

**[C++&fLex]**

**编译原理词法分析**

**实验报告**

**学院：计算机学院**

**2020211376 马天成**

**2022年10月7日**

**北京邮电大学《计算机网络》课程实验报告**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **实验**  **名称** | 词法分析 | | **学 院** | 计算机 | **指导教师** | 刘辰 |
| **班 级** | **班内序号** | **学 号** | | **学生姓名** | **成绩** | |
| **2020211305** | **2** | **2020211376** | | **马天成** |  | |
| **实**  **验**  **内**  **容** |  | | | | | |
| **学生**  **实验**  **报告** | （详见“实验报告和源程序”册） | | | | | |
| **课**  **程**  **设**  **计**  **成**  **绩**  **评**  **定** | **评语**:  **成绩**:  指导教师签名：  年 月 日 | | | | | |

注：评语要体现每个学生的工作情况，可以加页。

目录

[1. 实验目的和要求 4](#_Toc116048533)

[1.1 实验目的 4](#_Toc116048534)

[1.2 实验要求 4](#_Toc116048535)

[2. C++程序 4](#_Toc116048536)

[2.1 设计各种词语类型及词法识别自动机 4](#_Toc116048537)

[2.2 代码结构及编写 5](#_Toc116048538)

[2.2.1 模块设计 5](#_Toc116048539)

[2.2.2 Main函数 5](#_Toc116048540)

[2.2.3 Lex-识别进入各类词法函数 6](#_Toc116048541)

[2.2.4 Lex-各类词法识别函数 7](#_Toc116048542)

[2.2.4 Lex-读取单个字符函数 9](#_Toc116048543)

[2.2.5 Lex-输入输出函数 9](#_Toc116048544)

[2.2.6 output类 & error类 10](#_Toc116048545)

[2.3 实验结果 10](#_Toc116048546)

[3. Lex程序 10](#_Toc116048547)

[3.1 Lex介绍 10](#_Toc116048548)

[3.2 Lex编写结构 11](#_Toc116048549)

[3.2.1 定义段 11](#_Toc116048550)

[3.2.2 词法规则段 11](#_Toc116048551)

[3.2.3 辅助函数段 15](#_Toc116048552)

[3.3 实验结果 15](#_Toc116048553)

[4. 实验总结 16](#_Toc116048554)

# 1. 实验目的和要求

## 1.1 实验目的

词法分析程序的设计与实现

## 1.2 实验要求

设计并实现C语言的词法分析程序，要求实现以下功能：

（1）可以识别出C语言编写的源程序的每个单词符号，并以记号的形式输出每个符号。

（2）可以识别并跳过源程序中的注释

（3）可以统计源程序中的语句行数，各类单词的个数，以及字符总数，并输出统计结果

（4）检察院程序中存在的语法错误，并报告错误所在的位置

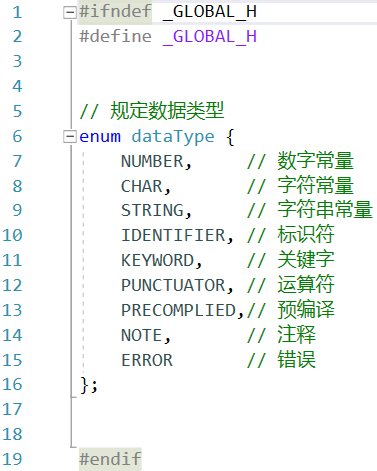
（5）对源程序中出现的错误进行适当的恢复，使词法分析可以继续进行，对源程序进行一次扫描，即可检查并报告源程序中所有存在的语法错误。

用C++和Lex实现

# 2. C++程序

## 2.1 设计各种词语类型及词法识别自动机

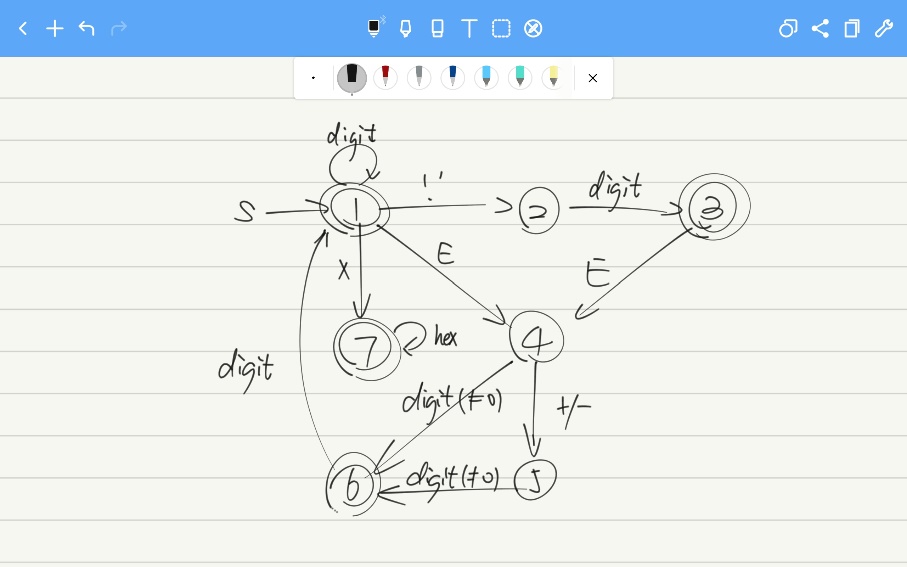
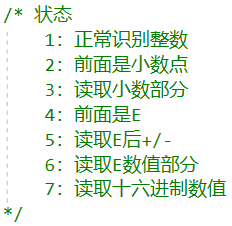
一共有9种数据类型，前面八种识别的时候出错自然会归为ERROR。



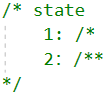
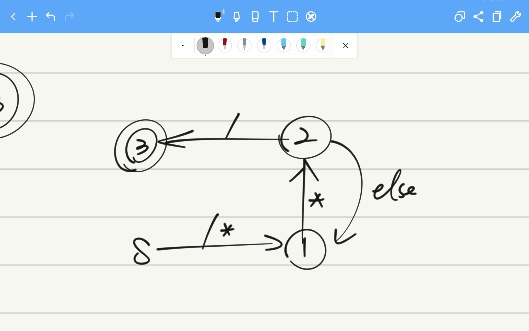
语法自动机：

鉴于有些词语比较简单，所以只有数字和注释做了自动机处理。

* 数字

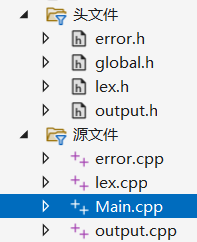


* 多行注释

## 2.2 代码结构及编写

### 2.2.1 模块设计

lex, output, error有对应.h&.c

1. global.h：规定数据类型

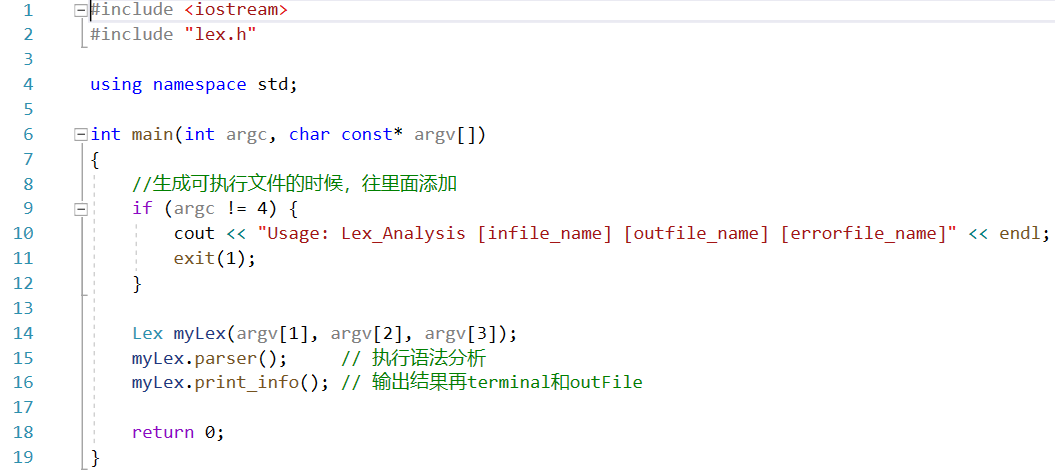
2. Main.c：主函数，运行

3. lex：词法分析模块

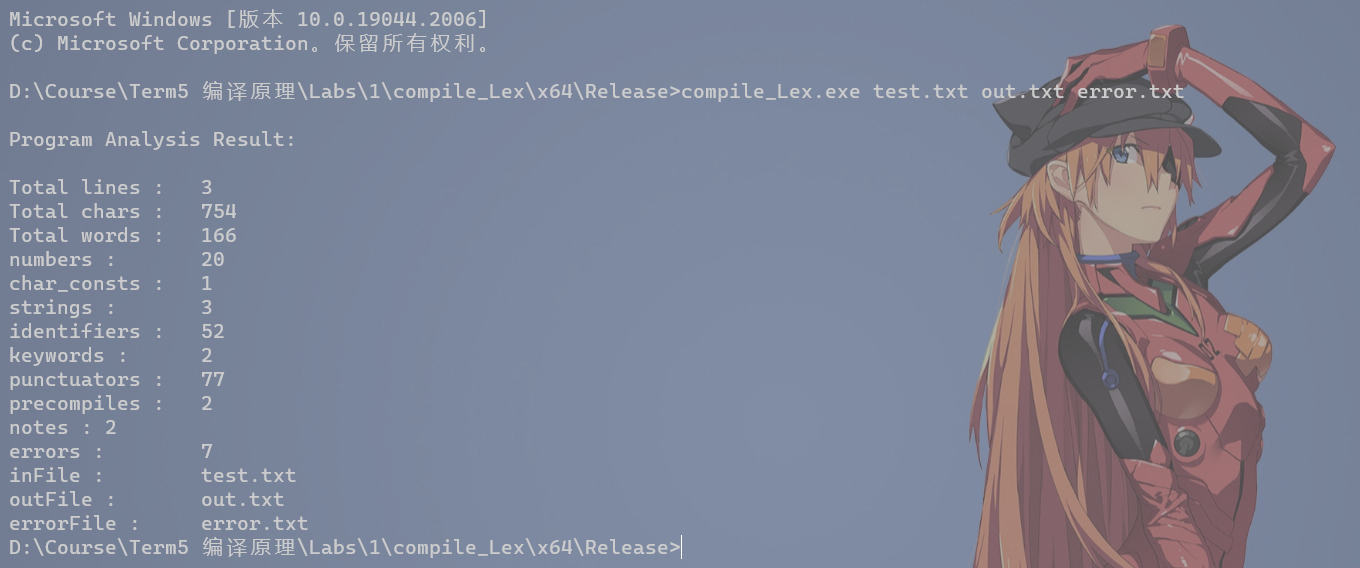
4. output：输出正确词法模块

5. error：输出错误词法模块

### 2.2.2 Main函数



Main函数的主要功能：



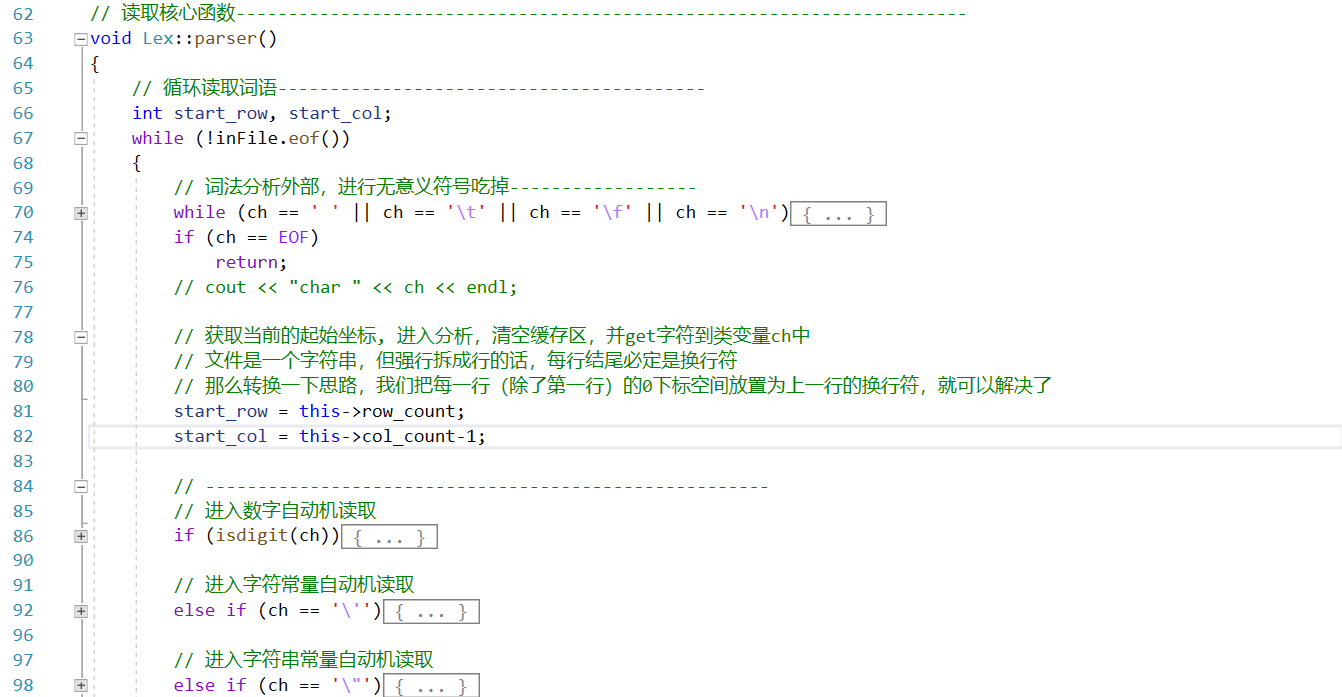
1. 获得输入参数，分别为**[infile\_name] [outfile\_name] [errorfile\_name]**

若格式不正确则报错exit(1)。

2. 运行lex类分析相应的inFile。

3. 用lex统计输出的函数输出有关的所有统计信息（同时打印在out和terminal里）。

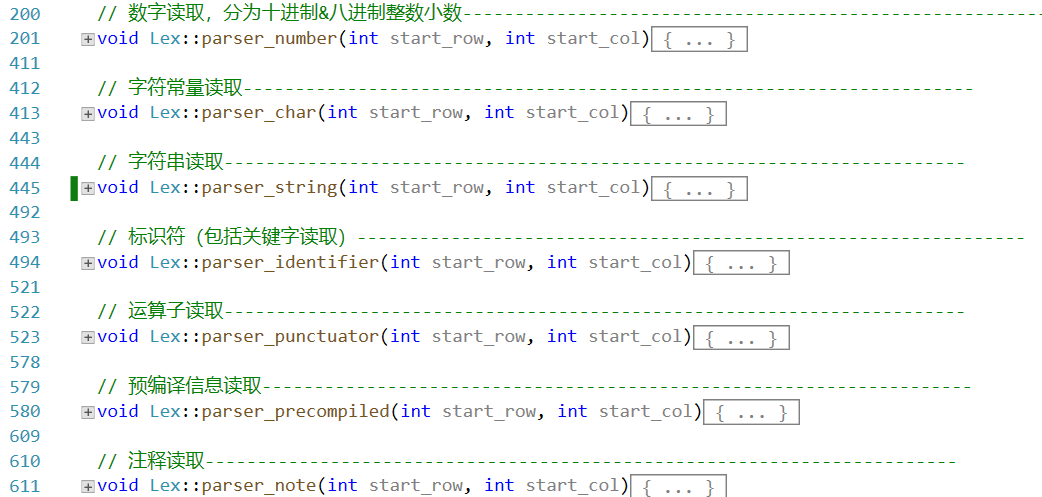
### 2.2.3 Lex-识别进入各类词法函数





上图是读取函数核心部分。这里的策略是每次进入判断时，我们已经读取了该词法的第一个字符。然后进行判断，进入相应的此法识别程序。

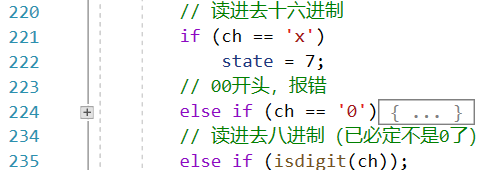
### 2.2.4 Lex-各类词法识别函数



一共有七个函数。其中标识符包括关键字的读取。

这里只提示重要的操作策略：

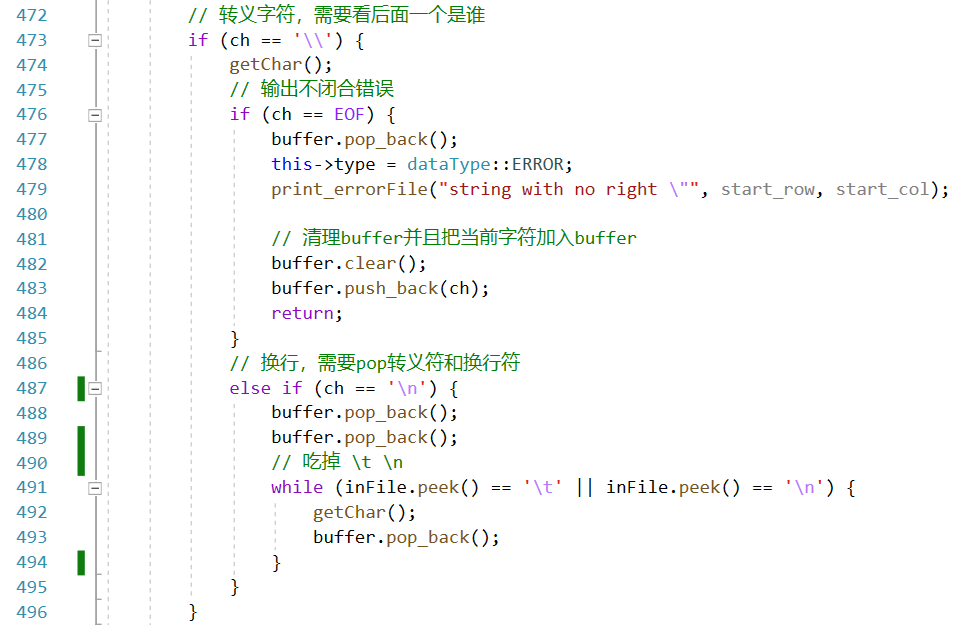
* 数字
* 读取中的自动机策略（如上文所示）
* 0开头会有**十六进制**和**八进制**两种合法读取方式



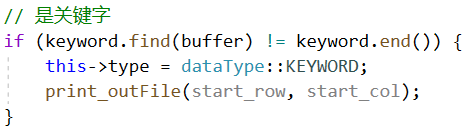
* 字符常量
* 字符需转义的符号不能直接读取，需转义



* 字符串常量
* 字符串里 **\ + \n** 能换行



* 标识符
* 标识符结束进行关键字的识别

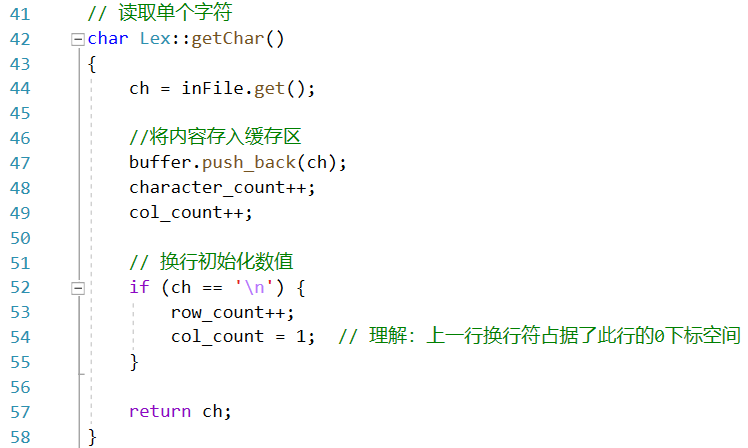


* 运算子
* 长度为2或者3的运算符要分**前缀读取**
* 预编译信息
* 预编译里 **\ + \n** 能换行



* 注释
* 读取中的自动机策略（如上文所示）
* 分为单行注释和多行注释

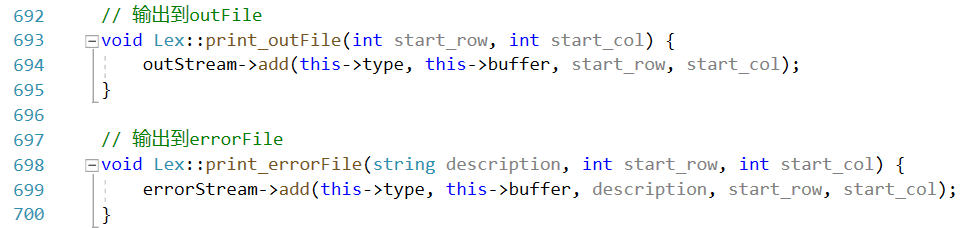
### 2.2.4 Lex-读取单个字符函数



我们把换**行符理解为每一行的第0号元素**。这样就解决了直观上下标不对应，以及输出词法位置种换行符如何理解的问题。

### 2.2.5 Lex-输入输出函数

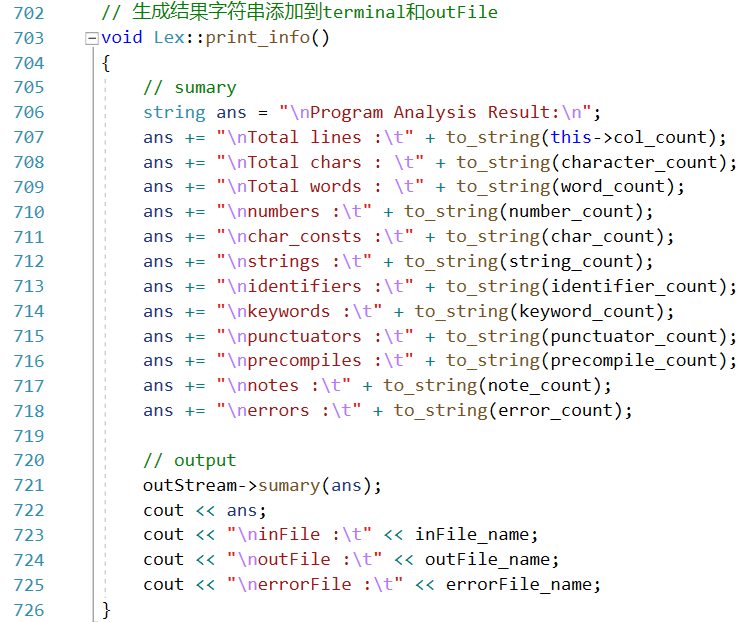
* out 和 error



**调用output类**输出词法信息到outFile；

**调用error类**输出错误信息到errorFile。

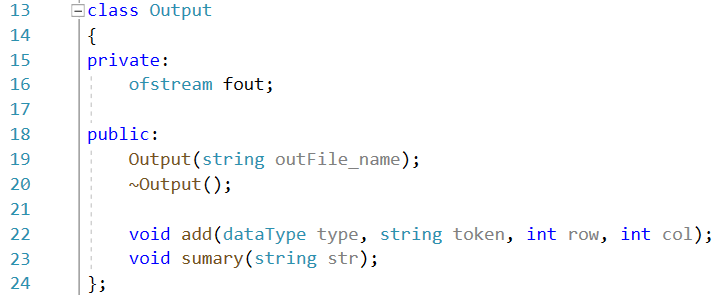
* summary

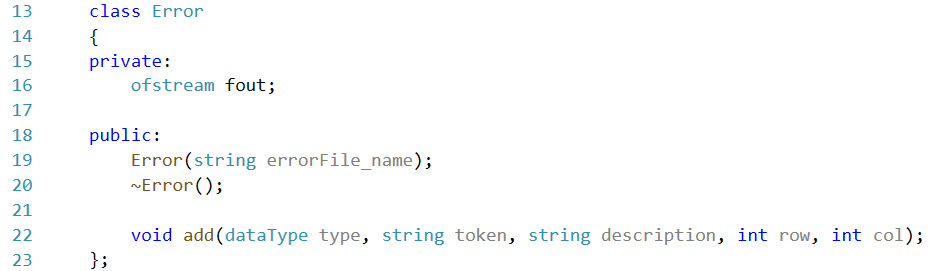


生成统计信息输出到terminal和outFile。

### 2.2.6 output类 & error类

主要为类封装，和outFile和errorFile交互





## 2.3 实验结果

* 实验用例：

test.txt

* 实验结果：

output.txt error.txt

# 3. Lex程序

## 3.1 Lex介绍

Lex是linux下的工具，是一个词法分析程序的自动生成工具。

Lex的基本工作原理为：由正规式生成NFA，将NFA变换成DFA，DFA经化简后，模拟生成词法分析器。

其中正规式由开发者使用Lex语言编写，其余部分由Lex翻译器完成.翻译器将Lex源程序翻译成一个名为lex.yy.c的C语言源文件，此文件含有两部分内容：一部分是根据正规式所构造的DFA状态转移表，另一部分是用来驱动该表的总控程序yylex()。当主程序需要从输入字符流中识别一个记号时，只需要调用一次yylex()就可以了。为了使用Lex所生成的词法分析器，我们需要将lex.yy.c程序用C编译器进行编译，并将相关支持库函数连入目标代码。Lex的使用步骤可如下图所示：



 [第一部分：定义段]

 %%

 [第二部分：词法规则段]

 %%

 [第三部分：辅助函数段]

## 3.2 Lex编写结构

### 3.2.1 定义段

第一部分以符号%{和%}包裹，里面为以C语法写的一些定义和声明：例如，文件包含，宏定义，常数定义，全局变量及外部变量定义，函数声明等。这一部分被Lex翻译器处理后会全部拷贝到文件lex.yy.c中。

%{

int words = 0;

int character\_cnt = 0;

int lines = 1;

void charCnt\_add(int n);

void line\_add(void);

%}

### 3.2.2 词法规则段

\*) 正规表达式规则如下：

（a）正文字符：除元字符以外的其他字符，这些字符在正规式中可以被匹配。

若单个正文字符c作为正规式，则可与字符c匹配，元字符无法被匹配，如果元字符想要被匹配，则需要通过“转义”的方式，即用” ”包括住元字符，或在元字符前加 \ 。例如 ”+” 和 \+ 都表示加号。

C语言中的一些转义字符也可以出现在正规式中，例如 \t \n \b 等。

（b）元字符：元字符是lex语言中作特殊用途的一些字符，包括：\* + ? | { } [ ] ( ). ^ $ “ \ - / < >。

（c）^ : 表示补集：[^…]表示补集，即匹配除 ^ 之后所列字符以外的任何字符。如 [^0-9] 表示匹配除数字字符 0-9 以外的任意字符。

除 ^ - \ 以外，任何元字符在方括号内失去其特殊含义。

如果要在方括号内表示负号 - ，则要将其至于方括号内的第一个字符位置或者最后一个字符位置，例如[-+0-9][+0-9-]都匹配数字及+ - 号。

（d） . ^$ /：

点运算符 . 匹配除换行之外的任何字符，一般可作为最后一条翻译规则。

（e）^ 匹配行首字符。如：^begin匹配出现在行首的begin

（f） $ 匹配行末字符。如：end$ 匹配出现在行末的end

（g）R1/R2（R1和R2是正规式）表示超前搜索：若要匹配R1，则必须先看紧跟其后的超前搜索部分是否与R2匹配。

digit [0-9]

space [ \t]

letter [\_A-Za-z]

lineComment (\/\/([^\n]\*(\\+\n)\*)\*\n)

comment (\/\\*(.\*\n\*)\*\\*\/)

string (\"([^"\n]\*(\\\n)\*(\\\")\*)\*\")

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

number {digit}+(\.{digit}+)?([eE][+-]?{digit}+)?

keyword "int"|"long"|"short"|"float"|"double"|"char"|"unsigned"|"signed"|"const"|"void"|"volatile"|"enum"|"struct"|"union"|"if"|"else"|"goto"|"switch"|"case"|"do"|"while"|"for"|"continue"|"break"|"return"|"default"|"typedef"|"auto"|"register"|"extern"|"static"|"sizeof"

operator ">"|">>"|">="|">>="|"<"|"<<"|"<="|"<<="|"!"|"!="|"="|"=="|"/"|"/="|"\*"|"\*="|"%"|"%="|"^"|"^="|"|"|"||"|"|="|"?"|"&"|"&&"|"&="|"+"|"+="|"++"|"-"|"-="|"--"

delimiter "("|")"|"{"|"}"|";"

char (\'[^']\*\')

illString (\"([^"\n]\*(\\\")\*)\*\n)

illNum ({digit}+(\.{digit}+)?([eE][+-]?[^0-9]))|({digit}+\.[^0-9])

macro #.\*\n

\*) 正规表达式的书写结束之后，就是对相应的规则进行描述。

**Lex源程序中常用到的变量及函数：**

* yyin和yyout：这是Lex中本身已定义的输入和输出文件指针。这两个变量指明了lex生成的词法分析器从哪里获得输入和输出到哪里。默认：键盘输入，屏幕输出。
* yytext和yyleng：这也是lex中已定义的变量，直接用就可以了。
* yytext：指向当前识别的词法单元（词文）的指针
* yyleng：当前词法单元的长度。
* ECHO：Lex中预定义的宏，可以出现在动作中，相当于fprintf(yyout, “%s”,yytext)，即输出当前匹配的词法单元。
* yylex()：词法分析器驱动程序，用Lex翻译器生成的lex.yy.c内必然含有这个函数。
* yywrap()：词法分析器遇到文件结尾时会调用yywrap()来决定下一步怎么做：

若yywrap()返回0，则继续扫描

若返回1，则返回报告文件结尾的0标记。

由于词法分析器总会调用yywrap，因此辅助函数中最好提供yywrap，如果不提供，则在用C编译器编译lex.yy.c时，需要链接相应的库，库中会给出标准的yywrap函数（标准函数返回1）。

{keyword} {

printf("< %s , - >\n",yytext);

charCnt\_add(yyleng);

words++ ;

}

{id} {

printf("< id , %s >\n",yytext);

charCnt\_add(yyleng);

words++;

}

{number} {

printf("< num , %s >\n",yytext);

charCnt\_add(yyleng);

}

{operator} {

printf("< operator , %s >\n",yytext);

charCnt\_add(yyleng);

}

{char} {

printf("< char , %s >\n",yytext);

charCnt\_add(yyleng);

}

{string} {

printf("< string , %s >\n",yytext);

line\_add();

charCnt\_add(yyleng);

}

{lineComment} {

printf("< comment , %s >\n",yytext);

line\_add();

charCnt\_add(yyleng);

}

{comment} {

line\_add();

printf("< comment , %s >\n",yytext);

charCnt\_add(yyleng);

}

{delimiter} {

printf("< %s , - >\n",yytext);

charCnt\_add(yyleng);

}

{illNum} {

printf("wrong numbers");

printf(" wrong location: %d line\n",lines);

line\_add();

}

{illString} {

printf("double quotes not match");

printf(" wrong location: %d line\n",lines);

charCnt\_add(yyleng);

line\_add();

}

{space} {

charCnt\_add(1);

}

{macro} {

printf("< macro , %s >\n",yytext);

lines++;

charCnt\_add(yyleng);

}

. {

charCnt\_add(1);

printf("unknown character: %s",yytext);

printf(" wrong location: %d line\n",lines);

}

\n {

charCnt\_add(1);

lines++;

}

其主要用到了yytext以及yyleng这两个变量。

yytext变量存储的是此时根据正则表达式匹配成功的字符串，当其匹配成功后，会进入到相应的分支结构中来，进行相应的统计运算。

### 3.2.3 辅助函数段

辅助过程主要是对翻译规则的补充，翻译规则部分中某些动作需要调用的过程，如果不是C语言的 库函数，则要在此给出具体的定义。

int main (void) {

yylex();

printf("characters count: %d\n",character\_cnt );

printf("total lines: %d\n",lines);

printf("word count: %d\n",words);

return 0;

}

int yywrap() {

return 1;

}

void charCnt\_add(int n) {

character\_cnt+=n;

}

void line\_add(void) {

for(int i=0;i<yyleng;i++)

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

lines++;

}

通过构建上述的lex源程序，以此来进行自动的词法分析程序的生成，通过相应的命令生成lex.yy.c程序后，通过gcc对该程序进行编译，最终会生成一个.out文件，这个文件是一个而执行文件，是一个生成的词法分析程序，通过运行该程序，并将自己所写的文件进行导入，然后表明结果的输出文件即可。

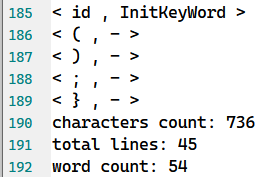
## 3.3 实验结果

* 实验用例：

文件中的test.c

* 实验结果：

文件中的output.txt



# 4. 实验总结

本次实验，实践了如何设计自动机，以及各种读取单个字符和匹配之间的冲突和解决。

主要有以下几个重点问题：

Q: 读取单个字符和回退引起的行列数值冲突

A: 全局**不采用回退**，直接进行单个字符的读取后直接判断词法匹配。

Q: 数字状态机转化和**十六进制八进制**的转换

A: 设计多状态自动机，并进行状态简化形成上述自动机。

Q: **转义符**问题

A: 预编译信息换行，字符串换行，单个字符转义。

Q: 关键字存储结构

A: 使用map存储，方便查找。