Mid Exam Preparation – 14 June 2023

1. SoftUni Reception

Link: https://judge.softuni.org/Contests/Practice/Index/2474#0

Every day, thousands of students pass by the reception at SoftUni with different questions to ask. The employees have to help everyone by providing all the information and answering all of the questions.

Three employees are working on the reception all day. Each of them can handle a **different number of students per hour**. Your task is to **calculate how much time** it will take to **answer all the questions** of a given number of students.

First, you will receive 3 lines with integers, representing the number of students that each **employee can help per hour.** On the following line, you will receive **students count as a single integer**.

Every fourth hour, all employees have a break, so they don't work for an hour. It is the only break for the employees, because they don't need rest, nor have a personal life. Calculate the time needed to answer all the student's questions and print it in the following format: **`"Time needed: {time}h."`**

Input / Constraints

- On the first three lines - **each employee efficiency** - integer in the range **[1 - 100]**
- On the fourth line - **students count** – integer in the range **[0 – 10000]**
- Input will always be valid and in the range specified

Output

- Print a single line: **`"Time needed: {time}h."`**
- Allowed working **time** / **memory**: **100ms** / **16MB**

Examples

Input	Output	Comment
5 6 4 20	`Time needed: 2h.`	All employees can answer 15 students per hour. After the first hour, there are 5 students left to be answered. All students will be answered in the second hour.
1 2 3 45	`Time needed: 10h.`	All employees can answer **6** students per hour. In the first 3 hours, they have answered **6 * 3 = 18 students**. **Then they have a break for an hour.** After the **next 3 hours,** there are 18 + 6 * 3 = **36 answered students**. After the break for an hour, there are only 9 students to answer. So in the **10th hour,** all of the student's questions would be answered.

`3`
`2`
`5`
`40`
	`Time needed: 5h.`	

2. Shoot for the Win

Link: https://judge.softuni.org/Contests/Practice/Index/2305#1

Write a program that helps you keep track of your **shot targets**. You will receive a **sequence with integers**, separated by a single space, representing targets and their value. Afterward, you will be receiving indices until the **"End"** command is given, and you need to print the **targets** and the **count of shot targets**.

Every time you receive an **index**, you need to shoot the target on that index, **if it is possible**.

Every time you **shoot a target**, its value becomes **-1, and it is considered shot**. Along with that, you also need to:

- **Reduce** all the other **targets**, which have **greater values** than your **current** target, **with its value**.
- **Increase** all the other **targets**, which **have less than or equal** value to the **shot target**, **with its value.**

Keep in mind that you can't shoot a target, which is already shot. You also can't increase or reduce a target, which is considered shot.

When you receive the **"End"** command, print the targets in their current state and the **count of shot targets** in the following format:

`"Shot targets: {count} -> {target1} {target2}… {targetn}"`

Input / Constraints

- On the **first line** of input, you will receive a **sequence** of **integers**, **separated** by **a single space – the targets sequence**.
- On the **following lines**, until the **"End"** command, you be receiving **integers** each on a single line – **the index of the target to be shot.**

Output

- The format of the output is described above in the problem description.

Examples

Input	Output	Comments

`24 50 36 70`
`0`
`4`
`3`
`1`
`End`
	`Shot targets 3 -> -1 -1 130 -1`	First, we shoot the target on index 0. It becomes equal to -1, and we start going through the rest of the targets. Since 50 is more than 24, we reduce it to 26 and 36 to 12 and 70 to 46. The sequence looks like that:
		-1 26 12 46
		The following index is invalid, so we don't do anything. Index 3 is valid, and after the operations, our sequence should look like that:
		-1 72 58 -1

		Then we take the first index with value 72, and our sequence looks like that: **-1 -1 130 -1** Then we print the result after the **"End"** command.

`30 30 12 60 54 66`
`5`
`2`
`4`
`0`
`End`
	`Shot targets: 4 -> -1`
	`120 -1 66 -1 -1`	

3. Heart Delivery

Link: 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.`	