Stacks and Queues – Exercise

This document defines the exercises for the "C++ Advanced" course @ Software University. Please submit your solutions (source code) to all below-described problems in Judge.

Write C++ code for solving the tasks on the following pages.

Code should compile under the C++03 or the C++11 standard.

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. If there are **no elements** in the sequence, print **0** on the console.

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`	

2. Basic Queue Operations

Play around with a queue. You will be given an integer **N** representing the number of elements to **add**, an integer **S** representing the **number of elements** to **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 printed the **smallest element** is 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 add **5** elements. Then we remove **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. 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 queries, 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
	32
	66
	8
	8, 16, 25, 32, 66, 47

1 25
1 16
1 8
4	

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 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 the 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

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** up 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 than 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`

6. Truck Tour

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

8
10 1
10 1
10 1
6 6
6 6
6 15
10 5
6 12
	0

7. 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 **closing 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 ≤ len_s ≤ 1000, where the len_s is the length of the sequence.
- Each character of the sequence **will be one of** {, }, (,), [,].

Examples

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