编译原理实验一：词法分析

一、实验要求：

1.输出基本的词法分析结果

2.输出未定义的标识符

3.识别单行注释

4.识别八进制数和十六进制数

5.识别指数形式浮点数

6.识别多行注释

二、实验分工：

实验由我们小组三个人共同完成。其中，代码的基础框架编写由邵一峰完成，主函数的读文件功能由文伦鹏完成，正则表达式的编写由我完成。

三、实验环境：

实验采用vmware14 workstation搭建虚拟机环境，使用ubuntu-18.04.1-desktop-amd64进行光盘刻录，并在此虚拟机上进行实验。

四、实验设计：

实验在 flex 的帮助下省去了很多困难，我们根据实验要求分为三部分由三个人分别独立完成，其中我们首先根据网页的实验指导完成了代码基础框架的编写——即接受文件输入的 main 函数，按照书本上的描述写出了除去整数、浮点数、标识符、注释、八进制/十六进制以及错误识别之外的所有正则表达式。

1.问题一：文中的 token 正则表达式如何确定？

根据实验指导中的思考与提示，大部分正则表达式的编写基于此完成。

（1）整型正则表达式：0 或者非零开头的数字串：(0|[1-9][0-9]*)

（2）八进制正则表达式：以 0 开头，0-7 组成的数字串： 0[0-7]*

（3）十六进制表达式：以 0x 或 0X 开头 0-F 组成的字符串： 0[xX][0-9a-fA-F]+

（4）正常的浮点数：以.或者数字开头且.后含有数字的数字串：([0-9]*[\.][0-9]+|[0-9]+[\.][0-9]*)

（5）科学计数法：以浮点数开头以 e 与整数相连的数字： {NORMAL_FLOAT}[eE][+-]?{INT}

（6）浮点数：两种浮点数的结合： ({NORMAL_FLOAT}|{SCI_FLOAT})

（7）标识符：以字母或下划线开头并且仅含有数字字母下划线： [A-Za-z_][A-Za-z0-9_]*

正则表达式除此之外还有注释和报错两个部分，其中注释部分：

（1）单行注释：匹配//，之后进行循环 input 直到读到换行符。

（2）多行注释：匹配/*，之后循环 input 直到读到*，此时进行循环读取*，若读取到/则结束多行注释并打印读取到的值，否则跳出循环直到再次遇到*。

报错部分：

（1）初步设想：当读取到数字或者标识符时，继续读取进行一次 input，若该 input 为空格、换行符、文件结束符其中之一则正确，否则一直读取直到空格、换行符、文件结束符其中之一并输出错误。

（2）最终实现：第一种方法可行，但是由于过于冗余，我们将其功能变少，写成了由正则表达式进行错误的匹配。

浮点：e 后出现非数字： ({NORMAL_FLOAT}|{INT})[eE]([0-9a-zA-Z]+|{NORMAL_FLOAT})?

十六进制：出现 F 之后字母： 0[xX][0-9a-zA-Z]+

八进制：出现 8/9：　0[0-9]+

2.问题二：如何忽略空白字符?

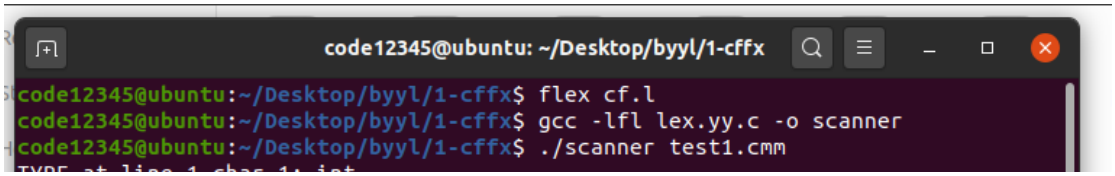
使用正则表达式\s 匹配所有非换行的空格制表符，匹配之后不做操作即可忽略。

3.问题三：如何获得 token 的字符数?

获得字符数使用 flex 内置变量 yyleng 即可，但是输出其所在位置则需要一定的计算。实验中设置了一个 base 表示当前行当前读取位置的位置。当读取到换行符时 base 置为 0，否则加上当前匹配的字数长度，即可每次匹配到之后获得当前的位置，结合 yylineno 输出当前行和偏移。

五、实验结果：

***编译*：**



***测试一运行*：**

```
code12345@ubuntu:~/Desktop/byyl/1-cffx$ ./scanner test1.cmm
TYPE at line 1,char 1: int
ID at line 1,char 5: main
LP at line 1,char 9: (
RP at line 1,char 10: )
LC at line 1,char 11: {
TYPE at line 2,char 5: float
ID at line 2,char 11: f
ASSIGNOP at line 2,char 13: =
FLOAT at line 2,char 15: 2.5
SEMI at line 2,char 18: ;
TYPE at line 3,char 5: int
ID at line 3,char 9: n_num
ASSIGNOP at line 3,char 15: =
INT data at line 3,char 17: 30
SEMI at line 3,char 19: ;
IF at line 4,char 5: if
LP at line 4,char 7: (
ID at line 4,char 8: n
RELOP at line 4,char 10: >
FLOAT at line 4,char 12: 0.15
RP at line 4,char 16: )
LC at line 4,char 17: {
ID at line 5,char 9: printf
LP at line 5,char 15: (
Error Type A at line 5,char 16: Myterious character: '"'
Error Type A at line 5,char 17: Myterious character: '"'
RP at line 5,char 18: )
SEMI at line 5,char 19: ;
RC at line 6,char 5: }
ELSE at line 6,char 6: else
LC at line 6,char 10: {
ID at line 7,char 9: _f2
ASSIGNOP at line 7,char 13: =
ID at line 7,char 15: _f
STAR at line 7,char 18: *
FLOAT at line 7,char 20: 0.15
SEMI at line 7,char 24: ;
RELOP at line 8,char 9: <
RELOP at line 8,char 11: >
RELOP at line 8,char 13: ==
Error Type A at line 9,char 9: Myterious character: '#'
Error Type A at line 9,char 11: Myterious character: '%'
```

```
TYPE at line 1,char 1: int
ID at line 1,char 5: main
LP at line 1,char 9: (
RP at line 1,char 10: )
LC at line 1,char 11: {
TYPE at line 2,char 5: float
ID at line 2,char 11: f
ASSIGNOP at line 2,char 13: =
FLOAT at line 2,char 15: 2.5
SEMI at line 2,char 18: ;
TYPE at line 3,char 5: int
ID at line 3,char 9: n_num
ASSIGNOP at line 3,char 15: =
INT data at line 3,char 17: 30
SEMI at line 3,char 19: ;
IF at line 4,char 5: if
LP at line 4,char 7: (
ID at line 4,char 8: n
RELOP at line 4,char 10: >
FLOAT at line 4,char 12: 0.15
RP at line 4,char 16: )
LC at line 4,char 17: {
ID at line 5,char 9: printf
LP at line 5,char 15: (
Error Type A at line 5,char 16: Myterious character: '"'
Error Type A at line 5,char 17: Myterious character: '"'
RP at line 5,char 18: )
SEMI at line 5,char 19: ;
RC at line 6,char 5: }
ELSE at line 6,char 6: else
LC at line 6,char 10: {
ID at line 7,char 9: _f2
ASSIGNOP at line 7,char 13: =
ID at line 7,char 15: _f
STAR at line 7,char 18: *
FLOAT at line 7,char 20: 0.15
SEMI at line 7,char 24: ;
RELOP at line 8,char 9: <
RELOP at line 8,char 11: >
RELOP at line 8,char 13: ==
Error Type A at line 9,char 9: Myterious character: '#'
Error Type A at line 9,char 11: Myterious character: '%'
AND at line 9,char 13: &&
DIV at line 10,char 9: /
NOTE at line 10,char 13: //note
RC at line 11,char 5: }
RETURN at line 12,char 5: return
INT data at line 12,char 12: 0
SEMI at line 12,char 13: ;
RC at line 13,char 1: }
```

***测试二运行*:**

```
code12345@ubuntu:~/Desktop/byyl/1-cffx$ ./scanner test2.cmm
INT8 at line 1,char 1: 0547
Error Type A at line 1,char 6: Illegal octal number: '089'
INT16 at line 1,char 10: 0x5c4ad
INT16 at line 1,char 18: 0X345
INT16 at line 1,char 24: 0X1D7E
Error Type A at line 1,char 31: Illegal hexadecimal number: '0x4m4'
FLOAT at line 2,char 1: 1.23
FLOAT at line 2,char 6: 1.3e0
FLOAT at line 2,char 12: 13.5e9
FLOAT at line 2,char 19: 2.e-23
FLOAT at line 2,char 26: 3.
FLOAT at line 2,char 29: .08
Error Type A at line 2,char 33: Illegal float number: '2er'
Error Type A at line 2,char 37: Illegal float number: '15e'
Error Type A at line 2,char 41: Illegal float number: '1e2.5'
NOTE at line 3,char 1: // note1
NOTE at line 6,char 1: /* this
is a long long comment
*/
ID at line 7,char 1: h
ASSIGNOP at line 7,char 3: =
INT data at line 7,char 5: 5
DIV at line 7,char 7: /
INT data at line 7,char 9: 2
NOTE at line 7,char 11: // note2
code12345@ubuntu:~/Desktop/byyl/1-cffx$
```

```
INT8 at line 1,char 1: 0547
Error Type A at line 1,char 6: Illegal octal number: '089'
INT16 at line 1,char 10: 0x5c4ad
INT16 at line 1,char 18: 0X345
INT16 at line 1,char 24: 0X1D7E
Error Type A at line 1,char 31: Illegal hexadecimal number: '0x4m4'
FLOAT at line 2,char 1: 1.23
FLOAT at line 2,char 6: 1.3e0
FLOAT at line 2,char 12: 13.5e9
FLOAT at line 2,char 19: 2.e-23
FLOAT at line 2,char 26: 3.
FLOAT at line 2,char 29: .08
Error Type A at line 2,char 33: Illegal float number: '2er'
Error Type A at line 2,char 37: Illegal float number: '15e'
Error Type A at line 2,char 41: Illegal float number: '1e2.5'
NOTE at line 3,char 1: // note1
NOTE at line 6,char 1: /* this
is a long long comment
*/
ID at line 7,char 1: h
ASSIGNOP at line 7,char 3: =
INT data at line 7,char 5: 5
DIV at line 7,char 7: /
INT data at line 7,char 9: 2
NOTE at line 7,char 11: // note2
```

六、实验反思：

词法分析面向的对象是单个的字符，目的是把它们组成有效的单词（字符串）；而语法的分析则是利用词法分析的结果作为输入来分析是否符合语法规则并且进行语法制导下的语义分析，最后产生四元组(中间代码)，进行优化（可有可无）之后最终生成目标代码。因此词法分析作为语法分析的基础，虽然简单却具有重要地位。