**Problem 1 – Fruit Market**

|  |  |
| --- | --- |
| The local fruit market offers fruits and vegetables with the following standard **price list**:   * banana 🡪 1.80 * cucumber 🡪 2.75 * tomato 🡪 3.20 * orange 🡪 1.60 * apple 🡪 0.86 | The market owner decided to introduce the following **discounts**:   * Friday 🡪 10% off for all products * Sunday 🡪 5% off for all products * Tuesday 🡪 20% off for fruits * Wednesday 🡪 10% off for vegetables * Thursday 🡪 30% off for bananas |

Write a program that helps the fruit market owner to **calculate the total price** for orders that consist of **day**, **3 products** with **quantities**.

**Input**

The input data should be read from the console. The input data consists of exactly 7 lines:

* At the first line you will be given the **day of week**.
* At the next 6 lines you will be given: **quantity1**, **product1**, **quantity2**, **product2**, **quantity3**, **product3**.

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

**Output**

You have to print at the console the **total price** for the specified 3 products at the specified day of week.

**Constraints**

* The **day of week** is one of the values: **Monday**, **Tuesday**, **Wednesday**, **Thursday**, **Friday**, **Saturday**, and **Sunday**.
* The product quantities (**quantity1**, **quantity2**, **quantity3**) will be a number in the range [1…100], with up to 2 digits after the decimal point. The will be used "**.**" as decimal separator.
* The products names (**product1**, **product2**, **product3**) is one of the values: **banana**, **cucumber**, **tomato**, **orange**, and **apple**.
* The total price should be rounded to exactly 2 digits after the decimal point (use "**.**" as decimal separator).
* Allowed work time for your program: 0.1 seconds.
* Allowed memory: 16 MB.

**Examples**

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Input** | **Output** |  | **Input** | **Output** |  | **Input** | **Output** |  | **Input** | **Output** |
| Friday  3  banana  5  tomato  2  cucumber | 24.21 | Tuesday  1.5  apple  2.50  orange  0.5  tomato | 5.83 |  | Monday  10  tomato  6  cucumber  10  orange | 64.50 | Thursday  3  banana  6.5  apple  2.33  tomato | 16.83 |

**Problem 2 – Biggest Triple**

We are given **n numbers**, e.g. {3, 7, 2, 8, 1, 4, 6, 9}. We split them into **triples**: sequences of 3 consecutive numbers (except the last sequence that could have 1 or 2 numbers). In our example, the numbers are split into these triples: the first three numbers {3, 7, 2}; the second three numbers {8, 1, 4}; the last two numbers {6, 9}. Write a program that enters **n** numbers and **finds the triple with biggest sum of numbers**. In our example this is the last triple: {6, 9}. In case there are several triples with the same biggest sum, print the leftmost of them.

**Input**

The input data should be read from the console. The input data consists of **exactly one line** holding the input numbers, separated one from another by a space.

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

**Output**

You have to print at the console the **leftmost biggest triple** as sequence of up to 3 numbers, separated by a space.

**Constraints**

* The **input numbers** will be integers in range [-1000 … 1000].
* The number of the input numbers **n** will be between 1 and 1000.
* Allowed work time for your program: 0.1 seconds.
* Allowed memory: 16 MB.

**Examples**

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

**Problem 3 – Wine Glass**

Bulgarians are famous for being enchanted by the magic of the red wine. Its magic is very powerful and yet unpredictable. Some people report being struck by a memory loss charm, others lose control over their speech, to others it acts like a love potion. You’re asked to help the Bulgarians by printing a few of their beloved magical wine glasses for them.

**Input**

The input data should be read from the console.

* You have an integer number **N** (always **even number**) specifying the height of the wine glass.

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 wine glass on the console following the examples below.

* The glass has exactly **N rows**, each of which contains exactly **N symbols**.
* The first row should contain the backslash (“**\**”) symbol, a total of (**N-2**) asterisks (“**\***”) and the slash (“**/**”) symbol.
* The second row should contain exactly one dot (”**.**”) before the backslash, one after the slash and two less (compared to the row above) asterisks between the slash and backslash.
* The third row should contain one more dot at each side and two less asterisks and so on, until the (**N /2**) row, where there should be no asterisks between the slashes.
* On the next **(N/2)-2 rows, if N >= 12** or **(N/2)-1 rows, if N < 12,** you should print the stem that should look like the following: a count of **(N/2)-1** dots (“**.**”), followed by two vertical lines (“**|**”) and **(N/2)-1** dots after the lines. The remaining one or two rows (up to a total count of N) should be filled with exactly **N** dashes (“-”) on each row.

**Constraints**

* The number **N** will be an **even** integer between 4 and 60, inclusive.
* Allowed working time for your program: 0.1 seconds.
* Allowed memory: 16 MB.

**Examples**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Input** | **Output** |  | **Input** | **Output** |  | **Input** | **Output** |
| 6 | \\*\*\*\*/  .\\*\*/.  ..\/..  ..||..  ..||..  ------ | 8 | \\*\*\*\*\*\*/  .\\*\*\*\*/.  ..\\*\*/..  ...\/...  ...||...  ...||...  ...||...  -------- | 12 | \\*\*\*\*\*\*\*\*\*\*/  .\\*\*\*\*\*\*\*\*/.  ..\\*\*\*\*\*\*/..  ...\\*\*\*\*/...  ....\\*\*/....  .....\/.....  .....||.....  .....||.....  .....||.....  .....||.....  ------------  ------------ |

**Problem 4 – Longest Alphabetical Word**

Nakov enjoys playing with words. Recently he invented the following puzzle game. He starts by given word **w** (e.g. "**softwareuniversity**") and he fills a square block of size **n**\***n** (e.g. n=7) with this word as many times as it fits, from left to right and from up to down (see the example on the right). It is also called Nakov's square block of word **w** and size **n**.

softwar

euniver

sitysof

tw**a**reun

iversit

ysoftwa

reunive

Nakov defines an **alphabetical word** as a sequence of letters, where each letter is alphabetically after its previous letter in the word. For example, "abc", "fo" and "aeou" are alphabetical words, but "zabc", "srevi" and "ntaeou" are not.

Now Nakov wants to find the **longest alphabetical word** in the obtained square block. The word can start anywhere in the square block and can run in left, right, up or down direction and cannot go outside of the square block. In our example, if we start from row 3 and column 2 in our 7 x 7 square block, we find the following alphabetical words: "**aw**" (left direction), "**ar**" (right direction), "**at**" (up direction) and "**aeou**" (down direction).

Write a program that reads a word **w** and a number **n** and finds the **longest alphabetical word** in Nakov's square block of word **w** and size **n**. If more than one longest alphabetical words exist in the block, find the smallest of them in the standard lexicographical order.

**Input**

The input data should be read from the console. The input data consists of **exactly two lines**:

* The first line will hold the word **w**.
* The second line will hold the size **n**.

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

**Output**

You have to print at the console the **longest alphabetical word**.

**Constraints**

* The word **w** will be a non-empty string, consisting of lower Latin letters, up 1000.
* The size of the square **n** will be an integer value in the range [1…50].
* Allowed work time for your program: 0.25 seconds.
* Allowed memory: 16 MB.

**Examples**

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Input** | **Output** | **Block** |  | **Input** | **Output** | **Block** |  | **Input** | **Output** | **Block** |
| softwareuniversity  7 | aeou | softwar  euniver  sitysof  tw**a**reun  iv**e**rsit  ys**o**ftwa  re**u**nive | alpha  6 | ahp | alphaa  l**pha**al  phaalp  haalph  aalpha  alphaa | java  3 | aj | **j**av  **a**ja  vaj |

**Problem 5 – Bit Shooter**

We are given a **bit field** in the form of **64-bit integer number**. We **shoot it 3 times**. Each shoot has a **center** and a **size**. The shoot damages **size** bits around the shoot **center** (makes these bits 0). Finally, the bit field is split into left side (bits 63 … 32) and right side (bits 31 … 0). Write a program that calculates **how many bits survive** (have value 1) after the shoots in the **left** side and in the **right** side of the bit field. The bits are numbered as traditionally in programming: from right to left from 0 to 63.

**Input**

The input data should be read from the console. It will consist of exactly 4 lines:

* At the first line you will have a 64-bit integer, corresponding to the bit field.
* At each of the next 3 lines we have 2 numbers: shoot **center** and shoot **size** – integers, split by a 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 consists of exactly 2 lines:

* The first line print "**left: …**" and the number of alive bits in the left side.
* The second line print "**right: …**" and the number of alive bits in the right side.

**Constraints**

* The **bit field** will be a 64-bit integer in the range [0 … 18 446 744 073 709 551 615].
* The values for the **center** will be integers will be integers in range [0 … 63].
* The values for the **size** will be **odd** integers in range [1 … 99].
* Allowed working time for your program: 0.25 seconds.
* Allowed memory: 16 MB.

**Examples**

|  |  |
| --- | --- |
| **Input** | **Output** |
| 4227378815862876842  1 5  22 3  58 7 | left: 10  right: 11 |
| **Comments** | |
| 4227378815862876842(10) = 0011101010101010101010101010001010101010100001001010101010101010 🡪 001110101010101010101010101000101010101010000100101010101010**0000** 🡪 0011101010101010101010101010001010101010**000**001001010101010100000 🡪 00**0000000**0101010101010101010001010101010000001001010101010100000 🡪  00000000001010101010101010100010 | 10101010000001001010101010100000 🡪 left: 10; right: 11 | |

|  |  |
| --- | --- |
| **Input** | **Output** |
| 9223372036854775806  15 7  43 15  58 1 | left: 15  right: 24 |