Problem 2011: Final Value of Variable After Performing Operations

Problem Information

Difficulty: Easy
Acceptance Rate: 0.00%
Paid Only: No

Problem Description

There is a programming language with only

four

operations and

one

variable

X

:

++X

and

X++

increments

the value of the variable

X

by

1

.

--X

and

X--

decrements

the value of the variable

X

by

1

.

Initially, the value of

X

is

0

.

Given an array of strings

operations

containing a list of operations, return

the

final

value of

X

after performing all the operations

.

Example 1:

Input:

operations = ["--X","X++","X++"]

Output:

1

Explanation:

The operations are performed as follows: Initially, X = 0. --X: X is decremented by 1, X = 0 - 1 = -1. X++: X is incremented by 1, X = -1 + 1 = 0. X++: X is incremented by 1, X = 0 + 1 = 1.

Example 2:

Input:

operations = ["++X","++X","X++"]

Output:

3

Explanation:

The operations are performed as follows: Initially, X = 0. ++X: X is incremented by 1, X = 0 + 1 = 1. ++X: X is incremented by 1, X = 1 + 1 = 2. X++: X is incremented by 1, X = 2 + 1 = 3.

Example 3:

Input:

operations = ["X++","++X","--X","X--"]

Output:

0

Explanation:

The operations are performed as follows: Initially, X = 0. X++: X is incremented by 1, X = 0 + 1 = 1. ++X: X is incremented by 1, X = 1 + 1 = 2. --X: X is decremented by 1, X = 2 - 1 = 1. X--: X is decremented by 1, X = 1 - 1 = 0.

Constraints:

1 <= operations.length <= 100

operations[i]

will be either

"++X"

,

"X++"

,

"--X"

, or

"X--"

.

Code Snippets

C++:

```
class Solution {
public:
int finalValueAfterOperations(vector<string>& operations) {

}
};
```

Java:

```
class Solution {
public int finalValueAfterOperations(String[] operations) {

}
}
```

Python3:

```
class Solution:
def finalValueAfterOperations(self, operations: List[str]) -> int:
```

Python:

```
class Solution(object):
def finalValueAfterOperations(self, operations):
"""
:type operations: List[str]
:rtype: int
"""
```

JavaScript:

```
/**
* @param {string[]} operations
* @return {number}
*/
var finalValueAfterOperations = function(operations) {

};
```

TypeScript:

```
function finalValueAfterOperations(operations: string[]): number {

};
```

C#:

```
public class Solution {
public int FinalValueAfterOperations(string[] operations) {

}
}
```

C:

```
int finalValueAfterOperations(char** operations, int operationsSize) {

}
```

Go:

```
func finalValueAfterOperations(operations []string) int {

}
```

Kotlin:

```
class Solution {
fun finalValueAfterOperations(operations: Array<String>): Int {

}
}
```

Swift:

```
class Solution {
func finalValueAfterOperations(_ operations: [String]) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn final_value_after_operations(operations: Vec<String>) -> i32 {

}
}
```

Ruby:

```
# @param {String[]} operations
# @return {Integer}
def final_value_after_operations(operations)

end
```

PHP:

```
class Solution {

/**
* @param String[] $operations
* @return Integer
*/
function finalValueAfterOperations($operations) {

}
}
```

Dart:

```
class Solution {
int finalValueAfterOperations(List<String> operations) {

}
}
```

Scala:

```
object Solution {
def finalValueAfterOperations(operations: Array[String]): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec final_value_after_operations(operations :: [String.t]) :: integer
def final_value_after_operations(operations) do

end
end
```

Erlang:

```
-spec final_value_after_operations(Operations :: [unicode:unicode_binary()])
-> integer().
final_value_after_operations(Operations) ->
.
```

Racket:

```
(define/contract (final-value-after-operations operations)
(-> (listof string?) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Final Value of Variable After Performing Operations
* Difficulty: Easy
* Tags: array, string
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
```

```
*/

class Solution {
public:
int finalValueAfterOperations(vector<string>& operations) {

}
};
```

Java Solution:

```
/**
* Problem: Final Value of Variable After Performing Operations
* Difficulty: Easy
* Tags: array, string
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

class Solution {
public int finalValueAfterOperations(String[] operations) {

}
}
```

Python3 Solution:

```
"""
Problem: Final Value of Variable After Performing Operations
Difficulty: Easy
Tags: array, string

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(1) to O(n) depending on approach
"""

class Solution:
def finalValueAfterOperations(self, operations: List[str]) -> int:
```

```
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def finalValueAfterOperations(self, operations):
"""
:type operations: List[str]
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Final Value of Variable After Performing Operations
* Difficulty: Easy
* Tags: array, string
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

/**
* @param {string[]} operations
* @return {number}
*/
var finalValueAfterOperations = function(operations) {

};
```

TypeScript Solution:

```
/**
* Problem: Final Value of Variable After Performing Operations
* Difficulty: Easy
* Tags: array, string
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
```

```
* Space Complexity: O(1) to O(n) depending on approach
*/

function finalValueAfterOperations(operations: string[]): number {

};
```

C# Solution:

```
/*
* Problem: Final Value of Variable After Performing Operations
* Difficulty: Easy
* Tags: array, string
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

public class Solution {
public int FinalValueAfterOperations(string[] operations) {

}
}
```

C Solution:

```
/*
* Problem: Final Value of Variable After Performing Operations
* Difficulty: Easy
* Tags: array, string
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

int finalValueAfterOperations(char** operations, int operationsSize) {

}
```

Go Solution:

```
// Problem: Final Value of Variable After Performing Operations
// Difficulty: Easy
// Tags: array, string
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(1) to O(n) depending on approach

func finalValueAfterOperations(operations []string) int {

}
```

Kotlin Solution:

```
class Solution {
fun finalValueAfterOperations(operations: Array<String>): Int {

}
}
```

Swift Solution:

```
class Solution {
func finalValueAfterOperations(_ operations: [String]) -> Int {

}
}
```

Rust Solution:

```
// Problem: Final Value of Variable After Performing Operations
// Difficulty: Easy
// Tags: array, string
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(1) to O(n) depending on approach

impl Solution {
pub fn final_value_after_operations(operations: Vec<String>) -> i32 {
```

```
}
}
```

Ruby Solution:

```
# @param {String[]} operations
# @return {Integer}
def final_value_after_operations(operations)

end
```

PHP Solution:

```
class Solution {

/**
* @param String[] $operations
* @return Integer
*/
function finalValueAfterOperations($operations) {

}
}
```

Dart Solution:

```
class Solution {
int finalValueAfterOperations(List<String> operations) {

}
}
```

Scala Solution:

```
object Solution {
def finalValueAfterOperations(operations: Array[String]): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec final_value_after_operations(operations :: [String.t]) :: integer
def final_value_after_operations(operations) do

end
end
```

Erlang Solution:

```
-spec final_value_after_operations(Operations :: [unicode:unicode_binary()])
-> integer().
final_value_after_operations(Operations) ->
.
```

Racket Solution:

```
(define/contract (final-value-after-operations operations)
(-> (listof string?) exact-integer?)
)
```