Problem 150: Evaluate Reverse Polish Notation

Problem Information

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

Problem Description

You are given an array of strings

tokens

that represents an arithmetic expression in a

Reverse Polish Notation

.

Evaluate the expression. Return

an integer that represents the value of the expression

.

Note

that:

The valid operators are

'+'

,

'-'

,

'*'

, and

'/'

.

Each operand may be an integer or another expression.

The division between two integers always

truncates toward zero

.

There will not be any division by zero.

The input represents a valid arithmetic expression in a reverse polish notation.

The answer and all the intermediate calculations can be represented in a

32-bit

integer.

Example 1:

Input:

tokens = ["2","1","+","3","*"]

Output:

9

Explanation:

((2 + 1) * 3) = 9

Example 2:

Input:

tokens = ["4","13","5","/","+"]

Output:

6

Explanation:

(4 + (13 / 5)) = 6

Example 3:

Input:

tokens = ["10","6","9","3","+","-11","*","/","*","17","+","5","+"]

Output:

22

Explanation:

((10 * (6 / ((9 + 3) * -11))) + 17) + 5 = ((10 * (6 / (12 * -11))) + 17) + 5 = ((10 * (6 / -132)) + 17) + 5 = ((10 * 0) + 17) + 5 = (0 + 17) + 5 = 17 + 5 = 22

Constraints:

1 <= tokens.length <= 10

4

tokens[i]

is either an operator:

"+"

,

"-"

,

"*"

, or

"/"

, or an integer in the range

[-200, 200]

.

Code Snippets

C++:

```
class Solution {
public:
int evalRPN(vector<string>& tokens) {

}
};
```

Java:

```
class Solution {
public int evalRPN(String[] tokens) {
```

```
}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

```
function evalRPN(tokens: string[]): number {

};
```

C#:

```
public class Solution {
public int EvalRPN(string[] tokens) {

}
}
```

C:

```
int evalRPN(char** tokens, int tokensSize) {

}
```

Go:

```
func evalRPN(tokens []string) int {

}
```

Kotlin:

```
class Solution {
fun evalRPN(tokens: Array<String>): Int {

}
}
```

Swift:

```
class Solution {
func evalRPN(_ tokens: [String]) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn eval_rpn(tokens: Vec<String>) -> i32 {

}
}
```

Ruby:

```
# @param {String[]} tokens
# @return {Integer}
def eval_rpn(tokens)

end
```

PHP:

```
class Solution {

/**
* @param String[] $tokens
* @return Integer
*/
function evalRPN($tokens) {

}
}
```

Dart:

```
class Solution {
int evalRPN(List<String> tokens) {

}
}
```

Scala:

```
object Solution {
def evalRPN(tokens: Array[String]): Int = {

}
}
```

Elixir:

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

end
end
```

Erlang:

```
-spec eval_rpn(Tokens :: [unicode:unicode_binary()]) -> integer().
eval_rpn(Tokens) ->
.
```

Racket:

```
(define/contract (eval-rpn tokens)
(-> (listof string?) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Evaluate Reverse Polish Notation
* Difficulty: Medium
* Tags: array, string, math, stack
*
* 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 evalRPN(vector<string>& tokens) {

}
};
```

Java Solution:

```
/**
* Problem: Evaluate Reverse Polish Notation
* Difficulty: Medium
* Tags: array, string, math, stack
*
* 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 evalRPN(String[] tokens) {
```

```
}
}
```

Python3 Solution:

```
"""
Problem: Evaluate Reverse Polish Notation
Difficulty: Medium
Tags: array, string, math, stack

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 evalRPN(self, tokens: List[str]) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Evaluate Reverse Polish Notation
* Difficulty: Medium
* Tags: array, string, math, stack
*
* 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[]} tokens
* @return {number}
*/
var evalRPN = function(tokens) {

};
```

TypeScript Solution:

```
/**
* Problem: Evaluate Reverse Polish Notation
* Difficulty: Medium
* Tags: array, string, math, stack
*
* 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 evalRPN(tokens: string[]): number {

};
```

C# Solution:

```
/*
* Problem: Evaluate Reverse Polish Notation
* Difficulty: Medium
* Tags: array, string, math, stack
*
* 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 EvalRPN(string[] tokens) {

}
```

```
}
```

C Solution:

```
/*
* Problem: Evaluate Reverse Polish Notation
* Difficulty: Medium
* Tags: array, string, math, stack
*
* 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 evalRPN(char** tokens, int tokensSize) {

}
```

Go Solution:

```
// Problem: Evaluate Reverse Polish Notation
// Difficulty: Medium
// Tags: array, string, math, stack
//
// 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 evalRPN(tokens []string) int {

}
```

Kotlin Solution:

```
class Solution {
fun evalRPN(tokens: Array<String>): Int {

}
}
```

Swift Solution:

```
class Solution {
func evalRPN(_ tokens: [String]) -> Int {

}
}
```

Rust Solution:

```
// Problem: Evaluate Reverse Polish Notation
// Difficulty: Medium
// Tags: array, string, math, stack
//
// 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 eval_rpn(tokens: Vec<String>) -> i32 {

}
}
```

Ruby Solution:

```
# @param {String[]} tokens
# @return {Integer}
def eval_rpn(tokens)

end
```

PHP Solution:

```
class Solution {

/**
* @param String[] $tokens
* @return Integer
*/
function evalRPN($tokens) {

}
}
```

Dart Solution:

```
class Solution {
int evalRPN(List<String> tokens) {

}
}
```

Scala Solution:

```
object Solution {
def evalRPN(tokens: Array[String]): Int = {

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

```
-spec eval_rpn(Tokens :: [unicode:unicode_binary()]) -> integer().
eval_rpn(Tokens) ->
.
```

Racket Solution:

```
(define/contract (eval-rpn tokens)
(-> (listof string?) exact-integer?)
)
```