Exercises: Stacks and Queues

Problems for exercises and homework for the "CSharp Advanced" course @ Software University.

Problem 1. Basic Stack Operations

Play around with a stack. You will be given an integer **N** representing the number of elements to push into the stack, an integer **S** representing the number of elements to pop from the stack and finally an integer **X**, an element that you should look for in the stack. If it's found, print "**true**" on the console. If it isn't, print the **smallest** element currently present in the stack.

Input

- On the first line you will be given **N**, **S** and **X,** separated by a single space
- On the next line you will be given **N** number of integers

Output

- On a single line print either **true** if **X** is present in the stack, otherwise print the **smallest** element in the stack. If the stack is **empty**, print 0

Examples

Input	Output	Comments
5 2 13 1 13 45 32 4	true	We have to **push 5** elements. Then we **pop 2** of them. Finally, we have to check whether 13 is present in the stack. Since it is we print **true.**
4 1 666 420 69 13 666	13	

Problem 2. 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 13 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	

Problem 3. Maximum and Minimum Element

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

Examples

Input	Output
9 1 97 2 1 20	26 20 91, 20, 26

2 1 26 1 20 3 1 91 4	
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

Problem 4. Fast Food

You have a fast food restaurant and most of the food that you're offering is previously prepared. You need to know if you will have enough food to serve lunch to all of your customers. Write a program that checks the orders' quantity. You also want to know the client with the **biggest** order for the day, because you want to give him a discount the next time he comes.

First, you will be given the **quantity of the food** that you have for the day (an integer number). Next, you will be given **a sequence of integers**, each representing the **quantity of an order**. Keep the orders in a **queue**. Find the **biggest order** and **print** it. You will begin servicing your clients from the **first one** that came. Before each order, **check** if you have enough food left to complete it. If you have, **remove the order** from the queue and **reduce** the amount of food you have. If you succeeded in servicing all of your clients, print:

`"Orders complete".`

If not, print:

`"Orders left: {order1} {order2} .... {orderN}".`

Input

- On the first line you will be given the quantity of your food - **an integer** in the range [0, 1000]
- On the second line you will receive a sequence of integers, representing each order, **separated by a single space**

Output

- Print the quantity of biggest order

- Print "Orders complete" if the orders are complete
- If there are orders left, print them in the format given above

Constraints

- The input will always be valid

Examples

Input	Output
348 20 54 30 16 7 9	54 Orders complete
499 57 45 62 70 33 90 88 76	90 Orders left: 76

Problem 5. 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

Problem 6. Auto Repair and Service

Winter is coming. All vehicles must prepare for it by changing their tires and other necessary procedures. It's a very busy time of the year for all of the auto services and they need help with the organization of their work.

Write a program that keeps track of the vehicles. The **first** vehicle that arrives for service, should be the **first** that **gets served**. After a vehicle gets served **it leaves** the auto service, but the workers also **keep** track of the **served vehicles**. If a client calls and asks for his vehicle, you should **look for** it and tell the client if it already has gotten served.

You will be given **a sequence of car models**, separated by a single space. After that you will be given a **few commands** until you receive the command "End". The commands are - "**`Service`**", "**`CarInfo`**-{**`modelName`**}" and "**`History`**". If you receive the command "**`Service`**", you should serve the **first received vehicle** and afterwards **print**:

"**`Vehicle {vehicleName} got served.`**"

and **delete** it from the awaiting vehicles. If you receive the command "**`CarInfo`**", **check** if the car is waiting for service. If it is – print:

`"Still waiting for service."` and if not, print `"Served."`

If you receive the command "**`History`**", print the served vehicles starting from **the last served to the first**. After the End command, first print the **waiting** (if there are any left) and then the **served vehicles** in the following format:

`"Vehicles for service: {modelName}, {modelName}, …, {modelName}"`

`"Served vehicles: {modelName}, {modelName}, …, {modelName}"`

Input

- On the first line, you will be given a sequence of strings
- On the next n lines you will be given commands

Output

- While receiving the commands, print the proper messages described above
- After the command "End", print the waiting vehicles from the **first to** the **last** and the served vehicles from the **last** to the **first**

Constraints

- The input **will always be valid** and in the **formats** described above
- The **count** of the commands might be **more** than the **count of the vehicles**
- There will always be **at least one** served vehicle
- The vehicles that you should check for will always **exist** in the **list of vehicles** given to you

Examples

Input	Output
BMWE60 SKODAOctavia MERCEDESClk PEUGEOT607 AUDI80 FIATPunto Service Service Service CarInfo-MERCEDESClk CarInfo-AUDI80 History End	Vehicle BMWE60 got served. Vehicle SKODAOctavia got served. Vehicle MERCEDESClk got served. Served. Still waiting for service. MERCEDESClk, SKODAOctavia, BMWE60 Vehicles for service: PEUGEOT607, AUDI80, FIATPunto Served vehicles: MERCEDESClk, SKODAOctavia, BMWE60

Problem 7. 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: (1) the **amount of petrol** that particular petrol pump will give, and (2) 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 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

Problem 8. 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

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

Examples

Input	Output
{[()]}	YES

{[(])}	NO
{{[[(())]]}}	YES

Problem 9. 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 ≤ N ≤ 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.