Problem 3612: Process String with Special Operations I

Problem Information

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

Problem Description

You are given a string

s

consisting of lowercase English letters and the special characters:

*

,

#

, and

%

.

Build a new string

result

by processing

s

according to the following rules from left to right:

If the letter is a

lowercase

English letter append it to

result

.

A

'*'

removes

the last character from

result

, if it exists.

A

'#'

duplicates

the current

result

and

appends

it to itself.

A

'%'

reverses

the current

result

.

Return the final string

result

after processing all characters in

s

.

Example 1:

Input:

s = "a#b%*"

Output:

"ba"

Explanation:

i

s[i]

Operation

Current

result

0

'a'

Append

'a'

"a"

1

'#'

Duplicate

result

"aa"

2

'b'

Append

'b'

"aab"

3

'%'

Reverse

result

"baa"

4

'*'

Remove the last character

"ba"

Thus, the final

result

is

"ba"

.

Example 2:

Input:

s = "z*#"

Output:

""

Explanation:

i

s[i]

Operation

Current

result

0

'z'

Append

'z'

"z"

1

'*'

Remove the last character

""

2

'#'

Duplicate the string

""

Thus, the final

result

is

""

.

Constraints:

1 <= s.length <= 20

s

consists of only lowercase English letters and special characters

*

,

#

, and

%

.

Code Snippets

C++:

```
class Solution {
public:
string processStr(string s) {

}
};
```

Java:

```
class Solution {
public String processStr(String s) {

}
}
```

Python3:

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

Python:

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

JavaScript:

```
/**
* @param {string} s
* @return {string}
*/
var processStr = function(s) {

};
```

TypeScript:

```
function processStr(s: string): string {

};
```

C#:

```
public class Solution {
public string ProcessStr(string s) {

}
}
```

C:

```
char* processStr(char* s) {

}
```

Go:

```
func processStr(s string) string {

}
```

Kotlin:

```
class Solution {
fun processStr(s: String): String {

}
}
```

Swift:

```
class Solution {
func processStr(_ s: String) -> String {

}
}
```

Rust:

```
impl Solution {
pub fn process_str(s: String) -> String {

}
}
```

Ruby:

```
# @param {String} s
# @return {String}
def process_str(s)

end
```

PHP:

```
class Solution {

/**
* @param String $s
* @return String
*/
function processStr($s) {

}
}
```

Dart:

```
class Solution {
String processStr(String s) {

}
}
```

Scala:

```
object Solution {
def processStr(s: String): String = {

}
}
```

Elixir:

```
defmodule Solution do
@spec process_str(s :: String.t) :: String.t
def process_str(s) do

end
end
```

Erlang:

```
-spec process_str(S :: unicode:unicode_binary()) -> unicode:unicode_binary().
process_str(S) ->
.
```

Racket:

```
(define/contract (process-str s)
(-> string? string?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Process String with Special Operations I
* Difficulty: Medium
* Tags: string
*
* 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:
string processStr(string s) {

}
};
```

Java Solution:

```
/**
* Problem: Process String with Special Operations I
* Difficulty: Medium
* Tags: string
*
* 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 String processStr(String s) {
```

```
}
}
```

Python3 Solution:

```
"""
Problem: Process String with Special Operations I
Difficulty: Medium
Tags: string

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 processStr(self, s: str) -> str:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Process String with Special Operations I
* Difficulty: Medium
* Tags: string
*
* 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 {string}
*/
var processStr = function(s) {

};
```

TypeScript Solution:

```
/**
* Problem: Process String with Special Operations I
* Difficulty: Medium
* Tags: string
*
* 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 processStr(s: string): string {

};
```

C# Solution:

```
/*
* Problem: Process String with Special Operations I
* Difficulty: Medium
* Tags: string
*
* 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 string ProcessStr(string s) {

}
```

```
}
```

C Solution:

```
/*
* Problem: Process String with Special Operations I
* Difficulty: Medium
* Tags: string
*
* 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
*/

char* processStr(char* s) {

}
```

Go Solution:

```
// Problem: Process String with Special Operations I
// Difficulty: Medium
// Tags: string
//
// 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 processStr(s string) string {

}
```

Kotlin Solution:

```
class Solution {
fun processStr(s: String): String {

}
}
```

Swift Solution:

```
class Solution {
func processStr(_ s: String) -> String {

}
}
```

Rust Solution:

```
// Problem: Process String with Special Operations I
// Difficulty: Medium
// Tags: string
//
// 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 process_str(s: String) -> String {

}
}
```

Ruby Solution:

```
# @param {String} s
# @return {String}
def process_str(s)

end
```

PHP Solution:

```
class Solution {

/**
* @param String $s
* @return String
*/
function processStr($s) {

}
}
```

Dart Solution:

```
class Solution {
String processStr(String s) {

}
}
```

Scala Solution:

```
object Solution {
def processStr(s: String): String = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec process_str(s :: String.t) :: String.t
def process_str(s) do

end
end
```

Erlang Solution:

```
-spec process_str(S :: unicode:unicode_binary()) -> unicode:unicode_binary().
process_str(S) ->
.
```

Racket Solution:

```
(define/contract (process-str s)
(-> string? string?)
)
```