Exercises: Stacks and Queues

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

You can check your solutions here: https://judge.softuni.bg/Contests/184/Stacks-and-Queues-Exercise.

Problem 1. Reverse Numbers with a Stack

Write a program that reads **N integers** from the console and **reverses them using a stack**. Use the **`Stack<int>`** class. Just put the input numbers in the stack and pop them. Examples:

Examples

Input	Output
1 2 3 4 5	5 4 3 2 1
1	1

Problem 2. Basic Stack Operations

Play around with a stack. You will be given an integer **N** representing the number of elements to push onto 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 Format:

- 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 Format:

- 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 3. Maximum 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.

Input Format:

- 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 Format:

- For each type 3 query, print the **maximum** element in the stack on a new line

Constraints:

- 1 ≤ N ≤ 105
- 1 ≤ x ≤ 109
- 1 ≤ type ≤ 3

Examples

Input	Output
9 1 97 2 1 20 2 1 26 1 20 3 1 91 3	26 91

Problem 4. 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 90 90	0	

Problem 5. Calculate Sequence with Queue

We are given the following sequence of numbers:

- $S_1 = N$
- $S_2 = S_1 + 1$
- $S_3 = 2*S_1 + 1$
- $S_4 = S_1 + 2$
- $S_5 = S_2 + 1$
- $S_6 = 2*S_2 + 1$
- $S_7 = S_2 + 2$
- $S_8 = S_3 + 1$
- …

Using the **Queue<T>** class, write a program to print its first 50 members for given N.

Constraints:

- -4000000000 ≤ N ≤ 2000000000

Examples

Input	Output
2	2 3 5 4 4 7 5 6 11 7 5 9 6 …
-1	-1 0 -1 1 1 1 2 …
1000	1000 1001 2001 1002 1002 2003 1003 …

Problem 6. 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 Format:

- 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 Format:

- 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 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 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 Format:

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

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 {, }, (,), [,].

Output Format:

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

Examples

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

Problem 8. Stack Fibonacci

Calculate the Fibonacci sequence **using a stack**. Set the Fibonacci sequence to start from 0, i.e. 0, 1, 1, 2, 3, 5, 8… and so on. First **push** 0 and 1 and then use **popping, peeking** and **pushing** to generate every consecutive number.

Examples

Input	Output
7	13

15	610
33	3524578

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 format:

- 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 Format:

- 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.

Problem 10. Poisonous Plants

You are given **N** plants in a garden. Each of these plants has been added with some amount of pesticide. After each day, if any plant has **more pesticide** than the plant at **its left**, being weaker (more GMO) than the left one, **it dies**. You are given the initial values of the amount of pesticide and the position of each plant. Print the number of days **after** which no plant dies, i.e. the time after which there are no plants with more pesticide content than the plant to their left.

Input Format:

- The input consists of an integer **N** representing the number of plants
- The next **single line** consists of **N** integers where every integer represents the position and the amount of pesticides of each plant

Output Format:

- Output a single value equal to the number of days after which no plants die

Constraints:

- $1 \leq N \leq 100000$
- Pesticides amount on a plant is between 0 and 1000000000
-

Examples

Input	Output
7 6 5 8 4 7 10 9	2

Explanation

Initially all plants are alive.
Plants = {(6,1), (5,2), (8,3), (4,4), (7,5), (10,6), (9,7)}.
Plants[k] = (i,j) => j^{th} plant has pesticide amount = i.
After the 1^{st} day, 4 plants remain as plants 3, 5, and 6 die.
Plants = {(6,1), (5,2), (4,4), (9,7)}.
After the 2^{nd} day, 3 plants survive as plant 7 dies. Plants = {(6,1), (5,2), (4,4)}.
After the 3^{rd} day, 3 plants survive, and no more plants die.
Plants = {(6,1), (5,2), (4,4)}.
After the 2^{nd} day the plants stop dying.