Exercises: Stacks and Queues

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

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<Integer>`** 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 amount of elements to push onto the stack, an integer **S** representing the amount of elements to pop from the stack and finally an integer **X**, an element that you should check whether is present in the stack. If it is print **true** on the console, if it's not 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** amount of integers.

Output Format: On a single line print either **true** if **X** is present in the stack otherwise print **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:
$1 \leq N \leq 10^5$
$1 \leq x \leq 10^9$
$1 \leq type \leq 3$

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 amount of elements to enqueue (add), an integer **S** representing the amount of elements to dequeue (remove/poll) from the queue and finally an integer **X**, an element that you should check whether is present in the queue. If it is print **true** on the console, if it's not print the smallest element currently present in the queue.

Examples

Input	Output	Comments
5 2 32 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 1 666 666 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:

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

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

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, S, 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. Recursive Fibonacci

The Fibonacci sequence is quite a famous sequence of numbers. Each member of the sequence is calculated from the sum of the two previous elements. The first two elements are 1, 1. Therefore the sequence goes as 1, 1, 2, 3, 5, 8, 13, 21, 34…
The following sequence can be generated with an array, but that's easy, so your task is to implement recursively.

So if the function getFibonacci(n) returns the n'th Fibonacci number we can express it using getFibonacci(n) = getFibonacci(n-1) + getFibonacci(n-2)

However, this will never end and in a few seconds a StackOverflow Exception is thrown. In order for the recursion to stop it has to have a "bottom". The bottom of the recursion is getFibonacci(1) should return 1 and getFibonacci(0) should return 1.

Input Format: On the only line in the input the user should enter the wanted Fibonacci number.

Output Format: The output should be the n'th Fibonacci number counting from 0

Constraints:

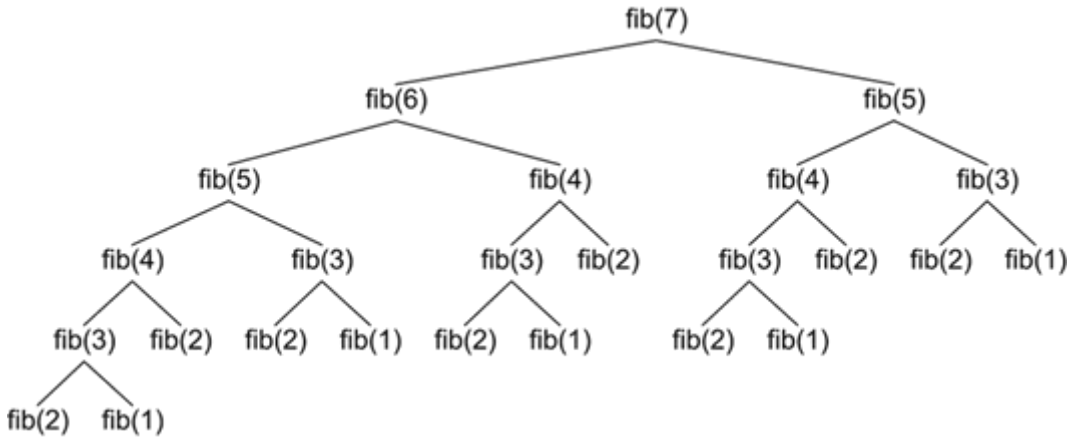
$1 \leq N \leq 49$

Examples

Input	Output

5	8
10	89
21	17711

For the Nth Fibonacci number, we calculate the N-1th and the N-2th number, but for the calculation of N-1th number we calculate the N-1-1th(N-2th) and the N-1-2th number, so we have a lot of repeated calculations.



If you want to figure out how to skip those unnecessary calculations, you can search for a technique called memoization.

Problem 9. Stack Fibonacci

There is another way of calculating the Fibonacci sequence using a stack. It is non recursive, so it does not make any unnecessary calculations. Try implementing it. First push 1 and 1 and them use popping peeking and pushing to generate every consecutive number.

Examples

Input	Output
7	21
15	987
33	5702887

Problem 10. *Simple Text Editor

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

- 1 someString - **append**s 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***.

For example:

3

1 abc

2 2

4

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 11. **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 pesticide and 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 amount of pesticides of each plant.

Constraints: 1 ≤ N ≤ 100000
Pesticides amount on a plant is between 0 and 1000000000

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

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.