Problem 1 – C# Clean Code

Pesho always writes useless comments in his code to remind him about some non-important things.

You are given **N** lines with valid C# code written by Pesho. Your task is to write a program that removes all comments and all empty lines from the given code. Empty line is a line without C# code after removing the comments.

Input

The input data should be read from the console.

The first line of the input will contain the number **N** of C# code lines.

On the next **N** lines your program should read the C# code lines.

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.

You should output only the cleaned code on the console. See the examples below.

Constraints

- **N** will be between 1 and 2000. The length of all lines will be between 0 and 1000 symbols.
- Allowed working time for your program: 0.1 seconds. Allowed memory: 16 MB.

Examples

Input example

```
9
using System; // no comment...
class JustClass
{ /* Just
multiline
comment  */private void JustMethod()
 {
     // string str="inception/*//*/";
 }
}
```

Output example

```
using System;
class JustClass
{ private void JustMethod()
 {
 }
}
```

Input example

```
10
class HardTest

{
 public HardMethod()
 {
  string str = @"//not a
comment ;)";//(y)
  string str2 = "/*no\"oo\\oo*/";/*noo*/
 }
}
```

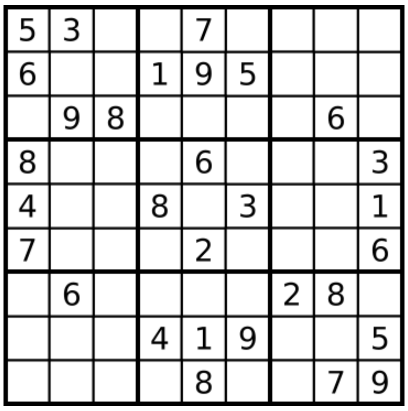
Output example

```
class HardTest
{
 public HardMethod()
 {
  string str = @"//not a
comment ;)";
  string str2 = "/*no\"oo\\oo*/";
 }
}
```

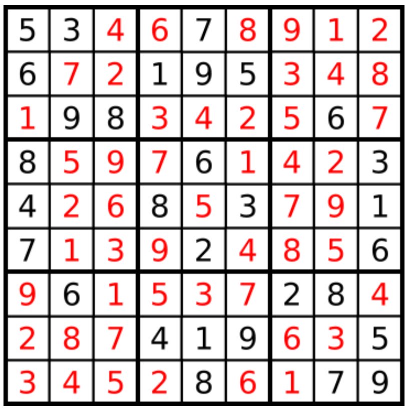
Problem 2 – Sudoku

Sudoku is a logic-based, combinatorial number-placement puzzle. The objective is to fill a 9×9 grid with digits so that **each column**, **each row**, and **each of the nine 3×3 sub-grids** that compose the grid contain all of the digits from 1 to 9.

On the pictures bellow you can see a Sudoku puzzle and its solution:



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



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

You are given a partially completed grid, which always has a unique solution. Your task is to solve the given Sudoku puzzle.

Input

The input data should be read from the console.

You will be given 9 lines with 9 symbols with numbers and dashes. The dashes represent empty cells.

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.

You should print the solved Sudoku puzzle. See the examples bellow.

Constraints

- Allowed working time for your program: 0.2 seconds. Allowed memory: 16 MB.

Examples

Input example	Output example
53--7---- 6--195--- -98----6- 8---6---3 4--8-3--1 7---2---6 -6----28- ---419--5 ----8--79	534678912 672195348 198342567 859761423 426853791 713924856 961537284 287419635 345286179

Input example	Output example
---2---63 3----54-1 --1--398- -------9- ---538--- -3------- -263--5-- 5-37----8 47---1---	854219763 397865421 261473985 785126394 649538172 132947856 926384517 513792648 478651239

Problem 3 – Employees

Mitko is the boss of a big software company called "MCPF" (Mitko Can't Play Football). MCPF has **M** employees with **N** different positions. Each position has a rating (number indicating how important the position is in the company's hierarchy). This rating will be positive integer number between 0 and 10000, inclusive.

Your task is to write a program that orders Mitko's employees by their position. If two employees' positions are equally rated, then your program should order them lexicographically by their last (family) name. And if two employees have equally rated positions and same family names, your program must also sort them by their first name lexicographically.

Input

The input data should be read from the console.

On the first line there will be the number **N** – the total number of positions in MCPF. On each of the next **N** lines there will be one job title and its rating, separated by dash ("-"). See the example below.

On the very next line there will be the number **M** – the total number of employees in MCPF. On each of the next **M** lines there you will find one employee's name (first name and last name separated by single space) and the employee's position. The employee's name and his position will be separated by dash ("-"). See the example below.

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.

Your program must print exactly **M** lines, containing the employee's names, ordered by the rules explained above. Each name must be shown on a single line.

Constraints

- **N** will be between 1 and 1000, inclusive.
- **M** will be between 1 and 1000, inclusive.
- The length of all position names, first and last names will be between 1 and 100, inclusive.
- Allowed working time for your program: 0.1 seconds.
- Allowed memory: 16 MB.

Example

Input example	Output example
9 Trainee - 0 Owner - 100 CEO - 98 Junior Developer - 30 Unit Manager - 95 Project Manager - 95 Team Leader - 94	Dimitar Dimitrov Blagoy Makendzhiev Michel Platini Petar Atanasov Ademar Junior Ivan Bandalovski Apostol Popov Atanas Georgiev

`Senior Developer - 50`
`Developer - 40`
`10`
`Georgi Georgiev – Trainee`
`Ademar Júnior – Unit Manager`
`Dimitar Dimitrov – Owner`
`Petar Atanasov – Project Manager`
`Atanas Georgiev – Trainee`
`Júnior Moraes – Trainee`
`Ivan Bandalovski – Developer`
`Apostol Popov – Developer`
`Michel Platini – CEO`
`Blagoy Makendzhiev - CEO`
	`Georgi Georgiev`
	`Junior Moraes`

Problem 4 – 3D Max Walk

You are given a **rectangular cuboid** of size **W** (width), **H** (height) and **D** (depth) consisting of **W** * **H** * **D** cubes, each containing an integer number. A **3D max walk** in the cuboid starts from the cube located at the cuboid's center (**W**, **H** and **D** are odd numbers). At each step the walk continues from the current cube in one of the 6 possible directions (left, right, up, down, deeper, shallower) to the cube which holds the maximal value among all possible cubes different than the current. The walk stops at some of the following conditions:

- Several cubes hold the same maximal value.
- There is only cube holding the maximal value but it is already visited (falls into a loop).

Your task is to write a program that finds the sum of the numbers in the cubes that are visited during the 3D max walk.

Input

The input data should be read from the console. At the first line 3 integers **W**, **H** and **D** are given separated by a space. These numbers specify the width, height and depth of the cuboid. At the next **H** lines the colors of the cubes in the cuboid are given as **D** sequences of exactly **W** integers. Each of these sequences consists of **W** integers separated by a single space. The sequences of **W** integers are separated one from another by " **|** " (space + vertical line + space).

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.

At the first line of the output print the **sum of the cubes visited by the 3D max walk**.

Constraints

- The numbers **W**, **H** and **D** are odd integers in the range [1…101].
- The integers in the cuboid are in the range [-1000…1000]
- Allowed work time for your program: 0.3 seconds.
- Allowed memory: 16 MB.

Examples

Input	Output
5 3 3	34
3 4 1 9 1 | 0 1 2 3 8 | 1 2 5 6 7	
2 **7** 3 1 **9** | 2 **5** **5** 2 1 | 8 6 3 5 **8**	
1 8 2 1 5 | 9 1 3 8 6 | 4 5 6 3 2	

Input	Output
3 3 1	19
1 2 3	
5 **6** 4	
8 4 5	

At the first example the visited cubes are: 5 -> 5 -> 7 -> 9 -> 8. After 8 the maximal possible number is 9, which is already visited (falls into a loop). At the second example the visited cubes are 6 -> 5 -> 8. After 8 there are two maximal numbers (value 5) so the walk stops.

Problem 5 – Liquid

You are given a **rectangular cuboid** of size **W** (width), **H** (height) and **D** (depth) consisting of **W** * **H** * **D** cubes, each containing an integer number specifying its **capacity** – how much liquids can pass through the cube per fixed unit of time (a non-negative integer number). Two cubes are **neighbors** if they share the same side. There are holes with unlimited capacity at the top side of all cubes staying at the top side of the cuboid. There are also holes with unlimited capacity at the bottom side of all cubes staying at the bottom side of the cuboid.

Suppose we have an unlimited supply of liquid spilled at the top of the cuboid and that the liquid can pass freely from one cube to all its neighbors except the cube above it. Your task is to write a program which **calculates the amount of liquid that could pass** from the top side of the cuboid through the top holes through its cubes to the bottom side of the cuboid and then outside of it through the bottom holes.

Input

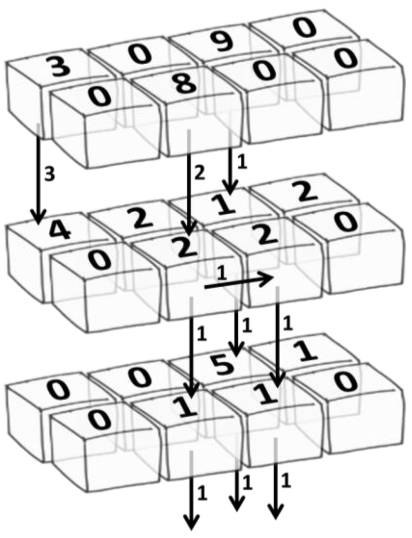
The input data should be read from the console. At the first line 3 integers **W**, **H** and **D** are given separated by a space. These numbers specify the width, height and depth of the cuboid. At the next **H** lines the colors of the cubes in the cuboid are given as **D** sequences of exactly **W** integers. Each of these sequences consists of **W** integers separated by a single space. The sequences of **W** integers are separated one from another by " **|** " (space + vertical line + space).

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.

At the first line of the output print the **amount of liquid** that can pass from the cuboid's top side to the cuboid's bottom side.



Constraints

- The numbers **W**, **H** and **D** are integers in the range [1…15].
- The integers in the cuboid are in the range [0…100]
- Allowed work time for your program: 0.5 seconds.
- Allowed memory: 64 MB.

Examples

Input	Output

`4 2 3`
`3 0 9 0 | 4 2 1 2 | 0 0 5 1`
`0 8 0 0 | 0 2 2 0 | 0 1 1 0`
	`3`

Input	Output

`3 2 1`
`1 2 3`
`4 5 6`
	`21`

The flow of the liquids for first of the above examples is shown on the figure above.