01. Resurrection

You ever heard of Phoenixes? Magical Fire Birds that are practically immortal – they reincarnate from an egg when they die. Naturally, it takes time for them to reincarnate. You will play the role of a scientist who calculates the time to reincarnate for each phoenix, based on its body parameters.

You will receive **N**, an **integer** – the **amount** of **phoenixes**.
For each **phoenix**, you will **receive 3 input lines**:

- On the **first input line** you will receive an **integer** – the **total length** of the **body** of the phoenix.
- On the **second input line** you will receive a **floating-point number** – the **total width** of the **body** of the phoenix.
- On the **third input line** you will receive an **integer** – the **length** of **1 wing** of the phoenix.

For each phoenix, you must **print** the **years** it will take for it to **reincarnate**, which is **calculated** by the following formula:

The **`totalLength` powered** by **2**, **multiplied** by the **sum of the `totalWidth`** and the **`totalWingLength`** (**`2 * wingLength`**).

```
totalYears = {totalLength} ^ 2 * ({totalWidth} + 2 * {wingLength})
```

Input

- On the **first input line** you will receive **N**, an **integer** – the **amount** of **phoenixes**.
- On the **next N * 3 input lines** you will be receiving **data** for **each phoenix**.

Output

- As output, you must print the **total years needed for reincarnation** for each phoenix.
- Print each phoenix's years **when you've calculated** them.
- Print each phoenix's years **on a new line**.

Constrains

- The **amount** of **phoenixes** will be an **integer** in **range [0, 1000]**.
- The **total length** of the **body** of the **phoenix** will be an **integer** in **range [-2^{31}, 2^{31}]**.
- The **total width** of the **body** of the **phoenix** will be a **floating-point number** in **range [-2^{31}, 2^{31}]**.
- The **total width** of the **body** of the **phoenix** will have up to **20 digits** after the **decimal point**.
- The **total length** of the **wing** of the **phoenix** will be an **integer** in **range [-2^{31}, $2^{31} - 1$]**.
- The **total years** is a **product** of **integers** and **floating-point numbers**, thus it is a **floating-point number**.
- The **total years** should have the **same accuracy** as the **total width**.
- Allowed working time / memory: **100ms / 16MB**.

Input	Output	Comments
2 100 50 30 150 25 10	1100000 1012500	**2 phoenixes:** **P1:** Body length: 100 Body width: 50 Length of 1 wing: 30 Total years: 100 ^ 2 * (50 + 2 * 30) = 1100000 **P2:** Body length: 150 Body width: 25 Length of 1 wing: 10 Total years: 150 ^ 2 * (25 + 2 * 10) = 1012500
2 100 50.243 31 154 23.132 11	1122430.000 1070350.512	**2 phoenixes:** **P1:** Body length: 100 Body width: 50.243 Length of 1 wing: 31 Total years: 100 ^ 2 * (50.243 + 2 * 31) = 1122430.000 **P2:** Body length: 154 Body width: 23.132 Length of 1 wing: 11 Total years: 154 ^ 2 * (23.132 + 2 * 11) = 1070350.512

02. Icarus

Icarus is the majestic phoenix who has been alive from the beginning of creation. Icarus travels through different planes. When Icarus travels through a plane, he damages Reality itself with his overwhelming, beyond godlike flames. You will receive a **sequence** of **integers** – the **plane**. After that you will receive **1 integer** – an **index** in that **sequence**, which is Icarus's **starting position**. Icarus's **INITIAL DAMAGE** is **1**. You will then begin **receiving commands** in the following format: "**`{direction} {steps}`**". The direction will be either "**`left`**" or "**`right`**", and the **steps** will be an **integer**. Depending on the direction, Icarus must **step** through the sequence of **integers to the left** or **right**. Each time he **steps** on a **NEW position**, he **damages** it. In other words, he **SUBTRACTS** his **current damage from** the **integer** at **that position**. Walking left and right has its conditions though:

- If Icarus **passes beyond** the **start** of the **sequence** (**`index: -1`**) while going **left**, he must go at the **end** of the **sequence** (**`index: length - 1`**).
- If Icarus **passes beyond** the **end** of the **sequence** (**`index: length - 1`**) while going **right**, he must go at the **start** of the **sequence** (**`index: 0`**).

If **1** of the **2 cases stated above** happens, Icarus **increments** his **damage** by **1**. The input ends when you receive the command "**Supernova**". When that happens you must print what is **left** of the **sequence**.

Input

- On the **first input line** you will get the **sequence** of **integers**, **separated** by **spaces**.
- On the **second input line** you will get Icarus's **starting position**.
- On the **next several input lines** you will get the **commands**.

Output

- As output you must print a **single line** containing the **remaining elements** of the **sequence**, **separated** by **spaces**.

Constrains

- The **integers** in the **sequence** will be in **range [0, 1000]**.
- The **initial position** of Icarus will **always** be **valid** and **inside** the **sequence's indexes**.
- The **direction** will always be either "**`left`**" or "**`right`**".
- The **steps** will be in **range [0, 1000]**.
- There will be **NO invalid** input lines.
- Allowed working time / memory: **100ms / 16MB**.

Examples

Input	Output	Comments
50 50 25 50 50 3 left 2 right 2 left 2 right 2 Supernova	50 48 21 48 50	Initial index: 3 Initial state: 50 50 25 50 50 Go left 2 steps: 50 50 24 50 50 50 49 24 50 50 Go right 2 steps: 50 49 23 50 50 50 49 23 49 50 Go left 2 steps: 50 49 22 49 50 50 48 22 49 50 Go right 2 steps: 50 48 21 49 50 50 48 21 48 50 Final state: 50 48 21 48 50
5 3 5 5 5 2 left 5 left 5 Supernova	2 0 0 0 0	Initial index: 2 Initial state: 5 3 5 5 5 Go left 5 steps: 5 2 5 5 5 4 2 5 5 5 4 2 5 5 3 4 2 5 3 3 4 2 3 3 3 Go left 5 steps: 4 0 3 3 3 2 0 3 3 3 2 0 3 3 0 2 0 3 0 0 2 0 0 0 0 Final state: 2 0 0 0 0

03. Phoenix Grid

The Phoenix Grid is an ancient artifact created by the Linguistics miracle – Mozilla, The "Fire Bird". It is used to translate Phoenix language. You are the newest scientist, researching the Grid and as the research team was almost out of hope, you came up with the genius idea to use Regular Expressions! You saved the day! You are a Hero!

You will begin **receiving encoded messages**. You must **CHECK** each **one** of **them** and if it's a **VALID**.

A **valid encoded message** consists of **one phrase** or **more phrases**, separated by **DOTS** ('**.**').

- A **phrase** consists of exactly **3 characters**.
- A **phrase CANNOT** contain **whitespace** characters or the '**_**' (**underscore**) character.

Valid messages: "**asd.dsa**", "**123.312**", "**3@a.231**", "**111**", "**@sd**", "**132.31$.ddd**" . . .

Invalid messages: "**123asdasd.dsa**", "**_@a. sd**", "**a.s.d**" . . .

When you have found a valid message, you must **check** if it a **PALINDROME** – if it reads the same backward as forward.

Palindrome messages: "**asd.dsa**", "**123.321**", "**cat.php.tac**" . . .

If the **message** is **VALID** and is a **PALINDROME** print "**YES**". In any other case, print "**NO**".

The input ends when you receive the command "**ReadMe**".

Input

- As input you will receive several input lines containing encoded messages.

Output

- As output you must print **for each message** "**YES**" or "**NO**" if its **valid** or **not**.

Constrains

- The input lines may contain **any ASCII character**.
- There will be no more than **1000 input lines**.
- Allowed working time / memory: **100ms / 16MB**.

Examples

Input	Output
asd asd.asd asd.dsa 123.323.321 _ds._sad.sds jss.csh.php.hsc.ssj ReadMe	NO NO YES YES NO YES
asa igi.igi ___.___ . sds.dsd.sds.dsd.sds.dsd.sds xha.ahx ReadMe	YES YES NO NO YES YES

04. CODE: Phoenix Oscar Romeo November

The fire creatures are assembling in squads to fight The Evil Phoenix God. You have been tasked to determine which squad is the strongest, so it will be sent as The Vanguard. You will begin receiving input lines containing information about fire creatures in the following format:

`{creature} -> {squadMate}`

The `creature` and the `squadMate` are **strings**. You should store every **creature**, and his **squad mates**. If the **creature** already **exists**, you should **add** the **new squad mate** to it.

- If there is **already** a **squad mate** with the **given name** in the **given creature's squad**, **IGNORE** that **line** of **input**.
- If the **given squad mate name** is the **same** as the **given creature**, **IGNORE** that **line** of **input**.

The **input sequence ends** when you receive the command "`Blaze it!`".

When that happens you must **print** the **creatures ordered** in **descending** order by **count** of **squad mates**. Sounds simple right? But there is one little **DETAIL**.

If a particular `creature` has a `squadMate`, and that `squadMate` has that `creature` in his `squadMates`, you **should NOT consider** them as **part** of the **count** of **squad mates**.

Example:

Creature 1: **Mozilla** -> {Tony, Dony, Mony}

Creature 2: **Tony** -> {Mozilla, Franzilla, Godzilla}

Mozilla has **2 squad mates** in total, because **Tony** also has **Mozilla** in his **squad mates**.

Tony has **2 squad mates** in total, because **Mozilla** also has **Tony** in his **squad mates**.

Input

- As input you will receive several input lines containing information about the fire creatures.
- The input sequence ends when you receive the command "`Blaze it!`".

Output

- As output you must print each of the creatures the following information: **{creature} : {countOfSquadMates}**
- As it was stated above, mind the **count** of **squad mates**. If **2 creatures** have themselves in their **squad mates**, they should **NOT** be **counted**.

Constrains

- The `creature` and the `squadMate` will be **strings** which may contain **any ASCII character**.
- There will be **NO invalid** input lines.
- Allowed time / memory: **100ms / 16MB**.

Examples

Input	Output
Mozilla -> Tony Tony -> Godzilla Mozilla -> Dony Tony -> Franzilla Mozilla -> Mony Tony -> Mozilla Blaze it!	Mozilla : 2 Tony : 2
FireBird -> FireMane Phoenix -> FireVoid FireVoid -> FireMane FireSnow -> FireMane Phoenix -> FireBird FireMane -> FireBird FireMane -> FireVoid Phoenix -> FireSnow FireMane -> FireSnow FireMane -> FireMane Phoenix -> FireMane Phoenix -> FireVoid Blaze it!	Phoenix : 4 FireBird : 0 FireVoid : 0 FireSnow : 0 FireMane : 0

05. Anonymous Downsite

The Anonymous informal group of activists have hacked a few commercial websites and the CIA has hired you to write a software which calculates the losses. Based on the given data, use the appropiate data types.

You will receive **2 input lines** – each containing an **integer**.

- The **first** is **N** – the **number** of **websites** which are down.
- The **second** is the **`security key`**.

On the **next N lines** you will receive **data** about **websites** in the following format:
`{siteName} {siteVisits} {siteCommercialPricePerVisit}`

You must **calculate** the **`site loss`** by the following formula: **`siteVisits * siteCommercialPricePerVisit`**

When you **finish reading all data**, you must print the **affected sites' names** – each on a **new line**.
Then you must print the **`total money loss`** – **sum** of all **`site loss`**, on a **new line**.
Finally you must print the **`security token`**, which is the **`security key`**, **POWERED** by the **COUNT** of **affected sites**.

Input

- On the **first input line** you will get **N** – the **count** of **affected websites**.
- On the **second input line** you will the **security key**.
- On the **next N input lines** you will get **data** about the **websites**.

Output

- As output you must print **all affected websites' names** – **each** on a **new line**.
- **After** the **website names** you must print the **total loss** of **data**, printed to the **20th digit** after the **decimal point**. The format is "**`Total Loss: {totalLoss}`**".
- Finally you must **print** the **security token**. The format is "**`Security Token: {securityToken}`**".

Constrains

- The integer **N** will be in **range [0, 100]**.
- The **security token** will be in **range [0, 10]**.
- The **website name** may contain any **ASCII character** except **whitespace**.
- The **site visits** will be an **integer** in **range [0, 2^{31}].**
- The **price per visit** will be a **floating point number** in **range [0, 100]** and will have **up** to **20 digits** after the decimal point.
- Allowed working **time/memory**: **100ms / 16MB**.

Examples

Input	Output
3 8 www.google.com 122300 94.23233 www.abv.bg 2333 11 www.kefche.com 12322 23.3222	www.google.com www.abv.bg www.kefche.com Total Loss: 11837653.10740000000000000000 Security Token: 512
1 1 www.facebook.com 100000 10.45	www.facebook.com Total Loss: 1045000.00000000000000000000 Security Token: 1

06. Anonymous Threat

The Anonymous have created a cyber hypervirus which steals data from the CIA. You, as the lead security developer in CIA, have been tasked to analyze the software of the virus and observe its actions on the data. The virus is known for his innovative and unbeleivably clever technique of merging and dividing data into partitions.

You will receive a **single input line** containing **STRINGS** separated by **spaces**.
The strings may contain **any ASCII** character except **whitespace**.

You will then begin receiving commands in one of the following formats:

- `merge {startIndex} {endIndex}`
- `divide {index} {partitions}`

Every time you receive the **`merge command`**, you must merge all elements from the **`startIndex`**, till the **`endIndex`**. In other words, you should concatenate them.
Example: `{abc, def, ghi} -> merge 0 1 -> {abcdef, ghi}`

If **any** of the **given indexes** is **out of the array**, you must take **ONLY** the **range** that is **INSIDE** the **array** and **merge** it.

Every time you receive the **`divide command`**, you must **DIVIDE** the **element** at the **given index**, into **several small substrings** with **equal length**. The **count** of the **substrings** should be **equal** to the **given partitions**.

Example: `{abcdef, ghi, jkl} -> divide 0 3 -> {ab, cd, ef, ghi, jkl}`

If the string **CANNOT** be **exactly divided** into the **given partitions**, **make all partitions except** the **LAST** with **EQUAL LENGTHS**, and make the **LAST one** – **the LONGEST**.

Example: `{abcd, efgh, ijkl} -> divide 0 3 -> {a, b, cd, efgh, ijkl}`

The **input ends** when you receive the command "**`3:1`**". At that point you must print the **resulting elements**, **joined** by a **space**.

Input

- The **first input line** will contain the **array** of **data**.
- On the **next several input** lines you will **receive commands** in the **format specified above**.
- The **input ends** when you receive the command "**`3:1`**".

Output

- As output you must print a single line containing the elements of the array, **joined** by a **space**.

Constrains

- The **strings** in the **array** may contain any **ASCII character** except **whitespace**.
- The **`startIndex`** and the **`endIndex`** will be in **range [-1000, 1000]**.
- The **`endIndex`** will **ALWAYS** be **GREATER** than the **`startIndex`**.
- The **`index`** in the **`divide command`** will **ALWAYS** be **INSIDE** the array.
- The **`partitions`** will be in **range [0, 100]**.
- Allowed working **time/memory**: **100ms / 16MB**.

Examples

Input	Output
Ivo Johny Tony Bony Mony merge 0 3 merge 3 4 merge 0 3 3:1	IvoJohnyTonyBonyMony
abcd efgh ijkl mnop qrst uvwx yz merge 4 10 divide 4 5 3:1	abcd efgh ijkl mnop qr st uv wx yz

07. Anonymous Vox

The Anonymous's main communication channel is based on encoded messages. The CIA has targetted that channel, assuming that it holds sensitive information. You have been hired to decode and break their internal com. system.

You will receive an input line containing a **single string** – the **encoded text**. Then, on the **next line** you will receive several values in the following format: "**`{value1}{value2}{value3}...`**".

You must find the **encoded placeholders** in the **text** and **REPLACE** each one of them with the **value** that corresponds to its **index**.
Example: **placeholder1 – value1**, **placeholder2 – value2** etc. There may be **more values** than **placeholders** or **more placeholders** than **values**.

The **placeholders** consist of 3 blocks **`{start}{placeholder}{end}`**. The **start** should consist only of **English alphabet letters**. The **placeholder** may contain **ANY ASCII** character. The **end** should be **EXACTLY EQUAL** to the **start**. The idea is that you have to find the **placeholders**, and **REPLACE** their **`placeholder`** block with the **value** at that **index**.

Example Placeholders: "**`a.....a`**", "**`b!d!b`**", "**`asdxxxxxasd`**", "**`peshogoshopesho`**"...

You **must ALWAYS** match the placeholder with the **LONGEST `start`** and the **RIGHTMOST `end`**. For example if you have "**`asddvdasd`**" you should **NOT** match "**`dvd`**" as a placeholder, you should match "**`asddvdasd`**".

At the end you must **print** the **result text**, after you've **replaced** the **values**.

Input

- On the **first input line** you will receive the **encoded text**.
- On the **second input line** you will receive the **placeholders**.

Output

- As output you must print a **single line** containing the **resulting text**, after the replacing of values.

Constrains

- The **given text** may contain **ANY ASCII** character.
- The **given values** may contain **ANY ASCII** character except '**{**' and '**}**'.
- The **given values** will **AWLAYS** follow the format specified above.
- Allowed working **time/memory**: **100ms / 16MB**.

Examples

Input	Output
`Hello_mister,_Hello` `{ Jack }`	`Hello Jack Hello`
`ASD___asdfffasd` `{this}{exam}{problem}{is}{boring}`	`ASD___asdthisasd`
`Whatsup_ddd_sup` `{Dude}`	`WhatsupDudesup`
`HeypalHey_____how_ya_how_doin_how` `{first}{second}`	`HeyfirstHey_____howsecondhow`

08. Anonymous Cache

The Anonymous are storing data on their dataservers about their activities. The CIA has higher the greatest hacker in the world – You. Your job is to extract their data and send it to the CIA. It won't be an easy task, Get Ready!

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

- **`{dataSet}`**
- **`{dataKey} -> {dataSize} | {dataSet}`**

The **`dataSet`** and **`dataKey`** are both strings. The **`dataSize`** is an **integer**. The **`dataSets`** hold **`dataKeys`** and their **`dataSizes`**.

If you receive only a **`dataSet`** you should **add** it. If you receive a **`dataKey`** and a **`dataSize`**, you should add them to the **given `dataSet`**.

And here's where the fun begins. If you receive a **`dataKey`** and a **`dataSize`**, but the given **`dataSet` does NOT exist**, you should **STORE** those **keys** and **values** in a **`cache`**. When the corresponding **`dataSet`** is **added**, you should **check** if the **`cache`** holds any **keys** and **values** referenced to it, and you should **add** them to the **`dataSet`**.

You should end your program when you receive the command "**`thetinggoesskrra`**". At that point you should extract the **`dataSet`** from the **`data`** with the **HIGHEST `dataSize`** (**SUM** of all its **`dataSizes`**), and you should print it.

NOTE: Elements in the **`cache`**, **should be CONSIDERED NON-EXISTANT**. You should **NOT** count them in the **final output**.

In case there are **NO `dataSets`** in the **`data`**, you **should NOT do anything**.

Input

- The input comes in the form of commands in one of the formats specified above.
- The input ends when you receive the command "**`thetinggoesskrra`**".

Output

- As output you must print the **`dataSet`** with the **HIGHEST SUM** of all **`dataSizes`**.
- The output format is:

  `Data Set: {dataSet}, Total Size: {sumOfAllDataSizes}`

  `$.{dataKey1}`

  `$.{dataKey2}`

- In case there are **NO `dataSets`** in the **`data`**, print **nothing**.

Constrains

- The **`dataSet`** and **`dataKey`** are **both strings** which may contain **ANY ASCII** character except ' ', '**-**', '**>**', '**|**'.
- The **`dataSize`** is a **valid integer** in **range [0, 1.000.000.000]**.
- There will be **NO invalid input lines**.
- There will be **NO `dataSets`** with **EQUAL SUMMED `dataSize`**.
- There will be **NO DUPLICATE** keys.
- Allowed working **time/memory**: **100ms / 16MB**.

Examples

Input	Output
Users BankAccounts ADDB444 -> 23111 | BankAccounts Students -> 2000 | Users Workers -> 24233 | Users thetinggoesskrra	Data Set: Users, Total Size: 26233 $.Students $.Workers
Cars Car1 -> 233333 | Cars Car23 -> 266666 | Cars Warehouse2 -> 10000 | Buildings Warehouse3 -> 480000 | Buildings Warehouse5 -> 100000 | Buildings Buildings thetinggoesskrra	Data Set: Buildings, Total Size: 590000 $.Warehouse2 $.Warehouse3 $.Warehouse5

09. Raindrops

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 [-2^{31}, 2^{31}]**.
- 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

10. 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	

11. 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

12. 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