Problem A. a+b

Input file:	`standard input`
Output file:	`standard output`
Time limit:	1 second
Memory limit:	256 megabytes

Given two integer numbers a, b. Calculate their sum.

Input

Two integer numbers on single line: $-10^9 \leq a, b \leq 10^9$

Output

Single integer number: $a + b$

Examples

standard input	standard output
2 2	4
100 -100	0
1000000000 1000000000	2000000000

Problem B. Regular Bracket Sequence

Input file:	`standard input`
Output file:	`standard output`
Time limit:	1 second
Memory limit:	256 megabytes

Let's define Bracket sequence (BS) as a string which consists of any natural number (+zero) of brackets. Brackets can be one of the following three types: parentheses: ( ), square brackets: [ ], and braces: { }. Examples of bracket sequences:

- `( [ )`
- `} (`
- `{ )`
- `( { [ ] } )`

Set of Regular Bracket Sequences (RBS) is defined using the following rule:

- Empty string is an RBS.
- If `A` is an RBS, then `(A)`, `[A]`, `{A}` are also RBS.
- If `A` and `B` are RBS, then `AB` and `BA` are also RBS.

Examples of regular bracket sequences:

- `( [ ] )`
- `{ } ( [ ] )`
- `( { [ ] } )`

Input

You are given a bracket sequence terminated by line break symbol. The length of the sequence is $0 \le N \le 200000$ characters. You need to check if this bracket sequence is regular.

Output

Single line: `yes` if given sequence is regular, and `no` otherwise.

Examples

standard input	standard output
`[()]`	`yes`
`[(){}[]]`	`yes`
`{(((())`	`no`

Note

Please note that you need your algorithm to have complexity $O(N)$ to fit the time limit in this problem.

Problem C. Deque

Input file:	`standard input`
Output file:	`standard output`
Time limit:	1 second
Memory limit:	256 megabytes

Implement deque data structure. This data structure was not covered in the lecture, but it's quite easy. Deque is also called double-ended queue (Double-Ended QUEue). This is a data structure, defined by functionality, and it should support the following operations:

- `push_back`;
- `push_front`;
- `len`;
- `pop_back`;
- `pop_front`.

You are given list of $1 \leq N \leq 100000$ requests to deque data structure. You need to process these requests, and check for possible errors (pop from empty dequeue). If the request cannot be processed, it should not make any changes in deque.

Request types are encoded by integer numbers:

0 — `push_back`;

1 — `push_front`;

2 — `len`;

3 — `pop_back`;

4 — `pop_front`;

-1 — End of list.

Input

Each line contains a request description, which consists of 1 or 2 integer numbers.

First number is a request type. For request types $0, 1$, line also contains space character followed by one more integer number: value to be pushed to deque: $0 \leq v \leq 10^9$.

Last line of input file contains one integer number: -1, which denotes end of list of requests.

Total number of requests do not exceed 100000.

Output

For each request of types 2-4, write a result on a separate line (requested length, or value to be removed by pop). If request cannot be performed, you should write `"Error!"` on separate line.

Examples

standard input	standard output
0 7 0 2 0 4 2 0 2 3 3 3 3 -1	3 2 4 2 7
1 8 1 2 0 4 4 1 3 4 4 4 -1	2 3 8 4
3 1 2 0 6 3 0 7 0 8 3 0 4 3 1 9 3 3 4 4 -1	Error! 6 8 4 7 2 9 Error!

Note

Please, notice that all mentioned deque operations should have $O(1)$ complexity to meet time limit in this task.

Python has `collections.deque` data structure built in. Please, don't use it in this problem. But you can use it to test your own deque.

Problem D. EyeQueue

Input file:	`standard input`
Output file:	`standard output`
Time limit:	2 seconds
Memory limit:	256 megabytes

Pear company plans to start sales of its brand new flagship device "eyePhone Y". Since the queues at the start of sales of the Company's devices are usually very long, this year the Company proposed an innovation called eyeQueue: The company sold several certificates in advance that allow the buyer not to go through all the queue, but through only a half: the owner of such a certificate has the right to enter the queue in the middle, instead of the back. If there is an odd number of people in the queue, he can enter the queue right behind the central person (further from the store from central person in the queue).

Of course, this behavior of certificate owners can cause irritation, that's why the Company asks you to develop a program that will simulate the state of eyeQueues in all Company stores in order to provide the fairness of the process and to avoid disturbances.

The company has $0 \le N \le 100000$ stores. You are given the sequence of events, including customers entering and leaving the queue of each store, as well as store requests on number of customers in the queue. Events are given in the order they happened. Types of these events are indicated by symbols:

`'+'` — a customer without a certificate entered the queue;

`'!'` — a customer with a certificate entered the queue;

`'?'` — a request from the store, how many people are currently in the queue;

`'-'` — the customer left the queue (made a purchase and left the store, being absolutely happy, holding his brand new flagship device);

`'#'` — end of the working day in stores of the Company.

All certificate owners are supposed to use their privilege.

Input

The first line of input file contains the only integer number: $0 \le N \le 100000$ – the number of shops your program should consider.

The following $0 \le M \le 100000$ lines contain events your program need to process. The line defining i-th event starts with symbol c_i which denotes type of event (according to the list given above). Then, for events (`'+'`, `'!'`, `'?'`, `'-'`), goes a space character followed by integer number $0 \le q_i < N$ — a number of a store, this event belongs to. Then, for events `'+'` and `'!'` goes space character followed by integer number id_i — id of the customer who entered the queue during this event (customers are enumerated from zero, enumeration is common for all the stores).

The last event in the list is `'#'` event. This character appears only on the last line of the input file.

Output

For each event `'-'` or `'?'`, print a single integer number on a separate line — response for this event:

For `'-'` event — the id of the customer who left the queue of given store during this event.

For `'?'` event — the number of customers present in the queue of given store in the moment this event happened.

Examples

standard input	standard output
1 + 0 0 + 0 1 + 0 2 ? 0 - 0 - 0 ? 0 + 0 3 + 0 4 ? 0 - 0 - 0 - 0 #	3 0 1 1 3 2 3 4
1 + 0 0 - 0 + 0 1 + 0 2 + 0 3 ! 0 4 - 0 - 0 - 0 - 0 #	0 1 2 4 3
2 + 0 0 + 0 1 + 1 2 + 1 3 - 0 - 1 ? 0 + 1 4 + 0 5 - 0 - 0 ? 1 - 1 - 1 #	0 2 1 1 5 2 3 4

Note

Please, note that all the events processing should have $O(1)$ complexity to meet time limit of this problem.