# Problem 2 – Fun with Matrices

Let's have some fun! You are given a 4x4 matrix of numbers. At the console you'll receive a **start value**, a **step** and a number of **operations to perform**. The start value shows the value of cell [0, 0]. Moving from left to right along each row you obtain each cell's value by incrementing the previous cell's value by the step.

Then, you'll receive some commands from the console in one of the following formats:

* **"[row] [col] multiply [num]"** – upon receiving this command you should multiply the value of the specified cell with coordinates [row, col] by [num];
* **"[row] [col] sum [num]"** – add the value of [num] to the value of the specified cell;
* **"[row] [col] power [num]"** – raise the value of the cell to the power [num];
* **"Game Over!"** – denotes the end of input.

After the game is over, you need to check the sum of cell values on each row, column and the two diagonals and print on the console one of the following (whichever has the **largest sum**):

* **"ROW[index] = sum"**
* **"COLUMN[index] = sum"**
* **"LEFT-DIAGONAL = sum"**
* **"RIGHT-DIAGONAL = sum"**

The index is the **index** of the row/column with biggest sum and is surrounded with square brackets at the output, **sum** is the sum itself. **Left-diagonal** is the diagonal starting at [0, 0] and ending on [3, 3], **right-diagonal** is the diagonal starting at [3, 0] and ending at [0, 3].

## Input

* The input data should be read from the console.
* The first input line holds **the start number**.
* The second input line holds **the step**.
* On the next lines you'll receive **orders in the described formats,** ending with the string "Game Over!"
* 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 you must print the **row, column or diagonal with the largest sum and the sum rounded to two digits after the decimal sign**. Use dot ("**.**") as the decimal separator.
* In case there are two equal values **priority** is as follows: rows (upper-most), columns (left-most), left diagonal, right diagonal.

## Constraints

* All numbers are in the range [±5.0 × 10−324 … ±1.7 × 10308].
* The **[row]** and **[col]** indexes will be valid, i.e. between [0 … 3].
* Allowed working time for your program: 0.1 seconds. Allowed memory: 16 MB.

## Examples

|  |  |  |
| --- | --- | --- |
| **Input** | **Comments** | **Output** |
| 5.3  -2.09  2 3 multiply -0.8  Game Over! | |  |  |  |  |  | | --- | --- | --- | --- | --- | |  | 0 | 1 | 2 | 3 | | 0 | 5.3 | 3.21 | 1.12 | -0.97 | | 1 | -3.06 | -5.15 | -7.24 | -9.33 | | 2 | -11.42 | -13.51 | -15.6 | **-17.69 => 14.152** | | 3 | -19.78 | -21.87 | -23.96 | -26.05 | | ROW[0] = 8.66 |

|  |  |  |
| --- | --- | --- |
| **Input** | **Output** | |
| 55  0.1  0 0 power -2.4  2 1 multiply 0  0 2 sum 777  3 1 power 2.4  Game Over! | ROW[3] = 16062.62 | |
|  |  | |
| **Input** | **Output** | | |
| -2444411  500000  2 2 power 3  2 3 multiply -0.05  2 0 multiply 1.11  0 0 sum 13.99  Game Over! | COLUMN[2] = 16690641554532500000.00 | | |
|  |  | | |
| **Input** | **Output** |
| 0.01  0.01  0 0 multiply 0.01  0 1 multiply -0.01  0 2 multiply 0.01  0 3 multiply 0.01  1 2 sum -10  2 0 sum -10  3 3 sum -10  2 2 sum -10  1 1 sum -10  Game Over! | ROW[0] = 0.00 |