**Problem 1 – Cinema**

Most people like going to the cinema. If you are such person, help the cinema director Kircho solve his financial task.

Kircho is trying to calculate how much his **incomes** are for a single projection, if the movie hall is completely full. He knows the **type of projection** (and how much a ticket for each type costs), the number of **rows** and the number of **columns** in the hall (the cimena hall is rectangular).

There are three types of movies projected in Kircho’s cinema:

* **Premiere** projection – **12.00** leva
* **Normal** projection – **7.50** leva
* **Discount** projection for kids, students and retirees – **5.00** leva

Your task is to calculate the incomes Kircho gets for a single projection full of people. You need to calculate how many people are watching the movie depending on the rows and columns in the hall and find the incomes in levas depending on the type of projection.

**Input**

* The input data is read from the console.
* A string representing the type of projectionstays at the first input line. It can be one of the following: “**Premiere**”, “**Normal**”, “**Discount**” (without the quotes).
* The numberof **rows** stays at the second input line.
* The number of **columns** stays at the third input line.
* The input data will always be valid and in the format described. There is no need to check it explicitly.

**Output**

* The output data must be printed on the console.
* On the only output line you must print the **incomes** with two digits after the decimal point with “leva” appended at the end (see the example tests). Use "**.**" as decimal separator.

**Constraints**

* The **rows** and **columns** are numbers between 1 and 100, inclusively.
* Time limit: 0.25 seconds.
* Allowed memory: 16 MB.

**Examples**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Input** | **Output** |  | **Input** | **Output** |
| Premiere  10  12 | 1440.00 leva | Normal  21  13 | 2047.50 leva |

**Problem 2 – Illuminati**

The illuminati are an ancient, very secret society that very secretly rules the world. They’ve managed to keep their existence secret by using a very secretive way of communication. Their secret is that they incorporate their secret messages into popular (non-secret) movies.

Your task is to extract the meaning of some movie lines. The messages are actually codes that can be extracted by summing the secret values of every vowel in the message. The values of the vowels are as follows: **A = 65, E = 69, I = 73, O = 79, U = 85.** The values apply for both upper and lowercase letters. For example, ‘**I am Batman!**’ has a total of **4 vowels**: three times ‘**A**’ and one time ‘**I’** and their sum is: **3 \* 65 + 1 \* 73 = 268**.

**Input**

* The input data should be read from the console.
* The only input line holds the message that has to be deciphered.

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 you must print the **number of vowels** in the message.
* On the second output line you must print the **sum of all the vowels** in the message.

**Constraints**

* The length of the message will be between 1 and 800 000 characters.
* Allowed working time for your program: 0.1 seconds.
* Allowed memory: 16 MB.

**Examples**

|  |  |
| --- | --- |
| **Input** | **Output** |
| Listen very carefully, I shall say this only once. | 13  933 |
| Heeeeeeere’s JOHNNY! | 9  631 |
| Don’t ask me about my business! | 9  669 |

**Problem 3 – New House**

Little Joro likes to build huts. After he built all the huts in his whole village, he decided to go to the big city and start building houses. But his knowledge of how to do this is limited. Help Joro to design the façade of a beautiful house by printing it to the console. The house must have a roof and one floor. The roof must contains only the symbols ‘**\***’ and ‘**-**’ and the floor must contains the symbols ‘**\***’ and ‘**|**’ (see the examples below).

**Input**

* The input data should be read from the console.
* At the only input line you are given an integer number **N** (always an **odd number**) showing the height of the house (without the roof).

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.
* You should print the house on the console, just like in the examples below. Each row can contain only the following characters: “**-**” (dash), “**\***” (asterisk) and “**|**” (pipe).

**Constraints**

* The number **N** will be a positive odd integer number between 3 and 101, inclusive.
* Allowed working time for your program: 0.25 seconds.
* Allowed memory: 16 MB.

**Examples**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Input** | **Output** |  | **Input** | **Output** |  | **Input** | **Output** |
| 3 | -\*-  \*\*\*  |\*|  |\*|  |\*| | 5 | --\*--  -\*\*\*-  \*\*\*\*\*  |\*\*\*|  |\*\*\*|  |\*\*\*|  |\*\*\*|  |\*\*\*| | 7 | ---\*---  --\*\*\*--  -\*\*\*\*\*-  \*\*\*\*\*\*\*  |\*\*\*\*\*|  |\*\*\*\*\*|  |\*\*\*\*\*|  |\*\*\*\*\*|  |\*\*\*\*\*|  |\*\*\*\*\*|  |\*\*\*\*\*| |

**Problem 4 – Crossing Sequences**

We’re dealing with two sequences: the **Tribonacci sequence**, where every number is the sum of the previous three, and the **number spiral**, defined by walking over a grid of numbers as a spiral (right, down, left, up, right, down, up, left, …) and writing down the current number every time we take a turn. Find the first number that appears in both sequences.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Example**  Let the Tribonacci sequence start with 1, 2 and 3. It will therefore contain the numbers 1, 2, 3, 6, 11, 20, 37, 68, 125, 230, 423, 778, 1431, 2632, 4841, 8904, 16377, 30122, 55403, 101902, and so on.  Also, let the number spiral start with 5 and have a step of 2; it then contains he numbers 5, 7, 9, 13, 17, 23, 29, 37, etc. Since 37 is the first number that is both in the Tribonacci sequence and in the spiral, it is the answer.  **Input**  The input data should be read from the console. | |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | | **45** | … | … | … | … | **55** | | … | **17** | 19 | 21 | **23** | … | | … | 15 | **5** | **7** | … | … | | … | **13** | 11 | **9** | … | … | | **37** | … | … | … | **29** | … | | … | … | … | … | … | **65** | |

* On the first three lines of input, you will read **three integers**, representing the three initial numbers of the Tribonacci sequence.
* On the next two lines of input, you will read **two integers**, representing the initial number and the step of each grid cell for the number spiral.

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

**Output**

The output must be printed on the console.

On the single line of output you must print **the smallest number that appears in both sequences**. If no number **in the range [1 … 1 000 000]** exists that appears in both sequences, print "**No**".

**Constraints**

* All numbers in the input will be in the range [1 … 1 000 000].
* Allowed work time for your program: 0.25 seconds.
* Allowed memory: 16 MB.

**Examples**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Input** | **Output** |  | **Input** | **Output** |  | **Input** | **Output** |  | **Input** | **Output** |  | **Input** | **Output** |
| 1  2  3  5  2 | 37 | 1  1  1  1  1 | 1 | 13  25  99  5  2 | 13 | 99  99  99  2  2 | No | 4  1  7  23  3 | 71 |

**Problem 5 – Catch the Bits**

You are given a **sequence of bytes**. Consider each byte as sequence of exactly 8 bits. You are given also a number **step**. Write a program to extract all the bits at positions: **1**, **1 + step**, **1 + 2\*step**, ... Print the output as a sequence of bytes. In case the last byte have less than 8 bits, add trailing zeroes at its right end. Bits in each byte are counted from the leftmost to the rightmost. Bits are numbered starting from 0.

**Input**

* The input data should be read from the console.
* The number **n** stays at the first line.
* The number **step** stays at the second line.
* At each of the next **n** lines **n** bytes are given, each at a separate line.

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. Print the output bytes, each at a separate line.

**Constraints**

* The number **n** will be an **integer** number in the range [1…100].
* The number **step** will be an **integer** number in the range [1…20].
* The **n numbers** will be integers in the range [0…255].
* Allowed working time for your program: 0.25 seconds.
* Allowed memory: 16 MB.

**Examples**

|  |  |  |
| --- | --- | --- |
| **Input** | **Output** | **Comments** |
| 2  11  109  87 | 128 | We have the following input sequence of 16 bits (2 bytes):  0**1**101101 0101**0**111.  We take the bits 1 and 12 (step=11). We obtain the sequence **10**.  We pad the sequence with 6 trailing zeroes. Result: **10000000**. |

|  |  |  |
| --- | --- | --- |
| **Input** | **Output** | **Comments** |
| 3  2  45  87  250 | 63  192 | We have the following input sequence of 24 bits (3 bytes):  0**0**1**0**1**1**0**1** 0**1**0**1**0**1**1**1** 1**1**1**1**1**0**1**0**  We take bits 1, 3, 5, …, 23 (step=2). We obtain the sequence:  00111111 1100. We pad it with 4 zeroes to obtain 2 full bytes. Result: **00111111 11000000**. |

|  |  |  |
| --- | --- | --- |
| **Input** | **Output** | **Comments** |
| 2  2  45  87 | 63 | We have the following input sequence of 16 bits (2 bytes):  0**0**1**0**1**1**0**1** 0**1**0**1**0**1**1**1**  We take bits 1, 3, 5, …, 15 (step=2). We obtain the sequence:  00111111 (8 bits). No padding is needed. Result: **00111111**. |