Exercise: Stacks, Queues, Tuples, and Sets

Problems for exercise and homework for the Python Advanced Course @SoftUni.

Submit your solutions in the SoftUni judge system at https://judge.softuni.org/Contests/3159.

1. Numbers

First, you will be given **two sequences** of **integer** values on **different lines**. The values of the sequences are separated by a **single space** between them.

Keep in mind that each sequence should contain only unique values.

Next, you will receive a **number** - **N**. On the following **N** lines, you will receive one of the following commands:

- **`"Add First {numbers, separated by a space}"`** - add the given numbers at the end of the **first sequence of numbers**.
- **`"Add Second {numbers, separated by a space}"`** - add the given numbers at the end of the **second sequence of numbers**.
- **`"Remove First {numbers, separated by a space}"`** - remove **only the numbers contained** in the **first** sequence.
- **`"Remove Second {numbers, separated by a space}"`** - remove **only the numbers contained** in the **second** sequence.
- **`"Check Subset"`** - check if **any of the given sequences** are a **subset** of the other. If it is, print **`"True"`**. Otherwise, print **`"False"`**.

In the end, print the **final sequences**, separated by a comma and a space **`", "`**. The values in each sequence should be **sorted** in **ascending order**.

Examples

Input	Output
1 2 3 4 5 1 2 3 3 Add First 5 6 Remove Second 8 9 11 Check Subset	True 1, 2, 3, 4, 5, 6 1, 2, 3
5 4 2 9 9 5 4 1 1 1 5 6 5 4 Add First 5 6 9 3 Add Second 1 2 3 3 3 Check Subset Remove Second 1 2 3 4 5	False 2, 3, 4, 5, 6, 9 6

2. Expression Evaluator

Write a program that evaluates a string expression. You will be given that **string expression** on the first line in the form of **numbers** and **operators** separated with a **single space** from each other. Your job is to take that string expression and calculate the result after evaluating it.

To do that, you have to follow a simple rule. If, for example, we have this string **`"2 4 * 1 3 -"`**, the first time we meet an operator (**`*`**), we should take **all the numbers** we have so far (**2**, **4**), apply that operation to them, and **save** the result (**8**). The next time we meet an operator (**`-`**), we again take **all the numbers** we have (**8**, **1**, **3**) and apply the operator to them **in that order** (**`8 - 1 - 3 = 4`**). In the **end**, we **print** the result.

All the numbers will always be **integers,** and the possible operators are **`"*"`**, **`"+"`**, **`"-"`**, **`"/"`**. It is important to **keep the order** of the numbers (especially for **`"/"`** and **`"-"`** because the **order matters**). When having a **division**, you should **round** the result to the **lower integer**.

Input

- **Single line**: a string containing integers and operators

Output

- **Single number**: the result after the evaluation

Constraints

- When reaching an operator, it is sure that you will have a **minimum of one number** to evaluate
- The string will always **end** with an **operator**, so you get one number as a result
- Operators and numbers will **always** be **valid**
- There will be **no** case of **division by zero**
- There might be **negative numbers** in the string

Examples

Input	Output	Comment

`6 3 - 2 1 * 5 /`	`1`	`6 - 3 = 3`
		`3 * 2 * 1 = 6`
		`6 / 5 = 1`

`2 2 - 1 *`	`0`	`2 - 2 = 0`
		`0 * 1 = 0`

`10 23 * 4 2 / 30 10 + 100 50 - 2 -1 *`	`164`	`10 * 23 = 230`
		`230 / 4 / 2 = 28`
		`28 + 30 + 10 = 68`
		`68 - 100 - 50 = -82`
		`-82 * 2 * -1 = 164`

3. Milkshakes

You are learning how to make milkshakes.

First, you will be given **two sequences of integers** representing **chocolates** and **cups of milk**.

You have to start with the **last chocolate** and try to match it with the **first cup of milk**. If their **values** are **equal**, you should **make a milkshake** and **remove both** ingredients. Otherwise, you should **move the cup of milk** at the end of the **sequence** and **decrease** the **value** of the **chocolate** by **5 without moving it from its position**.

If any of the values are **equal to or below 0**, you should **remove them** from the records **right before mixing them** with the other ingredient.

When you **successfully prepare 5 chocolate milkshakes, or** you have **no more chocolate or cups of milk left**, you need to **stop making chocolate milkshakes.**

Input

- On the **first line** of input, you will receive the integers representing the **chocolate**, **separated** by **", "**.
- On the **second line** of input, you will receive the integers representing the **cups of milk**, **separated** by **", "**.

Output

- On the **first** line, print:
  - If you **successfully made** 5 milkshakes: "**`Great! You made all the chocolate milkshakes needed!`**"
  - Otherwise: "**`Not enough milkshakes.`**"
- On the **second** line - print:
  - If there **are** chocolates left: "**`Chocolate: {chocolate1}, {chocolate2}, (…)`**"
  - Otherwise: "**`Chocolate: empty`**"
- On the **third** line - print:
  - If there **are** cups of milk left: "**`Milk: {milk1}, {milk2}, {milk3}, (…)`**"
  - Otherwise: "**`Milk: empty`**"

Constraints

- All given **numbers** will be valid integers in the range **[-100 … 100]**.

Examples

Input	Output

`20, 24, -5, 17, 22, 60, 26`
`26, 60, 22, 17, 24, 10, 55`
	`Great! You made all the chocolate milkshakes needed!`
	`Chocolate: 20`
	`Milk: 10, 55`

Comment	
1) 26 == 26 -> You made chocolate milkshake. Remove both ingredients. 2) 60 == 60 -> You made chocolate milkshake. Remove both ingredients. 3) 22 == 22 -> You made chocolate milkshake. Remove both ingredients. 4) 17 == 17 -> You made chocolate milkshake. Remove both ingredients. 5) -5 is invalid, so it is removed before mixing. 6) 24 == 24 -> You made chocolate milkshake. Remove both ingredients. You made enough chocolate milkshakes. The program ends.	

Input	Output

`-10, -2, -30, 10`
`-5`
	`Not enough milkshakes.`
	`Chocolate: -10, -2, -30, 10`
	`Milk: empty`

`2, 3, 3, 7, 2`
`2, 7, 3, 3, 2, 14, 6`
	`Great! You made all the chocolate milkshakes needed!`
	`Chocolate: empty`
	`Milk: 14, 6`

4. Honey

We think the process of making honey is amazing! Let's learn more about how bees make honey.

Worker-bees **collect** nectar. When a worker-bee has found **enough** nectar, she returns to the hive to **drop off the load** and pass the nectar to waiting bees so they can really **start the honey-making process**.

You will receive **3 sequences**:

- a sequence of **integers**, representing **working bees**
- a sequence of **integers**, representing **nectar**
- a sequence of **symbols** – **`"+", "-", "*"`** or **`"/"`**, representing the **honey-making process**.

Your task is to **check** if the bee has **collected enough nectar** to return to the hive and keep track of the total honey waiting bees **made with the collected nectar**.

Step one: **check** if a bee has **collected enough nectar.** You should take the **first bee** and try to match it with the **last nectar**:

- If the nectar value is **more or equal** to the value of **the bee,** it is considered **collected,** and the bee returns to the hive to drop off the load (step two).
- If the nectar value is **less** than the value of **the bee**, you should **remove the nectar** and try to match **the bee** with the **next nectar's** value until the bee collects enough nectar.

Step two: When a bee **successfully collects nectar**, she returns to the hive, and you should **calculate the honey made**. Take the **first symbol** in the sequence of **symbols** (**`"+", "-", "*"`** or **`"/"`**) and make the **corresponding calculation**:

`"{matched_bee} {symbol} {matched_nectar}"`

The result represents **the honey that is made** from the passed nectar. The calculation should **always** return **the absolute value of the result.** After the calculation**, remove the bee, the nectar, and the symbol.**

- If the **symbol** is **`"/"`** and the **nectar's value** is **0 (zero), skip** the calculation and **remove the bee, the nectar, and the symbol**.

Stop making honey when you are out of bees or nectar.

Input

- On the **first** line, you will be given the values of **bees** - **integers**, separated by a **single space**
- On the **second** line, you will be given the **nectar's** values - **integers**, separated by a **single space**
- On the third line, you will be given symbols - **`"+"`**, **`"-"`**, **`"*"`** or **`"/"`**, separated by a **single space**

Output

- On the first line - print the total honey made:
  - `"Total honey made: {total honey}"`
- On the next two lines print the **bees** or the **nectar** that are **left**, **if there are any**, **otherwise skip the line:**
  - `"Bees left: {bee1}, {bee2}, … {beeN}"`
  - `"Nectar left: {nectar1}, {nectar2}, … {nectarN}"`

Constraints

- All the bee's values will be **integers** in the range **[0, 150]**
- Nectar's values will be **integers** in the range **[0, 150]**
- There always will be enough symbols in the sequence of symbols

Examples

Input	Output	Comment

`20 62 99 35 0 150`
`120 60 10 1 70 10`
`+ - + + / * - - /`
	`Total honey made: 148`
	`Bees left: 99, 35, 0, 150`
		First, compare 20 to 10. 20 is bigger than 10, so remove 10. Then compare 20 to 70. 20 is less than 70, so the bee could return to the hive. Honey made with given nectar is 20 + 70 = 90.
		Next, compare 62 to 1. 62 is bigger than 1, so remove 1. Compare 62 to 10. 62 is bigger than 10, so remove 10. Compare 62 to 60. 62 is bigger than 60, so remove 60. Compare 62 to 120. 60 is less than 120, so the bee could return to the hive. Honey made with given nectar is 62 - 120 = (-58). (-58) is negative, and its absolute value is 58, so the calculation result is 58.
		Total honey made: 90 + 58 = 148.
		Print desired text.

`30`
`15 9 5 150 8`
`* + + * -`
	`Total honey made: 4500`
	`Nectar left: 15, 9, 5`	

5. Santa's Present Factory

This year Santa has decided to share his secret with you. Get ready to learn how his elves craft all the presents.

First, you will receive a sequence of **integers** representing the number of **materials for crafting toys** in one box. After that, you will be given another sequence of **integers** – their **magic level**.

Your task is to **mix materials with magic** so you can **craft presents**, listed in the table below with the **exact** magic level:

Present	Magic needed
`Doll`	`150`
`Wooden train`	`250`
`Teddy bear`	`300`
`Bicycle`	`400`

You should take the **last box with materials** and the **first magic level** value to craft a toy. Their multiplication calculates the total magic level. If the result **equals** one of the levels described in the table above, you craft the present and **remove both** materials and magic value. **Otherwise**:

- If the product of the operation is a **negative number**, you should sum the values together, remove them both from their positions and add the result to the materials.
- If the product **doesn't equal** one of the magic levels in the table and is a **positive** number, remove only the magic value and **increase** the material value by **15**.
- If the magic or material (or both) **equals 0**, remove it (or both) and continue crafting the presents.

Stop crafting presents when you **run out** of boxes of materials **or** magic level values.

Your task is considered done if you manage to craft **either one of the pairs**:

- **a doll and a train**
- **a teddy bear and a bicycle**

Input

- The first line of input will represent the values of boxes with materials - **integers**, separated by a **single space**
- On the second line, you will be given the magic values - **integers** again, separated by a **single space**

Output

- On the first line - print whether you've succeeded in crafting the presents:
  - **`"The presents are crafted! Merry Christmas!"`**
  - **`"No presents this Christmas!"`**
- On the next two lines print the **materials** and **magic** that are **left**, **if there are any (otherwise skip the line)**
  - **`"Materials left: {material_N}, {material_N-1}, … {material_1}"`**
  - **`"Magic left: {magicValue_1}, {magicValue_2}, … {magicValue_N}"`**
- On the next lines print the presents you **have crafted,** ordered **alphabetically** in the format:

  -
    ```
    "{toy_name1}: {amount}
    {toy_name2}: {amount}
    ...
    {toy_nameN}: {amount}"
    ```

Constraints

- All the materials' values will be **integers** in the range **[-100, 100]**
- Magic level values will be **integers** in the range **[-100, 100]**
- In all cases, **at least one present will be crafted**

Examples

Input	Output	Comment

`10 -5 20 15 -30 10`
`40 60 10 4 10 0`
	`The presents are crafted! Merry Christmas!`
	`Materials left: 20, -5, 10`
	`Bicycle: 1`
	`Teddy bear: 2`
		First, we have 40*10=400, which is the needed magic for a bicycle. Remove both.
		60*(-30) = -1800 (negative). 60+(-30) = 30. Remove 60 and -30. Add 30 to materials.
		30*10=300 (bear). Remove both.
		4*15=60, so remove 4, and the material is increased by 15 (15+15=30).
		10*30=300 (bear).
		Print desired text.

`30 5 15 60 0 30`
`-15 10 5 -15 25`
	`No presents this Christmas!`
	`Materials left: 20, 30`
	`Doll: 1`
	`Teddy bear: 1`	

`30 10`
`15 10 5 0 10`
	`No presents this Christmas!`
	`Magic left: 5, 0, 10`
	`Doll: 1`
	`Teddy bear: 1`	

6. Paint Colors

You will have to find all possible color combinations that can be used.

Write a program that finds colors in a string. You will be given a string on a **single line** containing **substrings** (separated by a **single space**) from which you will be able to form the following colors:

Main colors: **"red"**, **"yellow"**, **"blue"**

Secondary colors: **"orange"**, **"purple"**, **"green"**

To form a color, you should concatenate the **first** and the **last substrings** and check if you can get **any** of the **above colors' names**. If there is **only one substring left**, you should **use it** to do the same check.

You can only **keep** a **secondary color** if the **two main colors needed** for its creation could be **formed from the given substrings**:

- **orange = red + yellow**
- **purple = red + blue**

- **green = yellow + blue**

Note: You could find some of the main colors needed to keep a secondary color **after** it is found.

When you form a color, **remove both** substrings. Otherwise, you should **remove the last character** of **each** substring and **return** them in the **middle** of the **original string**. If the string contains an **odd number of substrings**, you should put the substrings **one position ahead**.

For example, if you are given the string **`"re yellow bye"`** you could not form a color with the substring **`"re"`** and **`"bye"`**, so you should remove the last character and return them in the middle of the string: **`"r by yellow"`**.

In the end, **print out the list with colors** in the order in which they are found.

Input

- Single line **string**

Output

- The **list** with the collected colors

Constraints

- You will not receive an empty string
- Please consider only the colors mentioned above
- There won't be any cases with repeating colors

Examples

Input	Output
`d yel blu e low redd`	`['yellow', 'blue', 'red']`
Comment	
First, we take "**d**" and "**redd**". After combining those substrings, we don't get any of the needed colors, so we remove the last characters from both substrings and return them in the middle of the original string, and it becomes "**yel blu red e low**". After that, we take "**yel**" and "**low**" so the first color we add to our list is yellow, and the string we are searching in looks as follows: "**blu red e**" Then we take "**blu**" and "**e**", and since this color is one of the searched ones (blue), we add it to our collection, and the state of the string is now "**red**". We should take the last substring and check if it matches some of the colors, and since it does, we add it (red) to our colors collection. Finally, we print all the colors found: yellow, blue, and red in the format shown above.	
Input	**Output**
`r ue nge ora bl ed`	`['red', 'blue']`
Comment	
We don't keep orange because we don't have yellow in the final list with colors (combining red and yellow gives us orange).	
Input	**Output**
`re ple blu pop e pur d`	`['red', 'purple', 'blue']`