Exercises: Stacks and Queues

Problems for exercises and homework for the "Programming Fundamentals: Arrays and Lists" course from the official "Applied Programmer" curriculum.

You can check your solutions here: https://judge.softuni.bg/Contests/2921.

1. Basic Queue Operations

Play around with a queue. You will be given an integer **N** representing the number of elements to enqueue (**add**), an integer **S** representing the **number of elements** to **dequeue** (**remove**) from the queue and finally an integer **X**, an element that you should look for in the **queue**. If it is, print **true** on the console. If it’s not print the **smallest element** currently present in the queue. If there are **no elements** in the sequence, print **0** on the console.

Examples

Input	Output	Comments
`5 2 32` `1 13 45 32 4`	`true`	We have to **enqueue 5** elements. Then we **dequeue 2** of them. Finally, we have to check whether 32 is present in the queue. Since it is we print **true**.
`4 1 666` `666 69 13 420`	`13`	
`3 3 90` `90 0 90`	`0`	

2. Maximum and Minimum Element

You have an empty sequence, and you will be given **N** queries. Each query is one of these types:

- 1 x – **Push** the element x into the stack.
- 2 – **Delete** the element present at the **top** of the **stack**.
- 3 – **Print** the **maximum** element in the stack.
- 4 – **Print** the **minimum** element in the stack.

After you go through all of the queries, print the stack in the following format:

"{n}, {n$_1$}, {n$_2$} …, {n$_n$}"

Input

- The first line of input contains an integer, **N**.
- The next **N** lines each contain an above-mentioned query. **(It is guaranteed that each query is valid.)**.

Output

- For each type 3 or 4 query, print the **maximum**/minimum element in the stack on a new line.

Constraints

- 1 ≤ N ≤ 105
- 1 ≤ x ≤ 109
- 1 ≤ type ≤ 4
- If there are **no elements** in the stack, **don’t print anything** on commands **3 and 4**.

Examples

Input	Output
9 1 97 2 1 20 2 1 26 1 20 3 1 91 4	26 20 91, 20, 26
10 2 1 47 1 66 1 32 4 3 1 25 1 16 1 8 4	32 66 8 8, 16, 25, 32, 66, 47

3. Fashion Boutique

You own a fashion boutique and you receive a delivery once a month in a huge box, which is full of clothes. You have to arrange them in your store, so you take the box and start **from the last piece** of clothing on the top of the pile **to the first one** at the bottom. Use a **stack** for the purpose. Each piece of clothing has its **value** (an integer). You have to **sum** their values, while you take them out of the box. You will be given an integer representing the **capacity** of a rack. While the sum of the clothes is **less** than the capacity, **keep summing** them. If the sum becomes **equal** to the capacity you have to **take a new rack** for the **next clothes**, if there are **any left** in the box. If it becomes **greater** than the capacity**, don't add** the piece of clothing to the current rack and take a new one. In the end, print **how many racks** you have used to hang all of the clothes.

Input

- On the first line you will be given **a sequence of integers**, representing the clothes in the box, separated **by a single space**.
- On the second line, you will be given **an integer**, representing the capacity of a rack.

Output

- Print the **number of racks**, needed to hang all of the clothes from the box.

Constraints

- The values of the clothes will be integers in the range [0,20].
- There will never be more than 50 clothes in a box.
- The capacity will be an integer in the range [0,20].

- **None** of the integers from the box will be **greater** than then the **value** of the **capacity**.

Examples

Input	Output
`5 4 8 6 3 8 7 7 9` `16`	`5`
`1 7 8 2 5 4 7 8 9 6 3 2 5 4 6` `20`	`5`

4. Songs Queue

Write a program that keeps track of a songs queue. The **first** song that is put in the queue, should be the **first** that **gets played**. A song cannot be added if it is currently in the queue.

You will be given **a sequence of songs**, separated by a comma and a single space. After that you will be given **commands until** there are **no songs enqueued**. When there are **no more songs** in the queue **print** "**`No more songs!`**" and **stop** the **program**.

The possible commands are:

- "**`Play`**" - plays a song (removes it from the queue).
- "**`Add {song}`**" - adds the song to the queue if it isn’t contained already, otherwise print "**`{song} is already contained!`**".
- "**`Show`**" - prints all songs in the queue separated by a comma and a white space **(start from the first song in the queue to the last)**.

Input

- On the first line, you will be given a sequence of strings, separated by a comma and a white space.
- On the next lines you will be given commands until there are no songs in the queue.

Output

- While receiving the commands, print the proper messages described above.
- After the command "Show", print the songs from the **first** to the **last**.

Constraints

- The input **will always be valid** and in the **formats** described above.
- There **might** be commands **even after** there are **no songs in the queue** (ignore them).
- There will never be duplicate songs in the initial queue.

Examples

Input	Output

`All Over Again, Watch Me`
`Play`
`Add Watch Me`
`Add Love Me Harder`
`Add Promises`
`Show`
`Play`
`Play`
	`Watch Me is already contained!`
	`Watch Me, Love Me Harder, Promises`
	`No more songs!`

Play Play	

5. Truck Tour

Suppose there is a circle. There are **N** petrol pumps on that circle. Petrol pumps are numbered 0 to (N−1) (both inclusive). You have **two pieces of information** corresponding to each of the petrol pump:

- the **amount of petrol** that particular petrol pump will give.
- the **distance from that petrol pump** to the next petrol pump.

Initially, you have a tank of infinite capacity carrying no petrol. You can start the tour at **any** of the petrol pumps. Calculate the **first point** from where the truck will be able to complete the circle. Consider that the truck will stop at **each of the petrol pumps**. The truck will move one kilometer for each liter of the petrol.

Input

- The first line will contain the value of **N** – the number of **lines**, which will **follow**.
- The next **N** lines will contain a pair of integers each, i.e. the **amount** of **petrol** that petrol pump will give and the **distance between** that petrol **pump** and the next petrol pump.

Output

- An integer which will be the smallest index of the petrol pump from which we can start the tour.

Constraints

- 1 ≤ N ≤ 1000001
- 1 ≤ Amount of petrol, Distance ≤ 1000000000

Examples

Input	Output
3 1 5 10 3 3 4	1

6. Balanced Parentheses

Given a sequence consisting of parentheses, determine whether the expression is **balanced**. A sequence of parentheses is balanced if every **open parenthesis** can be **paired uniquely** with a **closed parenthesis** that occurs **after** the former. Also, the **interval between** them **must** be **balanced**. You will be given **three** types of parentheses: (, {, and [.

{[()]} - This is a balanced parenthesis.

{[(])} - This is not a balanced parenthesis.

Input

- Each input consists of a single line, **the sequence of parentheses**.

Output

- For each test case, print on a new line "YES" if the parentheses are balanced.
  Otherwise, print "NO". Do not print the quotes.

Constraints

- $1 \le len_s \le 1000$, where len_s is the length of the sequence.
- Each character of the sequence **will be one of** {, }, (,), [,].

Examples

Input	Output
`{[()]}`	YES
`{[(])}`	NO
`{{[[(())]]}}`	YES

7. Simple Text Editor

You are given an empty text. Your task is to implement 4 commands related to manipulating the text

- **`1 {someString}`** - **appends `someString`** to the end of the text.
- **`2 {count}`** - **erases** the last **`count`** elements from the text.
- **`3 {index}`** - **returns** the element at position **`index`** from the text.
- **4** - **undoes** the last not undone command of type **`1/2`** and returns the text to the state before that operation.

Input

- The first line contains ***n***, the number of operations.
- Each of the following ***n*** lines contains the name of the operation followed by the command argument, if any, separated by space in the following format ***CommandName Argument***.

Output

- For each operation of type ***3*** print a single line with the returned character of that operation.

Constraints

- $1 \le N \le 105$.
- The length of the text will not exceed 1000000.
- All input characters are English letters.
- It is guaranteed that the sequence of input operation is possible to perform.

Examples

Input	Output
8 1 abc 3 3 2 3 1 xy 3 2 4 4 3 1	c y a

Explanation

- There are 8 operations. Initially, the text is empty.
- In the first operation, we append **abc** to the text.
- Then, we print its 3rd character, which is **c** at this point.
- Next, we erase its last 3 characters, **abc**.
- After that, we append **xy** to the text.
- The text becomes **xy** after these previous two modifications.
- Then, we are asked to return the 2nd character of the text, which is **y**.
- After that, we have to undo the last update to the text, so it becomes empty.
- The next operation asks us to undo the update before that, so the text becomes **abc** again.
- Finally, we are asked to print its 1st character, which is **a** at this point.

8. *Key Revolver

Our favorite super-spy action hero Sam is back from his mission in another exam, and this time he has an even more difficult task. He needs to **unlock a safe**. The problem is that the safe is **locked** by **several locks in a row**, which all have **varying sizes**.

Our hero posesses a special weapon though, called the **Key Revolver**, with special bullets. Each **bullet** can unlock a **lock** with a **size equal to or larger than** the **size** of the **bullet**. The bullet goes into the keyhole, then explodes, completely **destroying** it. Sam **doesn't know the size** of the locks, so he needs to just shoot at all of them, until the safe runs out of locks.

What's behind the safe, you ask? Well, intelligence! It is told that Sam's sworn enemy – **Nikoladze**, keeps his **top secret Georgian Chacha Brandy** recipe inside. It's valued differently across different times of the year, so Sam's boss will tell him what it's worth over the radio. One last thing, every bullet Sam fires will also cost him money, w**hich will be deducted from his pay** from the price of the intelligence.

Good luck, operative.

Input

- On the **first line** of input, you will receive the price of each **bullet** – an **integer in the range [0-100].**
- On the **second line**, you will receive the **size of the gun barrel** – an **integer in the range [1-5000].**
- On the **third line**, you will receive the **bullets** – a **space-separated integer sequence** with **[1-100] integers.**
- On the **fourth line**, you will receive the **locks** – a **space-separated integer sequence** with **[1-100] integers.**
- On the **fifth line**, you will receive the **value of the intelligence** – an **integer in the range [1-100000].**

After Sam receives all of his information and gear (**input**), he starts to **shoot the locks front-to-back**, while going through the bullets **back-to-front**.

If the **bullet** has a **smaller or equal** size to the **current lock**, print "`Bang!`", then **remove the lock**. If not, print **`"Ping!"`**, leaving the lock **intact**. The bullet is removed in **both cases**.

If Sam runs out of bullets in his barrel, print **`"Reloading!"`** on the console, then continue shooting. If there aren't any bullets left, **don't** print it.

The program ends when Sam **either runs out of bullets**, or the safe **runs out of locks**.

Output

- If Sam **runs out of bullets** before the safe runs out of **locks**, print:
  `"Couldn't get through. Locks left: {locksLeft}"`
- If Sam manages to **open the safe**, print:
  `"{bulletsLeft} bullets left. Earned ${moneyEarned}"`

Make sure to account for the **price of the bullets** when calculating the **money earned**.

Constraints

- The input will be **within the constaints** specified above and will **always be valid**. There is **no need** to check it explicitly.
- There will **never** be a case where Sam breaks the lock and ends up with a **negative balance**.

Examples

Input	Output	Comments

`50`
`2`
`11 10 5 11 10 20`
`15 13 16`
`1500`
	`Ping!`
	`Bang!`
	`Reloading!`
	`Bang!`
	`Bang!`
	`Reloading!`
	`2 bullets left. Earned $1300`
		20 shoots lock **15** (**ping**)
		10 shoots lock **15** (**bang**)
		11 shoots lock **13** (**bang**)
		5 shoots lock **16** (**bang**)
		
		Bullet cost: 4 * 50 = $200
		Earned: 1500 – 200 = $1300

`20`
`6`
`14 13 12 11 10 5`
`13 3 11 10`
`800`
	`Bang!`
	`Ping!`
	`Ping!`
	`Ping!`
	`Ping!`
	`Ping!`
	`Couldn't get through. Locks left: 3`
		5 shoots lock **13** (**bang**)
		10 shoots lock **3** (**ping**)
		11 shoots lock **3** (**ping**)
		12 shoots lock **3** (**ping**)
		13 shoots lock **3** (**ping**)
		14 shoots lock **3** (**ping**)

`33`
`1`
`12 11 10`
`10 20 30`
`100`
	`Bang!`
	`Reloading!`
	`Bang!`
	`Reloading!`
	`Bang!`
	`0 bullets left. Earned $1`
		10 shoots lock **10** (**bang**)
		11 shoots lock **20** (**bang**)
		12 shoots lock **30** (**bang**)
		
		Bullet cost: 3 * 33 = $99
		Earned: 100 – 99 = $1

9. *Cups and Bottles

You will be given a **sequence of integers** – each indicating a **cup's capacity**. After that you will be given **another sequence of integers** – a **bottle with water** in it. Your job is to try to **fill up** all of the cups.

Filling is done by picking **exactly one** bottle at a time. You must start picking from **the last received bottle** and start filling from **the first entered cup**. If the current bottle has **N** water, you **give** the **first entered cup N** water and **reduce** its integer value by **N**.

When a cup's **integer value** reaches **0 or less**, it **gets removed**. It is **possible** that the current cup's value is **greater** than the current bottle's value. **In that case** you **pick bottles until** you reduce the cup's integer value to **0 or less**. If a bottle's value is **greater or equal to** the cup's **current** value, you fill up the cup and **the remaining water becomes wasted**. You should **keep track of the wasted litters of water** and **print it at the end of the program**.

If you **have managed** to **fill up all of the cups**, print the **remaining water bottles**, from the **last entered – to the first**, otherwise you must print the **remaining cups**, by **order of entrance** – from the **first entered – to the last**.

Input

- On the **first line** of input you will receive the integers, representing the **cups' capacity**, **separated** by a **single space**.
- On the **second line** of input you will receive the integers, representing the **filled bottles**, **separated** by a **single space**.

Output

- On the first line of output you must print the remaining bottles, or the remaining cups, depending on the case you are in. Just **keep** the **orders of printing exactly as specified**.
  - **`"Bottles: {remainingBottles}"`** or **`"Cups: {remainingCups}"`**
- On the second line print the wasted litters of water in the following format: **`"Wasted litters of water: {wastedLittersOfWater}".`**

Constraints

- All of the given numbers will be valid integers in the range [1, 500].
- It is safe to assume that there will be **NO** case in which the water is **exactly as much** as the cups' values, so that at the end there are no cups and no water in the bottles.
- Allowed time/memory: 100ms/16MB.

Examples

Input	Output	Comment
4 2 10 5 3 15 15 11 6	Bottles: 3 Wasted litters of water: 26	**We take the first entered cup and the last entered bottle, as it is described in the condition.** **6 - 4 = 2 - we have 2 more so the wasted water becomes 2.** **11 - 2 = 9 -** again, it is more, so we add it to the previous amount, which is 2 and it becomes 11. **15 - 10 = 5 -** wasted water becomes 16. **15 - 5 = 10 -** wasted water becomes 26. **We've managed to fill up all of the cups, so we print the remaining bottles and the total amount of wasted water.**
1 5 28 1 4 3 18 1 9 30 4 5	Cups: 4 Wasted litters of water: 35	
10 20 30 40 50 20 11	Cups: 30 40 50 Wasted litters of water: 1	