**中国矿业大学计算机学院**

**2019级本科生课程设计报告**

课程名称 系统软件开发实践

报告时间 2022年2月25日

学生姓名 胡钧耀

学 号 06192081

专 业 计算机科学与技术

任课教师 张博

**成绩考核**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 编号 | 课程教学目标 | | | 占比 | | 得分 |
| 1 | **目标1：**针对编译器中词法分析器软件要求，能够分析系统需求，并采用FLEX脚本语言描述单词结构。 | | | 15% | |  |
| 2 | **目标2：**针对编译器中语法分析器软件要求，能够分析系统需求，并采用Bison脚本语言描述语法结构。 | | | 15% | |  |
| 3 | **目标3：**针对计算器需求描述，采用Flex/Bison设计实现高级解释器，进行系统设计，形成结构化设计方案。 | | | 30% | |  |
| 4 | **目标4：**针对编译器软件前端与后端的需求描述，采用软件工程进行系统分析、设计和实现，形成工程方案。 | | | 30% | |  |
| 5 | **目标5：**培养独立解决问题的能力,理解并遵守计算机职业道德和规范，具有良好的法律意识、社会公德和社会责任感。 | | | 10% | |  |
| 总成绩 | | | | | |  |
| 指导教师 | |  | 评阅日期 | |  | |

**目 录**

[实验（二） 基于Flex的C语言简单词法分析器 1](#_Toc96785874)

[2.1 实验目的 1](#_Toc96785875)

[2.2 C语言子集C1的定义 1](#_Toc96785876)

[2.3 *lex2-1.l*编写 1](#_Toc96785877)

[2.3.1 题目要求 1](#_Toc96785878)

[2.3.2 分析步骤 1](#_Toc96785879)

[2.3.3 *lex2-1.l*文件源代码 2](#_Toc96785880)

[2.3.4 双系统实验结果 2](#_Toc96785881)

[2.4 *lex2-2.l*编写 4](#_Toc96785882)

[2.4.1 题目要求 4](#_Toc96785883)

[2.4.2 分析步骤 4](#_Toc96785884)

[2.4.3 文件源代码 5](#_Toc96785885)

[2.4.4 双系统实验结果 8](#_Toc96785886)

[2.5 课后练习 9](#_Toc96785887)

[2.5.1 题目1 9](#_Toc96785888)

[2.5.2 题目2 9](#_Toc96785889)

[2.6 实验总结 10](#_Toc96785890)

[2.6.1 报错unrecognized rule error 10](#_Toc96785891)

[2.6.2 报错rule cannot be matched 10](#_Toc96785892)

[2.6.3 正则表达式太多混乱，理不清 10](#_Toc96785893)

[2.6.4 执行操作的编写 10](#_Toc96785894)

[2.6.5 程序评价与收获 10](#_Toc96785895)

1. 基于Flex的C语言简单词法分析器
   1. 实验目的

阅读*Flex/Bison.pdf*第一章，第二章，掌握Flex基础知识。利用Flex实现用于C语言子集C1的词法分析器。

* 1. C语言子集C1的定义

该子集主要由关键字、专用符号、标识符、整型常数、空格、注释构成。下面一一介绍各个部分：

关键字：所有的关键字都是保留字，并且必须是小写。

else if switch for int float return void while

专用符号。

+ - \* / < <= > >= == != = ; , ( ) [ ] { }

标识符（ID）和整型常数（NUM）通过下列正则表达式定义（注：小写和大写字母是有区别的）。

L = [a-zA-Z\_]

D = [0-9]

ID = {L}({L}|{D})\*

NUM = [1-9] {D}\*

空格由空白、换行符和制表符组成。

[space] \n \t

注释有单行注释和多行注释。

// /\*……\*/

* 1. *lex2-1.l*编写
     1. 题目要求

编写Flex代码*lex2-1.l*，实现对上述C1语言的词法分析。要求：输出所有的关键字KWORD、专用符号SYM、标识符ID、整型常数NUM。

* + 1. 分析步骤

对于关键字KWORD和专用符号SYM，只要设计两个正则表达式能识别出这些单词即可，识别到任何一个就可以输出，伪代码如下。

"else"|……|"while" {printf;}

\+|……|\} {printf;}

对于标识符ID、整型常数NUM，在题目要求中已经给出了识别的正则表达式，可以直接使用，代码如下。

L [a-zA-Z\\_]

D [0-9]

ID {L}({L}|{D})\*

NUM [1-9]{D}\*

注意：考虑到还有换行、空白等可能影响读取的情况，还有一些上述条件匹配不到的字符，需要单独考虑，如双引号，数字0，#等等，暂时先不加以考虑，统一认为是其他字符（这将在第二问中详细修改）。

* + 1. *lex2-1.l*文件源代码

L [a-zA-Z\\_]

D [0-9]

ID {L}({L}|{D})\*

NUM [1-9]{D}\*

enter \n

space [ \t]+

%%

"else"|"if"|"switch"|"for"|"int"|"float"|"return"|"void"|"while" {printf("KWORD:\t%s\n",yytext);}

\+|\-|\\*|\/|\<|\<\=|\>|\>\=|\=|\=\=|\!\=|\;|\,|\(|\)|[|]|\{|\} {printf("SYM:\t%s\n",yytext);}

{ID} {printf("ID:\t%s\n",yytext);}

{NUM} {printf("NUM:\t%s\n",yytext);}

{enter} {}

{space} {}

#|0|\"|!|\. {}

%%

void main()

{

yylex();

}

int yywrap()

{

return 1;

}

* + 1. 双系统实验结果

在Windows系统输入如下代码。

注：这次实验Windows端使用VS Code软件、软件自带调用CMD命令行程序，使用*gcc.exe*进行编译（已在上次实验中验证了gcc编译的可行性），这样避免了每次使用命令行还需要进行cd进入文件夹操作，也不用通过VS的命令行执行cl命令，这样更加快捷方便一些。

flex -o"lex2-1.yy.c" lex2-1.l

gcc -o"lex2-1" lex2-1.yy.c

lex2-1.exe < 2-1.cpp

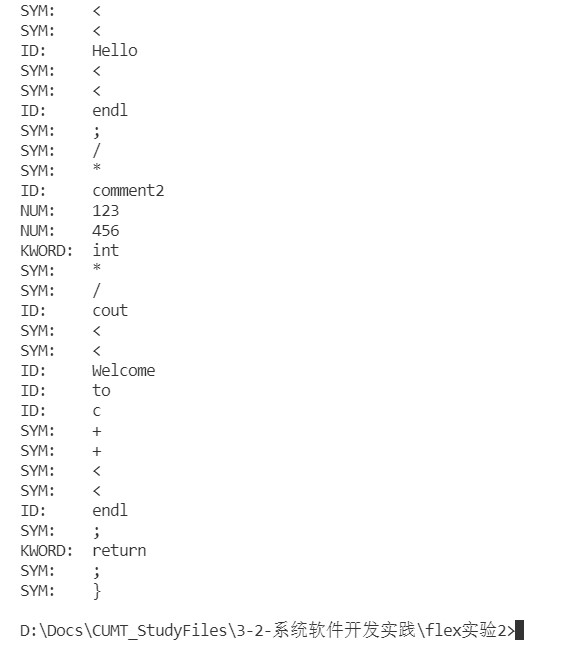
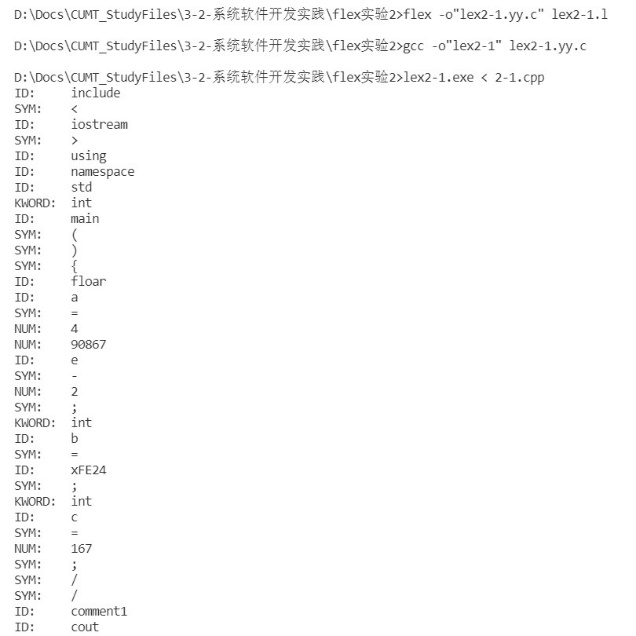


图 1 Windows系统下运行*lex2-1.l*结果

在Linux系统输入如下代码。

flex -o"lex2-1.yy.c" lex2-1.l

gcc -o"lex2-1.out" lex2-1.yy.c

./lex2-1.out < 2-1.cpp

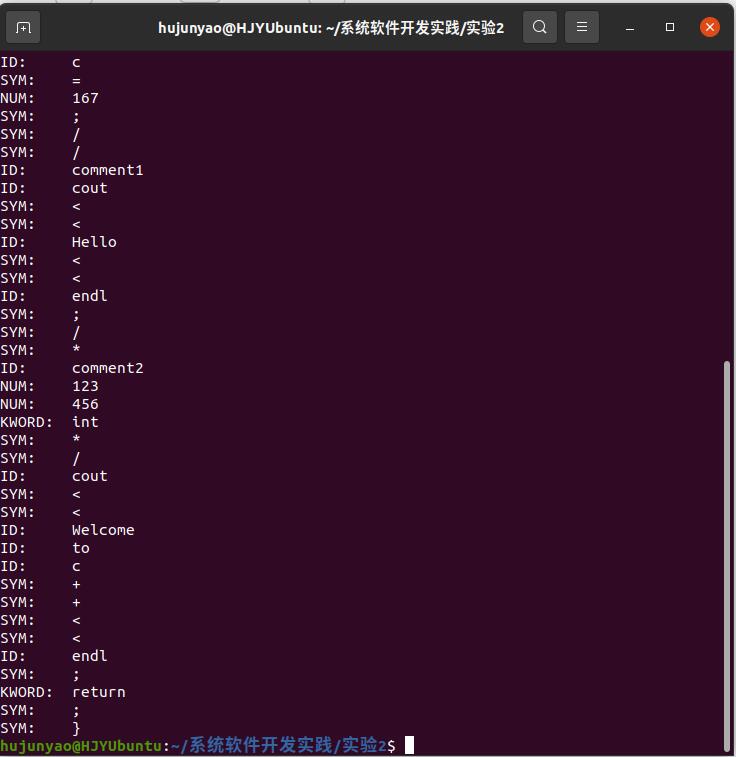
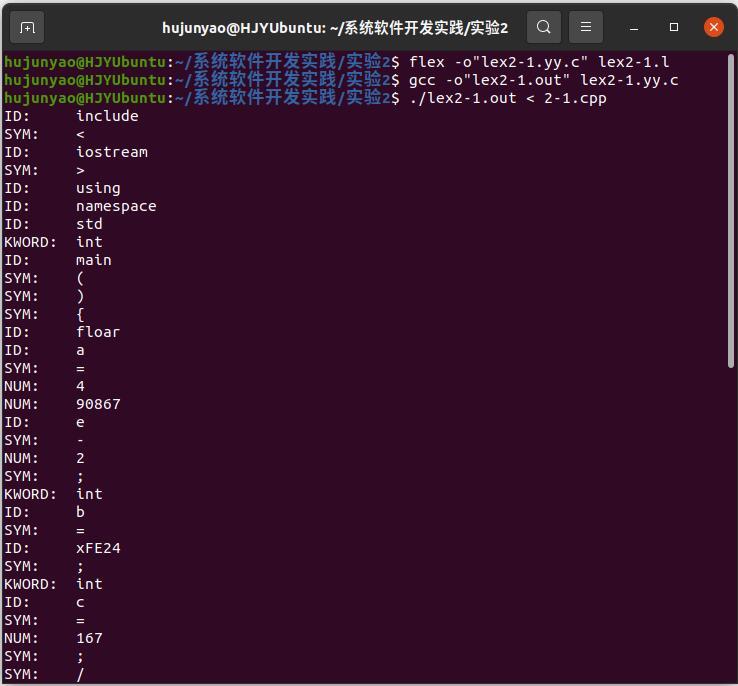


图 2 Linux系统下运行*lex2-1.l*结果

实验观察到诸如数字1.23会被识别成1和23、注释中的文字也被读取等等情况，这是因为实验1并没有对其进行匹配的条件，这将在第二问中进行优化。

* 1. *lex2-2.l*编写
     1. 题目要求

在实现以上基本功能的基础上，参考*ANSI C grammar (Lex).pdf*，实现以下功能，并另存为*lex2-2.l*文件：

输出上述标记所在的行号row、列号col；

忽略注释及其内容，如，注释中的数字/\*123\*/ , //123；

对NUM的识别，增加科学记数法、十六进制、八进制常数。

* + 1. 分析步骤

行号row、列号col的输出，参考了所给文件，稍作修改。yytext读出一段字符串，对于这一段字符串读取每一位，如果遇到\0也就是这个字符串已经读完，应该结束循环。如果中途遇到了\n，也就是相当于新增了一行，这需要重置列数为1，给行数+1。如果碰到的是\t应该根据当前列数计算这个tab补全了多少行，使用column += 8 - (column % 8)计算，也就是说8列是一组，余下的要补成8，就要加上column行。如果就是普通的情况，那么什么也不用考虑，只读一个普通的字符，给列数+1即可。

int column = 1;

void count(void)

{

int i;

for (i = 0; yytext[i] != '\0'; i++)

if (yytext[i] == '\n'){

column = 1;

row++;

}

else if (yytext[i] == '\t') column += 8 - (column % 8);

else column++;

}

忽略注释及其内容，如，注释中的数字/\*123\*/ , //123。对于单行注释很容易，使用正则表达式匹配上两个斜杠和回车前的内容即为单行注释。

"/""/"[^\n]\*

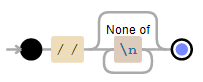


图 3 单行注释正则表达式NFA可视化

多行注释的正则表达式比较麻烦，需要考虑多种情况，参考网络上的C语言多行注释正则表达式，代码如下。

"/\*"([^\\*]|(\\*)\*[^\\*/])\*(\\*)\*"\*/"

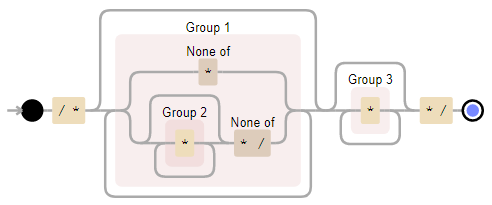


图 4 多行注释NFA可视化

对NUM的对于科学计数法，主要是考虑小数点（可有可无，有的话后面必须有数字）、E（可有可无，有的话后面必须有数字）、正负号（出现E之后，指数正负号可有可无），以及有需要的话，考虑C语言中数字的后缀如u和l等（可有可无，可出现一个，也可以出现两个，两个的时候无先后顺序）。代码如下。

[0-9]+(\.[0-9]+)?([eE][+-]?[0-9]+)?([uUlL]|[uU][lL]|[lL][uU])?

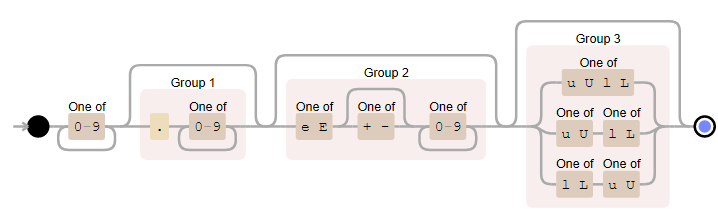


图 5 科学计数法NFA可视化

进制的两个正则表达式很类似。对NUM的八进制，只需要第一位是0，第二位是1-7，后面都是0-7或者没有即可。对NUM的十六进制，只需要第一位是0，第二位是x或者X，第三位是没有0，后面是十六进制任何字符或者没有。

0[1-7]{O}\*

0[xX][a-fA-F1-9][a-fA-F0-9]\*

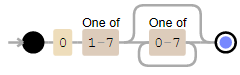
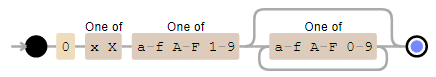
 

图 6 进制的NFA可视化

* + 1. 文件源代码

%{

#define SYM 1

#define ID 20

#define NUMBER 21

#define RELOP 22

#define MAIN 44

#define INT 45

#define FLOAT 46

#define RETURN 8

#define CONST 49

#define WS 51

#define INCLUDE 59

#define NEWLINE 23

#define OTHER 24

#define STRING 26

int yylval;

int column=0;

int row=0;

%}

delim [ \t \n]

ws {delim}+

letter [A-Za-z\_]

schar \'(\\.|[^"\\])\'

string \"(\\.|[^"\\])\*\"

digit [0-9]

O [0-7]

H [a-fA-F0-9]

id ({letter}|\\_)(\\_|{letter}|{digit})\*

number {digit}+(\.{digit}+)?([eE][+-]?{digit}+)?([uUlL]|([uU][lL])|([lL][uU]))?

%%

{ws} {return WS;}

"/\*"([^\\*]|(\\*)\*[^\\*/])\*(\\*)\*"\*/" {}

"/""/"[^\n]\* {}

main {yylval=MAIN; return(MAIN);}

int {yylval=INT; return(INT);}

float {yylval=FLOAT; return(FLOAT);}

return {yylval=RETURN; return(RETURN);}

0[1-7]{O}\* {yylval=NUMBER; return(NUMBER);}

0[xX][a-fA-F1-9]{H}\* {yylval=NUMBER; return(NUMBER);}

"#"include {yylval=INCLUDE; return(INCLUDE);}

{id} {return (ID);}

{number} {return (NUMBER);}

{string} {return (STRING);}

"<"|"<="|"="|"<>"|">"|">=" {yylval = SYM; return (RELOP);}

"<<"|"+"|"/"|"{"|"}"|";"|"("|")" {return(RELOP);}

. {yylval = OTHER; return OTHER;}

%%

int yywrap (){

return 1;

}

void count(){

int i;

for(i=0; yytext[i] != '\0'; i++){

if(yytext[i]=='\n'){

column = 1;

row++;

}

else if(yytext[i]=='\t') column += 8-(column%8);

else column++;

}

}

void writeout(int c){

switch(c){

case OTHER: break;

case RELOP: printf("[RELOP ] [%20s] ",yytext);break;

case NUMBER: printf("[NUM ] [%20s] ",yytext);break;

case ID: printf("[ID ] [%20s] ",yytext);break;

case NEWLINE:break;

case STRING: printf("[STRING ] [%20s] ",yytext);break;

case MAIN: printf("[MAIN ] [%20s] ",yytext);break;

case INT: printf("[INT ] [%20s] ",yytext);break;

case FLOAT: printf("[FLOAT ] [%20s] ",yytext);break;

case RETURN: printf("[RETURN ] [%20s] ",yytext);break;

case WS: break;

case INCLUDE:printf("[INCLUDE] [%20s] ",yytext);break;

default: break;

}

if(c!=WS && c!=OTHER){

printf("(%2d, %2d) \n", row+1, column+1);

}

count();

return;

}

int main (int argc, char \*\* argv){

int c=0;

while (c = yylex()){

writeout(c);

}

return 0;

}

int yyerror(char \*s){

fprintf(stderr,"%s\n",s);

return 1;

}

* + 1. 双系统实验结果

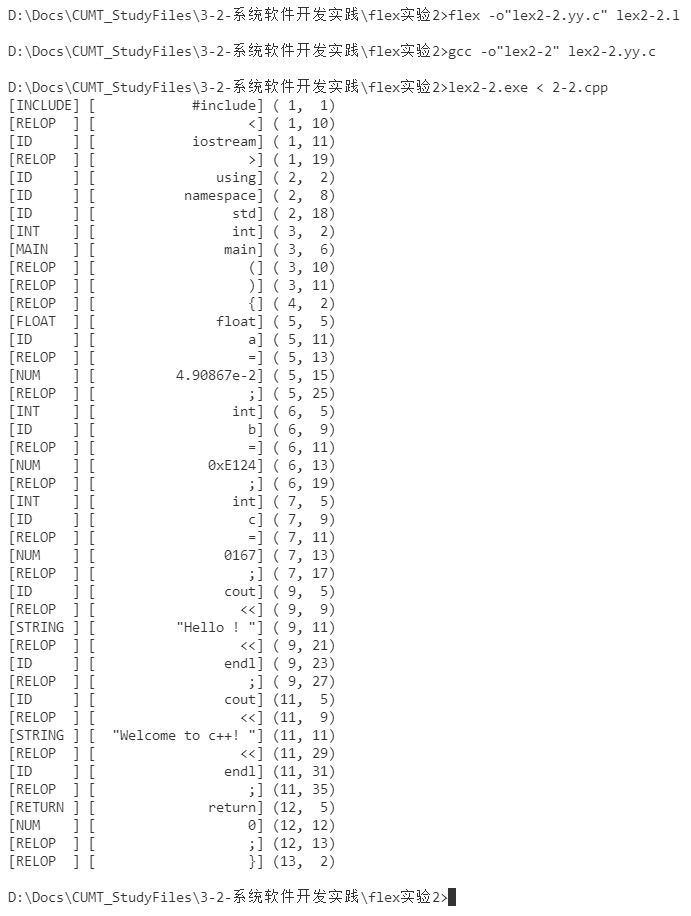


图 7 Windows环境运行lex2-2结果

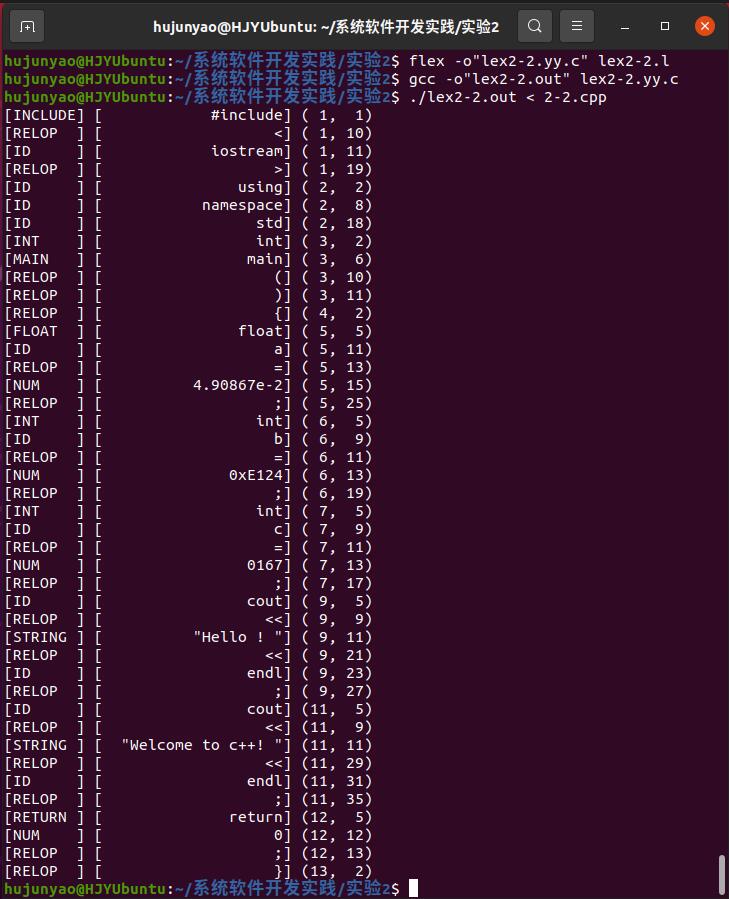


图 8 Linux环境运行lex2-2结果

* 1. 课后练习
     1. 题目1

给定如下的匹配规则，对于表达式i+=1，是+、=分别输出还是一起输出？

“+” {printf(“+”);}

“=” {printf(“=”);}

“+=” {printf(“+=”);}

解答：一次性输出+=符号。Lex的正则表达式是基于贪婪模式的，会找最大的符合条件的那个匹配输出出来。

* + 1. 题目2

阅读*ANSI C grammar (Lex).pdf*，分析其中关于浮点数的匹配规则并举例说明匹配情况。

D [0-9]

E ([Ee][+-]?{D}+)

FS (f|F|l|L)

{D}+{E}{FS}?

{D}\*"."{D}+{E}?{FS}?

{D}+"."{D}\*{E}?{FS}?

解答：D是整数，E是科学计数法，FS是float和long的后缀。下表即是三种表达式的异同分析。其中，通常2、3比较常用。

表 1 三种表达式的异同对比

| 表达式 | 底数整数 | 底数小数 | 科学计数 | 后缀 |
| --- | --- | --- | --- | --- |
| 1 | 1位及以上 | 未考虑 | 必须有 | 可有可无 |
| 2 | 0位及以上 | 1位及以上 | 可有可无 | 可有可无 |
| 3 | 1位及以上 | 0位及以上 | 可有可无 | 可有可无 |

* 1. 实验总结
     1. 报错unrecognized rule error

大概率是正则表达式写错了，匹配不到任何一条语句。需要修改正则表达式，注意引号和反斜杠的用法，有时候经常忘记需要转义。

* + 1. 报错rule cannot be matched

指这个规则不能够匹配到，应该是前面有规则已经给匹配到了，后面的就不会被匹配了。可以调整先后顺序或者修改优化正则表达式。

* + 1. 正则表达式太多混乱，理不清

多和同学讨论结果、查阅资料、对于复杂的正则表达式可以使用可视化网站（如[www.debuggex.com](http://www.debuggex.com)）或者一些编译集成环境的插件进行手工测试（如可以用Pycharm的Re库写一个正则后利用插件测试）。

* + 1. 执行操作的编写

行列值识别、注释的识别都比较难，单独识别\n、\r等方法尝试过但效果不好，参考代码的计算方法使用一个函数整合起来，构建了比较通用的方法，同时对tab的列数处理也很有意思，最后在同学交流和自主尝试下完成部分内容。有不懂的要多和同学交流经验，然后也要仔细阅读参考文献，才能有所收获。

* + 1. 程序评价与收获

之前自己学过一点爬虫的知识，这里Python的正则表达式和Lex正则表达式还是有一定差别的，不能直接搬过来用，需要做适当修改，该加双引号的要加双引号。最后，对于多行注释的行数问题还是有待研究，行数和文件还对应不上，大概是计算\n的问题，目前时间太短，未能完成，希望接下来可以多搜集一些资料再考虑一下这个问题。