Remove Comments

Given a C++ program, remove comments from it. The program `source` is an array 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 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: 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;", "}"]

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;
}

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

The line by line code is visualized as below:
int main()
{

int a, b, c;
a = b + c;
}

Explanation:
The string
```

```
/*
```

```
 denotes a block comment, including line 1 and lines 6-9. The string
```

```
//
```

```
 denotes line 4 as comments.
```

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 characte
rs are deleted, leaving the string "ab", which when delimited by newline characters b
ecomes ["ab"].
```

Note:

- The length of `source` is in the range `[1, 100]`.
- The length of `source[i]` is in the range `[0, 80]`.
- Every open block comment is eventually closed.
- There are no single-quote, double-quote, or control characters in the source code.

Solution 1

This problem *begs* for a regular expression solution...

Ruby:

```
def remove_comments(source)
  source.join($/).gsub(%r(//.*|/\*(.|\n)*?\*/), '').split($/).reject(&:empty?)
end
```

Python:

```
def removeComments(self, source):
    return filter(None, re.sub('//.*|/\*(.|\n)*?\*/', '', '\n'.join(source)).split(
'\n'))
```

written by StefanPochmann original link here

Solution 2

```cpp
class Solution {
public:
    vector<string> removeComments(vector<string>& s) {
        vector<string> ans;
        bool inBlock = false;
        string sf;
        for (auto &t:s) {
            for (int i = 0; i < t.size();) {
                if (!inBlock) {
                    if (i + 1 == t.size()) sf += t[i++];
                    else {
                        string m = t.substr(i,2);
                        if (m == "/*") inBlock = 1, i+=2;
                        else if (m == "//") break;
                        else sf += t[i++];
                    }
                }
                else {
                    if (i + 1 == t.size()) i++;
                    else {
                        string m = t.substr(i,2);
                        if (m == "*/") inBlock = 0, i+=2;
                        else i++;
                    }
                }
            }
            if (sf.size() && !inBlock) ans.push_back(sf), sf = "";
        }
        return ans;
    }
};
```

written by elastico original link here

Solution 3

We only need to check for two things:

1. If we see '//' we stop reading the current line, and add whatever characters we have seen to the result.
2. If we see '/*' then we start the multiline comment mode and we keep on ignoring characters until we see '*/'.
3. If the current character is neither of the above two and the multiline comment mode is off, then we add that character to the current line.

Once we parse one line (source[i]), then if the mode is off, we add the currently generated line (StringBuilder) to the result and repeat for source[i + 1].

We need to be careful not to insert empty lines in the result.

```
class Solution {
    public List<String> removeComments(String[] source) {
        List<String> res = new ArrayList<>();
        StringBuilder sb = new StringBuilder();
        boolean mode = false;
        for (String s : source) {
            for (int i = 0; i < s.length(); i++) {
                if (mode) {
                    if (s.charAt(i) == '*' && i < s.length() - 1 && s.charAt(i + 1)
== '/') {
                        mode = false;
                        i++;        //skip '/' on next iteration of i
                    }
                }
                else {
                    if (s.charAt(i) == '/' && i < s.length() - 1 && s.charAt(i + 1)
== '/') {
                        break;      //ignore remaining characters on line s
                    }
                    else if (s.charAt(i) == '/' && i < s.length() - 1 && s.charAt(i
+ 1) == '*') {
                        mode = true;
                        i++;           //skip '*' on next iteration of i
                    }
                    else    sb.append(s.charAt(i));     //not a comment
                }
            }
            if (!mode && sb.length() > 0) {
                res.add(sb.toString());
                sb = new StringBuilder();   //reset for next line of source code
            }
        }
        return res;
    }
}
```

Thanks to @ihaveayaya for suggestion to remove some duplicate code.

written by nrl original link here