

Problem 224: Basic Calculator

Problem Information

Difficulty: Hard

Acceptance Rate: 0.00%

Paid Only: No

Problem Description

Given a string

s

representing a valid expression, implement a basic calculator to evaluate it, and return

the result of the evaluation

Note:

You are

not

allowed to use any built-in function which evaluates strings as mathematical expressions,
such as

`eval()`

Example 1:

Input:

s = "1 + 1"

Output:

2

Example 2:

Input:

s = " 2-1 + 2 "

Output:

3

Example 3:

Input:

s = "(1+(4+5+2)-3)+(6+8)"

Output:

23

Constraints:

$1 \leq s.length \leq 3 * 10^5$

5

s

consists of digits,

'+'

,

'_'

,

'('

,

')'

, and

' '

.

s

represents a valid expression.

'+'

is

not

used as a unary operation (i.e.,

"+1"

and

"+(2 + 3)"

is invalid).

'_'

could be used as a unary operation (i.e.,

"-1"

and

"-(2 + 3)"

is valid).

There will be no two consecutive operators in the input.

Every number and running calculation will fit in a signed 32-bit integer.

Code Snippets

C++:

```
class Solution {  
public:  
    int calculate(string s) {  
  
    }  
};
```

Java:

```
class Solution {  
public int calculate(String s) {  
  
}  
}
```

Python3:

```
class Solution:  
    def calculate(self, s: str) -> int:
```

Python:

```
class Solution(object):  
    def calculate(self, s):  
        """  
        :type s: str  
        :rtype: int  
        """
```

JavaScript:

```
/**  
 * @param {string} s  
 * @return {number}  
 */  
var calculate = function(s) {  
  
};
```

TypeScript:

```
function calculate(s: string): number {  
  
};
```

C#:

```
public class Solution {  
    public int Calculate(string s) {  
  
    }  
}
```

C:

```
int calculate(char* s) {  
  
}
```

Go:

```
func calculate(s string) int {  
  
}
```

Kotlin:

```
class Solution {  
    fun calculate(s: String): Int {  
  
    }  
}
```

Swift:

```
class Solution {  
    func calculate(_ s: String) -> Int {  
  
    }  
}
```

Rust:

```
impl Solution {  
    pub fn calculate(s: String) -> i32 {  
  
    }  
}
```

Ruby:

```
# @param {String} s  
# @return {Integer}  
def calculate(s)  
  
end
```

PHP:

```
class Solution {  
  
    /**  
     * @param String $s  
     * @return Integer  
     */  
    function calculate($s) {  
  
    }
```

```
}
```

Dart:

```
class Solution {  
    int calculate(String s) {  
  
    }  
}
```

Scala:

```
object Solution {  
    def calculate(s: String): Int = {  
  
    }  
}
```

Elixir:

```
defmodule Solution do  
    @spec calculate(s :: String.t) :: integer  
    def calculate(s) do  
  
    end  
end
```

Erlang:

```
-spec calculate(S :: unicode:unicode_binary()) -> integer().  
calculate(S) ->  
.
```

Racket:

```
(define/contract (calculate s)  
  (-> string? exact-integer?)  
)
```

Solutions

C++ Solution:

```
/*
 * Problem: Basic Calculator
 * Difficulty: Hard
 * Tags: string, math, stack
 *
 * Approach: String manipulation with hash map or two pointers
 * Time Complexity: O(n) or O(n log n)
 * Space Complexity: O(1) to O(n) depending on approach
 */

class Solution {
public:
    int calculate(string s) {
}
```

Java Solution:

```
/**
 * Problem: Basic Calculator
 * Difficulty: Hard
 * Tags: string, math, stack
 *
 * Approach: String manipulation with hash map or two pointers
 * Time Complexity: O(n) or O(n log n)
 * Space Complexity: O(1) to O(n) depending on approach
 */

class Solution {
    public int calculate(String s) {
}
```

Python3 Solution:

```
"""
Problem: Basic Calculator
Difficulty: Hard
Tags: string, math, stack
```

```
Approach: String manipulation with hash map or two pointers
Time Complexity: O(n) or O(n log n)
Space Complexity: O(1) to O(n) depending on approach
"""

```

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

Python Solution:

```
class Solution(object):
    def calculate(self, s):
        """
        :type s: str
        :rtype: int
        """

```

JavaScript Solution:

```
/**
 * Problem: Basic Calculator
 * Difficulty: Hard
 * Tags: string, math, stack
 *
 * Approach: String manipulation with hash map or two pointers
 * Time Complexity: O(n) or O(n log n)
 * Space Complexity: O(1) to O(n) depending on approach
 */

/**
 * @param {string} s
 * @return {number}
 */
var calculate = function(s) {

};
```

TypeScript Solution:

```

/**
 * Problem: Basic Calculator
 * Difficulty: Hard
 * Tags: string, math, stack
 *
 * Approach: String manipulation with hash map or two pointers
 * Time Complexity: O(n) or O(n log n)
 * Space Complexity: O(1) to O(n) depending on approach
 */

function calculate(s: string): number {

};

```

C# Solution:

```

/*
 * Problem: Basic Calculator
 * Difficulty: Hard
 * Tags: string, math, stack
 *
 * Approach: String manipulation with hash map or two pointers
 * Time Complexity: O(n) or O(n log n)
 * Space Complexity: O(1) to O(n) depending on approach
 */

public class Solution {
    public int Calculate(string s) {

    }
}

```

C Solution:

```

/*
 * Problem: Basic Calculator
 * Difficulty: Hard
 * Tags: string, math, stack
 *
 * Approach: String manipulation with hash map or two pointers
 * Time Complexity: O(n) or O(n log n)
 * Space Complexity: O(1) to O(n) depending on approach

```

```
*/  
  
int calculate(char* s) {  
  
}
```

Go Solution:

```
// Problem: Basic Calculator  
// Difficulty: Hard  
// Tags: string, math, stack  
//  
// Approach: String manipulation with hash map or two pointers  
// Time Complexity: O(n) or O(n log n)  
// Space Complexity: O(1) to O(n) depending on approach  
  
func calculate(s string) int {  
  
}
```

Kotlin Solution:

```
class Solution {  
    fun calculate(s: String): Int {  
  
    }  
}
```

Swift Solution:

```
class Solution {  
    func calculate(_ s: String) -> Int {  
  
    }  
}
```

Rust Solution:

```
// Problem: Basic Calculator  
// Difficulty: Hard  
// Tags: string, math, stack
```

```
//  
// Approach: String manipulation with hash map or two pointers  
// Time Complexity: O(n) or O(n log n)  
// Space Complexity: O(1) to O(n) depending on approach  
  
impl Solution {  
    pub fn calculate(s: String) -> i32 {  
  
    }  
}
```

Ruby Solution:

```
# @param {String} s  
# @return {Integer}  
def calculate(s)  
  
end
```

PHP Solution:

```
class Solution {  
  
    /**  
     * @param String $s  
     * @return Integer  
     */  
    function calculate($s) {  
  
    }  
}
```

Dart Solution:

```
class Solution {  
    int calculate(String s) {  
  
    }  
}
```

Scala Solution:

```
object Solution {  
    def calculate(s: String): Int = {  
        }  
    }  
}
```

Elixir Solution:

```
defmodule Solution do  
  @spec calculate(s :: String.t) :: integer  
  def calculate(s) do  
  
  end  
end
```

Erlang Solution:

```
-spec calculate(S :: unicode:unicode_binary()) -> integer().  
calculate(S) ->  
.
```

Racket Solution:

```
(define/contract (calculate s)  
  (-> string? exact-integer?)  
)
```