Problem 1 - Guinea Pig

Problem for exam preparation for the Programming Fundamentals Course @SoftUni.
Submit your solutions in the SoftUni judge system at https://judge.softuni.org/Contests/Practice/Index/2031#0.

Merry has a guinea pig named Puppy, that she loves very much. Every month she goes to the nearest pet store and buys him everything he needs – food, hay, and cover.

On the **first three lines**, you will receive **the quantity of food**, **hay**, and **cover**, which Merry buys for a **month (30 days)**. On the **fourth line**, you will receive the **guinea pig's weight**.

Every day Puppy eats **300 gr of food**. **Every second** day Merry **first feeds the pet**, then gives it a **certain amount of hay equal to 5%** of the rest of the **food**. On **every third** day, Merry puts Puppy **cover** with **a quantity of 1/3** of its **weight**.

Calculate whether the quantity of **food, hay, and cover**, will be enough for a **month**.

If Merry runs out of food, hay, or cover, stop the program!

Input

- **On the first line – quantity food in kilograms** - a floating-point number in the range **[0.0 – 10000.0]**
- **On the second line – quantity hay in kilograms** - a floating-point number in the range **[0.0 – 10000.0]**
- **On the third line – quantity cover in kilograms** - a floating-point number in the range **[0.0 – 10000.0]**
- **On the fourth line – guinea's weight in kilograms** - a floating-point number in the range **[0.0 – 10000.0]**

Output

- If the food, the hay, and the cover are enough, print:
  - `"Everything is fine! Puppy is happy! Food: {excessFood}, Hay: {excessHay}, Cover: {excessCover}."`
- If one of the things is not enough, print:
  - `"Merry must go to the pet store!"`

The output values must be formatted to the second decimal place!

Examples

Input	Output
10 5 5.2 1	Everything is fine! Puppy is happy! Food: 1.00, Hay: 1.10, Cover: 1.87.
You receive food – **10000**, hay – **5000**, cover – **5200**, weight – **1000** (in grams). On the first day, Merry gives Puppy 300gr food – 9700gr food left. On the second day, the food left is **9400gr**, so the needed hay is **9400 * 5% = 470**, and the hay left is **4530.**	

On the third day, the cover left is **4866.67,** and the food left is **9100**, and so on. On the last day, Merry has: food – 1.00, hay – 1.10, and cover – 1.87.	
1 1.5 3 1.5	Merry must go to the pet store!
9 5 5.2 1	Merry must go to the pet store!

JS Examples

Input	Output
(["10", "5", "5.2", "1"])	Everything is fine! Puppy is happy! Food: 1.00, Hay: 1.10, Cover: 1.87
(["1", "1.5", "3", "1.5"])	Merry must go to the pet store!
(["9", "5", "5.2", "1"])	Merry must go to the pet store!

Problem 2 - Shopping List

Problem for exam preparation for the Programming Fundamentals Course @SoftUni.
Submit your solutions in the SoftUni judge system at https://judge.softuni.org/Contests/Practice/Index/2031#1.

It's the end of the week, and it is time for you to go shopping, so you need to create a shopping list first.

Input

You will receive an **initial list** with groceries separated by an exclamation mark **"!"**.

After that, you will be receiving **4 types** of commands until you receive **"Go Shopping!"**.

- **"Urgent {item}"** - **add** the item at the **start** of the list. If the item **already exists,** skip this command.
- **"Unnecessary {item}"** - **remove** the item with the given name, only **if it exists** in the list. Otherwise, skip this command.
- **"Correct {oldItem} {newItem}"** - if the item with the given **old name** exists, **change** its name with the **new** one. Otherwise, skip this command.
- **"Rearrange {item}"** - if the grocery exists in the list, **remove** it from its **current position** and **add** it at the **end** of the list. Otherwise, skip this command.

Constraints

- There won't be any duplicate items in the initial list

Output

- Print the **list** with all the groceries, joined by **", "**:
  "{firstGrocery}, {secondGrocery}, … {nthGrocery}"

Examples

Input	Output
Tomatoes!Potatoes!Bread Unnecessary Milk Urgent Tomatoes Go Shopping!	Tomatoes, Potatoes, Bread

Input	Output
Milk!Pepper!Salt!Water!Banana Urgent Salt Unnecessary Grapes Correct Pepper Onion Rearrange Grapes	Milk, Onion, Salt, Water, Banana

Correct Tomatoes Potatoes Go Shopping!	

JS Examples

Input	Output
(["Tomatoes!Potatoes!Bread", "Unnecessary Milk", "Urgent Tomatoes", "Go Shopping!"])	Tomatoes, Potatoes, Bread

Input	Output
(["Milk!Pepper!Salt!Water!Banana", "Urgent Salt", "Unnecessary Grapes", "Correct Pepper Onion", "Rearrange Grapes", "Correct Tomatoes Potatoes", "Go Shopping!"])	Milk, Onion, Salt, Water, Banana

Problem 3 - Heart Delivery

Problem for exam preparation for the Programming Fundamentals Course @SoftUni.
Submit your solutions in the SoftUni judge system at https://judge.softuni.org/Contests/Practice/Index/2031#2.

Valentine's day is coming, and Cupid has minimal time to spread some love across the neighborhood. Help him with his mission!

You will receive a **string** with **even integers,** separated by a **"@"** - this is our neighborhood. After that, a series of **Jump** commands will follow until you receive **"Love!"**. Every house in the neighborhood needs a certain number of **hearts** delivered by Cupid so it can celebrate Valentine's day. The integers in the neighborhood indicate those needed hearts.

Cupid starts at the position of the **first house** (index 0) and must jump by a **given length.** The jump commands will be in this format: **"Jump {length}"**.

Every time he jumps from one house to another, the needed hearts for the visited house are **decreased by 2**:

- If the needed hearts for a certain house become **equal to 0**, print on the console **`"Place {house_index} has Valentine's day."`**
- If **Cupid** jumps to a house where the needed hearts are **already 0,** print on the console **`"Place {house_index} already had Valentine's day."`**
- Keep in mind that **Cupid** can have a **larger jump length** than the **size of the neighborhood,** and if he does jump **outside** of it, he should **start** from the **first house** again (index 0)

For example, we are given this neighborhood: 6@6@6. Cupid is at the start and jumps with a length of 2. He will end up at index 2 and decrease the needed hearts by 2: [6, 6, 4]. Next, he jumps again with a length of 2 and goes outside the neighborhood, so he goes back to the first house (index 0) and again decreases the needed hearts there: [4, 6, 4].

Input

- On the first line, you will receive a **string** with **even integers** separated by **"@"** – the neighborhood and the number of hearts for each house.
- On the next lines, until **`"Love!"`** is received, you will be getting jump commands in this format: **`"Jump {length}"`**.

Output

In the end, print **Cupid's last position** and whether his mission was successful or not:

- **`"Cupid's last position was {last_position_index}."`**
- If **each house** has had Valentine's day, print:
  - **`"Mission was successful."`**
- If **not,** print the **count** of all houses that **didn't** celebrate Valentine's Day:
  - **`"Cupid has failed {houseCount} places."`**

Constraints

- The **neighborhood's** size will be in the range [1…20]
- Each **house** will need an **even number** of hearts in the range [2 … 10]
- Each **jump length** will be an integer in the range [1 … 20]

Examples

Input	Output	Comments
10@10@10@2 Jump 1 Jump 2 Love!	Place 3 has Valentine's day. Cupid's last position was 3. Cupid has failed 3 places.	Jump 1 ->> [10, 8, 10, 2] Jump 2 ->> [10, 8, 10, 0] so we print "Place 3 has Valentine's day." The following command is "Love!" so we print Cupid's last position and the outcome of his mission.
2@4@2 Jump 2 Jump 2 Jump 8 Jump 3 Jump 1 Love!	Place 2 has Valentine's day. Place 0 has Valentine's day. Place 0 already had Valentine's day. Place 0 already had Valentine's day. Cupid's last position was 1. Cupid has failed 1 places.	

JS Examples

Input	Output	Comments
[("10@10@10@2", "Jump 1", "Jump 2", "Love!"])	Place 3 has Valentine's day. Cupid's last position was 3. Cupid has failed 3 places.	Jump 1 ->> [10, 8, 10, 2] Jump 2 ->> [10, 8, 10, 0] so we print "Place 3 has Valentine's day." The following command is "Love!" so we print Cupid's last position and the outcome of his mission.
(["2@4@2", "Jump 2", "Jump 2", "Jump 8", "Jump 3", "Jump 1", "Love!"])	Place 2 has Valentine's day. Place 0 has Valentine's day. Place 0 already had Valentine's day. Place 0 already had Valentine's day. Cupid's last position was 1. Cupid has failed 1 places.	