实验报告：词法分析

涂奕腾 2020201018

1. 行、列计数

行和列的计数主要使用定义在声明部分的计数器实现：每当匹配一个字符后(除\n,\t)应当加上字符长度 yyleng，对于\n 应当将列计数器 col 重置为 1、将行计数器+1；对于\t 应当对列计数器+8

```
%{
#include <stdio.h>
#include <stdlib.h>
int row = 1;
int col = 1;
%}
%%
\n              {++row; col = 1;}
\t              {col += 8;}
" "             {++col;}
%%
```

2. 关键字、界符、算符

关键字、界符、算符通过在正则式中枚举出所有的情况暴力匹配：

```
KEYWORD
    "auto"|"break"|"continue"|"case"|"char"|"const"|"default"|"define"|"do"|"double"|"els
e"|"enum"|"extern"|"float"|"for"|"goto"|"if"|"int"|"long"|"main"|"register"|"return"|"sho
rt"|"signed"|"sizeof"|"static"|"struct"|"switch"|"typedef"|"union"|"unsigned"|"void"|"vol
atile"|"while"
BORDER      [\(\)\{\}\"\'\[\]\;\,\"\']
OPERATOR    "+"|"++"|"+="|"-"|"--"|"-
="|"*"|"*="|"/"|"/="|"%"|"%="|"<"|"<="|">"|">="|"!"|"!="|"="|"=="|"&&"|"||"|"&"|"|"|"^"|"
~"
%%
{KEYWORD}           {printf("%s:\t K, (%d, %d)\n",yytext,row,col); col += yyleng;}
{OPERATOR}          {printf("%s:\t O, (%d, %d)\n",yytext,row,col); col += yyleng;}
{BORDER}            {printf("%s:\t D, (%d, %d)\n",yytext,row,col); col += yyleng;}
%%
```

3. 标识符

标识符应当满足变量/函数的命名规则：以下划线或字母开头，后面由下划线、字母、数字组成，用正则表达式直接匹配

```
IDENTIFIER  [_a-zA-Z][_a-zA-Z0-9]*
other       [^ \n\t\v\r\f\(\)\{\}\"\'\[\]\;\,\"\'\+\-\*\/\%\<\>\=\&\|\!\^\~]
```

```
%%
{IDENTIFIER}              {printf("%s:\t I, (%d, %d)\n",yytext,row,col); col += yyleng;}
%%
```

4. 常数

八进制和十六进制的常数可以直接使用正则表达式判断，但对于十进制，我考虑了小数和科学计数法的部分。我自己的写法相较于规定的写法普遍性更强，是允许 0.5e3 这样的数存在的，但 sysy 定义中的十进制数字应当以 1-9 开头，如要符合 sysy 语言的规范定义，将 DECIMAL 中开头的{DIGIT}+改为[1-9] {DIGIT}*即可。

另一方面，为了符合词法定义，数字后面应当只能接空格符(SPACE、\n、\t)、界符和操作符，即排除了非法符号、关键字和标识符，储存在 afterNUMBER 中

```
DIGIT       [0-9]
FRACTION    \.[0-9]+
INDEX       [Ee][-\+][0-9]+
DECIMAL     {DIGIT}+{FRACTION}?{INDEX}?
OCTALCONS       0[0-7]+
HEXCONS         0[xX][0-9a-fA-F]+
NUMBER      {OCTALCONS}|{HEXCONS}|{DECIMAL}
afterNUMBER {OPERATOR}|{BORDER}|" "|\n|\t
%%
{NUMBER}/{afterNUMBER}      {printf("%s:\t C, (%d, %d)\n",yytext,row,col); col +=
yyleng;}
%%
```

5. 其他

主要是针对出现非法字符(如@)以及常数后接标识符(如变量名)/关键字的情况

```
other       [^ \n\t\v\r\f\(\)\{\}\"\'\[\]\;\,\"\'\+\-\*\/\%\<\>\=\&\|\!\^\~]
%%
{other}*         {printf("%s:\t T, (%d, %d)\n",yytext,row,col); col += yyleng;}
%%
```

6. 注释

对于注释，我是使用条件激活规则判断：

对于单行注释，当检测到“//”后激活单行注释模式，实际上本行的内容都没有意义了，可以直接将行计数器 row+1，列计数器重设置为 1，将本行读完即可(即读到\n)

对于多行注释，当检测到“/*”后激活单行注释模式，行、列的计数和正常情况下相同，直到读到“\n”退出多行注释模式

```
notebegin   "/*"
noteend     "*/"
linenotebegin   "//"
```

```
linenoteend     \n
%x      NOTE
%x      LINENOTE
%%
{notebegin}             {BEGIN NOTE; col += yyleng;}
<NOTE>\n            {++row; col = 1;}
<NOTE>\t            {col += 8;}
<NOTE>.             {col += yyleng;}
<NOTE>{noteend}         {BEGIN INITIAL; col += yyleng;}
{linenotebegin}         {BEGIN LINENOTE;}
<LINENOTE>.             {}
<LINENOTE>{linenoteend}     {BEGIN INITIAL; ++row; col=1;}
%%
```

7. 匹配优先级

优先级：注释>关键字>数字>算符>界符>标识符>空格符>其他

8. 完整代码

```
%{
#include <stdio.h>
#include <stdlib.h>

int row = 1;
int col = 1;

%}

KEYWORD     "auto"|"break"|"continue"|"case"|"char"|"const"|"default"|"define"|"do"|"doub
le"|"else"|"enum"|"extern"|"float"|"for"|"goto"|"if"|"int"|"long"|"main"|"register"|"retu
rn"|"short"|"signed"|"sizeof"|"static"|"struct"|"switch"|"typedef"|"union"|"unsigned"|"vo
id"|"volatile"|"while"
BORDER      [\(\)\{\}\"\'\[\]\;\,\"\']
OPERATOR        "+"|"++"|"+="|"-"|"--"|"-
="|"*"|"*="|"/"|"/="|"%"|"%="|"<"|"<="|">"|">="|"!"|"!="|"="|"=="|"&&"|"||"|"&"|"|"|"^"|"
~"
DIGIT       [0-9]
FRACTION    \.{DIGIT}+
INDEX       [Ee][+-]?{DIGIT}+
DECIMAL     {DIGIT}+{FRACTION}?{INDEX}?
OCTALCONS       0|0[0-7]+
```

```
HEXCONS         0[xX][0-9a-fA-F]+
NUMBER      {OCTALCONS}|{HEXCONS}|{DECIMAL}
afterNUMBER {OPERATOR}|{BORDER}|" "|\n|\t|\v|\r|\f
IDENTIFIER      [_a-zA-Z][_a-zA-Z0-9]*
other       [^ \n\t\v\r\f\(\)\{\}\"\'\[\]\;\,\"\'\+\-\*\/\%\<\>\=\&\|\!\^\~]


notebegin   "/*"
noteend     "*/"
linenotebegin   "//"
linenoteend     \n
%x      NOTE
%x      LINENOTE

%%

{notebegin}             {BEGIN NOTE; col += yyleng;}
<NOTE>\n            {++row; col = 1;}
<NOTE>\t            {col += 8;}
<NOTE>.             {col += yyleng;}
<NOTE>{noteend}         {BEGIN INITIAL; col += yyleng;}
{linenotebegin}         {BEGIN LINENOTE;}
<LINENOTE>.             {}
<LINENOTE>{linenoteend}     {BEGIN INITIAL; ++row; col=1;}

{KEYWORD}           {printf("%s:\t K, (%d, %d)\n",yytext,row,col); col += yyleng;}
{NUMBER}/{afterNUMBER}      {printf("%s:\t C, (%d, %d)\n",yytext,row,col); col +=
yyleng;}
{OPERATOR}          {printf("%s:\t O, (%d, %d)\n",yytext,row,col); col += yyleng;}
{BORDER}            {printf("%s:\t D, (%d, %d)\n",yytext,row,col); col += yyleng;}
{IDENTIFIER}            {printf("%s:\t I, (%d, %d)\n",yytext,row,col); col += yyleng;}
\n              {++row; col = 1;}
\t              {col += 8;}
" "             {++col;}
{other}*            {printf("%s:\t T, (%d, %d)\n",yytext,row,col); col += yyleng;}

%%
int yywrap(){
    return 1;
}

void main(int argc, char **argv){
    yylex();
}
```