# Problem 1 – Wormtest

The worms are having a contest – a Wormtest. In the Wormtest, every worm is given wormpoints depending on several statistics. Guess who’s going to calculate those statistics... Yup, that’s you!

You will be given input data about a single wormtestant. You must calculate his wormpoints, depending on the given data.

On the **first** line of input you will get the worm’s **length** in **meters** (**m**), which will be an **integer**.

On the **second** line of input you will get the worm’s **width**, in **centimeters** (**cm**), which will be a **floating-point** **number**.

You should **convert** the **length** in **centimeters** (**multiply** it by **100**). Then you should **divide** the **length** and the **width** and find the **remainder** of the **division**.

If it is **zero** or **cannot be calculated**, you should **print** the **product** of **the length** and **the width** (**length** \* **width**), **rounded** to the **second digit** after the **decimal point**.

If the **remainder** is **NOT zero**, you should print what **percentage** is **the length** of **the width**.   
Print it **rounded** to the **second** **digit** after the **decimal point**.

**Example**: length = 1m ; width = 30cm. percentage = 333.33 %.

### Input

* On the **first** input line you will receive the worm’s **length** in **meters** (**m**).
* On the **second** input line you will receive the worm’s **width** in **centimeters** (**cm**).

### Output

* As output you must print the product of the **length** and the **width** or how much **percent** is the **length** of the **width**.
* **Both** output results should be **rounded** and **printed** to the **second digit** after the **decimal point**.

### Constrains

* The worm’s **length** will be a valid **integer** in **range [0, 1000]**.
* The worm’s **width** will be a **floating-point number** in **range [0, 1000.00]**.

### Examples

|  |  |  |
| --- | --- | --- |
| **Input** | **Output** | **Comments** |
| 1000  200 | 20000000.00 | **length = 1000m. width = 200cm.**  **1000m \* 100 = 100000cm**.  **100000 % 200 = 0**. The **remainder** is **zero**. So we print the **product** of the two numbers, **rounded** to **the second digit** after the **decimal point**. |
| 40  600 | 666.67% | **40m \* 100 = 4000cm.**  **4000 % 600 = 400**. The **remainder** is **400**. So we print how much **percent** the **length** is of the **width**, **rounded** to **the second digit** after the **decimal point**.  **4000** is **666.67 percent** of **600**. |

# Problem 2 – Worm Ipsum

Dr. Worm has decided to create an AI which generates random sentences. However, in the worm language, there are no words which have 2 equal characters. The Doctor does not know how to program the AI to follow these rules so he asks you for help.

You will receive several input lines containing **sentences**. They represent the random data rendered by the AI. Your task is to **validate every input line** and **process the words** inside the sentence.   
The sentences will **ALWAYS** **start** with a **capital letter** and **end** with a **dot** “.”, because the AI is simple.

**IGNORE** all input lines which contain **more** than **1 sentence**.   
The **STRUCTURE** of all sentences will be **grammatically correct**.

You must then pass through each **VALID** input line, and go through the **words** in its **sentence**. If you find a **word** which has **2** or **more** **equal characters**, you must pick the **character** with the **MOST OCCURENCES** in that word.  
Then you must **REPLACE** every character in the word with **that** **character**. Example: “hello” = “lllll”.

Upon reading an input line, you should **validate** and **process** it. Then you should **print** the **result** of the processing. **Only after** **that** you should **read** the **next** **input** **line**.

The input **ends** when you **receive** the **command** “Worm Ipsum”.

### Input

* The input consists of several input lines containing one or more sentences.
* The input sequence **ends** when you **receive** the **command** “Worm Ipsum”.

### Output

* As output you should print **every valid** input line, **after** you’ve **processed** it.

### Constrains

* All sentences **WILL** be **grammatically correct**.
* There might be input lines with **2** or **more sentences** on them.

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| This is a random sentence.  This is another random sentence.  You have downloaded data.  Worm Ipsum | This is a random eeeeeeee.  This is another random eeeeeeee.  You have dddddddddd aaaa. |
| Hello World.  How are you.  What are you doing. What will you do.  Will you come.  Worm Ipsum | lllll World.  How are you.  llll you come. |

# Problem 3 – Wormhole

Dr. Worm has advanced technologically to level where he can travel through wormholes. Wormholes are holes in reality, which allow you to travel to other dimensions. Dr. Worm however, does not know anything about the wormholes, so he just enters random holes in hopes of getting to his target point.

You will be given a **sequence** of **integers** – the **wormholes**, on a **single input line**, **separated** by a **space**. The **integers** will either be **zeros** (**0**) or some **other integers**.

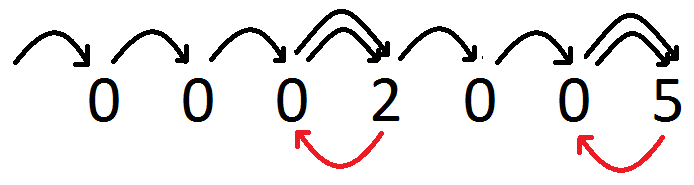
You must start from the first one and reach the last one.   
If the **integer** you are **currently at** is a **zero**, you must **continue forward** to the **next element** of the sequence.   
If the **integer** you are **currently at** is **NOT** **zero**, it is a **wormhole**. You must **go** to the **element** which is **at index** – thewormhole’s **value**.

**Every time** you **step** on a **wormhole**, you must **set** its **value** to **0**.

Your task is to **calculate** exactly **how many steps** it would take you to **reach** the **end** of the **sequence**.

The wormhole’s **teleportation** to the **target** index **does NOT count** as a **step**.

**Example**: If we have the input “0 0 0 2 0 0 5” . . . the **steps** would look like this:



### Input

* The input will come in the form of a **single input line**, containing a **sequence** of **integers**, **separated** by a **single** **space**.

### Output

* As **output** you must print **how many steps** it took you, to **reach** the **end** of the **sequence**.

### Constrains

* The **count** of **elements** in the input will be in **range [0, 100]**.
* The **integers** in the input will be in **range [0, N - 1]**, where **N** is the **count** of **elements**.

### Examples

|  |  |  |
| --- | --- | --- |
| **Input** | **Output** | **Comments** |
| 0 0 0 2 0 0 5 | 9 | You start walking trough the elements. You make **4 steps** and you step on a **wormhole**. You **teleport** at **index** (**2**), and you set the wormhole’s value to **0**. You make 4 more steps and you reach the next wormhole, which returns you to **index** (**5**). One last step and you have **reached** the **end** of the **sequence**.  The result is **9 steps**. |
| 0 0 1 0 2 0 4 0 | 13 |

# Problem 4 – Worms World Party

Worms World Party is a game where teams of worms fight each other with various weapons. Your task is to register the teams and the participants with their score, and sort out the teams by the total amount of their score.

You will receive several input lines in the following format:

{wormName} -> {teamName} -> {wormScore}

The wormName and teamName will be **strings**, consisting of **alphanumeric** **characters** and **spaces**.   
The wormScore will be an **integer**.   
Your task is to **store** every **team** and its **worms** with their **score**.   
If the **given** worm **already** **exists** in one of the **teams**, you should **IGNORE** that line of input.

When you receive the command “quit” the input sequence ends. When that happens, you must order all teams and print them.

The teams must be **ordered** **by total score** of their worms, in **descending order**, and if 2 teams have the **same total score**, they must be ordered by average score (**totalScore / wormCount**) in **descending order**. For each team you must print its worms, **ordered** by **descending order** of their **score**.

### Input

* The input comes in the form of input lines in the format, specified above.
* The input sequence ends when you receive the command “quit”.

### Output

* As output you must print all the teams and their worms in the following order:  
  “1. Team: {team1Name} – {totalScore}  
   ###{worm1Name} : {worm1Score}  
   ###{worm2Name} : {worm2Score}  
   . . .  
   2. Team: {team2Name} – {totalScore}  
   . . .”

### Constrains

* The wormName and teamName will consist of **alphanumeric characters** and **spaces**.
* The wormScore will be a valid integer in **range [0, 1.000.000.000]**.
* There will be **NO invalid** input lines.

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| Donald -> TheBest -> 3  Jefrey -> Overlords -> 2  Johnathan -> Overlords -> 3  George -> Psychos -> 5  Alex -> TheBest -> 7  quit | 1. Team: TheBest - 10  ###Alex : 7  ###Donald : 3  2. Team: Psychos - 5  ###George : 5  3. Team: Overlords - 5  ###Johnathan : 3  ###Jefrey : 2 |
| Tony Tony Chopper -> Straw Hats -> 10  Brook -> Straw Hats -> 5  Loffy -> Fake Hats -> 10  Brook -> Fake Hats -> 5  quit | 1. Team: Straw Hats - 15  ###Tony Tony Chopper : 10  ###Brook : 5  2. Team: Fake Hats - 10  ###Loffy : 10 |