A. Find Paired Brackets

time limit per test: 1 second
memory limit per test: 256 megabytes
input: standard input
output: standard output

You are given a correct (regular) bracket sequence of round brackets. For each index print, the index of the corresponding paired bracket.

For example, if you are given '(()())' then the resulting arrays is: $[5, 2, 1, 4, 3, 0]$.

Input

The first line contains the given sequence. Its length is between 2 and 100. It is guaranteed that you are given a correct (regular) bracket sequence of round brackets.

Output

Print the required sequence.

Examples

input	Copy
(()())	
output	Copy
5 2 1 4 3 0	

input	Copy
()()()()	
output	Copy
1 0 3 2 5 4 7 6	

input	Copy
((()))	
output	Copy
5 4 3 2 1 0	

B. Implement Queue Using Two Stacks

time limit per test: 3 seconds
memory limit per test: 256 megabytes
input: standard input
output: standard output

In a stack, we add elements in LIFO (Last In, First Out) order. This means that the last element inserted in the stack will be the first one removed. The basic operations of a stack are:

- push — insert an element at the top
- pop — remove an element from the top

In a queue, we add elements in FIFO (First In, First Out) order, meaning that the first element inserted is the first one to be removed. The basic operations of the queue are:

- enqueue — insert an element at the rear
- dequeue — remove an element from the front

To construct a queue using two stacks (s_1, s_2), we need to simulate the queue operations by using stack operations:

- enqueue — push an element to s_1;
- dequeue — if s_2 is empty then pop/push all elements from s_1 to s_2; after it do pop operation from s_2.

Implement this algorithm.

Input

The first line of the input contains the number of operations — n ($1 \le n \le 10^5$). The next n lines contain the description of operations one per line. The added element cannot exceed 10^9 by absolute value.

Output

Print lines, one line for each operation with a queue. Start a line with t ($1 \le t \le 2$):

- $t = 1$ if this operation is processed with the stack s_1;
- $t = 2$ if this operation is processed with the stack s_2.

In the case of a push operation, the second value in a line should be '+'. After print the pushed value.

In the case of a pop operation, the second value in a line should be '−'. After print the extracted value.

Use exactly the algorithm described in the statement.

Examples

input Copy

```
4
+ 1
+ 10
-
-
```

output Copy

```
1 + 1
1 + 10
1 - 10
2 + 10
1 - 1
2 + 1
2 - 1
2 - 10
```

C. Reverse Polish notation (RPN)

time limit per test: 1 second
memory limit per test: 256 megabytes
input: standard input
output: standard output

Reverse Polish notation (RPN), also known as Polish postfix notation or simply postfix notation, is a mathematical notation in which operators follow their operands, in contrast to Polish notation (PN), in which operators precede their operands. It does not need any parentheses as long as each operator has a fixed number of operands. The description "Polish" refers to the nationality of logician Jan Łukasiewicz, who invented Polish notation in 1924.

In reverse Polish notation, the operators follow their operands; for instance, to add `3` and `4`, one would write `3 4 +` rather than `3 + 4`. If there are multiple operations, operators are given immediately after their second operands; so the expression written `3 - 4 + 5` in conventional notation would be written `3 4 - 5 +` in reverse Polish notation: `4` is first subtracted from `3`, then `5` is added to it.

An advantage of reverse Polish notation is that it removes the need for parentheses that are required by infix notation. While `3 - 4 * 5` can also be written `3 - (4 * 5)`, that means something quite different from `(3 - 4) * 5`. In reverse Polish notation, the former could be written `3 4 5 * -`, which unambiguously means `3 (4 5 *) -` which reduces to `3 20 -` (which can further be reduced to `-17`); the latter could be written `3 4 - 5 *` (or `5 3 4 - *`, if keeping similar formatting), which unambiguously means `(3 4 -) 5 *`.

The expression is given in reverse Polish notation. Determine its meaning.

Input

The only line contains an expression in postfix notation containing single-digit numbers and operations `+`, `-`, `*`. The string contains no more than 100 numbers and operations.

Output

It is necessary to display the value of the written expression.

It is guaranteed that the result of the expression, as well as the results of all intermediate calculations, are in the range $[-2^{31}, 2^{31} - 1]$ (i.e. fit in 32-bit signed integer).

Examples

input	Copy
`8 9 + 1 7 - *`	
output	Copy
`-102`	

input	Copy
`3 7 +`	
output	Copy
`10`	

input	Copy
`2 3 4 5 * 6 - 7 + 8 9 2 * 3 4 1 - 3 4 + * 5 - + * + - * +`	
output	Copy
`-985`	

input	Copy
`9 9 9 9 9 9 9 9 * * * * * * * *`	
output	Copy
`387420489`	

D. Leftmost Ones

time limit per test: 3 seconds
memory limit per test: 256 megabytes
input: standard input
output: standard output

You are given a binary string s of '0's and '1's. The length of s is n. Also you are given m ($1 \le m \le n$). For each substring (window) of length m print the index of the leftmost 1 in it (or print -1).

Input

The first line contains t ($1 \le t \le 10^4$) — the number of test cases. Then t test cases follow.

The following $2t$ lines contain test cases. The first line contains n and m ($1 \le m \le n \le 10^6$). The second line contains only '0's and '1's. Its length is n.

The summary length of all s in a test doesn't exceed 10^6.

Output

Print t lines. The i-th line should contain the answer for the i-th test case.

Pay attention, that you print indices inside a window (not global indices in s).

Example

input Copy

```
4
6 3
101001
1 1
0
6 2
010010
3 1
111
```

output Copy

```
0 1 0 2
-1
1 0 -1 1 0
0 0 0
```

G. Improved Queue

time limit per test: 4 seconds
memory limit per test: 256 megabytes
input: standard input
output: standard output

Implement *improved queue* data structure that supports the following operations:

- "+ i" — enqueue value i (append it to the tail of a queue),
- "* i" — insert i in the middle of a queue (if the current queue length is even then i will take place of exact center; if the current queue length is odd then i will take place after the current center),
- "-" — dequeue element from a queue (remove element from the head).

Invent an efficient way to support all operations and implement it.

Input

The first line contains one integer n ($1 \le n \le 3 \cdot 10^5$) — the number of operations to process. The following n lines contain operation descriptions in the following format:

- "+ i" — enqueue i ($1 \le i \le n$);
- "* i" — insert i at the center of a queue;
- "-" — dequeue an element from a queue. It is guaranteed that at the moment of each such operation a queue will not be empty.

Output

For each operation of the third type, print on a separate line the dequeued element.

Examples

input Copy

```
7
+ 1
+ 2
-
+ 3
+ 4
-
-
```

output Copy

```
1
2
3
```