2011. Final Value of Variable After Performing Operations

Easy Array String Simulation

Problem Description

The problem presents a simple programming language with a single variable `X` and four operations that can modify the value of `X`. These operations are incrementing `X` by 1 (`++X` or `X++`) and decrementing `X` by 1 (`--X` or `X--`). Initially, `X` is 0. The task is to determine the final value of `X` after executing a sequence of these operations provided in an array of strings.

Intuition

The solution approach is quite straightforward due to the problem's simplicity. Since there are only two types of operations that affect `X` - either incrementing or decrementing by 1, we can iterate through the array of operations and simply count how many times each operation occurs. Since it's given that the operations are strings containing either `++` or `--`, we can inspect the second character of each operation string to determine whether we're dealing with an increment or a decrement operation. This is a valid approach because the second character explicitly determines operation type: if it's a `+`, we increment; if it's a `-`, we decrement. We do not need to differentiate between pre- or post-increment/decrement operations for this particular problem since the effect on the final value of `X` is the same.

Solution Approach

The implementation of the solution utilizes a one-pass loop through the list of operation strings. The Python code takes advantage of list comprehension—a compact syntax for processing lists—and the `sum` function to aggregate the results.

Here is a step-by-step explanation of the solution approach:

- **List Comprehension**: The `for s in operations` part of the list comprehension iterates over each string `s` in the `operations` list.
- **Condition Check**: For each string `s`, `s[1] == '+'` checks whether the second character is a `'+'`. The comparison yields a boolean value (`True` for increment, `False` for decrement).
- **Ternary Operation**: Based on the result of the condition check, `1 if s[1] == '+' else -1` evaluates to `1` if the character is a `'+'`, meaning the operation is an increment. If the character is not a `'+'` (thus a `'-'` since those are the only options), the result is `-1` for a decrement operation.
- **Summation**: The `sum(...)` function takes all these `1`s and `-1`s and sums them up to get the total net effect on `X`.

The solution does not make use of any additional data structures, resulting in a space complexity of `O(1)`. The time complexity is `O(n)` where `n` is the number of operations, since it only involves iterating through the list of operations once.

No complex algorithms or design patterns are necessary because we simply transform each operation into its numeric effect (`+1` or `-1`) and then summarize these effects to get the final result.

Example Walkthrough

Let's consider a small example to illustrate the solution approach. Suppose the array of operations is as follows:

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

Following the solution approach step by step:

1. Initialize a sum variable to 0 which will hold the final value of `X`.
2. Iterate over each operation in the list:
   - The first operation is "++X" which has a '+' as its second character. According to the approach, `X` should be incremented by 1.
   - The second operation is "X++" which also has a '+' as its second character. Again, `X` should be incremented by 1.
   - The third operation is "--X" which has a '-' as its second character. This time, `X` should be decremented by 1.
   - The fourth operation is "X--" which again has a '-' as its second character. We decrement `X` by 1 once more.
   - The fifth operation is "X++" with a '+' as the second character, and `X` is incremented by 1.
3. Apply the operations to the sum variable:
   - After the first operation, sum = 0 + 1 = 1
   - After the second operation, sum = 1 + 1 = 2
   - Following the third operation, sum = 2 - 1 = 1
   - After the fourth operation, sum = 1 - 1 = 0
   - After the fifth operation, sum = 0 + 1 = 1
4. We have now completed iterating through the operations, and the final sum value of 1 represents the final value of `X`.

Using the list comprehension and ternary operation, the Python code using this solution approach would look like this:

```
final_value_of_X = sum(1 if s[1] == '+' else -1 for s in operations)
```

And for our example array, executing this code will result in `final_value_of_X` being `1`, which matches our manual calculation.

Solution Implementation

Python

```python
from typing import List  # Need to import List for type hints

class Solution:
    def finalValueAfterOperations(self, operations: List[str]) -> int:
        """
        This function computes the final value after performing all the operations in the given list.

        Parameters:
        operations (List[str]): A list of string operations, each element is one of "++X", "X++", "--X", "X--".

        Returns:
        int: The final integer value after all operations are performed.
        """

        # Initialize the final value to 0
        final_value = 0

        # Loop through the list of operations
        for operation in operations:
            # Increment final_value for operations that have a '+' in the middle (i.e. '++X' or 'X++')
            if operation[1] == '+':
                final_value += 1
            # Decrement final_value for operations that have a '-' in the middle (i.e. '--X' or 'X--')
            else:
                final_value -= 1

        # Return the calculated final value
        return final_value

# Example usage:
# solution = Solution()
# result = solution.finalValueAfterOperations(["--X", "X++", "X++"])
# print(result)  # Output would be 1, since the final value is incremented twice and decremented once.
```

Java

```java
class Solution {

    // This method computes the final value after performing all the operations in the array.
    public int finalValueAfterOperations(String[] operations) {
        // Initialize the result to 0.
        int result = 0;

        // Loop through each operation string in the operations array.
        for (String operation : operations) {
            // Check the second character of the operation string to determine if it's an increment or decrement.
            // The second character will be '+' for increment and '-' for decrement operations.
            if (operation.charAt(1) == '+') {
                // If we have an increment operation, increase the result by 1.
                result++;
            } else {
                // If we have a decrement operation, decrease the result by 1.
                result--;
            }
        }

        // Return the final result after performing all operations.
        return result;
    }
}
```

C++

```cpp
#include <vector>
#include <string>

class Solution {
public:
    // Function to calculate the final value after performing all operations.
    int finalValueAfterOperations(std::vector<std::string>& operations) {
        int finalValue = 0; // Initialize the final value to 0.

        // Loop over each operation string in the vector.
        for (const auto& operation : operations) {
            // Check the second character of the string to determine operation type.
            if (operation[1] == '+') {
                finalValue++; // If it's a plus, increment the value.
            } else {
                finalValue--; // If it's a minus, decrement the value.
            }
        }

        // Return the result after performing all operations.
        return finalValue;
    }
};
```

TypeScript

```typescript
// Defines a function that calculates the final value after processing all operations
// @param operations - An array of strings representing operations, each being either "++X", "X++", "--X", or "X--"
// @returns The final value after performing all operations
function finalValueAfterOperations(operations: string[]): number {
    // Reduce the operations array to a single value that represents the final value
    // @param accumulator - The running total of the operations processed
    // @param currentValue - The current operation string being processed
    // @returns The updated running total after processing the current operation
    return operations.reduce((accumulator, currentValue) => {
        // Check if the second character of the operation string is a '+' or '-'
        // Increment or decrement the accumulator accordingly
        return accumulator + (currentValue[1] === '+' ? 1 : -1);
    }, 0); // Initialize the accumulator to 0, since the starting value is 0
}

// Example usage of the function:
// const result = finalValueAfterOperations(["--X", "X++", "X++"]);
// console.log(result); // Output will be 1
```

```python


from typing import List  # Need to import List for type hints

class Solution:
    def finalValueAfterOperations(self, operations: List[str]) -> int:
        """
        This function computes the final value after performing all the operations in the given list.

        Parameters:
        operations (List[str]): A list of string operations, each element is one of "++X", "X++", "--X", "X--".

        Returns:
        int: The final integer value after all operations are performed.
        """

        # Initialize the final value to 0
        final_value = 0

        # Loop through the list of operations
        for operation in operations:
            # Increment final_value for operations that have a '+' in the middle (i.e. '++X' or 'X++')
            if operation[1] == '+':
                final_value += 1
            # Decrement final_value for operations that have a '-' in the middle (i.e. '--X' or 'X--')
            else:
                final_value -= 1

        # Return the calculated final value
        return final_value

# Example usage:
# solution = Solution()
# result = solution.finalValueAfterOperations(["--X", "X++", "X++"])
# print(result)  # Output would be 1, since the final value is incremented twice and decremented once.
```

Time and Space Complexity

Time Complexity

The time complexity of the provided code is `O(n)`, where `n` is the number of operations in the input list. This is because the code iterates through each element in the operations list exactly once to determine if the operation is an increment or decrement.

Space Complexity

The space complexity of the provided code is `O(1)`. The only extra space used is for the accumulator in the sum operation which keeps track of the final result, regardless of the number of operations.