题目描述

有一个文件，包含以一定规则写作的文本，请统计文件中包含的文本数量。

规则如下：

1. 文本以";"分隔，最后一条可以没有";"，但空文本不能算语句，比如"COMMAND A; ;"只能算一条语句。注意，无字符/空白字符/ 制表符 都算作"空"文本；

2. 文本可以跨行，比如下面，是一条文本，而不是三条；

COMMAND A

AND

COMMAND B;

3. 文本支持字符串，字符串为成对的单引号(')或者成对的双引号(")，字符串可能出现用 转义字符 (\)处理的单双引号("your input is\"")和转义字符本身，比如

COMMAND A "Say \"hello\"";

4. 支持注释，可以出现在字符串之外的任意位置注释以"-"开头，到换行结束，比如：

COMMAND A; -this is comment

COMMAND -comment

A AND COMMAND B;

注意字符串内的"-"，不是注释。

输入描述

文本文件

输出描述

包含的文本数量

用例

输入	COMMAND TABLE IF EXISTS "UNITED STATE"; COMMAND A GREAT (ID ADSAB, download_length INTE-GER, -- test file_name TEXT, guid TEXT, mime_type TEXT, notifica-tionid INTEGER, original_file_name TEXT, pause_reason_type INTEGER, resumable_flag INTEGER, start_time INTEGER, state INTEGER, folder TEXT, path TEXT, total_length INTE-GER, url TEXT);
输出	2
说明	无

题目解析

题目的意思其实就是要求，输入的文本中有多少个有效的分号。

题目中给出了无效分号的情况：

1、比如"COMMAND A; ;"只能算一条语句。如果分号分隔的是空白，那么分号就是无效分号。比如例子中第二个分号。

2、注释后面的分号。比如 “how are you; - are you ok; ” 中第二个分号就是无效分号。

另外，分号能不能出现在引号中呢?

按照题目意思：3. 文本支持字符串，字符串为成对的单引号(')或者成对的双引号(")

即在一个文本中，引号都是成对出现的，而分号恰恰是一个文本的结束标志，因此分号不可能出现在成对的引号内。

而成对的引号内又有可能出现 ： 成对或不成对的转义引号

比如："your input is\""。

因此，为了找到有效分号，我们需要先对每行文本做如下处理：

1、将转义引号干掉，方便后面处理成对引号，这里干掉转移引号的方法是使用正则：

```
line.replaceAll(/\\[\"\']/g, "")
```

上面代码含义是 找到字符串中 \" 或者 \' 替换为 ''

2、将成对引号干掉，方便后面处理注释，这里干掉干掉成对引号的方法也是使用正则：

```
line.replaceAll(/([\"\']).*?\1/g, "")
```

上面代码正则中， \1是使用捕获组1的内容，而捕获组即()中的内容，如上面正则的捕获组就是([\"\'])，即成对引号的一端引号，如果捕获组匹配到的是双引号，则\1也匹配双引号，如果捕获组匹配到是单引号，则\1也匹配单引号。

另外捕获组和\1之间的正则是：.*?

这个是非贪婪模式匹配一段任意内容。需要注意和贪婪模式 .* 的区别。

比如："123"456"789"，如果使用 .* 匹配，则会匹配出"123"456"789"，如果使用 .*? 匹配，则只会匹配出"123"。

在处理完，转义引号和成对引号后，我们就可以去除文本行的注释，

我们只需要找到文本行中第一个"-"，将它和其后面的内容删除即可。

之后，我们先找每行文本内是否有分号；如果有，则将文本按照";"分割，过滤掉空白文本（使用正则/^\s*$/去匹配空白文本），剩余部分的数量就是有效分号的数量。

按照这种逻辑求出每行文本的有效分号数量，累加求和，但是这里的和不一定是最终结果。

因为，题目说：文本以";"分隔，最后一条可以没有";"

因此，我们还要检查输入的最后一行是否有分号，如果没有分号，则还需要在前面求得和的基础上+1。此时和才是最终题解。

JavaScript算法源码

```javascript
/* JavaScript Node ACM模式 控制台输入获取 */
const readline = require("readline");

const rl = readline.createInterface({
  input: process.stdin,
  output: process.stdout,
});

const lines = [];
rl.on("line", (line) => {
  if (line === "") {
    console.log(getResult(lines));
    lines.length = 0;
  } else {
    lines.push(line);
  }
});

```

```javascript
function getResult(lines) {
  let count = 0;
  lines.forEach((line) => {
    line = line.replaceAll(/\\[\"\']/g, "").replaceAll(/([\"\']).*?\1/g, "");

    const idx = line.indexOf("-");

    if (idx !== -1) {
      line = line.substring(0, idx);
    }

    if (line.indexOf(";") !== -1) {
      count += line.split(";").filter((str) => !/^\s*$/.test(str)).length;
    }
  });

  lines.at(-1).indexOf(";") === -1 ? count++ : null;

  return count;
}
```

Java算法源码

```java
import java.util.Arrays;
import java.util.LinkedList;
import java.util.Scanner;
import java.util.regex.Pattern;

public class Main {
  // 输入获取
  public static void main(String[] args) {
    Scanner sc = new Scanner(System.in);

    LinkedList<String> lines = new LinkedList<>();
    while (sc.hasNextLine()) {
      String line = sc.nextLine();

      if ("".equals(line)) {
        System.out.println(getResult(lines));
        sc.close();
        break;
      } else {
        lines.add(line);
      }
    }
  }

```

```java
  // 算法入口
  public static int getResult(LinkedList<String> lines) {
    int count = 0;

    for (String line : lines) {
      String tmp = line.replaceAll("\\[\"']", "").replaceAll("([\"']).*?\1", "");

      int idx = tmp.indexOf("-");

      if (idx != -1) {
        tmp = tmp.substring(0, idx);
      }

      Pattern compile = Pattern.compile("^\\s*$");

      if (tmp.contains(";")) {
        count +=
            Arrays.stream(tmp.split(";")).filter(str -> !compile.matcher(str).matches()).count();
      }
    }

    if (!lines.getLast().contains(";")) {
      count++;
    }

    return count;
  }
}
```

Python算法源码

```python
# 算法入口
import re


def getResult(lines):
    count = 0

    for line in lines:
        tmp = re.sub(r"([\"\']).*?\1", "", re.sub(r"\\[\"\']", "", line))

        idx = tmp.find("-")

        if idx != -1:
            tmp = tmp[:idx]

        reg = re.compile(r"^\s*$")
        if tmp.find(";") != -1:
            count += len(list(filter(lambda x: not reg.match(x), tmp.split(";"))))

    if lines[-1].find(";") == -1:
        count += 1

    return count


# 输入获取
lines = []

while True:
    line = input()

    if line != "":
        lines.append(line)
    else:
        print(getResult(lines))
        break
```