Problem 2810: Faulty Keyboard

Problem Information

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

Problem Description

Your laptop keyboard is faulty, and whenever you type a character

'i'

on it, it reverses the string that you have written. Typing other characters works as expected.

You are given a

0-indexed

string

s

, and you type each character of

s

using your faulty keyboard.

Return

the final string that will be present on your laptop screen.

Example 1:

Input:

s = "string"

Output:

"rtsng"

Explanation:

After typing first character, the text on the screen is "s". After the second character, the text is "st". After the third character, the text is "str". Since the fourth character is an 'i', the text gets reversed and becomes "rts". After the fifth character, the text is "rtsn". After the sixth character, the text is "rtsng". Therefore, we return "rtsng".

Example 2:

Input:

s = "poiinter"

Output:

"ponter"

Explanation:

After the first character, the text on the screen is "p". After the second character, the text is "po". Since the third character you type is an 'i', the text gets reversed and becomes "op". Since the fourth character you type is an 'i', the text gets reversed and becomes "po". After the fifth character, the text is "pon". After the sixth character, the text is "pont". After the seventh character, the text is "ponte". After the eighth character, the text is "ponter". Therefore, we return "ponter".

Constraints:

1 <= s.length <= 100

s

consists of lowercase English letters.

s[0] != 'i'

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

```
/**
* @param {string} s
```

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

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

```
char* finalString(char* s) {

}
```

Go:

```
func finalString(s string) string {

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

```
(define/contract (final-string s)
(-> string? string?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Faulty Keyboard
* Difficulty: Easy
* 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 finalString(string s) {

}
};
```

Java Solution:

```
/**
* Problem: Faulty Keyboard
* Difficulty: Easy
* 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 finalString(String s) {

}
}
```

Python3 Solution:

```
"""
Problem: Faulty Keyboard
Difficulty: Easy
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 finalString(self, s: str) -> str:
```

```
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Faulty Keyboard
* Difficulty: Easy
* 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 finalString = function(s) {

};
```

TypeScript Solution:

```
/**
* Problem: Faulty Keyboard
* Difficulty: Easy
* 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 finalString(s: string): string {

};
```

C# Solution:

```
/*
* Problem: Faulty Keyboard
* Difficulty: Easy
* 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 FinalString(string s) {

}
}
```

C Solution:

```
/*
* Problem: Faulty Keyboard
* Difficulty: Easy
* 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* finalString(char* s) {

}
```

Go Solution:

```
// Problem: Faulty Keyboard
// Difficulty: Easy
// 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 finalString(s string) string {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Faulty Keyboard
// Difficulty: Easy
// 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 final_string(s: String) -> String {
```

```
}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

```
(define/contract (final-string s)
(-> string? string?)
)
```