

# Exercises: Matrices

This document defines the **exercise assignments** for the ["CSharp Advanced" course @ Software University](#). Please submit your solutions (source code) of all below described problems in [Judge](#).

## Problem 1. Matrix of Palindromes

Write a program to generate and print the following **matrix of palindromes** of **3** letters with **r** rows and **c** columns like at the examples below.

- **Rows** define the first and the last letter: row 0 → 'a', row 1 → 'b', row 2 → 'c', ...
- **Columns + rows** define the middle letter:
  - column 0, row 0 → 'a', column 1, row 0 → 'b', column 2, row 0 → 'c', ...
  - column 0, row 1 → 'b', column 1, row 1 → 'c', column 2, row 1 → 'd', ...

### Input

- On the **first line**, you are given the integers **r** and **c**, separated by a space.

### Output

- On **r \* c** number of lines, print the matrix of palindromes as shown in the example.

### Constraints

- Constraints: **r** and **c** are integers in the range [1...26]; **r + c** ≤ 27.

### Examples

Input	Output
4 6	aaa aba aca ada aea afa bbb bcb bdb beb bfb bgb ccc cdc cec cfc cgc chc ddd ded dfd dgd dhd did

Input	Output
3 2	aaa aba bbb bcb ccc cdc

### Hints

- `char[] alphabet = "abcdefghijklmnopqrstuvwxyz".ToCharArray();`

## Problem 2. Diagonal Difference

Write a program that finds the **difference between the sums of the square matrix diagonals** (absolute value).

	0	1	2
0	11	2	4
1	4	5	6
2	10	8	-12

primary diagonal  
sum = 11 + 5 - 12 = 4

	0	1	2
0	11	2	4
1	4	5	6
2	10	8	-12

secondary diagonal  
sum = 4 + 5 + 10 = 19

## Input

- On the **first line**, you are given the integer **N** – the size of the square matrix
- The next **N lines** holds the values for **every row** – **N** numbers separated by a space

## Output

- Print **the absolute** difference between **the sums** of the primary and the secondary diagonal

## Examples

Input	Output	Comments
3 11 2 4 4 5 6 10 8 -12	15	<b>Primary diagonal:</b> $\text{sum} = 11 + 5 + (-12) = 4$ <b>Secondary diagonal:</b> $\text{sum} = 4 + 5 + 10 = 19$ <b>Difference:</b> $ 4 - 19  = 15$

## Problem 3. 2x2 Squares in Matrix

Find the count of **2 x 2 squares of equal chars** in a matrix.

## Input

- On the **first line**, you are given the integers **rows** and **cols** – the matrix's dimensions
- Matrix characters come at the next **rows** lines (space separated)

## Output

- Print the number of all the squares matrixes you have found

## Examples

Input	Output	Comments
3 4 A B B D E B B B I J B B	2	Two 2 x 2 squares of equal cells: A <b>B B</b> D      A B B D E <b>B B</b> B      E B <b>B B</b> I J B B      I J <b>B B</b>
2 2 a b c d	0	No 2 x 2 squares of equal cells exist.

## Problem 4. Maximal Sum

Write a program that reads a rectangular integer matrix of size **N x M** and finds in it the square **3 x 3** that **has maximal sum of its elements**.

## Input

- On the first line, you will receive the rows **N** and columns **M**. On the next **N lines** you will receive **each row with its columns**

## Output

- Print the **elements** of the 3 x 3 square as a matrix, along with their **sum**

## Examples

Input	Matrix	Output																				
4 5 1 5 5 2 4 2 1 4 14 3 3 7 11 2 8 4 8 12 16 4	<table><tr><td>1</td><td>5</td><td>5</td><td>2</td><td>4</td></tr><tr><td>2</td><td>1</td><td>4</td><td>14</td><td>3</td></tr><tr><td>3</td><td>7</td><td>11</td><td>2</td><td>8</td></tr><tr><td>4</td><td>8</td><td>12</td><td>16</td><td>4</td></tr></table>	1	5	5	2	4	2	1	4	14	3	3	7	11	2	8	4	8	12	16	4	Sum = 75 1 4 14 7 11 2 8 12 16
1	5	5	2	4																		
2	1	4	14	3																		
3	7	11	2	8																		
4	8	12	16	4																		

## Problem 5. Rubik's Matrix

Rubik's cube – everyone's favorite head-scratcher. Writing a program to solve it will be quite a difficult task for an exam, though. Instead, we have a Rubik's matrix prepared for you. You will be given a pair of dimensions: **R** and **C**. To prepare the matrix, fill each row with increasing integers, starting from one. For example, **2 x 4** matrix must look like this:

1	2	3	4
5	6	7	8

Next, you will receive series of commands, indicating which row or column you must move, in which direction, and how many times. For example, **1 up 1** means: column 1, direction: up, 1 move. After executing it, the matrix should look like:

1	6	3	4
5	2	7	8

Directions **left** and **right** means you must move the corresponding **row**, while **up** and **down** are related to the **columns**. After shuffling the Rubik's matrix, you have to **rearrange** it (meaning that the **values in each cell** should be in **increasing order**, such as the ones in the original matrix). The rearranging process should start at **top-left** and end at **bottom-right**. Find the **position** of the value you need, and print the **swap command** on the console, in the format described below.

## Input

- On the first line, you are given the integers **R** and **C**, separated by a space
- On the second line, you are given an integer **N**, indicating the number of commands to follow
- On the next N lines, you are given commands in format **{row/col} {direction} {moves}**

## Output

- On **R \* C** number of lines, print the **swap commands** needed to rearrange the matrix, either:
  - "Swap ({row}, {col}) with ({row}, {col})" or
  - "No swap required"

## Constraints

- R**, **C**, **N** are integers in range [1 ... 100]
- {row}** and **{col}** will always be inside the matrix
- {moves}** is in range [0 ...  $2^{31}-1$ ]
- Allowed time and memory: 0.25s / 16 MB

## Examples

Input	Output	Input	Output
3 3 2 1 down 1 1 left 1	No swap required Swap (0, 1) with (1, 0) No swap required Swap (1, 0) with (1, 2) Swap (1, 1) with (2, 1) Swap (1, 2) with (2, 1) No swap required No swap required No swap required	3 3 2 0 down 3 0 left 3	No swap required No swap required No swap required No swap required No swap required No swap required No swap required No swap required No swap required

## Problem 6. Target Practice

Cotton-eye Gosho has a problem. **Snakes!** An infestation of snakes! Gosho is a red-neck which means he doesn't really care about animal rights, so he bought some ammo, loaded his shotgun and started shooting at the poor creatures. He has a favorite spot – rectangular stairs which the snakes like to climb in order to reach Gosho's stash of whiskey in the attic. You're tasked with the horrible cleanup of the aftermath.

A **snake** is represented by a **string**. The **stairs** are a **rectangular matrix of size NxM**. A snake starts climbing the stairs from the **bottom-right corner** and slithers its way up in a **zigzag** – first it moves left until it reaches the left wall, it climbs on the next row and starts moving right, then on the third row, moving left again and so on. The first cell (bottom-right corner) is filled with the first symbol of the snake, the second cell (to the left of the first) is filled with the second symbol, etc. The snake is as long as it takes in order to **fill the stairs completely** – if you reach the end of the string representing the snake, start again at the beginning. Gosho is patient and a sadist, he'll wait until the stairs are completely covered with snakes and then will fire a shot.

The shot has three parameters – **impact row, impact column and radius**. When the projectile lands on the specified coordinates in the matrix it **destroys all symbols in the given radius (turns them into a space)**. You can check whether a cell is inside the blast radius using the Pythagorean Theorem (very similar to the "point inside a circle" problem).

The symbols above the impact area start **falling down until they land on other symbols (meaning a symbol moves down a row as long as there is a space below)**. When the horror ends, print on the console the **resulting staircase, each row on a new line**. You should really check out the examples at this point.

### Input

- The input data should be read from the console. It consists of exactly three lines
- On the first line, you'll receive the **dimensions** of the stairs in format: "**N M**", where **N** is the number of **rows**, and **M** is the number of **columns**. They'll be separated by a single space
- On the second line you'll receive the string representing the **snake**
- On the third line, you'll receive the **shot parameters (impact row, impact column and radius)**, all 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 should be printed on the console. It should consist of **N lines**
- Each line should contain a string representing the respective row of the final matrix

### Constraints

- The **dimensions** N and M of the matrix will be integers in the range [1 ... 12]
- The **snake** will be a string with length in the range [1 ... 20] and **will not contain any whitespace characters**
- The shot's **impact row and column** will always be **valid coordinates** in the matrix – they will be integers in the range [0 ... N – 1] and [0 ... M – 1] respectively

- The shot's **radius** will be an integer in the range [0 ... 4]
- Allowed working time for your program: 0.1 seconds. Allowed memory: 16 MB


## Examples

Input	Output	Comments																																																																																																																																																						
5 6 SoftUni 2 3 2	0 US t tn f iSi UU nUt oS	<p>The matrix has 5 rows and 6 columns. Fill it in the described pattern:</p> <table><tr><td>o</td><td>S</td><td>i</td><td>n</td><td>U</td><td>t</td></tr><tr><td>U</td><td>n</td><td>i</td><td>S</td><td>o</td><td>f</td></tr><tr><td>t</td><td>f</td><td>o</td><td>S</td><td>i</td><td>n</td></tr><tr><td>i</td><td>S</td><td>o</td><td>f</td><td>t</td><td>U</td></tr><tr><td>n</td><td>U</td><td>t</td><td>f</td><td>o</td><td>S</td></tr></table> <p>The shot lands on cell (2,3). It has a radius of 2 cells. The impact cell is shaded black and the other cells within the shot radius are shaded grey.</p> <table><tr><td>o</td><td>S</td><td>i</td><td>n</td><td>U</td><td>t</td></tr><tr><td>U</td><td>n</td><td>i</td><td>S</td><td>o</td><td>f</td></tr><tr><td>t</td><td>f</td><td>o</td><td></td><td>i</td><td>n</td></tr><tr><td>i</td><td>S</td><td>o</td><td>f</td><td>t</td><td>U</td></tr><tr><td>n</td><td>U</td><td>t</td><td>f</td><td>o</td><td>S</td></tr></table> <p>Replace all characters in the blast area with a space:</p> <table><tr><td>o</td><td>S</td><td>i</td><td></td><td>U</td><td>t</td></tr><tr><td>U</td><td>n</td><td></td><td></td><td></td><td>f</td></tr><tr><td>t</td><td></td><td></td><td></td><td></td><td></td></tr><tr><td>i</td><td>S</td><td></td><td></td><td></td><td>U</td></tr><tr><td>n</td><td>U</td><td>t</td><td></td><td>o</td><td>S</td></tr></table> <p>All characters start falling down until they land on other characters:</p> <table><tr><td>o</td><td>S</td><td>i</td><td></td><td>U</td><td>t</td></tr><tr><td>U</td><td>n</td><td></td><td></td><td></td><td>f</td></tr><tr><td>t</td><td>↓</td><td>↓</td><td></td><td>↓</td><td>↓</td></tr><tr><td>i</td><td>S</td><td>↓</td><td></td><td>↓</td><td>U</td></tr><tr><td>n</td><td>U</td><td>t</td><td></td><td>o</td><td>S</td></tr></table> <p>The resulting matrix is:</p> <table><tr><td>o</td><td></td><td></td><td></td><td></td><td></td></tr><tr><td>U</td><td>S</td><td></td><td></td><td></td><td>t</td></tr><tr><td>t</td><td>n</td><td></td><td></td><td></td><td>f</td></tr><tr><td>i</td><td>S</td><td>i</td><td></td><td>U</td><td>U</td></tr><tr><td>n</td><td>U</td><td>t</td><td></td><td>o</td><td>S</td></tr></table>	o	S	i	n	U	t	U	n	i	S	o	f	t	f	o	S	i	n	i	S	o	f	t	U	n	U	t	f	o	S	o	S	i	n	U	t	U	n	i	S	o	f	t	f	o		i	n	i	S	o	f	t	U	n	U	t	f	o	S	o	S	i		U	t	U	n				f	t						i	S				U	n	U	t		o	S	o	S	i		U	t	U	n				f	t	↓	↓		↓	↓	i	S	↓		↓	U	n	U	t		o	S	o						U	S				t	t	n				f	i	S	i		U	U	n	U	t		o	S
o	S	i	n	U	t																																																																																																																																																			
U	n	i	S	o	f																																																																																																																																																			
t	f	o	S	i	n																																																																																																																																																			
i	S	o	f	t	U																																																																																																																																																			
n	U	t	f	o	S																																																																																																																																																			
o	S	i	n	U	t																																																																																																																																																			
U	n	i	S	o	f																																																																																																																																																			
t	f	o		i	n																																																																																																																																																			
i	S	o	f	t	U																																																																																																																																																			
n	U	t	f	o	S																																																																																																																																																			
o	S	i		U	t																																																																																																																																																			
U	n				f																																																																																																																																																			
t																																																																																																																																																								
i	S				U																																																																																																																																																			
n	U	t		o	S																																																																																																																																																			
o	S	i		U	t																																																																																																																																																			
U	n				f																																																																																																																																																			
t	↓	↓		↓	↓																																																																																																																																																			
i	S	↓		↓	U																																																																																																																																																			
n	U	t		o	S																																																																																																																																																			
o																																																																																																																																																								
U	S				t																																																																																																																																																			
t	n				f																																																																																																																																																			
i	S	i		U	U																																																																																																																																																			
n	U	t		o	S																																																																																																																																																			

## Problem 7. Lego Blocks


You are given two **jagged arrays**. Each array represents a **Lego block** containing integers. Your task is first to **reverse** the second jagged array and then check if it would **fit perfectly** in the first jagged array.

First Jagged array




1	1	1	1	1	1
2	1	1	3		
2	1	1	2	3	
7	7	7	5	3	2

Second Jagged Array




1	1		
2	2	2	3
3	3	3	
4	4		

Reversed Second Array



		1	1
3	2	2	2
	3	3	3
		4	4

Matched Arrays



1	1	1	1	1	1	1	1
2	1	1	3	3	2	2	2
2	1	1	2	3	3	3	3
7	7	7	5	3	2	4	4

The picture above shows exactly what **fitting arrays** means. If the arrays fit perfectly you should **print out** the newly made rectangular matrix. If the arrays do not match (they do not form a rectangular matrix) you should print out the **number of cells** in the first array and in the second array combined. The examples below should help you understand the assignment better.

## Input

The first line of the input comes as an **integer number, n**, saying how many rows are there in both arrays. Then you have  $2 * n$  lines of numbers separated by whitespace(s). The first  $n$  lines are the rows of the first jagged array; the next  $n$  lines are the rows of the second jagged array. There might be leading and/or trailing whitespace(s).

## Output

You should print out the combined matrix in the format:

[*elem*, *elem*, ..., *elem*]

[*elem*, *elem*, ..., *elem*]

[*elem*, *elem*, ..., *elem*]

If the two arrays do not fit you should print out: **The total number of cells is: *count***

## Constraints

- The number  $n$  will be in the range [2...10]
- Time limit: 0.3 sec. Memory limit: 16 MB

## Examples

Input	Output
2 1 1 1 1 1 1 2 1 1 3 1 1 2 2 2 3	[1, 1, 1, 1, 1, 1, 1, 1] [2, 1, 1, 3, 3, 2, 2, 2]
2 1 1 1 1 1 1 1 1 1 1 1 1 1 1	The total number of cells is: 14

## Problem 8. Radioactive Mutant Vampire Bunnies

Browsing through GitHub, you come across an old JS Basics teamwork game. It is about very nasty bunnies that multiply extremely fast. There's also a player that has to escape from their lair. You really like the game so you

decide to port it to C# because that's your language of choice. The last thing that is left is the algorithm that decides if the player will escape the lair or not.

First, you will receive a line holding integers **N** and **M**, which represent the rows and columns in the lair. Then you receive **N** strings that can **only** consist of “.”, “B”, “P”. The **bunnies** are marked with “B”, the **player** is marked with “P”, and **everything** else is free space, marked with a dot “.”. They represent the initial state of the lair. There will be **only** one player. Then you will receive a string with **commands** such as **LLRRUDD** – where each letter represents the next **move** of the player (Left, Right, Up, Down).

**After** each step of the player, each of the bunnies spread to the up, down, left and right (neighboring cells marked as “.” **changes** their value to B). If the player **moves** to a bunny cell or a bunny **reaches** the player, the player has died. If the player goes **out** of the lair **without** encountering a bunny, the player has won.

When the player **dies** or **wins**, the game ends. All the activities for **this** turn continue (e.g. all the bunnies spread normally), but there are no more turns. There will be **no** stalemates where the moves of the player end before he dies or escapes.

Finally, print the final state of the lair with every row on a separate line. On the last line, print either “**dead: {row} {col}**” or “**won: {row} {col}**”. Row and col are the coordinates of the cell where the player has died or the last cell he has been in before escaping the lair.

## Input

- On the first line of input, the numbers **N** and **M** are received – the number of **rows** and **columns** in the lair
- On the next N lines, each row is received in the form of a string. The string will contain only “.”, “B”, “P”. All strings will be the same length. There will be only one “P” for all the input
- On the last line, the directions are received in the form of a string, containing “R”, “L”, “U”, “D”

## Output

- On the first N lines, print the final state of the bunny lair
- On the last line, print the outcome – “won:” or “dead:” + {row} {col}

## Constraints

- The dimensions of the lair are in range [3...20]
- The directions string length is in range [1...20]

## Examples

Input	Output
5 8 .....B ...B.... ....B..B ..... ..P..... ULLL	BBBBBBBB BBBBBBBB BBBBBBBB .BBBBBBB ..BBBBBB won: 3 0
4 5 ..... ..... .B... ...P. LLLLLLLL	.B... BBB.. BBBB.. BBB.. dead: 3 1

## Problem 9. Crossfire

You will receive **two integers** which represent the **dimensions** of a **matrix**. Then, you must **fill the matrix** with **increasing integers** starting from 1, and continuing on every row, like this:

first row: 1, 2, 3, ..., n

second row: n + 1, n + 2, n + 3, ..., n + n

third row: 2 \* n + 1, 2 \* n + 2, ..., 2 \* n + n

You will also receive several commands in the form of **3 integers** separated by a space. Those 3 integers will represent a **row** in the matrix, a **column** and a **radius**. You must then **destroy** the cells which correspond to those arguments **cross-like**.

**Destroying** a cell means that, **that cell** becomes completely **nonexistent** in the matrix. Destroying cells **cross-like** means that you form a **cross figure** with center point - equal to the cell with coordinates – the **given row** and **column**, and **lines** with length equal to the **given radius**. See the examples for more info.

The **input ends** when you receive the command “Nuke it from orbit”. When that happens, you must print what has remained from the initial matrix.

### Input

- On the first line you will receive the dimensions of the matrix. You must then fill the matrix according to those dimensions
- On the next several lines you will begin receiving 3 integers separated by a single **space**, which represent the row, col and radius. You must then destroy cells according to those coordinates
- When you receive the command “**Nuke it from orbit**” the input ends

### Output

- The output is simple. You must print what is left from the matrix
- Every row must be printed on a new line and every column of a row - separated by a space

### Constraints

- The dimensions of the matrix will be integers in the range [2, 100]
- The given rows and columns will be valid integers in the range  $[-2^{31} + 1, 2^{31} - 1]$
- The radius will be in range  $[0, 2^{31} - 1]$
- Allowed time/memory: 250ms/16MB

### Examples

Input	Output	Comment
5 5 3 3 2 4 3 2 Nuke it from orbit	1 2 3 4 5 6 7 8 10 11 12 13 16 21	Initial matrix: 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 Result from <b>first</b> destruction: 1 2 3 4 5 6 7 8 10 11 12 13 15 16 21 22 23 25 Result from <b>second</b> destruction:



		1 2 3 4 5 6 7 8 10 11 12 13 16 21
5 5 4 4 4 Nuke it from orbit	1 2 3 4 6 7 8 9 11 12 13 14 16 17 18 19	

## Problem 10. The Heigan Dance

At last, level 80. And what do level eighties do? Go raiding. This is where you are now – trying not to be wiped by the famous dance boss, Heigan the Unclean. The fight is pretty straightforward - dance around the Plague Clouds and Eruptions, and you'll be just fine.

Heigan's chamber is a 15-by-15 two-dimensional array. The player always starts at the **exact center**. For each turn, Heigan uses a spell that hits a certain cell and the neighboring **rows/columns**. For example, if he hits (1,1), he also hits (0,0, 0,1, 0,2, 1,0 ... 2,2). If the player's current position is within the area of damage, the player tries to move. First, he tries to move **up**, if there's **damage/wall**, he tries to move **right**, then **down**, then **left**. If he **cannot move** in any direction, because **the cell is damaged** or there is a **wall**, the player **stays** in place and takes the damage.

**Plague cloud** does 3500 damage **when it hits**, and 3500 damage **the next turn**. Then it **expires**. **Eruption** does 6000 damage **when it hits**. If a spell hits a player that also has an active Plague Cloud from the previous turn, the **cloud** damage is applied **first**. **Both** Heigan and the player **may** die in the same turn. If Heigan is **dead**, the spell he **would** have casted is **ignored**.

The player always starts at **18500** hit points; Heigan starts at **3,000,000** hit points. **Each** turn, the player does damage to Heigan. The fight is over either when the player is **killed**, or Heigan is **defeated**.

### Input

- On the first line you receive a floating-point number **D** – the damage done to Heigan each turn
- On the next several lines – you receive input in format **{spell} {row} {col}** – **{spell}** is either **Cloud** or **Eruption**

### Output

- On the first line
  - If Heigan is defeated: **"Heigan: Defeated!"**
  - Else: **"Heigan: {remaining}"**, where remaining is rounded to two digits after the decimal separator
- On the second line:
  - If the player is killed: **"Player: Killed by {spell}"**
  - Else **"Player: {remaining}"**
- On the third line: **"Final position: {row, col}"** -> the last coordinates of the player

### Constraints

- D** is a floating-point number in range [0 ... 500000]
- A damaging spell will always affect at least one cell
- Allowed memory: 16 MB
- Allowed working time: 0.25s

## Examples

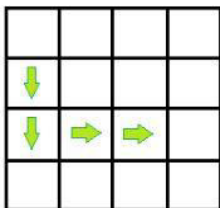
Input	Output
10000 Cloud 7 7 Eruption 6 7 Eruption 8 7 Eruption 8 7	Heigan: 2960000.00 Player: Killed by Eruption Final position: 8, 7
Input	Output
500000 Cloud 7 6 Eruption 7 8 Eruption 7 7 Cloud 7 8 Eruption 7 9 Eruption 6 14 Eruption 7 11	Heigan: Defeated! Player: 12500 Final position: 7, 11
Input	Output
12500.66 Cloud 7 7 Cloud 7 7 Cloud 7 7 Cloud 7 7	Heigan: 2949997.36 Player: Killed by Plague Cloud Final position: 7, 7

## Problem 11. Parking System

The parking lot in front of SoftUni is one of the busiest in the country, and it's a common cause for conflicts between the doorkeeper Bai Tzetzko and the students. The SoftUni team wants to proactively resolve all conflicts, so an automated parking system should be implemented. They are organizing a competition – Parkoniada – and the author of the best parking system will win a romantic dinner with RoYaL. That's **exactly** what you've been dreaming of, so you decide to join in.

The parking lot is a **rectangular** matrix where the **first** column is **always** free and all other cells are parking spots. A car can enter from **any cell** of the **first column** and then decides to go to a specific spot. If that spot is **not** free, the car searches for the **closest** free spot on the **same** row. If **all** the cells on that specific row are used, the car cannot park and leaves. If **two** free cells are located at the **same** distance from the **initial** parking spot, the cell which is **closer** to the entrance is preferred. A car can **pass** through a used parking spot.

Your task is to calculate the distance travelled by each car to its parking spot.



Example: A car enters the parking at row 1. It wants to go to cell 2, 2 so it moves through **exactly four** cells to reach its parking spot.

## Input

- On the first line of input, you are given the integers **R** and **C**, defining the dimensions of the parking lot

- On the next several lines, you are given the integers **Z, X, Y** where **Z** is the entry row and **X, Y** are the coordinates of the desired parking spot
- The input stops with the command '**stop**'. All integers are separated by a **single** space

## Output

- For each car, print the distance travelled to the desired spot or the first free spot
- If a car cannot park on its desired row, print the message '**Row {row number} full**'

## Constraints

- $2 \leq R, C \leq 10000$
- Z, X, Y are inside the dimensions of the matrix. Y is never on the first column
- There are no more than 1000 input lines
- Allowed time/space: 0.1s (C#) /16MB

## Examples

Input	Output
4 4 1 2 2 2 2 2 2 2 2 3 2 2 stop	4 2 4 Row 2 full

## Problem 12. String Matrix Rotation

You are given a **sequence of text lines**. Assume these text lines form a **matrix of characters** (pad the missing positions with spaces to build a rectangular matrix). Write a program to **rotate the matrix** by 90, 180, 270, 360, ... degrees. Print the result at the console as sequence of strings. Examples:

Input	Rotate(90)	Rotate(180)	Rotate(270)																																																															
hello softuni exam	<table><tr><td>e</td><td>s</td><td>h</td></tr><tr><td>x</td><td>o</td><td>e</td></tr><tr><td>a</td><td>f</td><td>l</td></tr><tr><td>m</td><td>t</td><td>l</td></tr><tr><td></td><td>u</td><td>o</td></tr><tr><td></td><td>n</td><td></td></tr><tr><td></td><td>i</td><td></td></tr></table>	e	s	h	x	o	e	a	f	l	m	t	l		u	o		n			i		<table><tr><td></td><td></td><td></td><td>m</td><td>a</td><td>x</td><td>e</td></tr><tr><td>i</td><td>n</td><td>u</td><td>t</td><td>f</td><td>o</td><td>s</td></tr><tr><td></td><td></td><td>o</td><td>l</td><td>l</td><td>e</td><td>h</td></tr></table>				m	a	x	e	i	n	u	t	f	o	s			o	l	l	e	h	<table><tr><td></td><td>i</td><td></td></tr><tr><td></td><td>n</td><td></td></tr><tr><td>o</td><td>u</td><td></td></tr><tr><td>l</td><td>t</td><td>m</td></tr><tr><td>l</td><td>f</td><td>a</td></tr><tr><td>e</td><td>o</td><td>x</td></tr><tr><td>h</td><td>s</td><td>e</td></tr></table>		i			n		o	u		l	t	m	l	f	a	e	o	x	h	s	e
e	s	h																																																																
x	o	e																																																																
a	f	l																																																																
m	t	l																																																																
	u	o																																																																
	n																																																																	
	i																																																																	
			m	a	x	e																																																												
i	n	u	t	f	o	s																																																												
		o	l	l	e	h																																																												
	i																																																																	
	n																																																																	
o	u																																																																	
l	t	m																																																																
l	f	a																																																																
e	o	x																																																																
h	s	e																																																																

## Input

- The first line holds a command in format "**Rotate(X)**" where **X** are the degrees of the requested rotation
- The next lines contain the **lines of the matrix** for rotation
- The input ends with the command "**END**"

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

## Output

- Print at the console the **rotated matrix** as a sequence of text lines

## Constraints

- The rotation **degrees** are positive integer in the range [0...90000], where **degrees** is **multiple of 90**
- The number of matrix lines is in the range [1...1 000]
- The matrix lines are **strings** of length 1 ... 1 000
- Allowed working time: 0.2 seconds. Allowed memory: 16 MB

## Examples

Input	Output
Rotate(90) hello softuni exam END	esh xoe afl mtl uo n i

Input	Output
Rotate(180) hello softuni exam END	maxe inutfos olleh

Input	Output
Rotate(270) hello softuni exam END	i n ou ltm lfa eox hse

Input	Output
Rotate(720) js exam END	js exam

Input	Output
Rotate(810) js exam END	ej xs a m

Input	Output
Rotate(0) js exam END	js exam