## Problem 1 – Torrent Pirate

Captain Jack Sparrow is a famous pirate. He loves to steal different stuff just for fun and he loves watching movies. He recently discovered a brand new technology called peer-to-peer or torrent. After he browsed a famous site, he made a **collection of movies** he would like to download. Assume 1 movie has a size of **1500MB**. Jack doesn’t want to pay for the internet, so he decided to go to the mall and use the free Wi-Fi there. The Wi-Fi has a fixed speed of **2MB/s**. Unfortunately for Jack, his wife will be going with him to the mall and this means that the download would not be free at all. She likes to buy sandals and other useless stuff. You are given the **money** his wife spends **per hour** at the mall.

Your task is to help Jack **calculate** whether it is **cheaper** to go to the mall and download the movies or go to the cinema to watch them. If the amount is the same, Jack still wants to make his wife happy, so he goes to the mall.

### Input

The input data should be read from the console. It consists of three input values, each at a separate line:

* Download data **d**: how much **megabytes** in total Jack should download.
* Price of cinema **p**: how much **money would cost** Jack to go to the cinema to watch one movie.
* Wife spending **w**: how much **money per hour** does Jack’s wife spend.

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 **“{place to go} -> {price to pay}lv”**.
* The **price to pay** should be formatted with 2 digits after the decimal sign.

### Constraints

* **d** is an integer number in range [0...2,147,483,647]. **p** is an integer number in range [0…30]. **w** is an integer number in range [0…200].
* Allowed working time for your program: 0.25 seconds.
* Allowed memory: 16 MB.

### Examples

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Input** | **Output** | | **Comments** | | |
| 30000  5  50 | cinema -> 100.00lv | | **Download time** =  = *(download data)/(fixed speed)/(seconds)/(minutes)* =  = 30000 / 2 / 60 / 60 = **4.1667** hours in the mall  **Price for download** = *(download time)\*(wife spending)* =  = 4.1667\*50 = **208.34**lv  **Number of movies downloaded** =  = *(download data)/(movie size)* = 30000/1500 = **20** movies  **Cinema price** =  = *(number of movies)\*(cinema price)* = 20\*5lv = **100**lv  Result on the console: **cinema -> 100lv** | | |
| **Input** | | **Output** | | **Input** | **Output** |
| 30000  15  50 | | mall -> 208.33lv | | 30000  15  72 | mall -> 300.00lv |

## Problem 2 – Basket Battle

Simeon likes to play a special basket game with Nakov called Basket Battle. The rules are very simple. Every player tries to score a basket from a different distance and if he succeeds, he wins a certain amount of points (based on the distance he shot from). You will receive the distance from which every player tries to score and the information whether the shot was successful or not.

The players decide who will start shooting first .The game is played in several rounds. Each round consists of the two players shooting. After **each** round, the players **switch turns** (if Simeon was first in the first round, he is shooting second in the second round). A player **wins** if he reaches **500 points**. **If someone reaches 500 points, the game stops and your program should break and print the output.**

A player **can’t** make more than **500 points** in the game. For example if a player has 450 points and he scores successfully 90 points, the player stays with 450 points after that round. You must help Simeon and Nakov calculate their points and determine the winner with a computer program. Example:

Simeon <- The player who starts shooting first.

3 <- The number of possible rounds.

300 <- Simeon tries to score 300 points.  
success <- Simeon succeeds and scores.

200 <- Nakov tries to score 200 points.

fail <- Nakov fails and still has 0 points.

400 <- Nakov tries to score 400 points. (New round starts and players switch turns)

success <- Nakov succeeds and scores.

200 <- Simeon tries to score 200 points.  
success <- Simeon succeeds and scores.

The game has ended since Simeon has scored a total of 500 points and wins the game.

### Input

The input data should be read from the console. It consists of three input values, each at a separate line:

* The first line holds a string **F** – the name of the player that starts shooting first in the first round
* The second line holds an integer **N** – the number of rounds in the game
* For each round you will receive an input **P** - the amount of points every player tries to score and the string **I** - information about whether the shot was successful or not (each input will be on 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 data should be printed on the console. You have 3 cases:

1. If there is a **winner** (someone reaches 500 points), your output should consist of **three lines**:

* On the first line you should print the name of the winner.
* On the second line you should print in which round the player won the game.
* On the third line you should print the points of the player who lost the game

1. If **no one won** the game and the players have the **same score**, you should print out **two lines**:

* On the first line you should print the text: "DRAW"
* On the second line you should print the points that the players have.

1. If **no one wins** the game and the players have **different amount of points** your output should consist of **two lines**:

* On the first line you should print the name of the player with more points.
* On the second line you should print the difference between the points of the players.

### Constraints

* **F** will be a string, either "**Simeon**" or "**Nakov**".
* **N** will be an integer number in the range [1...20].
* **P** will be integer in the range [1…500].
* **I** will be a string, either "**success**" or "**fail**".
* Allowed working time for your program: 0.1 seconds. Allowed memory: 16 MB.

### Examples

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Input** | **Output** |  | **Input** | **Output** |  | **Input** | **Output** |
| Simeon  3  300  success  200  fail  400  success  200  success | Simeon  2  400 | Simeon  2  250  success  250  success  150  fail  100  fail | DRAW  250 | Nakov  3  150  success  100  fail  150  success  50  success  150  success  70  success | Nakov  130 |

# Problem 3 – Dumbbell

As we all know programmers are not bodybuilders and vice versa. But there is one exception. His name is Marcho Zukerberov. At the moment he is very busy learning how to code in the well-known language “C Diez” and he has only 30 minutes per day for training. Help Marcho break the rule of becoming an ordinary and overweight programmer by making him a **dumbbell** with the weight needed (**N kg**) for his “gouemi ruki chuek”.

### Input

The input data should be read from the console.

On the only input line you have an integer number **N**, showing the **height** of the dumbbell.

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 **number of lines** should be equal to the height **N** of the dumbbell.

**Each line** should hold exactly **N** **x 3** symbols: "**.**" (dot) , "**\***" (asterisk), "=" (equal sign) or “&” (ampersand).

The **visible part** of the dumbbell **bar** should be exactly **N** symbols.

### Constraints

* The number **N** will always be an **odd** integer number in the range [5…39].
* Allowed working time for your program: 0.25 seconds.
* Allowed memory: 16 MB.

### Examples

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Input** | **Output** |  | **Input** | **Output** |
| 5 | ..&&&.....&&&..  .&\*\*&.....&\*\*&.  &\*\*\*&=====&\*\*\*&  .&\*\*&.....&\*\*&.  ..&&&.....&&&.. |  | 7 | ...&&&&.......&&&&...  ..&\*\*\*&.......&\*\*\*&..  .&\*\*\*\*&.......&\*\*\*\*&.  &\*\*\*\*\*&=======&\*\*\*\*\*&  .&\*\*\*\*&.......&\*\*\*\*&.  ..&\*\*\*&.......&\*\*\*&..  ...&&&&.......&&&&... |

# Problem 4 – Encrypted matrix

Bogi and Acho love to play with numbers, so they invented a game to encrypt a message and create a diagonal matrix with it. Your task is to write a program, which helps Acho and Bogi convert a message into a number, encrypt it and fill a matrix with it.

*First* you should **convert** the message into a **number**. This is done when you take the **last digit** of the **ASCII code** of each character in the message and **add** those digits **next to each other**.

For example the string **"Soft Uni"** is converted to **31262505:**  
(**'S'** => ASCII(8**3**), **'o'** => ASCII(11**1**), **'f'** => ASCII(10**2**), **'t'** => ASCII(11**6**), **' '** => ASCII(3**2**), **'U'** => ASCII(8**5**), **'n'** => ASCII(11**0**), **'i'** => ASCII(10**5**)).

*Then* you should **encrypt** the converted number **digit by digit**. The formula is the following: If the digit is **even or '0'** - you should **multiply it by itself**. If the digit is **odd** – you should **add** to its value the **neighboring digits**. If there is a missing neighboring digit, you should add 0 instead of it.

If the **result** after the encrypting of a digit **is a number with two digits, you should concatenate** the result to the new number.

For example **31262505** is encrypted to **464364705:  
3** => 3+0+1=**4**, **1** => 1+3+2=**6**, **2** => 2\*2=**4**, **6** => 6\*6=**36**, **2** => 2\*2=**4**, **5** => 5+2+0=**7**, **0** => 0\*0=**0**, **5** => 5+0+0=**5**.

*Finally* you should fill a square diagonal matrix with the encrypted number. The size of the matrix should be the same as the number of digits in the encrypted number. The diagonal to be filled comes from the console as a character: **'\' (backslash)** represents the **main diagonal**; **'/' (slash)** means the **anti-diagonal**.

For example **464364705** is filled in the following two matrices:

| **\ (main diagonal)** | **/ (anti-diagonal)** |
| --- | --- |
| **4** 0 0 0 0 0 0 0 0  0 **6** 0 0 0 0 0 0 0  0 0 **4** 0 0 0 0 0 0  0 0 0 **3** 0 0 0 0 0  0 0 0 0 **6** 0 0 0 0  0 0 0 0 0 **4** 0 0 0  0 0 0 0 0 0 **7** 0 0  0 0 0 0 0 0 0 **0** 0  0 0 0 0 0 0 0 0 **5** | 0 0 0 0 0 0 0 0 **5**  0 0 0 0 0 0 0 **0** 0  0 0 0 0 0 0 **7** 0 0  0 0 0 0 0 **4** 0 0 0  0 0 0 0 **6** 0 0 0 0  0 0 0 **3** 0 0 0 0 0  0 0 **4** 0 0 0 0 0 0  0 **6** 0 0 0 0 0 0 0  **4** 0 0 0 0 0 0 0 0 |

Take note of the **direction** of the number in the anti-diagonal matrix.

### Input

The input data should be read from the console.

On the first input line you have a string, containing the message.

On the second input line you have the direction as character: either **'\'** or **'/'**.

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.

You must print the matrix with a single space between the elements.

### Constraints

* The string length will be in the range [1-500].
* Allowed working time for your program: 1 second.
* Allowed memory: 16 MB.

### Examples

| **Input** | **Output** |  | **Input** | **Output** |  | **Input** | **Output** |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Soft  \ | 4 0 0 0 0  0 6 0 0 0  0 0 4 0 0  0 0 0 3 0  0 0 0 0 6 |  | Soft Uni  / | 0 0 0 0 0 0 0 0 5  0 0 0 0 0 0 0 0 0  0 0 0 0 0 0 7 0 0  0 0 0 0 0 4 0 0 0  0 0 0 0 6 0 0 0 0  0 0 0 3 0 0 0 0 0  0 0 4 0 0 0 0 0 0  0 6 0 0 0 0 0 0 0  4 0 0 0 0 0 0 0 0 |  | S  / | 3 |

# Problem 5 – Game of Bits

Vasko likes to play with odd and even numbers as well as bits. He has to make a game using bits but he really enjoys the course of Web Fundamentals so he doesn't have time to make the game. Please help him!

You have a 32-bit integer and commands: "Odd", "Even" or "Game Over!" When you read the "Odd" command you have to obtain a new number by extracting the values of all odd bit positions in the current number (positions are counted from right to left and the first bit has a position of 1). When you read the "Even" command you have to extract the bits at even positions. When you read the command "Game Over!" you must print on the console the count of bits with value '1' in the final number.

# Input

The input data should be read from the console. On the first line, you are given an integer number and on each of the next lines, you have an issued command.

The possible commands are as follows: "**Odd**" and "**Even**". On the last input line, you are given the order "**Game Over!**" which means that the game has ended.

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

# Output

On the only output line you should print the final number before the “Game over” command and the count of bits with value 1. The output format is as follows:

"***<final number after bit’s extraction> -> <number of bits with value 1>***"

# Constraints

* The **input number** will be a 32-bit integer in the range [0 … 4 294 967 295].
* The minimum number of commands is 1.
* The maximum number of commands will be 30.
* Allowed working time: 0.1 seconds.
* Allowed memory: 16 MB.

# Examples

|  |  |  |
| --- | --- | --- |
| **Input** | **Output** | **Comments** |
| 3586153387  Odd  Even  Game Over! | 232 -> 4 | 11010101110000000101101110101011(3586153387) → Odd → 1111100011010001 (63697) →  Even → 11101000(232) → Game Over! → 4 |