Trainers will give you the contest password.
Only your **last** code submission will be evaluated. You have exactly **6 hours** to solve all given problems.

Problem 1 – 9Gag Numbers

In 9Gag we like fun. We have a lot of time and we play with fun pictures and fun stories. Recently we invented a funny way to express numbers. We use the following 9 digits:

0	-!
1	**
2	!!!

3	&&
4	&-
5	!-

6	*!!!
7	&*!
8	!!**!-

We write the numbers as sequences digits from our 9 available digits given above. The last digit of the number (the most right one) has a value as shown in the above table. The next digit on the left has a value 9 times bigger than the shown in the above table, the next digit on the left has 9*9 times bigger value than the shown in the table and so on. Your task is to write a program to **convert a 9Gag-style number into its corresponding decimal representation**.

Input

The input data consists of a single string – a 9Gag-style number. The input data will always be valid and in the described format. There is no need to check it explicitly.

Output

The output data consists of a single line holding the calculated decimal representation of the given 9Gag-style number and should be printed at the console.

Constraints

- The input number will have between 1 and 20 digits.
- Allowed working time for your program: 0.1 seconds. Allowed memory: 16 MB.

Examples

Input	Output
*!!!	6

Input	Output
***!!!	15

Input	Output
!!!**!-	176

Input	Output
!!**!--!!-	653

Problem 2 – Special Value

You are given **N** number of rows. Each row contains a different number of columns. Each cell in a row contains a negative number or an index of a cell on the next row, i.e. rows[row][column] holds an index of cell in [row+1]. **The row after the last row is the first row**. If the number of columns on the next row is **C**, each of the cells on the current row are in the range **[-C, C-1] inclusive**.

A **path in the rows** is the max number of cells that can be passed, starting from any of the cells on the first row, and following the pattern above (i.e. each cell on row **R**, holds a negative number or an index on row **R+1**). The path ends when the path reaches an already visited cell in this path, or the number in the cell is negative.

A **special value** is the sum of the path + the absolute value of a negative number reached. If a path reaches a visited cell, this path cannot get a special value.

Your task is to find the biggest special value using the given rows.

Input

The input data should be read from the console.

On the first line you will be given the number **N**.

On the **next N** lines you will be given **the numbers in each row**, separated with ", " (comma and space).

Output

The output data should be printed on the console.

The output should contain **only** the **maximal special number**.

Constraints

- **The rows** will be between **1 and 1000** inclusive.
- **The cells on each row** will be between **1 and 1000** inclusive.
- Allowed working time for your program: 0.5 seconds. Allowed memory: 16 MB.

Examples

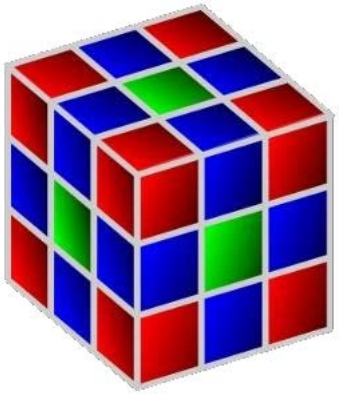
Input example	Output example
4 -1, 0, **-3**, -2, 0, -2 -1, 3, 1, 0, 2, 0 -9, 1, 1, -7 1, -5, -3, -1, 3, -2, 2, 1, 1	**4** (starting from column 2) //special value = 1 + |-3|
6 5, -4, **8**, -5, 0 1, -2, -1, 1, 0, -1, -1, -2, **1** 3, **-5** 4, -9, -4, 4, 0, 7 1, -2, -8, 4, -8, 7, -5, -4, -4 4, -1, 0, -3, 2, 4, -4, 1	**8** (starting from column 2) // special value = 3 + |-5|

2
0, -3, **0**, **3**
-3, 3, 0, **2**, 0
	7 (starting from column 3)
	//special value = 4 + |-3|
	//the path is [0][3] -> [1,3] -> [0,2] -> [1,0]

2
0, **-3**, 0, 3
0, 3, 0, 2, 0
	4 (starting from column 1)
	//the only path is from column 1
	//the other paths don't reach //negative number,

Problem 3 – Kukata is Dancing

"The Hook" (also known as Kukata) is a very famous Sofia Gangster. He is wanted for various crimes like drugs distribution, counterfeiting of credit cards and also breaking a bottle in a guy's neck while screaming "Eiiii murshiiiiiii". But the real passion of Kukata is the dancing, especially on top of cubes. And today he is dancing on a very special cube. It is of size 3x3x3 and there are nine squares on each of its six sides. Each side is colored according to the following rules:



- The four squares at the corners are red.
- The center square on each side is green.
- All of the remaining squares are blue.

Each dance of Kukata always begins from the green square on the top side of the cube and he is looking at one of the blue squares. Kukata knows only three dance moves, but he can do them flawlessly:

- 'L' – He stays in the current square and turns 90 degrees to the left.
- 'R' – He stays in the current square and turns 90 degrees to the right.
- 'W' – He walks on the next square in the current direction.

Please keep in mind that Kukata is a great dancer and he can cross an edge of the cube into another side. If that happens, the cube automatically rotates to keep him on top. You will receive a sequence of all movements made by Kukata. You should evaluate them and return the color of the last square that Kukata is standing on.

Input

The input data should be read from the console. On first input line you will receive an integer number **N**, showing the number of performed dances. On each of the next **N** lines you will receive a string containing only the letters '**L**', '**R**' and '**W**' which will refer to the sequence of moves. The input data will be correct and there is no need to check it explicitly.

Output

The output data should be printed on the console. On the **N** output lines you should print one of the words "**RED**", "**GREEN**", "**BLUE**" (without the quotes), showing the color of the last square of the dance.

Constraints

- **N** will be between **5** and **10** inclusive.
- The input strings will contain between **1** and **50** characters, inclusive.
- The input strings will contain only the letters '**L**', '**R**' and '**W**'.
- The answer will **not depend** on the **initial direction** of Kukata.
- Allowed working time for your program: 0.1 seconds. Allowed memory: 16 MB.

Examples

Example input	Example output
5 LLRR WWWWWWWWWWWWW WLWRW WWL LWRL	GREEN GREEN RED BLUE BLUE

Example input	Example output
5 WWRLLW RWLW WWL W LWWW	RED RED BLUE BLUE GREEN

Problem 4 – Fake Text Markup Language

Joro's trying to top the leaderboard on "weirdest things by the lectors" (yep, couldn't think of a better translation). Last time he used Paint for a color picker on a CSS lecture, now he's inventing a whole new markup language – the Fake Text Markup Language. It looks a bit like HTML – it has opening and closing tags, and those tags define the formatting of text… on the console! So, if we receive an FTML formatted text, we need to print the text content according to the following rules:

- Tags can be opening and closing, e.g.: *<upper> and </upper>*
- Tags can be nested, e.g.: *<upper> this is a <lower> nested tag</lower> in another tag </upper>*
- Nesting is measured in "**depth**", in this case <lower> is **deeper** than <upper>
- Tags **cannot** "intersect", e.g.: *<upper> this is <lower> very </upper> wrong </lower>* is **not valid**
- Tags don't appear in the resulting text, but they affect it
- When tags are nested, the effect of the "**deepest"** is considered first, then the effect of the less deeper and so on
  - e.g.: *<upper>Before nested <lower>Inside nested</lower> After nested</upper>* will apply the effects of <lower> and then of <upper>
  - The end result will be: *BEFORE NESTED INSIDE NESTED AFTER NESTED*
- The <upper> tag converts text to its uppercase variant
  - e.g.: *<upper>tExT</upper>* results in: *TEXT*
- The <lower> tag converts text to its lowercase variant
  - e.g.: *<lower>tExt</lower>* results in: *text*
- The <toggle> tag:
  - if a character is uppercase, it converts it to lowercase
  - if a character is lowercase, it converts it to uppercase
  - e.g. *<toggle>tExT<toggle>* results in: *TeXt*
- <upper>, <lower> and <toggle> tags don't affect punctuation or whitespaces
- The <del> tag deletes all text in it
  - e.g.: *this is <del> this is deleted </del> some text* results in: *this is some text*
- The <rev> tag reverses all text in it
  - e.g.: *123 reversed is <rev>123</rev>* results in: *123 reversed is 321*
- The FTML keeps all whitespaces and new lines (it doesn't remove them like HTML)
  - the only exception is, obviously, the <del> tag

Write a program, which formats code, according to the rules described.

Input

On the first line of the input, there will be the number N – the number of lines in the FTML text.

On each of the next lines there will be a line of the FTML text, consisting of English letters, punctuation and whitespaces (and of course FTML tags)

Output

On the console, you should print the lines of the formatted text.

Constraints

- 0 < N < 500
- The text input will NOT contain any '<' or '>', other than the ones for the tags
- Allowed working time for your program: 0.1 seconds.
- Allowed memory: 16 MB.

Examples

Input example	Output example
2 So<rev><upper>saw</upper> txet em</rev> <lower><upper>here</upper></lower>	Some text WAS here
3 <toggle><rev>ERa</rev></toggle> you <rev>noc</rev><lower>FUSED</lower> <rev>?<rev>already </rev></rev>	Are you confused already ?

Problem 5 – Three in One

Solving one task is too mainstream. Solving two task is also too mainstream.
You are given three tasks. Solve them all.

First task

A round of Blackjack has ended and it's time to reveal the winner. Use the following two rules to determine the outcome of the round:

- If a player has more than **21** points, he loses.
- Of all the players who have **21** or fewer points, the ones with the maximum number of points are the winners.

You are given a **zero-based** list of integers called **points**, where **points[i]** is the number of points received by the i-th player.

If there is exactly one winner, your program should print the 0-based index of the winning player. Otherwise, it should print **-1**.

Second task

You have a birthday party with your friends! On the table are a variety of cakes of different sizes, with size being measured in "bites". You've decided to take turns picking cakes until all of the cakes are picked by either you or one of your friends. You all like big cakes, so each person on their turn will pick the **biggest remaining cake**. If there's a tie for the biggest cake, it doesn't matter which of the tied cakes is picked. Since it's your birthday, you get to pick the first cake.

For example, suppose you have 2 friends and there are 5 cakes with sizes of 6, 7, 9, 6, and 4. You will pick the 9 bite cake, your first friend will pick the 7 bite cake, your other friend will get one of the 6 bite cakes, you will get the other 6 cake, and your first friend will get the 4. You will get a total of 15 cake bites.

F will be the number of friends you have with you (not including you). **sizes** will be a list of integers, each of which is the size of one cake - not necessarily in order of size.

Your program should find the **total bites of cake you will get**.

Third task

Your character in an RPG game has **G1** gold, **S1** silver and **B1** bronze coins. You need **G2** gold, **S2** silver and **B2** bronze coins to buy a new beer. A bank in the game supports four types of exchange operations:

- the bank will give you **9 silver** coins in exchange for **1 gold** coin
- the bank will give you **1 gold** coin in exchange for **11 silver** coins
- the bank will give you **9 bronze** coins in exchange for **1 silver** coin
- the bank will give you **1 silver** coin in exchange for **11 bronze** coins

Write a program to find the **minimal number of exchange operations** required for your character to hold at least **G2** gold, at least **S2** silver and at least **B2** bronze coins.

Print **-1** if your character doesn't have enough money for beer.

Input

The input data should be read from the console.

The elements of the **points** list are given on the first input line, separated by a single comma.

The elements of the **sizes** list will be given on the second input line, separated by a single comma. The number **F** will be given on the third input line.

Numbers **G1**, **S1**, **B1**, **G2**, **S2** and **B2** will be given on the fourth input line separated by a single space.

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

Output

The output data should be printed on the console.

On the first output line print the solution of the first task (*the index of the Blackjack winner*).

On the second output line print the solution of the second task (*the total bites of cake you will get*).

On the third output line print the solution of the third task (*minimal number of exchange operations*).

Constraints

- **points** will contain between 1 and 50 elements, inclusive. Each element of **points** will be between 1 and 35, inclusive.
- **sizes** will contain between 1 and 10 elements, inclusive. Each element of **sizes** will be between 1 and 10, inclusive. **F** will be between 1 and 5, inclusive.
- **G1**, **S1**, **B1**, **G2**, **S2** and **B2** will each be between 0 and 1 000 000, inclusive.
- Allowed working time for your program: 0.1 seconds.
- Allowed memory: 16 MB.

Examples

Example input	Example output
21,20,19 6,7,9,6,4 2 1 0 0 0 0 81	0 15 10

Example input	Example output
1,2,3,3,2 1,1,1,5,1 3 1 100 12 5 53 33	-1 6 7