Programming Fundamentals with Python: Exam Preparation

01. SoftUni Reception

Submit your solutions in the SoftUni judge system at 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.`	

JS 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.`	

02. Treasure Hunt

Submit your solutions in the SoftUni judge system at https://judge.softuni.org/Contests/Practice/Index/1773#1.

The pirates need to carry a treasure chest safely back to the ship, looting along the way.

Create a program that **manages** the **state** of the **treasure chest** along the way. On the **first line,** you will receive the **initial loot** of the treasure chest, which is a **string** of **items** separated by a **"|"**.

"{loot$_1$}|{loot$_2$}|{loot$_3$} … {loot$_n$}"

The following lines represent commands **until "Yohoho!"** which ends the treasure hunt:

- **"Loot {item$_1$} {item$_2$}…{item$_n$}"**:
  - Pick up treasure loot along the way. Insert the items at the **beginning** of the chest.
  - If an item is **already** contained, **don't** insert it.
- **"Drop {index}"**:
  - **Remove** the loot at the given **position** and **add** it at the **end** of the treasure chest.
  - If the index is **invalid,** skip the command.

- **"Steal {count}"**:
  - Someone steals the **last count** loot items. If there are **fewer items** than the given count, **remove as much** as there are.
  - Print the stolen items separated by **", "**:
    "{item1}, {item2}, {item3} … {itemn}"

In the end, output the **average treasure gain,** which is the **sum** of all treasure items **length** divided by the **count** of all items inside the chest **formatted** to the **second decimal** point:

"Average treasure gain: {averageGain} pirate credits."

If the chest is **empty,** print the following message:

"Failed treasure hunt."

Input

- On the **1st line,** you are going to receive the **initial treasure chest** (**loot** separated by **"|"**)
- On the following **lines**, until **"Yohoho!"**, you will be receiving commands.

Output

- Print the output in the **format described above**.

Constraints

- The **loot items** will be strings containing any ASCII code.
- The **indexes** will be integers in the range [**-200**…**200**]
- The **count** will be an integer in the range [**1**….**100**]

Examples

Input	**Output**
Gold|Silver|Bronze|Medallion|Cup Loot Wood Gold Coins Loot Silver Pistol Drop 3 Steal 3 Yohoho!	Medallion, Cup, Gold Average treasure gain: 5.40 pirate credits.
Comments	
The first command **"Loot Wood Gold Coins"** adds **Wood** and **Coins** to the chest but **omits** Gold since it is already contained. The chest now has the following items: **Coins Wood Gold Silver Bronze Medallion Cup** The **second** command adds **only Pistol** to the chest The **third** command **"Drop 3"** removes the **Gold** from the chest, but immediately adds it at the **end**:	

`Pistol Coins Wood Silver Bronze Medallion Cup Gold` The **fourth** command **"Steal 3"** removes the **last 3** items **Medallion**, **Cup**, **Gold** from the chest and prints them. In the end calculate the average treasure gain which is the sum of all items length Pistol(**6**) + Coins(**5**) + Wood(**4**) + Silver(**6**) + Bronze(**6**) = **27** and **divide** it by the count 27 / 5 = **5.4** and format it to the **second decimal** point.

Input	Output

`Diamonds|Silver|Shotgun|Gold`
`Loot Silver Medals Coal`
`Drop -1`
`Drop 1`
`Steal 6`
`Yohoho!`
	`Coal, Diamonds, Silver, Shotgun, Gold, Medals`
	`Failed treasure hunt.`

JS Examples

Input	Output

`(["Gold|Silver|Bronze|Medallion|Cup",`
`"Loot Wood Gold Coins",`
`"Loot Silver Pistol",`
`"Drop 3",`
`"Steal 3",`
`"Yohoho!"])`
	`Medallion, Cup, Gold`
	`Average treasure gain: 5.40 pirate credits.`

Comments	
The first command **`"Loot Wood Gold Coins"`** adds **Wood** and **Coins** to the chest but **omits** Gold since it is already contained. The chest now has the following items: **`Coins Wood Gold Silver Bronze Medallion Cup`** The **second** command adds **only Pistol** to the chest The **third** command **"`Drop 3`"** removes the **Gold** from the chest, but immediately adds it at the **end**: **`Pistol Coins Wood Silver Bronze Medallion Cup Gold`** The **fourth** command **"Steal 3"** removes the **last 3** items **Medallion**, **Cup**, **Gold** from the chest and prints them. In the end calculate the average treasure gain which is the sum of all items length Pistol(**6**) + Coins(**5**) + Wood(**4**) + Silver(**6**) + Bronze(**6**) = **27** and **divide** it by the count 27 / 5 = **5.4** and format it to the **second decimal** point.	

Input	Output

`(["Diamonds|Silver|Shotgun|Gold",`
`"Loot Silver Medals Coal",`
`"Drop -1",`
`"Drop 1",`
`"Steal 6",`
`"Yohoho!"])`
	`Coal, Diamonds, Silver, Shotgun, Gold, Medals`
	
	`Failed treasure hunt.`

03. Moving Target

Submit your solutions in the SoftUni judge system at https://judge.softuni.org/Contests/Practice/Index/2305#2.

You are at the shooting gallery again, and you need a program that helps you keep track of moving targets. On the first line, you will receive a **sequence of targets with their integer values**, split by a **single space**. Then, you will start receiving **commands for manipulating the targets** until the **"End"** command. The commands are the following:

- **"Shoot {index} {power}"**
  - Shoot the target at the index **if it exists** by **reducing** its **value** by the **given power** (**integer value**).
  - Remove the target **if it is shot**. A target is considered **shot** when **its value reaches 0**.
- **"Add {index} {value}"**
  - Insert a target with the received value at the received **index if it exists**.
  - If not, print: **"Invalid placement!"**
- **"Strike {index} {radius}"**
  - **Remove** the target at the given **index** and **the ones before and after it** depending on the **radius**.
  - If **any of the indices** in the range is **invalid**, print: **"Strike missed!"** and **skip** this command.

Example: "Strike 2 2"

	{radius}	{radius}	{strikeIndex}	{radius}	{radius}		

- **"End"**
  - **Print** the sequence with targets in the following format and **end the program**:
    "{target₁}|{target₂}…|{targetₙ}"

Input / Constraints

- On the **first line,** you will receive **the sequence of targets – integer values [1-10000]**.
- On the **following lines,** until the **"End"** will be receiving the command described above – **strings**.
- There will never be a case when the **"Strike"** command would empty the whole sequence.

Output

- Print the appropriate message in case of any command if necessary.
- In the end, print the sequence of targets in the format described above.

Examples

Input	Output	Comments

`52 74 23 44 96 110`
`Shoot 5 10`
`Shoot 1 80`
`Strike 2 1`
`Add 22 3`
`End`
	`Invalid placement!`
	`52|100`
		The first command is "**Shoot**", so we reduce the target on **index 5**, which is valid, with the given **power** – **10**.
		Then we receive the same command, but we need to reduce the target on the 1st index, with power 80. The value of this target is 74, so it is considered shot, and we **remove** it.
		Then we receive the "**Strike**" command on the 2nd index, and we need to check if the range with radius 1 is valid:
		52 23 44 96 100
		And it is, so we **remove** the targets.
		At last, we receive the "**Add**" command, but the index is **invalid**, so we print the appropriate **message**, and in the end, we have the following result:
		52|100

`1 2 3 4 5`
`Strike 0 1`
`End`
	`Strike missed!`
	`1|2|3|4|5`	