2. JavaScript: Step Counter

In this challenge, you are provided with the implementation of a simple counter object:

```
const counter = (function counter() {
  let value = 0;
  return {
    getValue: function() {
      return value;
    },
    changeBy: function(k) {
      value += k;
    },
  }
})();
```

Your task is to implement a function *stepCounter* that:

- takes a single parameter *k*
- returns a new object, representing a step counter with the initial value of 0 and with three methods:
  - increment() : increments the current value by *k*
  - decrement() : decrements the current value by *k*
  - getValue() : returns the current value

Your implementation must encapsulate the provided counter object and use it for its implementation. The object returned by *stepCounter* must not have a *changeBy* property.

Your implementation of the function will be tested by a provided code stub on several input files. Each input file contains a parameter for *stepCounter*, followed by several values denoting the operations to perform on the object returned by *stepCounter*. The results of performing the operations will be printed to the standard output by the provided code.

▼ Input Format Format for Custom Testing

In the first line, there is a single integer, *k denoting the parameter for the stepCounter function.*
In the second line, there is integer, n, denoting the number of operations to perform.
Next, n lines follow. Each of them contains a single character, either +, -, or ? denoting respectively calling the increment(), decrement(), and getValue() methods on the objected returned by stepCounter.

▼ Sample Case 0

Sample Input

```
STDIN    Function
-----    --------
1     →  parameter k = 1
4     →  number of operations, n = 1
+     →  increment()
?     →  getValue()
-     →  decrement()
?     →  getValue()
```

Sample Output

```
1
0
```

Explanation
In this test, the *k* parameter for *stepCounter* is 1, so each *increment* must increment the value by 1 and each *decrement* must decrement the value by 1. *Initially, the counter has the value 0. There are 4 operations to be performed. The first of them increments the counter, so it has the value 1 now. The second prints the current value of the counter. The third decrements the counter, so it has the value 0 now. The fourth prints the current value of the counter.*

▼ Sample Case 1

Sample Input

```
STDIN    Function
-----    --------
2     →  parameter k = 2
5     →  number of operations, n = 5
-     →  decrement()
?     →  getValue()
+     →  increment()
+     →  increment()
?     →  getValue()
```

Sample Output

```
-2
2
```

Explanation

In this test, the *k* parameter for *stepCounter* is 2, so each *increment* must increment the value by 2 and each *decrement* must decrement the value by 2. *Initially, the counter has the value 0. There are 5 operations to be performed. The first of them decrements the counter, so it has the value -2 now. The second prints the current value of the counter. The third increments the counter, so it has the value 0 now. The fourth increments the counter, so it has the value 2 now. The fifth prints the current value of the counter.*