Problem 1 – Sevenland Numbers

In Sevenland we use a numeral system of base 7. It consists of seven digits (instead of the traditional 10) and these 7 digits are: **0**, **1**, **2**, **3**, **4**, **5**, and **6**. The numbers in the system of base 7 are just like the decimal numbers, but after 6 the next number is 10. More general, the numbers in the 7-base numeral system are: 0, 1, 2, 3, 4, 5, 6, 10, 11, …, 16, 20, 21, …, 26, 30, …, 65, 66, 100, 101, …, 106, 110, …, 166, 200, …, 666, 1000.

Write a program that takes as input a 7-based integer number **K** and calculates and prints the next number following it (**K+1**).

Input

The input data should be read from the console and consists of a single line holding a 7-based integer **K**.

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.

The output should consist of a single line holding the number **K+1** (in 7-based numeral system).

Constraints

- The number **K** is in the range [0…666] inclusive.
- Allowed work time for your program: 0.1 seconds.
- Allowed memory: 16 MB.

Examples

Input	Output
5	6

Input	Output
56	60

Input	Output
166	200

Input	Output
200	201

Problem 2 – A-nacci

The A-nacci sequence (read "ei nachi"), often called "ei-nachi, aman ot teya zadachi" (read it however you like) is a sequence similar to the Fibonacci sequence – each element is formed by the sum of the previous two, but with a little different rules for the elements.

The elements in the A-nacci sequence are the capital letters from the English alphabet. Each letter has a code, determined by its position in the alphabet – A has the code 1, B has the code 2, …, Z has the code 26. Here are all the elements that can be in an A-nacci sequence, along with their codes:

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
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	26

The first two elements in the sequence can be any two of the letters above. **Every next element has a code equal to the sum of the codes of the previous two elements**.

For example, if A and B are the first two elements, the third element will be C (code(A) = 1, code(B) = 2, 1 + 2 = 3, code(C) = 3). Analogically, the fourth element's code will be determined by the sum of the codes of B and C, so the fourth element will be E.

If the sum of two codes is larger than 26, then that sum is taken by its modulus by 26 (you know this as the % operator in C#). For example, if the sum is 27, then the code will be (27 by modulus 26) = 1, which is the code of A. Another example – if the first two elements are Y and D, then the sum of their codes is 25 + 4 = 29, which is larger than 26, so the code of the next element will be (29 by modulus 26) = 3, which is the code of C.

The **A-nacci figure** consists of **lines of sequential elements from an A-nacci sequence**, printed out similarly to the letter A (but without the dash in the middle). **The first line contains exactly one element – the first element of an A-nacci sequence**. The **second line contains the second and third elements** of the sequence, **concatenated** (that is, not separated by anything). **Each of the next lines contains exactly two elements** – the **next elements of the sequence**, **separated by a certain number of whitespaces**. The number of whitespaces separating the elements on the third line, fourth line and so on, are as follows:

- **The third line has exactly one whitespace** between the two elements
- **The fourth line has exactly two whitespaces** between the two elements
- …
- **The N^th line has exactly N-2 whitespaces** between the two elements

Write a program, which, by given the first two elements (letters) of the A-nacci sequence and the number of lines in the A-nacci form, prints an A-nacci form on the console.

Input

The input data should be read from the console.

The **first two lines** will contain the values of the **first two elements** of the A-nacci sequence – each element will be a **capital English letter** on a separate line.

On the third line of the input there will be the number **L** – **the number of lines** in the A-nacci form.

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.

The output should contain **exactly L lines**. The **first line should contain exactly one capital English letter**. **The second line** (if L>1) **should contain exactly two capital English letters**. The **third line should contain two capital English letters, separated by a single whitespace** (" ") and so on. There **shouldn't be any whitespaces after the second** (i.e. last) **letter on a line**.

Constraints

- 1 ≤ **L** ≤ 42.
- All elements of the A-nacci sequence are characters, which are capital letters from the English alphabet.
- Allowed working time for your program: 0.1 seconds. Allowed memory: 16 MB.

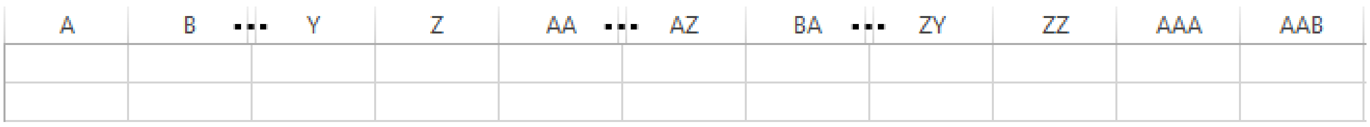
Examples

Input example	**Output example**

C B 3	C BE G L
A Q 1	A

Problem 3 – Excel Columns

Columns are a fundamental part of any spreadsheet program such as Microsoft Excel. Columns run vertically in a worksheet. Each column is identified by a capital Latin letters in the column header starting with column with **identifier** A and running through to column with identifier Z. After Z you get AA, AB, AC etc. until you get to AZ. Then it is BA, BB, BC, …, ZY, ZZ, AAA, AAB, …, AAZ, ABA, …, ZZY, ZZZ, AAAA, AAAB and so on… The last column is ZZZZZZZZZZ.



Recently Todor has learned how to work with Excel. Since then all that he do is giving **integer indices** to every possible column. Starting from the first column A (with index 1), through Z (with index 26), AA (with index 27), and so on. Of course this is a very, very hard and time-wasting job. You, as programmer, know that this can be solved easily with a computer program.

Help Todor by writing a program that converts excel column identifier to a column index.

Input

The input data should be read from the console.

On the first line of the input there will be an integer **N** – column identifier length.

On each of the next **N** lines there will be a single character. All characters together construct a column identifier.

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 output line print the column index.

Constraints

- **N** will be between 1 and 10, inclusive.
- Each column character in the input will be capital Latin letter (‘A’ – ‘Z’)
- Allowed work time for your program: 0.1 seconds. Allowed memory: 16 MB.

Examples

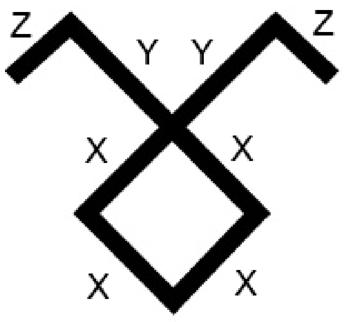
Example input	Example output
1 Z	26

Example input	Example output
2 A A	27

Example input	Example output
4 C E C A	56187

Example input	Example output
4 B E A R	38576

Problem 4 – Telerik Logo



Telerik Academy is considering opening a new office in Great Britain. Therefore the whole Trainers team is traveling to the United Kingdom for the important event. They've decided that they need to print some advertising posters and give them away to the local citizens. Please help them and print some posters with the Telerik Logo in different sizes. As a little reminder, here it is the logo itself (n. b. the letters are for reference only and denote the lengths of the sides):

Input

The input data should be read from the console. You will receive an integer number **X**. Note that **X** will always be equal to **Y** and **Z** will always be **(X / 2) + 1**. (Refer to the examples).

Output

The output should be printed on the console.

Use the "*" (asterisk) character to print the logo and "." (dot) for the rest.

Constraints

- **X** will always be a positive **odd** number between **3** and **27** inclusive.
- Allowed working time for your program: 0.1 seconds.
- Allowed memory: 16 MB.

Examples

Example input	Example output

3	`.*...*.`
	`*.*.*.*`
	`...*...`
	`..*.*..`
	`.*...*.`
	`..*.*..`
	`...*...`

Example input	Example output

5	`..*.......*..`
	`.*.*.....*.*.`
	`*...*...*...*`
	`.....*.*.....`
	`......*......`
	`.....*.*.....`
	`....*...*....`
	`...*.....*...`
	`..*.......*..`
	`...*.....*...`
	`....*...*....`
	`.....*.*.....`
	`......*......`

Problem 5 – Bit Ball

The life in Bitlandia is monotonous and boring. The only thing that makes the bits happy is the game of **Bit ball**. It's a simple game, similar to football, in which two teams of eleven bits play against each other. Let's refer to the teams as **"Top"** team and **"Bottom"** team. The playground is a grid of 8x8 cells, in which a **zero** denotes an **empty cell**, and **one** denotes a **player**.

You will receive information about the field as a list of **16** bytes (positive integers in the range [0…255]) n_0, n_1, …, n_{15}. The bits of the **first 8 numbers** (n_0, n_1, …, n_7) denote the positions of the players from the **"Top" team** and they will contain a total of eleven bits with a value of one in their binary representations. The same applies for the **second 8 numbers** (n_8, n_9, …, n_{15}), except that they denote the positions of the **"Bottom" team** players. For example the bits of n_0 and the bits of n_8 represent the first row of the field, the bits of n_1 and n_9 represent the second row and so on.

The game consists of several phases:

1. First the players of the "Top" team take places on the field. Refer to **Table 1**. Each **"T"** in the table represents a **player** (i.e. a bit with a value of **1**).
2. Then their opponents take their places. Refer to **Table 2**. Each **"B"** in the table represents a **player** (i.e. a bit with a value of **1**).
3. Here comes an interesting part. The bitballers (the players) are very similar to the Bulgarian football players - they are really rough. So, after the placement, if two players appear to be in the **same cell**, they instantly commit a foul and both receive a red card. Afterwards the cell becomes **empty**. Refer to **Table 3.** Positions [4, 2] and [4, 1] (row 4, columns 2 and 1) are now empty.
4. The final part is attacking. Refer to **Table 4**. Each bit from the **"Top" team** attacks **downwards** and each bit from the **"Bottom" team** attacks **upwards**. A goal is scored if a bit can get to the end of the field (the bottom end for "Top" team and the top end for the "Bottom" team). Each bit can only move in **straight line** and through **empty cells** (zeros) e.g. the "T" player on position [2, 5], (2 being the row and 5 being the column) can score, because no other player blocks his way to the end, but the "B" player on [6, 6] can't, because the players on [4, 6], [3, 6] and [1, 6] block his way.

Today is the important final of the Champions Bit League between the teams of Bitogorec and Bitev Plovdiv. Your task is to find what will be the final score of the game.

Input

The input data should be read from the console.
There will be exactly 16 lines each holding an integer number (n_0, n_1, …, n_{15}).
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 output row you should print the final result of the game in the following format "X:Y", where X is the score of the "Top" team, and "Y" is the score of the "Bottom" team.

Constraints

- The numbers $\mathbf{n_0}$, $\mathbf{n_1}$, …, $\mathbf{n_{15}}$ are positive integers in the range [0…255]
- Two players from the same team will never be in the same cell
- Allowed work time for your program: 0.1 seconds.
- Allowed memory: 16 MB.

	7	6	5	4	3	2	1	0	
0				T					$n_0 = 16$
1		T			T		T		$n_1 = 74$
2			T		T				$n_2 = 40$
3									$n_3 = 0$
4		T		T		T	T		$n_4 = 86$
5									$n_5 = 0$
6				T					$n_6 = 16$
7									$n_7 = 0$

Table 1 - The eleven bits of Top Team

	7	6	5	4	3	2	1	0	
0									$n_8 = 0$
1									$n_9 = 0$
2	B							B	$n_{10} = 129$
3		B			B			B	$n_{11} = 73$
4						B	B		$n_{12} = 6$
5						B			$n_{13} = 4$
6		B			B				$n_{14} = 72$
7				B					$n_{15} = 16$

Table 2 - The eleven bits of Bottom Team

	7	6	5	4	3	2	1	0
0				T				
1		T			T		T	
2	B		T		T			B
3		B			B			B
4		T		T				
5						B		
6		B		T	B			
7				B				

Table 3 – All the players left after the fouls

	7	6	5	4	3	2	1	0
0	↑			T		↑		↑
1	|	T			T	|	T	|
2	B		T		T	|	|	B
3		B	|		B	|	|	B
4		T	|	T		|	|	
5			|			B	|	
6		B	|	T	B		|	
7			↓	B			↓	

Table 4 – Attacking

Examples

Example input	**Example output**
16 74 40 0 86 0 16 0 0 0 129 73 6 4 72 16	2:3

Example input	**Example output**
240 0 240 0 96 0 0 0 0 0 0 6 0 15 0 15	4:4