# Problem 1 – Raindrops

Практически упражнения към курса [**"Programming Fundamentals" за ученици**](https://github.com/BG-IT-Edu/School-Programming/tree/main/Courses/Applied-Programmer/Programming-Fundamentals).

Тествайте задачата в judge: [https://judge.softuni.bg/Contests/2675](https://judge.softuni.bg/Contests/2675/Практически-изпит-10-12-2017)

The **Raindear Forecast Agency** (**RFA**) is an organization founded by an old and kind grandma which wanted quality forecasts. The Agency has hired you to write a software which finds the Rain Coefficient, by calculating simple input data.

You will receive **N**, an integer – the **amount** of **regions**. Then you will receive the **density** – a floating-point number.

For **each region**, you will receive an input line in the following format:

“{raindropsCount} {squareMeters}”

The raindropsCount and the squareMeters will be integers. Your task is to **calculate** **the regional coefficient** by the following formula: raindropsCount / squareMeters

**NOTE**: The **regional coefficient** **should** be a **floating-point number**.

Your task is to **sum all regional coefficients**, and then **divide** it by the density, and **print** the **result**.   
If a **division** is **not possible**, just print the **sum** of **all regional coefficients**.

## Input

* On the **first input line** you will receive **N** – the **amount** of **regions**.
* On the **second input line** you will receive the **density**.
* On the **next N input lines** you will receive **information** about the **regions**.

## Output

* As output you must print the sum of all regional coefficients divided by the density.
* If a division is not possible you must print the sum of all regional coefficients.
* The output should be **rounded** and **printed** to **3 places** after the **decimal point**.

## Constraints

* The **amount** of **regions** – **N** will be an **integer** in **range [0, 100]**.
* The **density** will be a **floating-point number** in **range [0, 9]**.
* The **raindropsCount** will be an **integer** in **range [-231, 231]**.
* The **squareMeters** will be an **integer** in **range [1, 10000]**.
* Allowed working **time** / **memory**: **100ms** / **16MB**.

## Examples

|  |  |  |
| --- | --- | --- |
| **Input** | **Output** | **Comment** |
| 4  4  2000 10  1000 5  5000 2000  3000 30 | 125.625 | 2000 / 10 = 200  1000 / 5 = 200  5000 / 2000 = 2.5  3000 / 30 = 100  200 + 200 + 2.5 + 100 = 502.5  502.5 / 4 = 125.625 |
| 2  2  100000 50  200000 25 | 5000.000 | 100000 / 50 = 2000  200000 / 25 = 8000  2000 + 8000 = 10000  10000 / 2 = 5000  (rounded till 3rd symbol) = 5000.000 |

# Problem 2 – Rainer

A Rainer is like a runner but in Rain. One who runs from the Rain. Donald is one good Rainer and he created a game where he dodges raindrops at lightning fast speed through some incomprehensible logic.

You will receive a **sequence** of **integers** – each of those integers, **except** the **last one**, **form** the **game field**.   
You must take the **last integer** from that sequence – that is the **initial index** at which **Donald steps**.

The game goes so – you must **decrease all** of the **integers** in the **sequence’ values** by **1**.   
Then you must **read** an **integer** – the **next index** at which **Donald steps**.   
You must **repeat** these steps until **Donald** gets **wet**.

If an integer **reaches 0**, that means a **raindrop** has **fallen there**. If **Donald** is **on that position**, he gets **wet**.

If an integer **reaches 0**, and **Donald** is **not there**, you must **return** the **integer** to its **original value**. (**initial** value)

When **Donald** gets **wet**, the **program ends**, and you must print the **current** **sequence** of **integers**, and the **count** of **steps** **Donald has made** (the **initial** index **does not count** as a step)

## Input

* On the **first input line** you will get the **sequence** of **integers**, **separated** by **spaces**.
* On the **next several input lines** you will be **getting integers** – the **indexes**.

## Output

* As output you must print the **sequence of integers**, **separated** by **spaces**, on one line.
* Then you must print the **steps Donald has made** on the **second line**.

## Constraints

* The **count** of the **integers** in the **sequence** will be **[3, 100]**.
* The **integers** in the **sequence** will be in **range [2, 100]**.
* The **indexes** that will be **given** to you will **always** be **valid** and **inside** the **sequence**.
* Allowed working **time** / **memory**: **100ms** / **16MB**.

## Examples

|  |  |  |
| --- | --- | --- |
| **Input** | **Output** | **Comment** |
| 5 2 3 4 5 3  0  1  4  1  1 | 4 0 0 2 4  5 | Sequence – 5 2 3 4 5, Initial Index – 3  We decrease all by 1, Sequence – 4 1 2 3 4  We check if Donald is on an element 0. He is not, so we read next step. Index – 0. Steps – 1.  Sequence – 3 0 1 2 3. There is an element with value 0, but Donald is not there, we return it to its original value (2).  Sequence – 3 2 1 2 3. Index – 1. Steps – 2.  Sequence – 2 1 3 1 2. Index – 4. Steps – 3.  Sequence – 1 2 2 4 1. Index – 1. Steps – 4.  Sequence – 5 1 1 3 5. Index – 1. Steps – 5.  We decrease by 1, and it gets 4 0 0 2 4. Donald is on Index 1 – which is currently 0. He dies. No other steps are made, and the program ends. |
| 2 3 4 5 6 2  1  2  3  4  0 | 0 0 2 4 0  5 |  |

# Problem 3 – Raincast

The Raindear Forecast Agency has hired you again, astonished by your previous works. This time you are hired to write a software which receives Telegram Raincasts, and validates them. The messages are quite scrambled so you only have to find the valid ones.

You will begin **receiving input lines** which may contain **any ASCII character**. Your task is to find the **Raincasts**.

The **Valid Raincast** consists of **3 lines**:

* Type: {type}
* Source: {source}
* Forecast: {forecast}

The type should either be “Normal”, “Warning” or “Danger”.  
The source should consist of **alphanumeric characters**.  
The forecast should **not contain** any of the following characters: ‘!’, ‘.’, ‘,’, ‘?’.

* When you **find** a type, you must **search** for a source.
* When you **find** a source you must **search** for a forecast.
* When you **find** a forecast, you have **completed** a **single Valid Raincast**. You must **start** **searching** for a type again, for the **next Raincast**.

There might be **invalid lines** **between** the **valid ones**. You should **keep** the **order** of **searching**.

**NOTE**: The **valid input lines** must be **exactly** in the format specified above. **Any difference** makes them **invalid**.

When you receive the command “Davai Emo”, the **input ends**. You must print **all valid raincasts** you’ve found, each in a **specific format**, each on a **new line**.

## Input

* The input will come in several input lines which may contain any ASCII character.
* The input ends when you receive the command “Davai Emo”.

## Output

* As output you must **print all** of the **valid raincasts** you’ve found, **each** on a **new line**.
* The **format** is: ({type}) {forecast} ~ {source}

## Constraints

* The input lines may contain **any ASCII character**.
* There will be **no more** than **100 input lines**.
* Allowed working **time** / **memory**: **100ms** / **16MB**.

## Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| Type: Normal  Source: JohnKutchur9  Forecast: A full rain program no sun  Type: Danger  Forecast: Invalid Input Line  Source: IvoAndreev  Type: Invalid Input Line  Forecast: Shte vali qko  Davai Emo | (Normal) A full rain program no sun ~ JohnKutchur9  (Danger) Shte vali qko ~ IvoAndreev |
| Forecast: Bau  Source: Myau  Type: Strong  Source: Good  Forecast: Valid  Type: Warning  Type: Danger  Source: Emo  Forecast: Nqma da se kefim mn na praznici  Davai Emo | (Warning) Nqma da se kefim mn na praznici ~ Emo |

# Problem 4 – RainAir

Before naming it RyanAir … Tony Ryan named it RainAir, because the day he named it, it was really rainy, and he liked rain. Anyways, you have been hired by Tony, to create a software which manipulates data about flights and customers. The future of RyanAir is in your hands.

You will receive input lines in one of the following formats:

* {customerName} {customerFlight1} {customerFlight2} {customerFlight3} ...
* {customerName} = {customer2Name}

The customerName is a string. The customerFlights are integers.

If you receive a customerName and customerFlights, you should **add the customer** and **the flights** to the customer.   
If the customer **already exists**, just **add** the **new flights** to him.

If you receive a customerName and customer2Name, you should **make** the **1st customer**’s flights **equal** to the **2nd customer**’s flights.

The input ends when you receive the command “I believe I can fly!”. When that happens, you must **print all customers**, **ordered** by **count** of **flights** in **descending order**, and then by **alphabetical order**.

The **flights** must be ordered in **ascending order**.

## Input

* The input consists of several input lines in the format specified above.
* The input ends when you receive the command “I believe I can fly!”.

## Output

* As output you must print all the customers ordered in the way specified above.
* The format is: #{customerName} ::: {flight1}, {flight2}, {flight3}...

## Constraints

* There will be **no invalid input lines**.
* The customerName is a string which may contain **any ASCII characters** **except** ‘ ’ (space) and ‘=’.
* The customerFlight is an integer in **range [0, 10000]**.
* There will be **no non-existent** customerNames in the commands that require customerNames.
* If **all data ordering fails**, you should **order** the data by **order** **of** **input**.
* Allowed working **time** / **memory**: **100ms** / **16MB**.

## Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| Donald 1549 4592 3945 111  Prakash 111 45  Gibbs 492 502  Isacc 204 544  I believe I can fly! | #Donald ::: 111, 1549, 3945, 4592  #Gibbs ::: 492, 502  #Isacc ::: 204, 544  #Prakash ::: 45, 111 |
| Prakash 111 134 2451 232  Sony 222  Prakash 555  Stamat 111  Stamat = Sony  I believe I can fly! | #Prakash ::: 111, 134, 232, 555, 2451  #Sony ::: 222  #Stamat ::: 222 |

## Министерство на образованието и науката (МОН)

* Настоящият курс (презентации, примери, задачи, упражнения и др.) е разработен за нуждите на Национална програма "**Обучение за ИТ кариера**" на МОН за подготовка по професия "Приложен програмист".



* Курсът е базиран на учебно съдържание и методика, предоставени от **фондация "Софтуерен университет"** и се разпространява под **свободен** **лиценз CC-BY-NC-SA** (Creative Commons Attribution-Non-Commercial-Share-Alike 4.0 International).

