Problem 3749: Evaluate Valid Expressions

Problem Information

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

Problem Description

You are given a string

expression

that represents a nested mathematical expression in a simplified form.

A

valid

expression is either an integer

literal

or follows the format

op(a,b)

, where:

op

is one of

"add"

,

"sub"

,

"mul"

, or

"div"

.

a

and

b

are each valid expressions.

The

operations

are defined as follows:

add(a,b) = a + b

sub(a,b) = a - b

mul(a,b) = a * b

div(a,b) = a / b

Return an integer representing the

result

after fully evaluating the expression.

Example 1:

Input:

expression = "add(2,3)"

Output:

5

Explanation:

The operation

add(2,3)

means

2 + 3 = 5

.

Example 2:

Input:

expression = "-42"

Output:

-42

Explanation:

The expression is a single integer literal, so the result is -42.

Example 3:

Input:

expression = "div(mul(4,sub(9,5)),add(1,1))"

Output:

8

Explanation:

First, evaluate the inner expression:

sub(9,5) = 9 - 5 = 4

Next, multiply the results:

mul(4,4) = 4 * 4 = 16

Then, compute the addition on the right:

add(1,1) = 1 + 1 = 2

Finally, divide the two main results:

div(16,2) = 16 / 2 = 8

Therefore, the entire expression evaluates to 8.

Constraints:

1 <= expression.length <= 10

5

expression

is valid and consists of digits, commas, parentheses, the minus sign

'-'

, and the lowercase strings

"add"

,

"sub"

,

"mul"

,

"div"

.

All intermediate results fit within the range of a long integer.

All divisions result in integer values.

Code Snippets

C++:

```
class Solution {
public:
long long evaluateExpression(string expression) {

}
};
```

Java:

```
class Solution {
public long evaluateExpression(String expression) {
```

```
}
}
```

Python3:

```
class Solution:
def evaluateExpression(self, expression: str) -> int:
```

Python:

```
class Solution(object):
def evaluateExpression(self, expression):
"""
:type expression: str
:rtype: int
"""
```

JavaScript:

```
/**
* @param {string} expression
* @return {number}
*/
var evaluateExpression = function(expression) {

};
```

TypeScript:

```
function evaluateExpression(expression: string): number {

};
```

C#:

```
public class Solution {
public long EvaluateExpression(string expression) {

}
}
```

C:

```
long long evaluateExpression(char* expression) {

}
```

Go:

```
func evaluateExpression(expression string) int64 {

}
```

Kotlin:

```
class Solution {
fun evaluateExpression(expression: String): Long {

}
}
```

Swift:

```
class Solution {
func evaluateExpression(_ expression: String) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn evaluate_expression(expression: String) -> i64 {

}
}
```

Ruby:

```
# @param {String} expression
# @return {Integer}
def evaluate_expression(expression)

end
```

PHP:

```
class Solution {

/**
* @param String $expression
* @return Integer
*/
function evaluateExpression($expression) {

}
}
```

Dart:

```
class Solution {
int evaluateExpression(String expression) {

}
}
```

Scala:

```
object Solution {
def evaluateExpression(expression: String): Long = {

}
}
```

Elixir:

```
defmodule Solution do
@spec evaluate_expression(expression :: String.t) :: integer
def evaluate_expression(expression) do

end
end
```

Erlang:

```
-spec evaluate_expression(Expression :: unicode:unicode_binary()) ->
integer().
evaluate_expression(Expression) ->
```

```
.
```

Racket:

```
(define/contract (evaluate-expression expression)
(-> string? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Evaluate Valid Expressions
* Difficulty: Hard
* Tags: string, math, hash, stack
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

class Solution {
public:
long long evaluateExpression(string expression) {

}
};
```

Java Solution:

```
/**
* Problem: Evaluate Valid Expressions
* Difficulty: Hard
* Tags: string, math, hash, stack
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/
```

```
class Solution {
public long evaluateExpression(String expression) {

}
}
```

Python3 Solution:

```
"""
Problem: Evaluate Valid Expressions
Difficulty: Hard
Tags: string, math, hash, stack

Approach: String manipulation with hash map or two pointers
Time Complexity: O(n) or O(n log n)
Space Complexity: O(n) for hash map
"""

class Solution:
def evaluateExpression(self, expression: str) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def evaluateExpression(self, expression):
"""
:type expression: str
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Evaluate Valid Expressions
* Difficulty: Hard
* Tags: string, math, hash, stack
*
* Approach: String manipulation with hash map or two pointers
```

```
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

/**
* @param {string} expression
* @return {number}
*/
var evaluateExpression = function(expression) {

};
```

TypeScript Solution:

```
/**
* Problem: Evaluate Valid Expressions
* Difficulty: Hard
* Tags: string, math, hash, stack
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

function evaluateExpression(expression: string): number {

};
```

C# Solution:

```
/*
* Problem: Evaluate Valid Expressions
* Difficulty: Hard
* Tags: string, math, hash, stack
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

public class Solution {
```

```
public long EvaluateExpression(string expression) {

}
}
```

C Solution:

```
/*
* Problem: Evaluate Valid Expressions
* Difficulty: Hard
* Tags: string, math, hash, stack
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

long long evaluateExpression(char* expression) {

}
```

Go Solution:

```
// Problem: Evaluate Valid Expressions
// Difficulty: Hard
// Tags: string, math, hash, stack
//
// Approach: String manipulation with hash map or two pointers
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) for hash map

func evaluateExpression(expression string) int64 {

}
```

Kotlin Solution:

```
class Solution {
fun evaluateExpression(expression: String): Long {

}
```

```
}
```

Swift Solution:

```
class Solution {
func evaluateExpression(_ expression: String) -> Int {

}
}
```

Rust Solution:

```
// Problem: Evaluate Valid Expressions
// Difficulty: Hard
// Tags: string, math, hash, stack
//
// Approach: String manipulation with hash map or two pointers
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) for hash map

impl Solution {
pub fn evaluate_expression(expression: String) -> i64 {

}
}
```

Ruby Solution:

```
# @param {String} expression
# @return {Integer}
def evaluate_expression(expression)

end
```

PHP Solution:

```
class Solution {

/**
* @param String $expression
* @return Integer
```

```
*/
function evaluateExpression($expression) {

}
}
```

Dart Solution:

```
class Solution {
int evaluateExpression(String expression) {

}
}
```

Scala Solution:

```
object Solution {
def evaluateExpression(expression: String): Long = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec evaluate_expression(expression :: String.t) :: integer
def evaluate_expression(expression) do

end
end
```

Erlang Solution:

```
-spec evaluate_expression(Expression :: unicode:unicode_binary()) ->
integer().
evaluate_expression(Expression) ->
.
```

Racket Solution:

```
(define/contract (evaluate-expression expression)
(-> string? exact-integer?)
)
```