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

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.

After the end of the exam you should leave all used papers (including this one) in the room!!!

Problem 1 – Durankulak Numbers

In Durankulak we have a special way to write numbers. We use the following 168 digits:

A	B	C	…	Z
0	1	2	…	25

…

aA	aB	aC	…	aZ
26	27	28	…	51

…

bA	bB	bC	…	bZ
52	53	54	…	77

…

fA	…	fJ	fK	fL
156	…	165	166	167

We write the numbers as sequences of digits. 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 168 times bigger than the shown in the above table, the next digit on the left has 168*168 times bigger value than the shown in the table and so on. Your task is to write a program to **convert a Durankulak-style number into its corresponding decimal representation**.

Input

The input data consists of a single string – a Durankulak-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 Durankulak-style number and should be printed at the console.

Constraints

- The input string will have between 1 and 16 characters.

- Allowed working time for your program: 0.1 seconds. Allowed memory: 16 MB.

Examples

Input	Output
U	20

Input	Output
bM	64

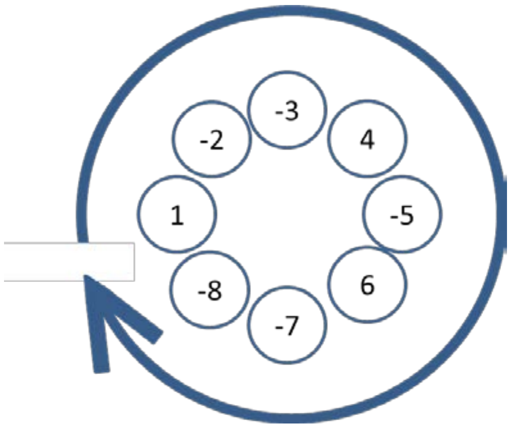
Input	Output
BaG	200

Input	Output
CfI	500

Problem 2 – Joro the Rabbit

Joro is a rabbit. But he is no ordinary rabbit – he just loves to jump around. But jumping around without any precalculated direction is too ordinary, so he likes jumping in just a given direction and to make it more fun, the jumping is done in a circle. By given terrain, help Joro find longest fun and not ordinary route of jumps.

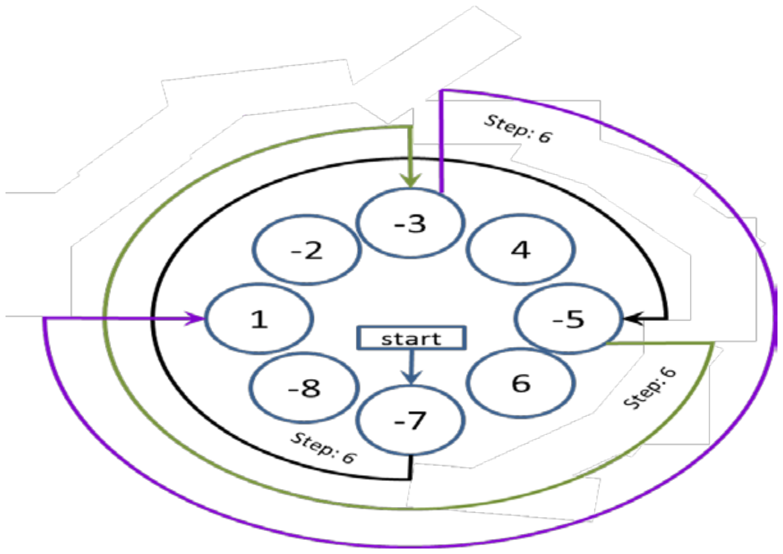
You are given the terrain as sequence of numbers. The terrain should form a circle, so the last number is before the first, and the first is after the last.



Joro can enter the terrain from every position, jump only on numbers **larger** than the one he is on, only in direction left-to-right and with the same step. Joro’s jumping steps range from 1 to the size of the terrain. **Joro cannot jump on position that he already visited**.

Example:

In the sample above, the best route is **-7**, **-5**, **-3**, **1** with **length 4** and **step 6**.



Input

The input data should be read from the console.

On the first line you will be given the terrain- numbers separated with “, ” (comma and space).

Output

The output data should be printed on the console.

The output should contain the maximal number of positions visited by Joro, using any of the possible steps.

Constraints

- **The numbers in the terrain** will be between 1 and 2 500 inclusive.
- Each of the numbers in the terrain will be **between -1000 and 1000**
- Allowed working time for your program: 0.2 seconds. Allowed memory: 16 MB.

Examples

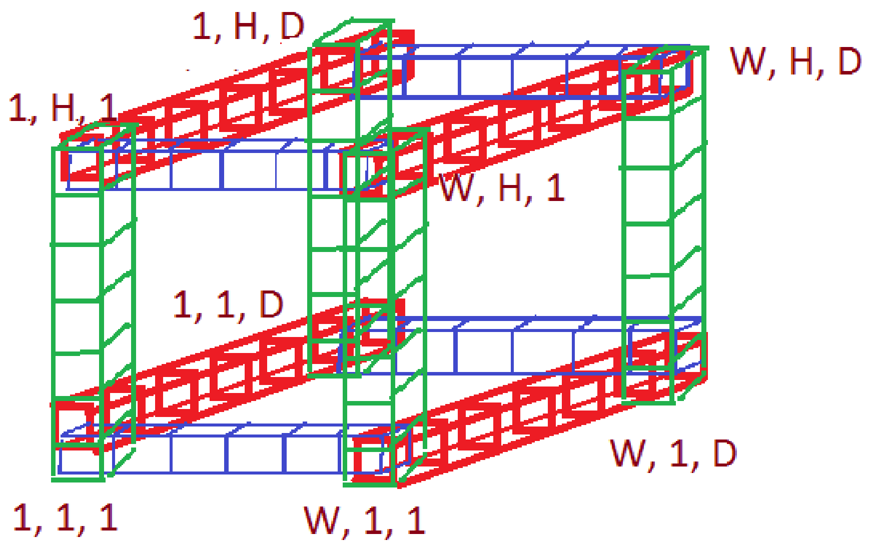
Input example	**Output example**
`1, -2, -3, 4, -5, 6, -7, -8`	`4`
`1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 0`	`11`
`1, 1, 1`	`1`

Problem 3 – Laser

You are given a rectangular cuboid of size W (width), H (height) and D (depth) consisting of W * H * D cubes. Adjacent cubes we will call any cubes which share a common wall, edge or vertex. So, any cube in the cuboid has 26 (8 + 9 + 9) adjacent cubes, except if it is on the edge or wall of the cuboid (then it has less).

Let’s say we go inside one of the cubes in the cuboid (which is not on the side, or on the edge of the cube) and shoot a laser from it into one of the adjacent cubes. We can define the direction of the shot with a 3D vector, which has only 1, -1 or 0 as its coordinate values. So, for example if we want to shoot a laser straight up, that direction will be (0, 1, 0), if we want to shoot it straight down, the direction will be (0, -1, 0), if we want to shoot it forwards and up the direction will be (0, 1, 1) and if we want to shoot it left, forward and up the direction will be (-1, 1, 1). We have a total of 26 possible directions (the number of adjacent cubes).

When the laser passes through a cube, it burns it and it cannot pass through it again. Furthermore, if the laser reaches the wall of the cuboid (that is, a cube which is on the wall), it burns the cube there and reflects back into the cuboid, continuing to burn cubes it passes through. The reflection happens according to the laws of light reflection. Basically, the component of the direction vector, which would cause the laser to leave the cuboid, becomes the opposite number. So if a laser is moving in direction “right” (1, 0, 0), once it reaches the right wall of the cube it will reflect back, moving in direction “left (-1, 0, 0), but, in this case, that will cause it to visit an already burnt cube, so the laser will stop there.



Furthermore, all cubes on the edges (red green and blue cubes on the picture) of our cuboid are burned to begin with. So a laser will always stop before reaching the edge, because it cannot go into the edge cubes.

Write a program, which by given the width, height and depth of a cuboid, along with the coordinates of the cube from which the laser is shot and the direction of the shot, determines the last position the laser can reach in the cube.

Note: the starting position is burned during the shot, so it cannot be the answer. The edges are burned by default, so no cube on the edge can be an answer either

Input

The input data should be read from the console.

On the first line of the input there will be the numbers W, H, D, separated by whitespaces – the width, height and depth of the cuboid

On the next line there will be the numbers startW, startH, startD, separated by whitespaces – the width, height and depth of the cube, from which the laser is shot.

On the third line there will be the numbers dirW, dirH, dirD, separated by whitespaces – defining the direction, in which the laser is shot.

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 only line of the output you should print exactly 3 numbers, separated by whitespaces – the width, height and depth of the last cube the laser will visit.

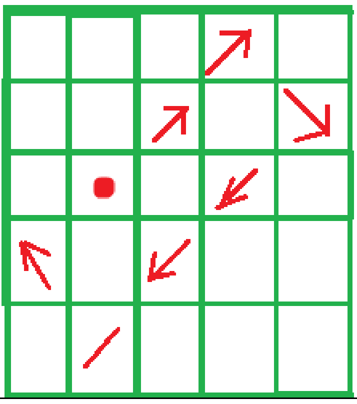
Constraints

- W, H, D, startW, startH, startD, dirW, dirH and dirD will all be integers
- 4 < W, H, D < 100
- 1 < startW, startH, startD < W, H, D
- Each of the numbers dirW, dirH, dirD can only have the values -1, 1 or 0
- Allowed working time for your program: 0.1 seconds. Allowed memory: 16 MB.

Example

Sample input	Sample output

`5 10 5`
`2 6 3`
`1 0 1`
	`0 6 1`

`Explanation: Here we are only moving on the "sixth floor" of cubes (so we can meet only walls and no edges) and the movement looks like this:` 	

Problem 4 – C# Brackets

As you may or may not know, Joro got drunk last Friday and deleted his Visual Studio’s IntelliSense. Now, he has to work without one, but he doesn’t like the idea. The worst thing is he can’t work properly without the feature, which formats his code – more specifically, he needs his opening and closing brackets to be properly aligned, and the text between them to be properly indented.

Here are some bullets on how Joro wants his code to be formatted:

1. The brackets Joro needs to be considered in the formatting are { and } – i.e. the ones defining bodies of loops, conditionals, methods, etc.
2. Joro doesn’t use tabulations to indent his code, he uses some specific string, for example /*--*/
    - Joro will tell you the string each time he needs to format something
3. Each bracket needs to be on a separate line, with the proper indentation (or lack thereof)
4. The code within an opening and closing bracket must be indented with one more indentation string than the brackets themselves
5. There must be no empty lines in the formatted code
6. There must be no empty spaces at the beginning or end of a formatted line, except if they are part of the indentation string
7. There must be no sequences of empty spaces, with a length of more than 1
    - That is, there can be single spaces between symbols in the code, but not two or more consecutive spaces

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

Input

The input data should be read from the console.

On the first line of the input there will be the number **N** – the total number of lines in Joro’s initial code.

On the next line of the input there will be a string **S** – the indentation string, which can consist of any symbols.

Each of the next **N** lines will contain a line of Joro’s code. The code will be of the C# syntax, but it is not guaranteed to be valid. The brackets, however, will be valid.

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

Output

The output data should be printed on the console.

You must print all lines of **the formatted code.**

Constraints

- 0 < N < 1000,   0 < S.Length < 100
- No line in the input will be more than 300 symbols.
- There could be empty lines in the input
- There could be sequences of whitespaces in the input
- The brackets will always be valid, but the code inside could have mistakes
- There will be no other { and } than the ones defining loop, method or conditional statement bodies – that means there will be no { or } in strings, comments, etc.
- Do not try to make "perfect C# formatting" – only what is required by the aforementioned rules
- Allowed working time for your program: 0.1 seconds.
- Allowed memory: 16 MB.

Examples

Input example	**Output example**

`3`
`>>`
`{a{`
`}`
`}`
	`{`
	`>>a`
	`>>{`
	`>>}`
	`}`

`5`
`....`
`using System;    namespace Stars`
`{class Program{`
`static     string[] separators`
`= new string[] { " " };}`
`}`
	`using System; namespace Stars`
	`{`
	`....class Program`
	`....{`
	`........static string[] separators`
	`........= new string[]`
	`........{`
	`............" "`
	`........}`
	`........;`
	`....}`
	`}`

Problem 5 – Two Is Better Than One

Solving one task is too mainstream. You are given two tasks. Solve them both.

First task

The **digits 3 and 5** are lucky digits, and all other digits are unlucky. A **lucky number** is a positive integer whose decimal representation is a **palindrome** that contains only lucky digits. A palindrome is a number that reads the same forward and backward. Write a program that counts the number of lucky numbers within a specified range.

You are given two integers **A** and **B**. Return the number of lucky numbers between **A** and **B**, inclusive.

Second task

Given a list of integer and a percentile **P** (integer between 0 and 100, inclusive), find the smallest element **E** in the list such that at least **P** percent of the elements in the list are less than or equal to **E**.

Input

The input data should be read from the console.

The number **A** and number **B** will be given on the first input line, separated by a single space.

On the second input line there will be the list of numbers mentioned in the second task. The numbers in the list will be separated by commas. On the third output line there will be the number **P**.

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 number of lucky numbers).

On the second output line print the solution of the second task (the element found).

Constraints

- **A** will be between 1 and 10^18, inclusive. **B** will be between **A** and 10^18, inclusive.
- The number of the elements in the list from second task will be between 1 and 50, inclusive.
- Each element in the list from the second task will be between -1000000 and 1000000, inclusive.
- Allowed working time for your program: 0.2 seconds.
- Allowed memory: 16 MB.

Examples

Example input	Example output

`1 99`
`-2,-1,-4,-3`
`50`
	`4`
	`-3`

Example input	Example output

`35 53`
`10,9,8,7,6,5,4,3,2,1`
`39`
	`0`
	`4`