实验遇到的问题&解决方法

实验过程

根据 lex 的语法，通过编写正则表达式对助教提供的实验模板进行补完。整体的难度可以接受，大部分是体力活，难度曲线较为平缓。

处理头文件

问题描述

预处理过后的代码，会将用到的头文件转化为类似于如下所示的信息。如何利用这些信息进行正确的词法分析是一个问题。

```
# 1 "<stdin>"
# 1 "<built-in>" 1
# 1 "<built-in>" 3
# 321 "<built-in>" 3
# 1 "<command line>" 1
# 1 "<built-in>" 2
# 1 "<stdin>" 2
```

解决方法

通过查阅资料我们发现，`# num_1 "<filename>" num_2` 中，`num_1` 表示当前我们处于`filename`这一文件的行数，`num_2`表示当前准备执行的操作。`num_2`的具体含义如下：

- `1` - 文件的起始
- `2` - 返回前一个文件
- `3` - 下面的内容来自于系统头文件
- `4` - 下面的内容来自于隐式的 `extern C` 代码块

处理字符串

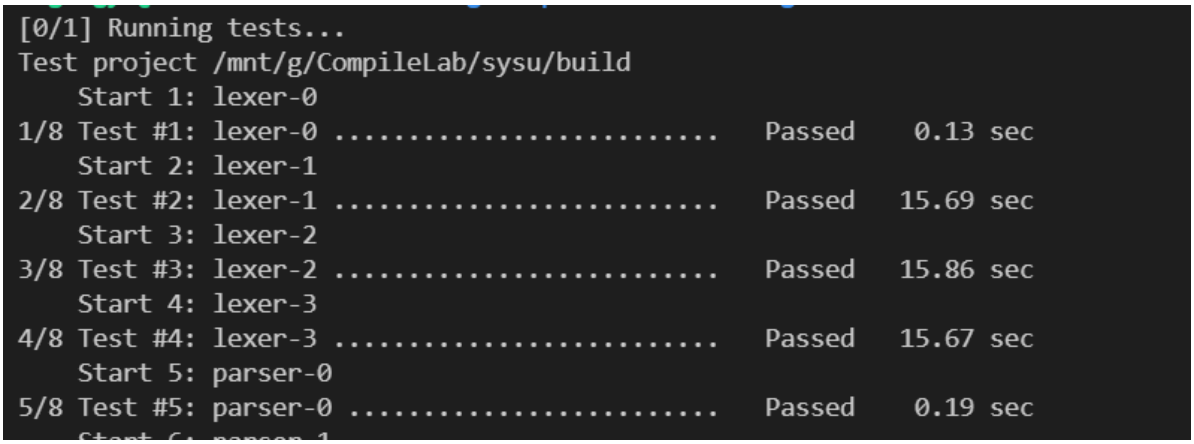
问题描述

最开始我直接是用最简单的方法来匹配的，即`\".*\"`。但是这种做法有两个问题，一个是当一行同时有多个字符串的时候，根据最长匹配原则，会直接将多个字符串合并在一起了。还有一个问题就是字符串中的转义字符 `\"`匹配的时候也会出问题。

解决方法

这个问题产生的原因个人觉得是我们对字符串的严格定义不清晰。C 语言中字符串应该以 `"`开头，并以`"`结尾。要注意的是，结尾的 `"` 的前面不应该包含 `\`，即应该要区别结尾 `"` 与字符串中转义字符`\"`。我们使用如下的正则表达式来匹配字符串：`\"(([^\"\n])?(\\\")?)*\"` 。对于包含在两个`\"`之间的字符串内容，我们定义为 `(([^\"\n])?(\\\")?)*"`，意思是我们先从全体字符集中去除掉`"` 与 `\n` ，这两个字符是不能出现在字符串内容中的；然后再添加上 `\"`，作为一个类似于补丁的东西。使用这种方法我们就可以正确地匹配字符串了。

评测结果


[0/1] Running tests...
Test project /mnt/g/CompileLab/sysu/build
Start 1: lexer-0
1/8 Test #1: lexer-0 ..... Passed 0.13 sec
Start 2: lexer-1
2/8 Test #2: lexer-1 ..... Passed 15.69 sec
Start 3: lexer-2
3/8 Test #3: lexer-2 ..... Passed 15.86 sec
Start 4: lexer-3
4/8 Test #4: lexer-3 ..... Passed 15.67 sec
Start 5: parser-0
5/8 Test #5: parser-0 ..... Passed 0.19 sec