Postfix expression (`postfix`)

A postfix expression is an expression in which operators follow their operands, for example the following postfix expression *5 7 +* is equivalent to the standard expression *5 + 7*.

The main advantage of a postfix expression is that there is no need for brackets: *2 3 + 2 5 + ** is equivalent to *(2 + 3) * (2 + 5)*.

Another great advantage is that it is more easy to evaluate than infix expressions.
Using a stack we can evaluate the expression from left to right. Each time we scan a number we push it into the stack, each time we scan an operands we pop two elements from the top of the stack, evaluate the operation between them and push the result into the stack again. At the end of the scan, the result is the only element in the stack.

Write a program that evaluates a given postfix expression.

Implementation

You should submit a single file, with either a `.c`, `.cpp`, `.java` or `.py` extension.

Your program must read the input data from `stdin` and write the output data into `stdout`.

`stdin` consists of 2 lines:

- Line 1: The integer N.
- Line 2: N space-separated strings, which can be positive numbers or operands.

`stdout` consists of only one line:

- Line 1: The integer result of the postfix expression evaluation.

No additional output should be printed.

Constraints

- $3 \leq N \leq 99$.
- All the expressions are well-formed.
- All the expressions contains only $+$, $-$ or $*$ operands.
- All the numbers, intermediate and final results fit in 31-bits integers.

Scoring

Your program will be tested on several test cases grouped in subtask.
To achieve the score of a subtask, you need to correctly solve all of its test cases.

- **Subtask 1** [**20 points**]: There are only + operands (see first example).
- **Subtask 2** [**20 points**]: All operands are after numbers (see second example).
- **Subtask 3** [**60 points**]: All possible operands are present (see third example).

Examples

stdin	stdout

`11`
`1 2 + 3 + 4 + 5 + 6 +`
	`21`

`11`
`1 2 3 4 5 6 + - * + -`
	`20`

`9`
`10 2 4 * - 15 3 * +`
	`47`

Explanations

In the **first example** the input is equivalent to the following expression:

$$1 + 2 + 3 + 4 + 5 + 6 = 21$$

In the **second example** the input is equivalent to the following expression:

$$1 - (2 + 3 * (4 - (5 + 6))) = 20$$

In the **third example** the input is equivalent to the following expression:

$$(10 - (2 * 4)) + (15 * 3) = 47$$