Problem 439: Ternary Expression Parser

Problem Information

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

Problem Description

Given a string

expression

representing arbitrarily nested ternary expressions, evaluate the expression, and return

the result of it

.

You can always assume that the given expression is valid and only contains digits,

'?'

,

':'

,

'T'

, and

'F'

where

'T'

is true and

'F'

is false. All the numbers in the expression are

one-digit

numbers (i.e., in the range

[0, 9]

).

The conditional expressions group right-to-left (as usual in most languages), and the result of the expression will always evaluate to either a digit,

'T'

or

'F'

.

Example 1:

Input:

expression = "T?2:3"

Output:

"2"

Explanation:

If true, then result is 2; otherwise result is 3.

Example 2:

Input:

expression = "F?1:T?4:5"

Output:

"4"

Explanation:

The conditional expressions group right-to-left. Using parenthesis, it is read/evaluated as: "(F ? 1 : (T ? 4 : 5))" --> "(F ? 1 : 4)" --> "4" or "(F ? 1 : (T ? 4 : 5))" --> "(T ? 4 : 5)" --> "4"

Example 3:

Input:

expression = "T?T?F:5:3"

Output:

"F"

Explanation:

The conditional expressions group right-to-left. Using parenthesis, it is read/evaluated as: "(T ? (T ? F : 5) : 3)" --> "(T ? F : 3)" --> "F" "(T ? (T ? F : 5) : 3)" --> "(T ? F : 5)" --> "F"

Constraints:

5 <= expression.length <= 10

4

expression

consists of digits,

'T'

,

'F'

,

'?'

, and

':'

.

It is

guaranteed

that

expression

is a valid ternary expression and that each number is a

one-digit number

.

Code Snippets

C++:

```
class Solution {
public:
string parseTernary(string expression) {

}
};
```

Java:

```
class Solution {
public String parseTernary(String expression) {

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

```
function parseTernary(expression: string): string {
```

```
};
```

C#:

```
public class Solution {
public string ParseTernary(string expression) {

}
}
```

C:

```
char* parseTernary(char* expression) {

}
```

Go:

```
func parseTernary(expression string) string {

}
```

Kotlin:

```
class Solution {
fun parseTernary(expression: String): String {

}
}
```

Swift:

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

}
}
```

Rust:

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

```
}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

```
class Solution {
String parseTernary(String expression) {

}
}
```

Scala:

```
object Solution {
def parseTernary(expression: String): String = {

}
}
```

Elixir:

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

end
end
```

Erlang:

```
-spec parse_ternary(Expression :: unicode:unicode_binary()) ->
unicode:unicode_binary().
parse_ternary(Expression) ->
.
```

Racket:

```
(define/contract (parse-ternary expression)
(-> string? string?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Ternary Expression Parser
* Difficulty: Medium
* Tags: string, 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:
string parseTernary(string expression) {

}
};
```

Java Solution:

```
/**
* Problem: Ternary Expression Parser
* Difficulty: Medium
* Tags: string, 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 String parseTernary(String expression) {

}
}
```

Python3 Solution:

```
"""
Problem: Ternary Expression Parser
Difficulty: Medium
Tags: string, 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 parseTernary(self, expression: str) -> str:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Ternary Expression Parser
* Difficulty: Medium
* Tags: string, 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} expression
* @return {string}
*/
var parseTernary = function(expression) {

};
```

TypeScript Solution:

```
/**
* Problem: Ternary Expression Parser
* Difficulty: Medium
* Tags: string, 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 parseTernary(expression: string): string {

};
```

C# Solution:

```
/*
* Problem: Ternary Expression Parser
* Difficulty: Medium
* Tags: string, 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 string ParseTernary(string expression) {

}
}
```

C Solution:

```
/*
* Problem: Ternary Expression Parser
* Difficulty: Medium
* Tags: string, 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
*/

char* parseTernary(char* expression) {

}
```

Go Solution:

```
// Problem: Ternary Expression Parser
// Difficulty: Medium
// Tags: string, 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 parseTernary(expression string) string {

}
```

Kotlin Solution:

```
class Solution {
fun parseTernary(expression: String): String {

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Ternary Expression Parser
// Difficulty: Medium
// Tags: string, 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 parse_ternary(expression: String) -> String {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

```
class Solution {
String parseTernary(String expression) {

}
}
```

Scala Solution:

```
object Solution {
def parseTernary(expression: String): String = {

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

```
-spec parse_ternary(Expression :: unicode:unicode_binary()) ->
unicode:unicode_binary().
parse_ternary(Expression) ->
.
```

Racket Solution:

```
(define/contract (parse-ternary expression)
(-> string? string?)
)
```