LAB1 实验报告

学号 201908030406 姓名 陈叹

一、实验要求：

本次实验需要各位同学根据 `cminux-f` 的词法补全 lexical_analyer.l 文件，完成词法分析器，能够输出识别出的 `token`，`type`,`line(刚出现的行数)`，`pos_start(该行开始位置)`，`pos_end(结束的位置,不包含)`。

二、实验难点

具体难点详见步骤三。
这里说几点重要的：
(1)环境配置以及成功编译运行，由于初次探索，这一步花费时间较长。以及熟悉这门语言的基本模式和结构，语法等。
(2)正则表达式的设计。主要是数字和注释的正则表达式的设计，错了好几次，细节比较多。
(3)当前位置的记录。不仅在读到单个的换行符，空格等，在注释的时候，也要跟踪记录这些值，确保不出错。

三、实验设计

(1)环境安装配置：
我已经有 ubuntu20.04 了，因此，直接执行 `sudo apt-get install llvm bison flex` 即可，然后查看版本，如下图所示。(我已经安装好了)

```
nidhs@nidhs-VirtualBox:~$ sudo apt-get install llvm bison flex
[sudo] nidhs 的密码：
正在读取软件包列表... 完成
正在分析软件包的依赖关系树
正在读取状态信息... 完成
bison 已经是最新版 (2:3.5.1+dfsg-1)。
flex 已经是最新版 (2.6.4-6.2)。
llvm 已经是最新版 (1:10.0-50~exp1)。
升级了 0 个软件包，新安装了 0 个软件包，要卸载 0 个软件包，有 134 个软件包未被升级。
nidhs@nidhs-VirtualBox:~$ flex -V
flex 2.6.4
nidhs@nidhs-VirtualBox:~$ bison -V
bison (GNU Bison) 3.5.1
由 Robert Corbett 和 Richard Stallman 编写。

版权所有 (C) 2020 Free Software Foundation, Inc.
这是自由软件；请参考源代码的版权声明。本软件不提供任何保证，甚至不会包括
可售性或适用于任何特定目的的保证。
nidhs@nidhs-VirtualBox:~$
```

然后，运行仓库中那个识别单词数量的程序，命名为 test.l。验证结果如下图。

```
nidhs@nidhs-VirtualBox:~/byyl/lab1$ flex test.l
nidhs@nidhs-VirtualBox:~/byyl/lab1$ gcc lex.yy.c -lfl
nidhs@nidhs-VirtualBox:~/byyl/lab1$ ./a.out
hello world ! aaa ab1

look, I find 4 words of 15 chars
nidhs@nidhs-VirtualBox:~/byyl/lab1$
```

说明环境配置完成。

(2)

观察仓库中给出的代码，发现没有 main 函数，并且我们需要为 flex 的模式与动作进行补全，以及 analyzer 函数进行完善。

首先，main 函数主要作用就是调用 analyzer 函数。编写代码如下：

```
int main(int argc,char** argv)
{
    char input_file[5]={'t','e','x','t','\0'};
    Token_Node* tok;
    analyzer(input_file,tok);
}
```

其中保存需要识别的文本的文件名问 text，当然，也可以改成 scanf 输入或者直接从 stdin 输入文本。设定默认的话方便调试。

然后，analyzer 函数开头输出调试信息，并将仓库中给的其他代码暂时注释，以进行初步调试，代码如下图。

```
void analyzer(char* input_file, Token_Node* token_stream){
    printf("11111\n");
    lines = 1;
    pos_start = 1;
    pos_end = 1;
    if(!(yyin = fopen(input_file,"r"))){
        printf("[ERR] No input file\n");
        exit(1);
    }
    printf("[START]: Read from: %s\n", input_file);

    int token;
    int index = 0;
```

运行结果如下图：

```
nidhs@nidhs-VirtualBox:~/byyl/lab1$ flex src
nidhs@nidhs-VirtualBox:~/byyl/lab1$ gcc lex.yy.c -lfl -o lex.out
nidhs@nidhs-VirtualBox:~/byyl/lab1$ ./lex.out
11111
[START]: Read from: text
```

证明文件链接完成，可以开始正式编写了。

(3)将所有特殊的单个终结符号加入正则规则

容易发现，例如'+','('等符号被读取时，可以立即返回值，因为它一定是单个符号组成的。因此，在正则表达式中加入以下内容：

```
"+" {return ADD;}
"-" {return SUB;}
"*" {return MUL;}
"/" {return DIV;}
"<" {return LT;}
"<=" {pos_end++;return LTE;}
">" {return GT;}
">=" {pos_end++;return GTE;}
"==" {pos_end++;return EQ;}
"!=" {pos_end++;return NEQ;}
";" {return SEMICOLON;}
"," {return COMMA;}
"(" {return LPARENTHESE;}
")" {return RPARENTHESE;}
"[" {return LBRACKET;}
"]" {return RBRACKET;}
"{" {return LBRACE;}
"}" {return RBRACE;}
```

相应的，在 analyzer 函数中加入以下内容：

```
while(token = yylex()){
    switch(token){
        case ADD:
            printf("%c   %d   %d   %d   %d\n",'+',(int)ADD,lines,pos_start,pos_end);
            break;
        case SUB:
            printf("%c   %d   %d   %d   %d\n",'-',(int)SUB,lines,pos_start,pos_end);
            break;
        case MUL:
            printf("%c   %d   %d   %d   %d\n",'*',(int)MUL,lines,pos_start,pos_end);
            break;
        case DIV:
            printf("%c   %d   %d   %d   %d\n",'/',(int)DIV,lines,pos_start,pos_end);
            break;
        case LT:
            printf("%c    %d   %d   %d   %d\n",'<',(int)LT,lines,pos_start,pos_end);
            break;
        case LTE:
            printf("%c%c   %d   %d   %d   %d\n",'<','=',(int)LTE,lines,pos_start,pos_end);
            break;
        case GT:
            printf("%c   %d   %d   %d   %d\n",'>',(int)GT,lines,pos_start,pos_end);
            break;
        case GTE:
            printf("%c%c   %d   %d   %d   %d\n",'>','=',(int)GTE,lines,pos_start,pos_end);
            break;
        case EQ:
            printf("%c%c   %d   %d   %d   %d\n",'=','=',(int)EQ,lines,pos_start,pos_end);
            break;
        case NEQ:
            printf("%c%c   %d   %d   %d   %d\n",'!','=',(int)NEQ,lines,pos_start,pos_end);
            break;
        case ASSIN:
            printf("%c   %d   %d   %d   %d\n",'+',(int)ASSIN,lines,pos_start,pos_end);
            break;
        case SEMICOLON:
            printf("%c   %d   %d   %d   %d\n",';',(int)SEMICOLON,lines,pos_start,pos_end);
            break;
        case COMMA:
            printf("%c   %d   %d   %d   %d\n",',',(int)COMMA,lines,pos_start,pos_end);
            break;
        case LPARENTHESE:
            printf("%c   %d   %d   %d   %d\n",'(',(int)LPARENTHESE,lines,pos_start,pos_end);
            break;
```

后面没有截完，代码类似。

其中 yylex()函数能够读取文本内容，并与正则表达式匹配，并执行匹配之后的动作。

另外，还需要定位到相应符号的位置。通过 3 个变量 `pos_start,pos_end,lines` 来标记位置。那么，每次 yylex()返回 token 后，pos_start 应该等于 pos_end+1，同时，将 pos_end 也赋为该值。在正则表达式匹配的长度>=2 时，需要改变 pos_end 的值，如上面和下图的代码所示：

```
pos_start=pos_end+1;
pos_end=pos_end+1;
```

同时，注意到读取到空白字符时，需要改变这三个位置。暂时先只考虑' '和'\n'，模式动作如下图所示：

```
"\n" {lines++;pos_start=1;pos_end=1;}
" " {pos_start++,pos_end++;}
```

动作中没有返回值，因为没必要返回，直接继续读取即可。

编辑 text 中的内容如下：

```
1 , {
2 } ; <= >=
3 ====
4 () [[
5 +*-/
```

运行结果如下图：

```
nidhs@nidhs-VirtualBox:~/byyl/lab1$ ./lex.out
11111
[START]: Read from: text
,    271    1    1    1
{    276    1    3    3
}    277    2    1    1
;    270    2    3    3
<=    264    2    5    6
>=    266    2    8    9
==    267    3    1    2
==    267    3    3    4
(    272    4    1    1
)    273    4    2    2
[    274    4    4    4
[    274    4    5    5
+    259    5    1    1
*    261    5    2    2
-    260    5    3    3
/    262    5    4    4
[END]: Analysis completed.
```

可以看到，运行结果完全正确，该部分编写代码正确。

(3)接下来，要处理变量、数字等较为复杂的符号。

①首先，考虑变量 IDENTIFIER。

其正则表达式比较简单：$[\_a-zA-Z][\_a-zA-Z0-9]*$

编写模式动作如下图所示

```
[_a-zA-Z][_a-zA-Z0-9]* {
    strcpy(cur.text,yytext);

    int i=0;
    while(1){
        char tmp=cur.text[i];
        if(tmp!='_'&&(tmp<'a'||tmp>'z')&&(tmp<'A'||tmp>'Z')&&(tmp<'0'||tmp>'9')){
            break;
        }
        i++;
    }
    pos_end+=i-1;
    return IDENTIFIER;
}
```

其中麻烦的点在于计算长度，也可以考虑看到'\0'结束来判断长度。

yytext 是一个标记匹配当前正则表达式的字符数组，cur 是我定义的一个全局 Token_Node，用来记录当前符号的信息。

在 analyzer 函数中：

```
case IDENTIFIER:
    printf("%s   %d   %d   %d   %d\n",cur.text,(int)IDENTIFIER,lines,pos_start,pos_end);
    break;
```

就将当前存储于 cur.text 中的文本输出，并输出相关行列信息即可。

下面是部分运行结果：

```
int    280    3    1    3
float    281    3    5    9
intfloat    285    4    1    8
_int    285    5    1    4
a    285    5    6    6
;    270    5    7    7
```

可以看到，词法分析器很好的识别了保留字符"int","float"和一般的变量"intfloat","_int"等，并正确输出了他们的位置。

②整数 INTEGER

正则表达式：$[1-9][0-9]*$

动作代码：

```
[1-9][0-9]* {
    strcpy(cur.text,yytext);
    int i=0;
    while(1){
        if(cur.text[i]=='\0') break;
        i++;
    }
    pos_end+=i-1;
    return INTEGER;
}
```

识别文本：

```
1 123 4568
2 01345
3 _int a;
```

部分运行结果：

```
[START]: Read from: text
123    286    1    1    3
4568    286    1    5    8
[ERR]: unable to analysize 0 at 2 line, from 1 to 1
1345    286    2    2    5
_int    285    3    1    4
```

我们发现，本来 01345 是不匹配我编写的任何一个正则表达式的。但是在读取完 0，输出一个 error 之后，后面的 1345 匹配的我的表达式，这个到后面错误处理再来解决，其他运行结果还是非常正确的。

并且现在，我只编写了纯粹由数字组成的数字，其实 C 中还可以有类似"0x3f"这样的十六进制定义，它也表示一个整数。这个后面有机会再加。

③浮点数 FLOATPOINT

正则表达式：$[1-9][0-9]*.[0-9]*$

动作代码：

```
[1-9][0-9]*.[0-9]* {
    strcpy(cur.text,yytext);
    int i=0;
    while(1){
        if(cur.text[i]=='\0') break;
        i++;
    }
    pos_end+=i-1;
    return FLOATPOINT;
}
```

部分运行结果：

```
nidhs@nidhs-VirtualBox:~/byyl/lab1$ ./a.out
[START]: Read from: text
123 4568   287   1   1   8
13045   286   2   1   5
13.45   287   3   1   5
13.045   287   3   7   12
 int   285   4   1   4
```

④数组 ARRAY

其实有点疑惑数组竟然是算作一整个词法单元的。

那么它的正则表达式是什么呢？

应当是一个变量名+'['+数字+']',其中'['和']'要转义。

那么，数组的正则表达式：

$$[\_a-zA-Z][\_a-zA-Z0-9]*\[[1-9][0-9]*\]$$

不过我用 C 编了代码发现，数组的各个部分是可以分开的，中间可以有若干个空白字符。

因此，需要在中间加入空白字符，包括空格，换行，还包括制表符，换页符等。

单个空白字符的正则表达式为：$[\ \f\n\r\t\v]$

因此，最终数组的正则表达式为：

$$[\_a-zA-Z][\_a-zA-Z0-9]*[\ \f\n\r\t\v]*\[[\ \f\n\r\t\v]*[1-9][0-9]*[\ \f\n\r\t\v]*\]$$

另外，还需要注意输出的位置问题，遇到'\n'时要执行合理的位置变换。

代码：

```
[_a-zA-Z][_a-zA-Z0-9]*[ \f\n\r\t\v]*\[[ \f\n\r\t\v]*[1-9][0-9]*[ \f\n\r\t\v]*\] {
    int p=0,q=0;
    for(int j=0;j<256;j++) cur.text[j]='\0';
    while(1){
        if(yytext[p]=='\0') break;
        if(yytext[p]==13||yytext[p]==32||yytext[p]==11||yytext[p]==12) p++,pos_end++;
        else if(yytext[p]==10) lines++,pos_end=0;
        else cur.text[q++]=yytext[p++],pos_end++;
    }
    return ARRAY;
}
```

测试文本：

```
a   [ 112341 ]
_int a[1];
```

运行结果：

```
a[112341]   288   1   1   15
_int   285   2   1   4
a[1]   288   2   6   10
```

⑤LETTER

只要在 IDENTIFIER 之前匹配即可。因为显然，所有的 letter 都是 indentifier

代码：

```
[a-zA-Z] {
    strcpy(cur.text,yytext);
    int i=0;
    while(1){
        if(cur.text[i]=='\0') break;
        i++;
    }
    pos_end+=i-1;
    return LETTER;
}
[_a-zA-Z][_a-zA-Z0-9]* {
    strcpy(cur.text,yytext);
    int i=0;
    while(1){
        if(cur.text[i]=='\0') break;
        i++;
    }
    pos_end+=i-1;
    return IDENTIFIER;
}
```

运行结果：

```
a   289   1   1   1
Z   289   1   3   3
```

可以看到，还是识别出来了。

因此该部分编写完成。

(4)最后，处理其他，例如注释等。

①注释

以连续的/*开头，*/结尾，中间可以有任意字符，但是不能有连续的*/.

因此，构造正则表达式：\/*([^*]|*[^\/])**\/

其中'/','*'均需转义，中间部分表示任意非'*'字符或者是*后面跟一个任意不是'/'的字符。这样的闭包。

但是由于这样会把'\n'包含进去，导致行数，列数计算完全错误。需要遍历这个 yytext，当有'\n'时，做出相应变换。

代码：

```
\/\*([^\*]|\*[^\/])*\*\/ {
    strcpy(cur.text,yytext);
    int i=0;
    while(1){
        if(cur.text[i]=='\0') break;
        if(cur.text[i]=='\n') lines++,pos_end=0;
        else pos_end++;
        i++;
    }
    pos_end+=i-1;
    return COMMENT;
}
```

运行结果：

```
/*aslh
***dskajf
ljfg*/   291   7   5   28
```

合理测试出了较难的 case。

②EOL，文件结束符，读取不到，不用管

③BLANK，空字符.
正则表达式：我将'\n'分开考虑了，具体见代码
其中'\n'需要换行，其他只需要位置+1 即可。
代码：

```
"\n" {lines++;pos_start=1;pos_end=1;}
[ \f\r\t\v] {pos_start++,pos_end++;}
```

注意无需返回值。
④ERROR
这个非常麻烦，后面根据样例调试。

(5)
至此，基本代码框架已经搭建完成。
我找到了 Git 上的测试样例进行测试：
①1.cminus
我之前很多规范跟标准的不一样，例如：
结束位置需要+1；
ASSGN 是赋值的意思，将其加上；
ARRAY 是[]，而带下标的数组，如 a[1]，认为是 4 个字符。我好多都白写了，将其改为和标准一致，代码如下图：

```
\[[ \f\n\r\t\v]*\] {
    int p=0,q=0;
    for(int j=0;j<256;j++) cur.text[j]='\0';
    while(1){
        if(yytext[p]=='\0') break;
        if(yytext[p]==13||yytext[p]==32||yytext[p]==11||yytext[p]==12) p++,pos_end++;
        else if(yytext[p]==10) lines++,pos_end=0,p++;
        else cur.text[q++]=yytext[p++],pos_end++;
    }
    return ARRAY;
}
```

数字的部分我遗漏了单独一个 0 的情况；
标准输出用的是制表符，我用的是三个空格，全部改掉；
发现单个的变量，比如'u',标准输出将其识别为 IDENTIFIER，而我将其识别为 LETTER,改成先识别 IDENTIFIER，只不过这样的话 LETTER 似乎永远无法识别到了。
改完这些之后，终于没有铺天盖地的不匹配了。
运行完后的 diff 结果如下图所示：

```
nidhs@nidhs-VirtualBox:~/byyl/lab1$ diff ans1 1.token
101a102
>
```

发现标准输出最后输出了一个空行。
那我也加一个。
终于没有错误了。
②2.cminus
发现我的浮点数也没有考虑整数部分为 0 的情形，加上之后代码如下：

```
([0-9]+.|[0-9]*.[0-9]+) {
    strcpy(cur.text,yytext);
    int i=0;
    while(1){
        if(cur.text[i]=='\0') break;
        i++;
    }
    pos_end+=i;
    return FLOATPOINT;
}
```

```
nidhs@nidhs-VirtualBox:~/byyl/lab1$ diff ans2 2.tokens
nidhs@nidhs-VirtualBox:~/byyl/lab1$
```

完全匹配。

③3.cminus

发现标准输出的注释直接不输出。

白忙活了，把注释输出相关全部删除。

但是，注释中仍然要记录当前的位置。

改动之后的 COMMENT 动作如下图：

```
\/\*([^\*]|\*[^\/])*\*\/ {
    strcpy(cur.text,yytext);
    int i=0;
    while(1){
        if(cur.text[i]=='\0') break;
        if(cur.text[i]=='\n') lines++,pos_end=0;
        else pos_end++;
        i++;
    }
    return COMMENT;
}
```

运行结果如下图

```
nidhs@nidhs-VirtualBox:~/byyl/lab1$ diff ans3 3.tokens
nidhs@nidhs-VirtualBox:~/byyl/lab1$
```

完全匹配。

④4.cminus

```
nidhs@nidhs-VirtualBox:~/byyl/lab1$ diff ans4 4.tokens
nidhs@nidhs-VirtualBox:~/byyl/lab1$
```

直接正确

⑤5.cminus

不匹配如下。说明没识别出注释。

```
nidhs@nidhs-VirtualBox:~/byyl/lab1$ diff ans5 5.tokens
1,14d0
< /     262     1       1       2
< *     261     1       2       3
< lsjdlsjflsjf  285     1       3       15
< [ERR]: unable to analysize & at 1 line, from 15 to 15
< [ERR]: unable to analysize & at 1 line, from 15 to 15
< [ERR]: unable to analysize | at 1 line, from 15 to 15
< [ERR]: unable to analysize | at 1 line, from 15 to 15
< [ERR]: unable to analysize % at 1 line, from 15 to 15
< wdalkds       285     1       15      22
< *     261     1       22      23
< *     261     1       23      24
< *     261     1       24      25
< *     261     1       25      26
< /     262     1       26      27
```

观察注释：

```
/*lsjdlsjflsjf&&||%wdalkds****/
```

我的正则表达式是*后面连任意的单个字符，只要不包括'/'，因此在句子末尾，*两两匹配导致'/'没有匹配而出错，修改正则表达式：

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

修改之后的规则为至少一个*，后面跟一个非'/'的字符，

再次运行就通过了，如下图

```
nidhs@nidhs-VirtualBox:~/byyl/lab1$ diff ans5 5.tokens
nidhs@nidhs-VirtualBox:~/byyl/lab1$
```

⑥6.cminus

发现注释中换行的时候，pos_end 设置成了 0，其实应该是 1.错位了。

修正之后运行通过，如下图

```
nidhs@nidhs-VirtualBox:~/byyl/lab1$ diff ans6 6.tokens
nidhs@nidhs-VirtualBox:~/byyl/lab1$
```

6 个测试样例全部通过，加一张完整截图：

```
nidhs@nidhs-VirtualBox:~/byyl/lab1$ ./a.out >ans1
1.cminus
nidhs@nidhs-VirtualBox:~/byyl/lab1$ diff ans1 1.tokens
nidhs@nidhs-VirtualBox:~/byyl/lab1$ ./a.out >ans2
2.cminus
nidhs@nidhs-VirtualBox:~/byyl/lab1$ diff ans2 2.tokens
nidhs@nidhs-VirtualBox:~/byyl/lab1$ ./a.out >ans3
3.cminus
nidhs@nidhs-VirtualBox:~/byyl/lab1$ diff ans3 3.tokens
nidhs@nidhs-VirtualBox:~/byyl/lab1$ ./a.out >ans4
4.cminus
nidhs@nidhs-VirtualBox:~/byyl/lab1$ diff ans4 4.tokens
nidhs@nidhs-VirtualBox:~/byyl/lab1$ ./a.out >ans5
5.cminus
nidhs@nidhs-VirtualBox:~/byyl/lab1$ diff ans5 5.tokens
nidhs@nidhs-VirtualBox:~/byyl/lab1$ ./a.out >ans6
6.cminus
nidhs@nidhs-VirtualBox:~/byyl/lab1$ diff ans6 6.tokens
nidhs@nidhs-VirtualBox:~/byyl/lab1$
```

(6)

①

发现数字和变量的处理都错了，得按实验指导书上的来。

不过总体来说，是将复杂的正则表达式变得简单。
因此，更新代码如下：

```
[0-9]+ {
    strcpy(cur.text,yytext);
    int i=0;
    while(1){
        if(cur.text[i]=='\0') break;
        i++;
    }
    pos_end+=i;
    return INTEGER;
}
[0-9]+\.|[0-9]*\.[0-9]+ {
    strcpy(cur.text,yytext);
    int i=0;
    while(1){
        if(cur.text[i]=='\0') break;
        i++;
    }
    pos_end+=i;
    return FLOATPOINT;
}
[a-zA-Z]+ {
    strcpy(cur.text,yytext);
    int i=0;
    while(1){
        if(cur.text[i]=='\0') break;
        i++;
    }
    pos_end+=i;
    return IDENTIFIER;
}
[a-zA-Z] {
    strcpy(cur.text,yytext);
    int i=0;
    while(1){
        if(cur.text[i]=='\0') break;
        i++;
    }
    pos_end+=i;
    return LETTER;
}
```

②
自己设计样例测试时，输入如下文本

```
1 /**********/
2 aa
3 /*********/
```

发现中间的 aa 竟然被当成注释，没有任何输出。显然是我注释的正则表达式写错了。
更新注释的正则表达式：\/* ([^*]|[*] + [^\/*]) * [*] +\/
这样就没问题了。
③发现我没有按照要求来做，其实不用这么麻烦，很多接口都已经设计好了。。
因此将代码修改如下：

```
\[\] {
    pos_end+=2;
    return ARRAY;
}
"else" {pos_end+=4;return ELSE;}
"if" {pos_end+=2;return IF;}
"int" {pos_end+=3;return INT;}
"float" {pos_end+=5;return FLOAT;}
"return" {pos_end+=6;return RETURN;}
"void" {pos_end+=4;return VOID;}
"while" {pos_end+=5;return WHILE;}
[a-zA-Z]+ {
    int i=0;
    while(1){
        if(yytext[i]=='\0') break;
        pos_end++;i++;
    }
    return IDENTIFIER;
}
[a-zA-Z] {
    int i=0;
    while(1){
        if(yytext[i]=='\0') break;
        pos_end++;i++;
    }
    return LETTER;
}
[0-9]+ {
    int i=0;
    while(1){
        if(yytext[i]=='\0') break;
        pos_end++;i++;
    }
    return INTEGER;
}
[0-9]+\.|[0-9]*\.[0-9]+ {
    int i=0;
    while(1){
        if(yytext[i]=='\0') break;
        pos_end++;i++;
    }
    return FLOATPOINT;
}
\/\*([^\*]|[\*]+[^\/\*])*[\*]+\/ {
    int i=0;
    while(1){
        if(yytext[i]=='\0') break;
        if(yytext[i]=='\n') lines++,pos_end=1;
        else pos_end++;
        i++;
    }
    return COMMENT;
}
```

```
"+" {pos_end++;return ADD;}
"-" {pos_end++;return SUB;}
"*" {pos_end++;return MUL;}
"/" {pos_end++;return DIV;}
"<" {pos_end++;return LT;}
"<=" {pos_end++;pos_end++;return LTE;}
">" {pos_end++;return GT;}
">=" {pos_end++;pos_end++;return GTE;}
"==" {pos_end++;pos_end++;return EQ;}
"!=" {pos_end++;pos_end++;return NEQ;}
";" {pos_end++;return SEMICOLON;}
"," {pos_end++;return COMMA;}
"(" {pos_end++;return LPARENTHESE;}
")" {pos_end++;return RPARENTHESE;}
"[" {pos_end++;return LBRACKET;}
"]" {pos_end++;return RBRACKET;}
"{" {pos_end++;return LBRACE;}
"}" {pos_end++;return RBRACE;}
"=" {pos_end++;return ASSIN;}
"\n" {lines++;pos_start=1;pos_end=1;return EOL;}
[ \f\r\t\v] {pos_start++,pos_end++;return BLANK;}
. {pos_end++;return ERROR;}


 /****请在此补全所有flex的模式与动作  end******/
%%
/****************C代码 start*************/
```

```
void analyzer(char* input_file, Token_Node* token_stream){
    lines = 1;
    pos_start = 1;
    pos_end = 1;
    if(!(yyin = fopen(input_file,"r"))){
        printf("[ERR] No input file\n");
        exit(1);
    }
    printf("[START]: Read from: %s\n", input_file);

    int token;
    int index = 0;

    while(token = yylex()){
        switch(token){
            case COMMENT:
                //STUDENT TO DO
                break;
            case BLANK:
                //STUDENT TO DO
                break;
            case EOL:
                //STUDENT TO DO
                break;
            case ERROR:
                printf("[ERR]: unable to analysize %s at %d line, from %d to %d\n", yytext, lines,
pos_start, pos_end);
```

```
135                 printf("[ERR]: unable to analysize %s at %d line, from %d to %d\n", yytext, lines,
    pos_start, pos_end);
136             default :
137                 if (token == ERROR){
138                     sprintf(token_stream[index].text, "[ERR]: unable to analysize %s at %d line, from
    %d", yytext, lines, pos_start, pos_end);
139                 } else {
140                     strcpy(token_stream[index].text, yytext);
141                 }
142                 token_stream[index].token = token;
143                 token_stream[index].lines = lines;
144                 token_stream[index].pos_start = pos_start;
145                 token_stream[index].pos_end = pos_end;
146                 index++;
147                 if (index >= MAX_NUM_TOKEN_NODE){
148                     printf("%s has too many tokens (> %d)", input_file, MAX_NUM_TOKEN_NODE);
149                     exit(1);
150                 }
151         }
152         pos_start=pos_end;
153     }
154     printf("[END]: Analysis completed.\n");
155     return;
156 }
157 /***************c代码 end*************/
```

无需 main 函数。

再次编译运行结果如下：

```
nidhs@nidhs-VirtualBox:~/byyl/cminus_compiler-2021-fall/build$ cd ..
nidhs@nidhs-VirtualBox:~/byyl/cminus_compiler-2021-fall$ python3 ./tests/lab1/te
st_lexer.py
Find 6 files
[START]: Read from: ./tests/lab1/testcase/2.cminus
[END]: Analysis completed.
[START]: Read from: ./tests/lab1/testcase/3.cminus
[END]: Analysis completed.
[START]: Read from: ./tests/lab1/testcase/1.cminus
[END]: Analysis completed.
[START]: Read from: ./tests/lab1/testcase/5.cminus
[END]: Analysis completed.
[START]: Read from: ./tests/lab1/testcase/4.cminus
[END]: Analysis completed.
[START]: Read from: ./tests/lab1/testcase/6.cminus
[END]: Analysis completed.
nidhs@nidhs-VirtualBox:~/byyl/cminus_compiler-2021-fall$ diff ./tests/lab1/token
 ./tests/lab1/TA_token
nidhs@nidhs-VirtualBox:~/byyl/cminus_compiler-2021-fall$
```

跟 token 全部匹配上了。

四、实验结果验证

详见第三点，再给出几个额外测试样例的文本和结果：

文本 7.cminus：

```
1 /****/
2 a
3 /****b***/
4 c/*/*/=/*//*/d/***///**/e//**f//g*/h;
5 int i[]={1,2,3,4,5};
6 while float ()
```

输出 7.tokens：

```
 1 a    285 2    1    2
 2 c    285 4    1    2
 3 =    269 4    7    8
 4 d    285 4    14   15
 5 /    262 4    20   21
 6 e    285 4    25   26
 7 /    262 4    26   27
 8 h    285 4    36   37
 9 ;    270 4    37   38
10 int  280 5    1    4
11 i    285 5    5    6
12 []   288 5    6    8
13 =    269 5    8    9
14 {    276 5    9    10
15 1    286 5    10   11
16 ,    271 5    11   12
17 2    286 5    12   13
18 ,    271 5    13   14
19 3    286 5    14   15
20 ,    271 5    15   16
21 4    286 5    16   17
22 ,    271 5    17   18
23 5    286 5    18   19
24 }    277 5    19   20
25 ;    270 5    20   21
26 while    284 6    1    6
27 float    281 6    7    12
28 (    272 6    13   14
29 )    273 6    14   15
```

文本 8.cminus：

```
1 for (int i==0,x+=y;z=w**4){}
2 /*asd
3 adf
4 ****//xm/***aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
5 aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
```

输出 8.tokens：

```
 1 for 285 1   1   4
 2 (   272 1   5   6
 3 int 280 1   6   9
 4 i   285 1   10  11
 5 ==  267 1   11  13
 6 0   286 1   13  14
 7 ,   271 1   14  15
 8 x   285 1   15  16
 9 +   259 1   16  17
10 =   269 1   17  18
11 y   285 1   18  19
12 ;   270 1   19  20
13 z   285 1   20  21
14 =   269 1   21  22
15 w   285 1   22  23
16 *   261 1   23  24
17 *   261 1   24  25
18 4   286 1   25  26
19 )   273 1   26  27
20 {   276 1   27  28
21 }   277 1   28  29
22 /   262 4   6   7
23 xm  285 4   7   9
24 /   262 4   9   10
25 *   261 4   10  11
26 *   261 4   11  12
27 *   261 4   12  13
28 aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa   285 4   13  58
29 aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa   285 5   1   58
```

五、实验反馈

总体体验良好，没有在哪一步卡特别久。也学会了许多新的知识。
编写代码的过程中，也可以注重于正则表达式的设计。特别是注释的正则表达式，经过我反复的修改优化，才终于正确了。
并且接触了解了 flex，git 等工具。
实验前面由于不清楚环境配置，做了许多不必要的工作，好在后来发现并将代码修改了回来。