Problem 722: Remove Comments

Problem Information

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

Problem Description

Given a C++ program, remove comments from it. The program source is an array of strings

source

where

source[i]

is the

i

th

line of the source code. This represents the result of splitting the original source code string by the newline character

'\n'

.

In C++, there are two types of comments, line comments, and block comments.

The string

"//"

denotes a line comment, which represents that it and the rest of the characters to the right of it in the same line should be ignored.

The string

"/*"

denotes a block comment, which represents that all characters until the next (non-overlapping) occurrence of

"*/"

should be ignored. (Here, occurrences happen in reading order: line by line from left to right.) To be clear, the string

"/*/"

does not yet end the block comment, as the ending would be overlapping the beginning.

The first effective comment takes precedence over others.

For example, if the string

"//"

occurs in a block comment, it is ignored.

Similarly, if the string

"/*"

occurs in a line or block comment, it is also ignored.

If a certain line of code is empty after removing comments, you must not output that line: each string in the answer list will be non-empty.

There will be no control characters, single quote, or double quote characters.

For example,

source = "string s = "/* Not a comment. */";"

will not be a test case.

Also, nothing else such as defines or macros will interfere with the comments.

It is guaranteed that every open block comment will eventually be closed, so

"/*"

outside of a line or block comment always starts a new comment.

Finally, implicit newline characters can be deleted by block comments. Please see the examples below for details.

After removing the comments from the source code, return

the source code in the same format

.

Example 1:

Input:

source = ["/*Test program */", "int main()", "{ ", " // variable declaration ", "int a, b, c;", "/* This is a test", " multiline ", " comment for ", " testing */", "a = b + c;", "}"]

Output:

["int main()","{ "," ","int a, b, c;","a = b + c;","}"]

Explanation:

The line by line code is visualized as below: /*Test program */ int main() { // variable declaration int a, b, c; /* This is a test multiline comment for testing */ a = b + c; } The string /* denotes a block comment, including line 1 and lines 6-9. The string // denotes line 4 as comments. The line by line output code is visualized as below: int main() { int a, b, c; a = b + c; }

Example 2:

Input:

source = ["a/*comment", "line", "more_comment*/b"]

Output:

["ab"]

Explanation:

The original source string is "a/*comment\nline\nmore_comment*/b", where we have bolded the newline characters. After deletion, the implicit newline characters are deleted, leaving the string "ab", which when delimited by newline characters becomes ["ab"].

Constraints:

1 <= source.length <= 100

0 <= source[i].length <= 80

source[i]

consists of printable

ASCII

characters.

Every open block comment is eventually closed.

There are no single-quote or double-quote in the input.

Code Snippets

C++:

```
class Solution {
public:
vector<string> removeComments(vector<string>& source) {

}
};
```

Java:

```
class Solution {
public List<String> removeComments(String[] source) {

}
}
```

Python3:

```
class Solution:
def removeComments(self, source: List[str]) -> List[str]:
```

Python:

```
class Solution(object):
def removeComments(self, source):
"""
:type source: List[str]
:rtype: List[str]
"""
```

JavaScript:

```
/**
* @param {string[]} source
* @return {string[]}
*/
var removeComments = function(source) {

};
```

TypeScript:

```
function removeComments(source: string[]): string[] {
```

```
};
```

C#:

```
public class Solution {
public IList<string> RemoveComments(string[] source) {

}
}
```

C:

```
/**
* Note: The returned array must be malloced, assume caller calls free().
*/
char** removeComments(char** source, int sourceSize, int* returnSize) {

}
```

Go:

```
func removeComments(source []string) []string {

}
```

Kotlin:

```
class Solution {
fun removeComments(source: Array<String>): List<String> {

}
}
```

Swift:

```
class Solution {
func removeComments(_ source: [String]) -> [String] {

}
}
```

Rust:

```
impl Solution {
pub fn remove_comments(source: Vec<String>) -> Vec<String> {

}
}
```

Ruby:

```
# @param {String[]} source
# @return {String[]}
def remove_comments(source)

end
```

PHP:

```
class Solution {

/**
* @param String[] $source
* @return String[]
*/
function removeComments($source) {

}
}
```

Dart:

```
class Solution {
List<String> removeComments(List<String> source) {

}
}
```

Scala:

```
object Solution {
def removeComments(source: Array[String]): List[String] = {

}
}
```

Elixir:

```
defmodule Solution do
@spec remove_comments(source :: [String.t]) :: [String.t]
def remove_comments(source) do

end
end
```

Erlang:

```
-spec remove_comments(Source :: [unicode:unicode_binary()]) ->
[unicode:unicode_binary()].
remove_comments(Source) ->
.
```

Racket:

```
(define/contract (remove-comments source)
(-> (listof string?) (listof string?))
)
```

Solutions

C++ Solution:

```
/*
* Problem: Remove Comments
* Difficulty: Medium
* Tags: array, string
*
* 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:
vector<string> removeComments(vector<string>& source) {

}
};
```

Java Solution:

```
/**
* Problem: Remove Comments
* Difficulty: Medium
* Tags: array, string
*
* 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 List<String> removeComments(String[] source) {

}
}
```

Python3 Solution:

```
"""
Problem: Remove Comments
Difficulty: Medium
Tags: array, string

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 removeComments(self, source: List[str]) -> List[str]:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def removeComments(self, source):
"""
:type source: List[str]
:rtype: List[str]
```

```
"""
```

JavaScript Solution:

```
/**
* Problem: Remove Comments
* Difficulty: Medium
* Tags: array, string
*
* 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[]} source
* @return {string[]}
*/
var removeComments = function(source) {

};
```

TypeScript Solution:

```
/**
* Problem: Remove Comments
* Difficulty: Medium
* Tags: array, string
*
* 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 removeComments(source: string[]): string[] {

};
```

C# Solution:

```
/*
* Problem: Remove Comments
* Difficulty: Medium
* Tags: array, string
*
* 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 IList<string> RemoveComments(string[] source) {

}
}
```

C Solution:

```
/*
* Problem: Remove Comments
* Difficulty: Medium
* Tags: array, string
*
* 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
*/

/**
* Note: The returned array must be malloced, assume caller calls free().
*/
char** removeComments(char** source, int sourceSize, int* returnSize) {

}
```

Go Solution:

```
// Problem: Remove Comments
// Difficulty: Medium
// Tags: array, string
//
// 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 removeComments(source []string) []string {

}
```

Kotlin Solution:

```
class Solution {
fun removeComments(source: Array<String>): List<String> {

}
}
```

Swift Solution:

```
class Solution {
func removeComments(_ source: [String]) -> [String] {

}
}
```

Rust Solution:

```
// Problem: Remove Comments
// Difficulty: Medium
// Tags: array, string
//
// 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 remove_comments(source: Vec<String>) -> Vec<String> {

}
}
```

Ruby Solution:

```
# @param {String[]} source
# @return {String[]}
def remove_comments(source)

end
```

PHP Solution:

```
class Solution {

/**
* @param String[] $source
* @return String[]
*/
function removeComments($source) {

}
}
```

Dart Solution:

```
class Solution {
List<String> removeComments(List<String> source) {

}
}
```

Scala Solution:

```
object Solution {
def removeComments(source: Array[String]): List[String] = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec remove_comments(source :: [String.t]) :: [String.t]
def remove_comments(source) do

end
```

```
end
```

Erlang Solution:

```
-spec remove_comments(Source :: [unicode:unicode_binary()]) ->
[unicode:unicode_binary()].
remove_comments(Source) ->
.
```

Racket Solution:

```
(define/contract (remove-comments source)
(-> (listof string?) (listof string?))
)
```