Problem 2 – Jan's Notation

Jan is a brilliant mathematician. With brilliance though comes bizzarity, and Jan has invented his own mathematic notation – he writes the operator after his operands, so he can save space by not writing any parenthesis. He asked from you to create an automated calculator that works with his notation, so he can prove to the world it's the better way to do math!

Write a JS program that parses a series of instructions written in **postfix notation** and executes them (postfix means the operator is written **after** the operands). You will receive a **series of instructions** – if the instruction is a **number**, **save it**; otherwise, the instruction is an **arithmetic operator** (**`+-*/`**) and you must apply it to the most two **most recently saved** numbers. **Discard** these two numbers and in their place, **save the result** of the operation – this number is now eligible to be an **operand** in a subsequent operation. Keep going until all input instructions have been exhausted, or you encounter an error.

In the end, if you're left with a **single saved number**, this is the **result** of the calculation and you must **print** it. If there are more numbers saved, then the user supplied **too many instructions** and you must print "**`Error: too many operands!`**". If at any point during the calculation you **don't have** two number saved, the user supplied **too few instructions** and you must print "**`Error: not enough operands!`**". ***See the examples for more details.***

Input

You will receive an array with numbers **and** strings – the numbers will be **operands** and must be saved; the strings will be **arithmetic operators** that must be applied to the operands.

Output

Print on the **console** on a single line the **final result** of the calculation or an **error message**, as instructed above.

Constraints

- The **numbers** (operands) will be integers
- The **strings** (operators) will always be one of **`+-*/`**
- The result of each operation will be in range [-2^{53}…2^{53}-1] (**`MAX_SAFE_INTEGER`** will **never** be exceeded)

Examples

Input	Output	Explanation

`[3,`
` 4,`
` '+']`
	7	The first instruction is a **number**, therefor we **save** it. The next one is also a **number**, we **save** it too.
		
		The third instruction is a **string**, so it must be an **operator** – we **remove the last two** numbers we saved, and perform the operation: **3+4=7**. The result of this operation is then **saved** where the two operands **used to be**.
		
		We've ran out of instructions, so we check the saved values – we only have **one**, so this must be **final result**. We **print** it on the console.

`[5,`
` 3,`
` 4,`
` '*',`
` '-']`
	-7	We save in order **5**, **3** and **4**. The result of the operation **3*4** is **12**, which we **save in place** of **3** and **4**.
		
		Currently we have **5** and **12** saved. The result of the operation **`5-12`** is **-7**, which we **save in place** of **5** and **12**.
		
		We have no more instructions and **only one** value saved, which we **print**.

Input	Output
[7, 33, 8, '-']	Error: too many operands!

Input	Output
[15, '/']	Error: not enough operands!

Input	Output	Explanation
[31, 2, '+', 11, '/']	3	**(31+2)/11**

Input	Output	Explanation
[-1, 1, '+', 101, '*', 18, '+', 3, '/']	6	**(-1+1)*101+18/3**