编译原理实验报告

实验一 词法分析

学号：202011998088　　姓名：臧祝利　　日期：2022年9月22日

实验要求

通过 Flex 实现对C--程序的词法分析；

基本要求

- 输出基本的词法分析结果 ✅
- 输出未定义的标识符 ✅
- 识别单行注释 ✅

附加要求

- 识别八进制和十六进制数 ✅
- 识别指数形式浮点数 ✅
- 识别多行注释 ✅

实验假设

> 对C--语言的部分说明以及小组自身的假设

对于 `INT` 类型：《编译原理实践与指导教程》P117 A.2补充说明中提及

> 词法单元INT表示的是所有（无符号）整型常数

但考虑到负数普遍存在于程序中，因此加上**正负号**判定，且认为给定的数字均存在于32位整型范围内，因此不用考虑 `溢出` 问题；

对于 `FLOAT` 类型：补充说明中提及

> 词法单元FLOAT表示的是所有（无符号）浮点型整数；
> 小数点前后必须有数字出现；

但根据C语言的浮点数格式，认为**小数点前无数字**也为合法，且有**正负**判定，不考虑 `溢出` 问题；

对于非法的 `指数型浮点数`，小组内对此进行了一下定义

> 假设e前面出现的全是数字（及一个小数点），e后面只会出现数字、小数点和字母；

实验分工

小组成员：臧祝利、陈金利、孙泽林、赵建锟

分工如下：

> 臧祝利：代码+思考题 ☆
>
> 陈金利：指数形式浮点数（包括合法和非法）
>
> 孙泽林：注释（单行|多行），识别八进制和十六进制
>
> 赵建锟：ID，FLOAT，INT的定义及调试

具体内容：

- 写好大部分的tokens识别和规则；
- 解决三个思考题；
- 整合小组内容；
- 测试、debug并进行改正；

实验环境

- Linux：Ubuntu 20.04 LTS
- Flex： V2.6.4
- GCC： V9.3.0

代码以及调试均在虚拟机下完成，测试情况良好；

部分正则表达式使用了 菜鸟工具 的正则表达式在线测试，经过测试，均通过普遍测试后写入文件中；

实验设计

代码

先将简单的tokens的正则表达式及规则写入文件，并收集整合小组其他人的代码，实现最终测试；

输出基本的词法分析结果

输出的结果可以使用Flex自带的内置变量 `yytext` 进行当前语素的输出，而输出的句子会根据规则的内容进行输出；

基本tokens的 定义 代码如下，根据实验说明，将所有括号类型定义为 `BRACKET` ，关键字定义为 `KEYWORD` ；

[最终代码见cf.l文件]

```
%option yylineno
DIGIT[0-9]
LETTER[a-zA-Z]
SEMI ;
COMMA ,
ASSIGNOP =
RELOP \>|\<|>=|<=|==|!=
PLUS \+
MINUS \-
STAR \*
DIV \/
AND &&
OR \|\|
DOT \.
NOT !
BRACKET \(|\)|\[|\]|\{|\}
TYPE int|float
KEYWORD struct|if|else|while|return
%%
```

规则 如下：

```
%%
\n {column=1;}
{TYPE} {printf("TYPE at line %d,char %d: %s\n",yylineno,column,yytext),column+=yyleng;}
{SEMI} {printf("SEMI at line %d,char %d: %s\n",yylineno,column,yytext),column+=yyleng;}
{COMMA} {printf("COMMA at line %d,char %d: %s\n",yylineno,column,yytext),column+=yyleng;}
{KEYWORD} {printf("KEYWORD at line %d,char %d: %s\n",yylineno,column,yytext),column+=yyleng;}
{PLUS} {printf("PLUS at line %d,char %d: %s\n",yylineno,column,yytext),column+=yyleng;}
{MINUS} {printf("MINUS at line %d,char %d: %s\n",yylineno,column,yytext),column+=yyleng;}
{STAR} {printf("STAR at line %d,char %d: %s\n",yylineno,column,yytext),column+=yyleng;}
```

```
{DIV} {printf("DIV at line %d,char %d: %s\n",yylineno,column,yytext),column+=yyleng;}
{AND} {printf("AND at line %d,char %d: %s\n",yylineno,column,yytext),column+=yyleng;}
{OR} {printf("OR at line %d,char %d: %s\n",yylineno,column,yytext),column+=yyleng;}
{BRACKET} {printf("BRACKET at line %d,char %d: %s\n",yylineno,column,yytext),column+=yyleng;}
{ASSIGNOP} {printf("ASSIGNOP at line %d,char %d: %s\n",yylineno,column,yytext),column+=yyleng;}
{RELOP} {printf("RELOP at line %d,char %d: %s\n",yylineno,column,yytext),column+=yyleng;}
{DOT} {printf("DOT at line %d,char %d: %s\n",yylineno,column,yytext),column+=yyleng;}
{NOT} {printf("NOT at line %d,char %d: %s\n",yylineno,column,yytext),column+=yyleng;}
" " {column+=1;}
\t {column+=4;}
. {printf("ERROR Type A at line %d,char %d: Mysterious charcter:'%s'\n",yylineno,column,yytext),column++;}
%%
```

输出未定义的标识符

这里按照实验给定的输出要求

```
. {printf("ERROR Type A at line %d,char %d: Mysterious charcter:'%s'\n",yylineno,column,yytext),column++;}
```

对于未定义的标识符，即没有经过规则定义的，将会进行单个字符的输出；

输出行、列号

对于每一行来说，使用Flex自带的变量`yylineno`,可以指示当前识别到哪一行；

而列则需要通过自定义一个变量`column`来实现；列的具体内容详见思考题的第三个”如何正确获得token的字符数？ “；

思考题

- 文法中token的正则表达式确定?

此实验的主要目的就是设计不同tokens的正则表达式，从而通过Flex实现词法分析；因此正确匹配内容是此实验的重中之重，即要根据不同类型的定义规则入手，进行正则表达式的书写，并详尽考虑冲突情况（即一种token可以被多种规则匹配，因此规则的顺序很重要）；

- 实验中如何忽略空白字符?

在最开始编写框架内容时，发现对于

```
. {printf("Error Type A at line %d,char %d: Mysterious charcter:'%s'\n",yylineno,column,yytext),column++;}
```

会打印出空格内容；

原因：空格也属于特殊字符，因此识别后会输出；

如何解决?

对空格和制表符添加对应的表达式匹配，碰到时只后移列指针$column$(见下一个问题)而不进行输出，即

```
" " {column+=1;}
\t {column+=4;}
```

- 实验中如何如何正确获得token的字符数?

已有工具： Flex内置变量`yyleng`表示当前语法单元对应的语素的长度

最初想法： 寻找Flex里与`yyline`相同的，用来指示列的内置变量；--> 没有找到 😭

第二个想法： 得知Flex的头部定义部分可以定义变量，因此定义了列指针$columm = 1;$，同时添加针对回车的规则，以实现换行对列指针的初始化

```
\n {column=1;}
```

此时的想法是对后面的代码入手，比如说在循环$yylex()$时对列指针进行操作，比如这样的代码：

```
while (yylex() != 0) column++;
或者
while (column++,yylex() != 0);
```

运行后发现$column$的值并没有任何变化 💭

最终想法： 突然想到还有个 `yyleng` 没有用，又想到下面的代码如果不能运行的话，可以将列指针的变化写到**规则**部分；

一开始写的是

```
printf("XXX at line %d,char %d: %s\n",yylineno,column+=yyleng,yytext);
```

运行出来发现数值发生了变化，但是值为加了长度之后的；

最后改成了

```
printf("XXX at line %d,char %d: %s\n",yylineno,column,yytext),column+=yyleng;
```

成功解决问题；

实验结果

1 test.cmm

```
int main(){
    float f = 2.5;
    int n_num = 30;
    if (n > 0.15){
        printf("");
    }else{
        _f2 = _f * 0.15;
        < > ==
        # % &&
        /     //note
    }
    return 0;
}
```

```
TYPE at line 1,char 1: int
ID at line 1,char 5: main
BRACKET at line 1,char 9: (
BRACKET at line 1,char 10: )
BRACKET 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: ;
KEYWORD at line 4,char 5: if
BRACKET at line 4,char 8: (
ID at line 4,char 9: n
RELOP at line 4,char 11: >
FLOAT at line 4,char 13: 0.15
BRACKET at line 4,char 17: )
```

```
BRACKET at line 4,char 18: {
ID at line 5,char 9: printf
BRACKET at line 5,char 15: (
ERROR Type A at line 5,char 16: Mysterious charcter:'"'
ERROR Type A at line 5,char 17: Mysterious charcter:'"'
BRACKET at line 5,char 18: )
SEMI at line 5,char 19: ;
BRACKET at line 6,char 5: }
KEYWORD at line 6,char 6: else
BRACKET 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: Mysterious charcter:'#'
ERROR Type A at line 9,char 11: Mysterious charcter:'%'
AND at line 9,char 13: &&
DIV at line 10,char 9: /
NOTE at line 10,char 15: //note
BRACKET at line 11,char 5: }
KEYWORD at line 12,char 5: return
INT data at line 12,char 12: 0
SEMI at line 12,char 13: ;
BRACKET at line 13,char 1: }
```

2 test2.cmm

```
0547 089 0x5c4ad 0X345 0X1D7E 0x4m4
1.23 1.3e0 13.5e9 2.e-23 3. .08 2er 15e 1e2.5
// note1
/* this
is a long long comment
*/
h = 5 / 2 // note2
```

```
INT8 at line 1,char 1: 0547
Error Type A at line 1,char 6: Illgal 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: Illgal 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: Illgal float number :2er
Error Type A at line 2,char 37: Illgal float number :15e
Error Type A at line 2,char 41: Illgal float number :1e2.5
NOTE at line 3,char 1: // note1
NOTEE 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
```

3 自测试test0.cmm

原则：尽量把各种复杂的类型的各种可能形式表现出来，检测是否能正确识别类型；

```
int main(){
    iff
    .  !
    , ≠
    struct
    while
    0x12f
    0x8rk
    retur
    3.
    ew
    2er
    -3.e2
    3.e
    .e2
    -4.e2.5
    -4.e-3
    -4.e-2.5
    .09
    8.
    elsee
    e2
    elwe
    we2e
    else
    retsdn0
    /*

    nihao

    */
    float f=12.;
    .56
    3.45
    int a = -114514;
    int b = -34.5 + 456;    /
    if (a>0&&b>0){  //this is a note
        a=a-b;
        if (a≥b) printf("haha");
    }
    b||c;
    $   @
    01234
    00
@ ^
    000.000012
    034567821
    return 0;
}
```

```
TYPE at line 1,char 1: int
```

```
ID at line 1,char 5: main
BRACKET at line 1,char 9: (
BRACKET at line 1,char 10: )
BRACKET at line 1,char 11: {
ID at line 2,char 5: iff
DOT at line 3,char 5: .
NOT at line 3,char 8: !
COMMA at line 4,char 5: ,
RELOP at line 4,char 7: ≠
KEYWORD at line 5,char 5: struct
KEYWORD at line 6,char 5: while
INT16 at line 7,char 5: 0x12f
Error Type A at line 8,char 5: Illgal hexadecimal number: 0x8rk
ID at line 9,char 5: retur
FLOAT at line 10,char 5: 3.
ID at line 11,char 5: ew
Error Type A at line 12,char 5: Illgal float number :2er
FLOAT at line 13,char 5: -3.e2
Error Type A at line 14,char 5: Illgal float number :3.e
Error Type A at line 15,char 5: Illgal float number :.e2
Error Type A at line 16,char 5: Illgal float number :-4.e2.5
FLOAT at line 17,char 5: -4.e-3
FLOAT at line 18,char 5: -4.e-2
FLOAT at line 18,char 11: .5
FLOAT at line 19,char 5: .09
FLOAT at line 20,char 5: 8.
ID at line 21,char 5: elsee
ID at line 22,char 5: e2
ID at line 23,char 5: elwe
ID at line 24,char 5: we2e
KEYWORD at line 25,char 5: else
ID at line 26,char 5: retsdn0
NOTEE at line 31,char 5: /*

    nihao

    */
TYPE at line 32,char 5: float
ID at line 32,char 11: f
ASSIGNOP at line 32,char 12: =
FLOAT at line 32,char 13: 12.
SEMI at line 32,char 16: ;
FLOAT at line 33,char 5: .56
FLOAT at line 34,char 5: 3.45
TYPE at line 35,char 5: int
ID at line 35,char 9: a
ASSIGNOP at line 35,char 11: =
INT data at line 35,char 13: -114514
SEMI at line 35,char 20: ;
TYPE at line 36,char 5: int
ID at line 36,char 9: b
ASSIGNOP at line 36,char 11: =
FLOAT at line 36,char 13: -34.5
PLUS at line 36,char 19: +
INT data at line 36,char 21: 456
SEMI at line 36,char 24: ;
DIV at line 36,char 29: /
KEYWORD at line 37,char 5: if
BRACKET at line 37,char 8: (
ID at line 37,char 9: a
```

```
RELOP at line 37,char 10: >
INT data at line 37,char 11: 0
AND at line 37,char 12: &&
ID at line 37,char 14: b
RELOP at line 37,char 15: >
INT data at line 37,char 16: 0
BRACKET at line 37,char 17: )
BRACKET at line 37,char 18: {
NOTE at line 37,char 21: //this is a note
ID at line 38,char 9: a
ASSIGNOP at line 38,char 10: =
ID at line 38,char 11: a
MINUS at line 38,char 12: -
ID at line 38,char 13: b
SEMI at line 38,char 14: ;
KEYWORD at line 39,char 9: if
BRACKET at line 39,char 12: (
ID at line 39,char 13: a
RELOP at line 39,char 14: ≥
ID at line 39,char 16: b
BRACKET at line 39,char 17: )
ID at line 39,char 19: printf
BRACKET at line 39,char 25: (
ERROR Type A at line 39,char 26: Mysterious charcter:'"'
ID at line 39,char 27: haha
ERROR Type A at line 39,char 31: Mysterious charcter:'"'
BRACKET at line 39,char 32: )
SEMI at line 39,char 33: ;
BRACKET at line 40,char 5: }
ID at line 41,char 5: b
OR at line 41,char 6: ||
ID at line 41,char 8: c
SEMI at line 41,char 9: ;
ERROR Type A at line 42,char 5: Mysterious charcter:'$'
ERROR Type A at line 42,char 9: Mysterious charcter:'@'
INT8 at line 43,char 5: 01234
INT8 at line 44,char 5: 00
ERROR Type A at line 45,char 1: Mysterious charcter:'@'
ERROR Type A at line 45,char 3: Mysterious charcter:'^'
FLOAT at line 46,char 5: 000.000012
Error Type A at line 47,char 5: Illgal octal number: 034567821
KEYWORD at line 48,char 5: return
INT data at line 48,char 12: 0
SEMI at line 48,char 13: ;
BRACKET at line 49,char 1: }
```

实验反思

$Q1$.有趣的小tips：

将代码给组员测试后，由于一开始是从虚拟机直接转移的到本地的，从本地再转移时候出现问题： “\r"的识别问题；

根据组员调查后发现：

> Windows下用enter换行等于在最后添加\r\n，
>
> 而Linux是在最后添加\n

用linux系统交换文件后，问题解决，因此本实验暂时忽略“\r”符号的识别；

$Q2.$ Flex 的规则句式前后顺序会影响程序的运行?

比如最后调试 `test.cmm` 时，由于规则中 `KEYWORD` 的位置靠后，导致 `else` 由于是e开头且前面没有数字，被认定为非法的浮点数，因此调换了顺序；

甚至也会发生因为顺序问题而出现的flex不通过的情况；

为什么会出现这种奇怪的问题呢?

我们假定非法型浮点数e前面必须有数字或者小数点或者都有，e后面会出现小数点、数字和字母的组合，因此我们按这样写的表达式里面其实会包括合法浮点数以及以e开头的关键词或者标识符；出现这种问题的原因就是在于规则的顺序，如果优先识别关键词，其次是标识符，将非法型浮点数的优先级靠后，可以实现较好的效果；

即 如果两个表示同时匹配一个词法单元，会按照规则的顺序选择动作

$Q3.$ 解决确定列位置的问题时，为什么写入while循环内部或者是条件的列指针不会执行？即使是尝试添加printf语句进行调试也无结果?

由于对flex了解不够，该问题尚待解决，但不影响实验的进行，只是个人碰到的小问题；

$Q4.$ 正则表达式可能会出现一些bug，希望可以通过他人独特的视角来提出bug，方便修改与改进；