

编译原理实验1

利用FLEX构造C-Minus-f词法分析器

计科210X 甘晴void 202108010XXX

实验要求

详细的实验项目文档为 https://gitee.com/coderwym/cminus_compiler-2023-fall/tree/master/Documentations/lab1

学习和掌握词法分析程序的逻辑原理与构造方法。通过 FLEX 进行实践，构造 C-Minus-f 词法分析器。具体完成过程如下：

1. 学习 C-Minus-f 的词法规则
2. 学习 FLEX 工具使用方法
3. 使用 FLEX 生成 C-Minus-f 的词法分析器，并进行验证

根据掌握的 C-Minus-f 的词法规则与 FLEX 工具使用方法, 补全 `lexical_analyer.l` 文件。要求实现功能：能够输出识别的 `token,type,line` (token所在行号), `pos_start` (token开始位置), `pos_end` (token结束位置, 不包含该位置, 即结束位置的后一个位置)

示例如下：

输入：（注意 `int` 前面有一个空格）

```
int a;
```

则识别结果应为：

<code>int</code>	280	1	2	5
<code>a</code>	285	1	6	7
<code>;</code>	270	1	7	8

实验难点

（1）实验环境配置

很折磨人，在附录I里给出

（2）理解C-Minus-f 的词法规则

C MINUS是C语言的一个子集，**cminus-f**在**C MINUS**上追加了浮点操作。简单来说就是一个微缩版的C语言，供编译原理学习研究。

相关规则如下：

1.关键字

```
else if int return void while float
```

2.专用符号

```
+ - * / < <= > >= == != = ; , ( ) [ ] { } /* */
```

3.标识符ID和整数NUM通过下列政策表达式定义

```
letter = a|...|z|A|...|Z  
digit = 0|...|9  
ID = letter+  
INTEGER = digit+  
FLOAT = (digit+. | digit*.digit+)
```

4.注释用/*...*/表示，可以超过一行。注释不能嵌套。

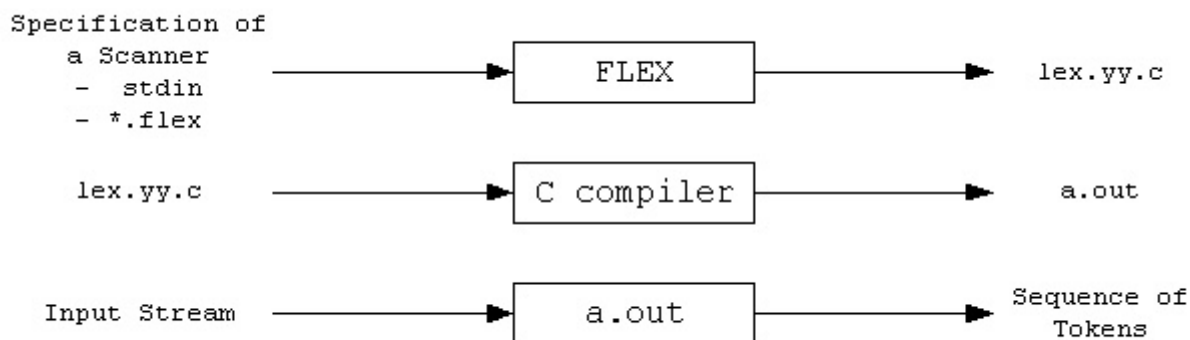
```
/*...*/
```

注意：**[,]**和**[]**是三个不同的最小单位，其中**[]**用于声明数组类型，且**[]**中间不能有空格，否则就该被识别为别的。

(3) 使用FLEX

①FLEX简单介绍

FLEX是一个生成词法分析器的工具。利用**FLEX**，我们只需提供词法的正则表达式，就可自动生成对应的C代码。整个流程如下图：



第一行是我们需要完成的，二三行在之前的计算机系统学科又涉及到，这里不再赘述。

使用不再赘述，在后面实操环节直接给出。

②Lex源程序

研究lexical_analyzer.l文件，可以总结出如下：

```
声明部分：
头文件引入，变量的定义和声明
//这一部分会直接复制到lex.yy.c的开头。
%%
转换规则：
形式为：模式{动作}，模式为正则表达式，动作则是代码片段
.{}可以处理其他出现的字符
//这一部分经过FLEX编译器转换为对应的C代码。
%%
辅助函数：
各个动作需要的辅助函数。
//这一部分由用户自定义，会直接复制到lex.yy.c末尾。
```

本实验主要是需要完成转换规则部分，给出cminux-f中词法单元的正则表达式和动作。此外，在辅助函数里还有三句需要补全。

③转换规则（**flex**的模式与动作）

对于运算、符号、关键字、ID和NUM这四类词法单元（**token**），在识别后要给出它的5个信息。

- **token**: 这个就是词法单元本身
- **type**: 由于在`cminus_token_type`表中定义了它们的编号，只要返回类型名就可以
- **line**（**token**所在行号）：行数的处理在辅助函数中进行（`lines++`即可）
- **pos_start**（**token**开始位置）：上一个**pos_end**的位置
- **pos_end**（**token**结束位置）：**pos_start**加上词素长度

总结模式如下：

```
RE {pos_start=pos_end;pos_end=pos_start+strlen(yytext);return token}
```

在“转换规则”中，只需要将所有待处理**token**按照这个模式进行书写就可以。

对于确定长度的**token**，可以从直接操作，不需再调用**len**。

④**FLEX**语法

I 了解一些**FLEX**常用的正则表达

- 匹配任意字符，除了 `\n`。
- 用来指定范围。例如：`A-Z` 指从A 到 Z 之间的所有字符。
- [] 一个字符集合。匹配括号内的 任意字符。如果第一个字符是 `^` 那么它表示否定模式。
例如：`[abc]` 匹配 `a`，`b`，和 `c`中的任何一个。
- * 匹配 0个或者多个上述的模式。
- + 匹配 1个或者多个上述模式。
- ? 匹配 0个或1个上述模式。
- \$ 作为模式的最后一个字符匹配一行的结尾。
- { } 指出一个模式可能出现的次数。 例如：`A{1,3}` 表示 A 可能出现1次或3次。
- \ 用来转义元字符。同样用来覆盖字符在此表中定义的特殊意义，只取字符的本意。
- ^ 否定。
- | 表达式间的逻辑或。
- "<一些符号>" 字符的字面含义。元字符具有。

II 了解一些FLEX常用的全局变量（无需在.l文件中定义，可直接使用）

<code>FILE *yyin/*yyout</code>	Lex中本身已定义的输入和输出文件指针。 这两变量指明了flex生成的词法分析器从哪里获得输入和输出到哪里。默认指向标准输入和标准输出。
<code>char *yytext</code>	指向当前是别的词法单元的指针。
<code>int yyleng</code>	当前词法单元的长度。
<code>yylineno</code>	提供当前的行数信息
<code>ECHO</code>	lex中预定义的宏，相当于fprintf(yyout, "%s", yytext) ，即输出当前匹配的词法单元。

实际上我们用lines模拟了这里的yyleng

III 了解一些FLEX常用的全局函数

<code>FILE *yyin/*yyout</code>	Lex中本身已定义的输入和输出文件指针。 这两变量指明了flex生成的词法分析器从哪里获得输入和输出到哪里。默认指向标准输入和标准输出。
<code>char *yytext</code>	指向当前是别的词法单元的指针。
<code>int yyleng</code>	当前词法单元的长度。
<code>yylineno</code>	提供当前的行数信息
<code>ECHO</code>	lex中预定义的宏，相当于fprintf(yyout, "%s", yytext) ，即输出当前匹配的词法单元。

以上这些FLEX中常用的全局变量和全局函数在代码中会涉及到，适当使用可以提高效率。可以通过查FLEX手册得到。

⑤注释

这是比较难理解的一个部分。

一开始的想法是这个

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

好像也可以。

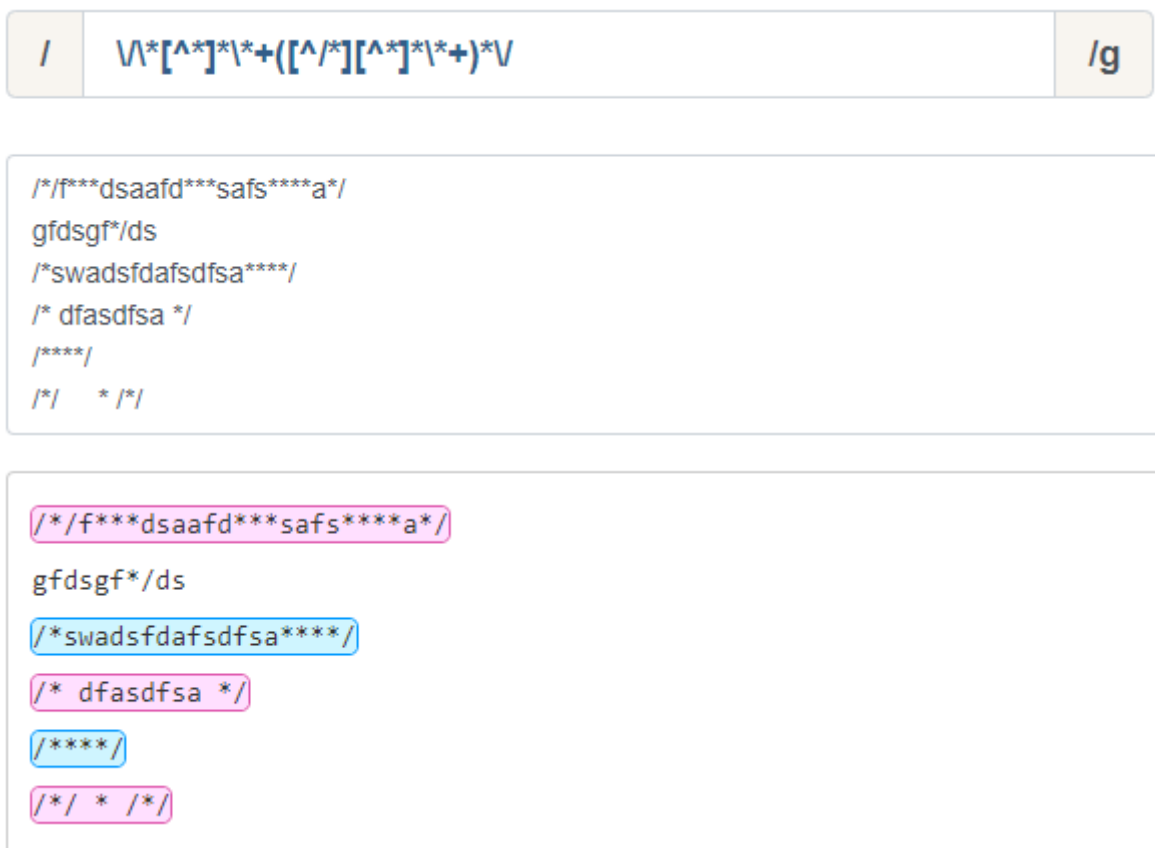
最后采用的是这个方法

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

在线验证正则表达式的正确性

<https://c.runoob.com/front-end/854/>

截图如下：



实验设计

（1）根据需要识别的token完成转换规则

需要识别的token定义在lexical_analyzer.h中，如下：

```
typedef enum cminus_token_type {  
    //运算  
    ADD = 259,  
    SUB = 260,  
    MUL = 261,  
    DIV = 262,  
    LT = 263,
```

```

    LTE = 264,
    GT = 265,
    GTE = 266,
    EQ = 267,
    NEQ = 268,
    ASSIN = 269,
    //符号
    SEMICOLON = 270,
    COMMA = 271,
    LPARENTHESIS = 272,
    RPARENTHESIS = 273,
    LBRACKET = 274,
    RBRACKET = 275,
    LBRACE = 276,
    RBRACE = 277,
    //关键字
    ELSE = 278,
    IF = 279,
    INT = 280,
    FLOAT = 281,
    RETURN = 282,
    VOID = 283,
    WHILE = 284,
    //ID和NUM
    IDENTIFIER = 285,
    INTEGER = 286,
    FLOATPOINT = 287,
    ARRAY = 288,
    LETTER = 289,
    //others
    EOL = 290,
    COMMENT = 291,
    BLANK = 292,
    ERROR = 258
} Token;

```

根据这里的每一个token，按照“难点”中的模式给出它们各自对应的转换规则如下

```

/* 运算 */
\+   {pos_start = pos_end; pos_end++; return ADD;}
\+   {pos_start = pos_end; pos_end++; return SUB;}

```

```

\*    {pos_start = pos_end; pos_end++; return MUL;}
\/    {pos_start = pos_end; pos_end++; return DIV;}
\<    {pos_start = pos_end; pos_end++; return LT;}
"<=" {pos_start = pos_end; pos_end+=2; return LTE;}
\>    {pos_start = pos_end; pos_end++; return GT;}
">=" {pos_start = pos_end; pos_end+=2; return GTE;}
"=="  {pos_start = pos_end; pos_end+=2; return EQ;}
"!="  {pos_start = pos_end; pos_end+=2; return NEQ;}
\=    {pos_start = pos_end; pos_end++; return ASSIN;}

/* 符号 */
\;    {pos_start = pos_end; pos_end++; return SEMICOLON;}
\,    {pos_start = pos_end; pos_end++; return COMMA;}
\[    {pos_start = pos_end; pos_end++; return LPARENTHESE;}
\)    {pos_start = pos_end; pos_end++; return RPARENTHESE;}
\[    {pos_start = pos_end; pos_end++; return LBRACKET;}
\]    {pos_start = pos_end; pos_end++; return RBRACKET;}
\{    {pos_start = pos_end; pos_end++; return LBRACE;}
\}    {pos_start = pos_end; pos_end++; return RBRACE;}

/* 关键字 */
else {pos_start = pos_end; pos_end+=4; return ELSE;}
if   {pos_start = pos_end; pos_end+=2; return IF;}
int  {pos_start = pos_end; pos_end+=3; return INT;}
float {pos_start = pos_end; pos_end+=5; return FLOAT;}
return {pos_start = pos_end; pos_end+=6; return RETURN;}
void  {pos_start = pos_end; pos_end+=4; return VOID;}
while {pos_start = pos_end; pos_end+=5; return WHILE;}

/* ID & NUM */
[a-zA-Z]+ {pos_start = pos_end; pos_end+=yyleng; return
IDENTIFIER;}
[0-9]+    {pos_start = pos_end; pos_end+=yyleng; return INTEGER;}
[0-9]+\.[0-9]*\.[0-9]+ {pos_start = pos_end; pos_end+=yyleng;
return FLOATPOINT;}
"[]" {pos_start = pos_end; pos_end+=2; return ARRAY;}
[a-zA-Z] {pos_start = pos_end; pos_end++; return LETTER;}

/* others */
\n {return EOL;}
\\\[^\*]\*\*+([^\*]\[\*]\*\*+)*\\[^\*]\*\*+ {return COMMENT;}
" " {pos_start = pos_end; pos_end+=yyleng; return BLANK;}

```



```
\t {pos_start = pos_end; pos_end+=yy leng; return BLANK;}  
. {return ERROR;}
```

（2）补全辅助函数

换行需要lines自增1，然后将pos_end换为1。

注释只需要考虑换行和根进目前处理的位置即可。

代码如下：

```
case COMMENT:  
    /*STUDENT TO DO*/  
    for (int i=0;i<yy leng;i++){  
        if (yytext[i]=='\n'){ /*换行操作*/  
            lines++;  
            pos_end=1;  
        }  
        else pos_end++;  
    }  
    break;  
case BLANK:  
    /*STUDENT TO DO*/  
    break;  
case EOL:  
    /*STUDENT TO DO*/  
    lines++;  
    pos_end=1;  
    break;
```

实验结果验证

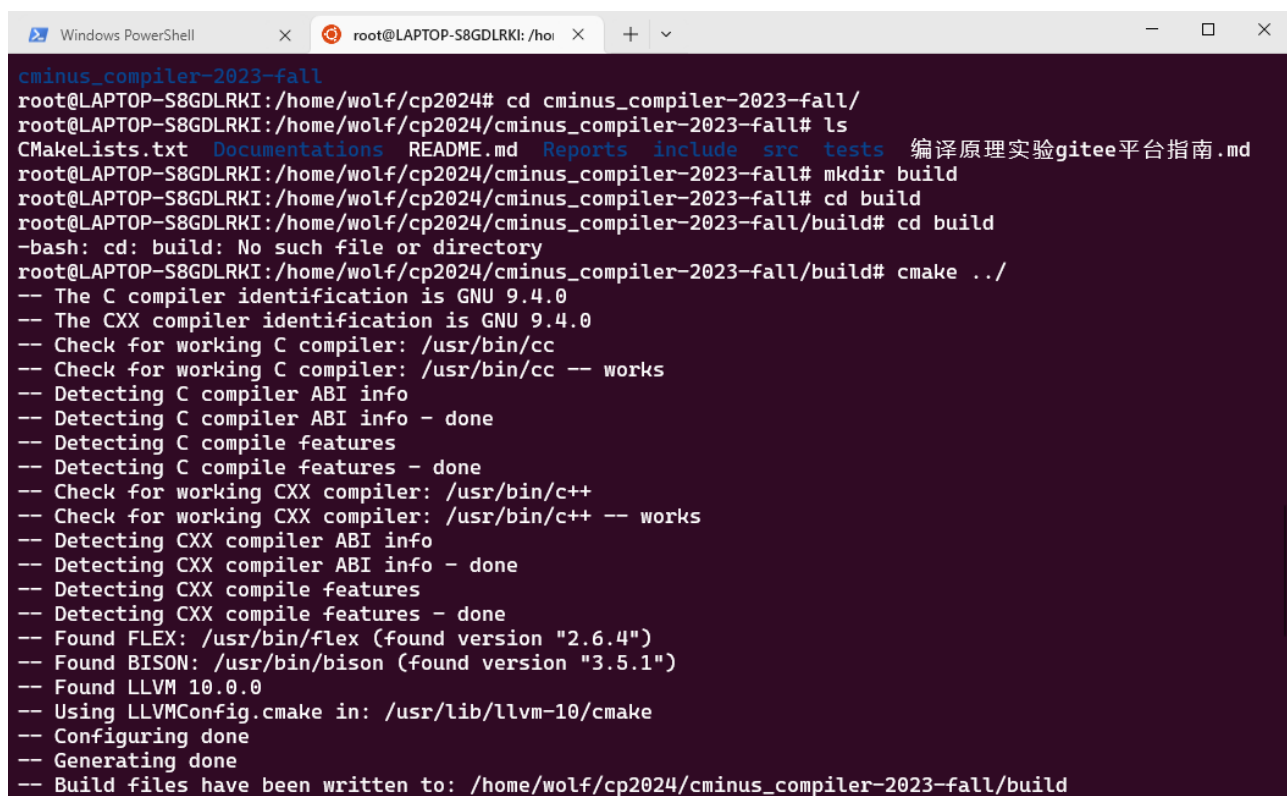
（1）编译

```
# 进入workspace
$ cd cminus_compiler-2023-fall

# 创建build文件夹，配置编译环境
$ mkdir build
$ cd build
$ cmake ../

# 开始编译
# 如果你只需要编译lab 1，请使用 make lexer
$ make
```

配置编译环境截图如下：



```
Windows PowerShell
root@LAPTOP-S8GDLRKI: /home/wolf/cp2024/

cminus_compiler-2023-fall
root@LAPTOP-S8GDLRKI:/home/wolf/cp2024# cd cminus_compiler-2023-fall/
root@LAPTOP-S8GDLRKI:/home/wolf/cp2024/cminus_compiler-2023-fall# ls
CMakeLists.txt  Documentations  README.md  Reports  include  src  tests  编译原理实验gitee平台指南.md
root@LAPTOP-S8GDLRKI:/home/wolf/cp2024/cminus_compiler-2023-fall# mkdir build
root@LAPTOP-S8GDLRKI:/home/wolf/cp2024/cminus_compiler-2023-fall# cd build
root@LAPTOP-S8GDLRKI:/home/wolf/cp2024/cminus_compiler-2023-fall/build# cd build
-bash: cd: build: No such file or directory
root@LAPTOP-S8GDLRKI:/home/wolf/cp2024/cminus_compiler-2023-fall/build# cmake ../
-- The C compiler identification is GNU 9.4.0
-- The CXX compiler identification is GNU 9.4.0
-- Check for working C compiler: /usr/bin/cc
-- Check for working C compiler: /usr/bin/cc -- works
-- Detecting C compiler ABI info
-- Detecting C compiler ABI info - done
-- Detecting C compile features
-- Detecting C compile features - done
-- Check for working CXX compiler: /usr/bin/c++
-- Check for working CXX compiler: /usr/bin/c++ -- works
-- Detecting CXX compiler ABI info
-- Detecting CXX compiler ABI info - done
-- Detecting CXX compile features
-- Detecting CXX compile features - done
-- Found FLEX: /usr/bin/flex (found version "2.6.4")
-- Found BISON: /usr/bin/bison (found version "3.5.1")
-- Found LLVM 10.0.0
-- Using LLVMConfig.cmake in: /usr/lib/llvm-10/cmake
-- Configuring done
-- Generating done
-- Build files have been written to: /home/wolf/cp2024/cminus_compiler-2023-fall/build
```

开始编译截图如下：

```
Windows PowerShell
root@LAPTOP-S8GDLRKI: /home/wolf/cp2024/cminus_compiler-2023-fall/build# make lexer
[ 20%] [FLEX][lex] Building scanner with flex 2.6.4
lexical_analyzer.l:61: warning, rule cannot be matched
Scanning dependencies of target flex
[ 40%] Building C object src/lexer/CMakeFiles/lexer.dir/lex.yy.c.o
lexical_analyzer.l: In function 'analyzer':
lexical_analyzer.l:96:11: warning: suggest parentheses around assignment used as truth value [-Wparentheses]
At top level:
/home/wolf/cp2024/cminus_compiler-2023-fall/build/src/lexer/lex.yy.c:1366:16: warning: 'input' defined but not used [-Wunused-function]
1366 |         static int input (void)
      |
/home/wolf/cp2024/cminus_compiler-2023-fall/build/src/lexer/lex.yy.c:1323:17: warning: 'yyunput' defined but not used [-Wunused-function]
1323 |         static void yyunput (int c, char * yy_bp )
      |
[ 60%] Linking C static library ../../libflex.a
[ 60%] Built target flex
Scanning dependencies of target lexer
[ 80%] Building C object tests/lab1/CMakeFiles/lexer.dir/main.c.o
[100%] Linking C executable ../../lexer
[100%] Built target lexer
```

(2) 运行

直接使用python文件对所有的.cminus文件进行分析

```
python3 ./tests/lab1/test_lexer.py
```

截图如下：

```
Windows PowerShell
root@LAPTOP-S8GDLRKI: /home/wolf/cp2024/cminus_compiler-2023-fall/build# cd ..
root@LAPTOP-S8GDLRKI: /home/wolf/cp2024/cminus_compiler-2023-fall# python3 ./tests/lab1/test_lexer.py
Find 6 files
[START]: Read from: ./tests/lab1/testcase/6.cminus
[END]: Analysis completed.
[START]: Read from: ./tests/lab1/testcase/4.cminus
[END]: Analysis completed.
[START]: Read from: ./tests/lab1/testcase/2.cminus
[END]: Analysis completed.
[START]: Read from: ./tests/lab1/testcase/5.cminus
[END]: Analysis completed.
[START]: Read from: ./tests/lab1/testcase/1.cminus
[END]: Analysis completed.
[START]: Read from: ./tests/lab1/testcase/3.cminus
[END]: Analysis completed.
```

由于中间没有出错，故中间无多余的输出，一个[START]对应一个[END]，表示中间分析过程没有问题。

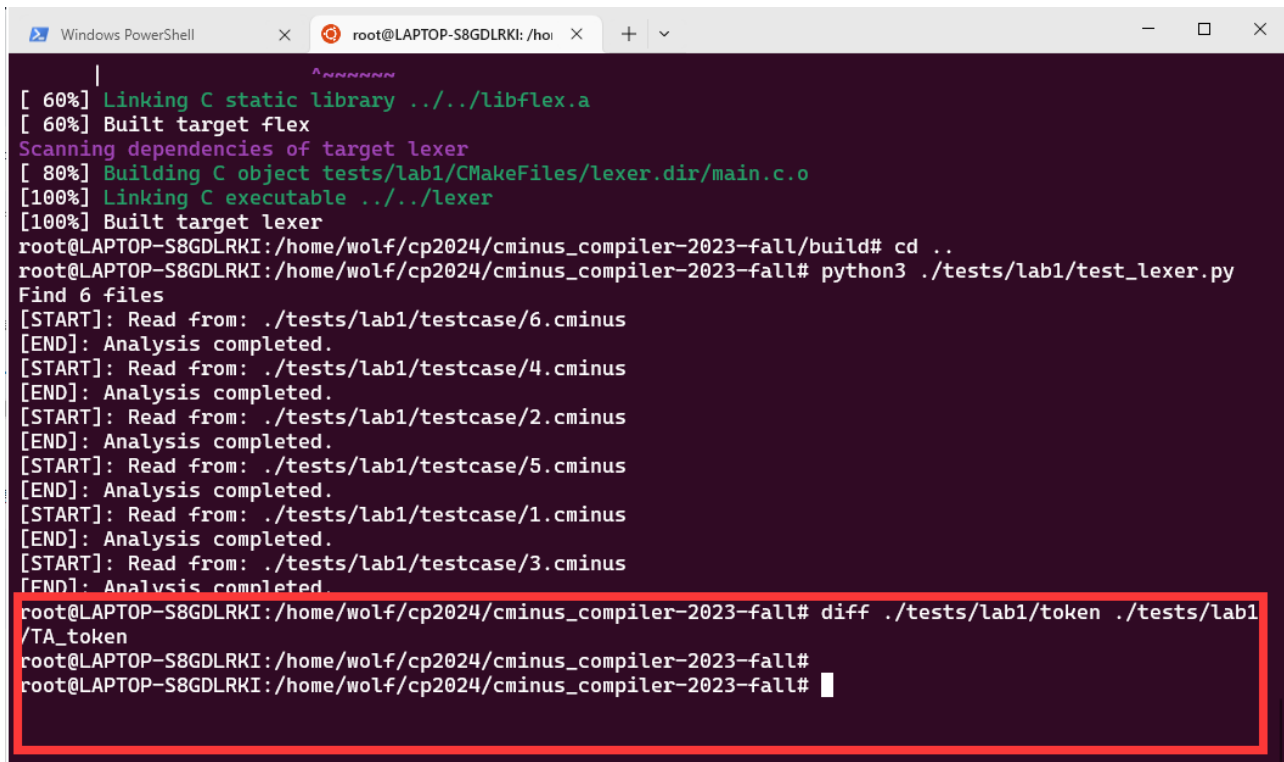
(3) 验证

使用diff工具可以比对我们的结果与标准结果。

```
diff ./tests/lab1/token ./tests/lab1/TA_token
```

如果没有输出，则表示两个对比之后完全一致，也就是结果正确。

截图如下：



```
Windows PowerShell
root@LAPTOP-S8GDLRKI: /home/wolf/cp2024/cminus_compiler-2023-fall/build# cd ..
[ 60%] Linking C static library ../../libflex.a
[ 60%] Built target flex
Scanning dependencies of target lexer
[ 80%] Building C object tests/lab1/CMakeFiles/lexer.dir/main.c.o
[100%] Linking C executable ../../lexer
[100%] Built target lexer
root@LAPTOP-S8GDLRKI:/home/wolf/cp2024/cminus_compiler-2023-fall/build# cd ..
root@LAPTOP-S8GDLRKI:/home/wolf/cp2024/cminus_compiler-2023-fall# python3 ./tests/lab1/test_lexer.py
Find 6 files
[START]: Read from: ./tests/lab1/testcase/6.cminus
[END]: Analysis completed.
[START]: Read from: ./tests/lab1/testcase/4.cminus
[END]: Analysis completed.
[START]: Read from: ./tests/lab1/testcase/2.cminus
[END]: Analysis completed.
[START]: Read from: ./tests/lab1/testcase/5.cminus
[END]: Analysis completed.
[START]: Read from: ./tests/lab1/testcase/1.cminus
[END]: Analysis completed.
[START]: Read from: ./tests/lab1/testcase/3.cminus
[END]: Analysis completed.
root@LAPTOP-S8GDLRKI:/home/wolf/cp2024/cminus_compiler-2023-fall# diff ./tests/lab1/token ./tests/lab1/TA_token
root@LAPTOP-S8GDLRKI:/home/wolf/cp2024/cminus_compiler-2023-fall#
```

关于diff还有更多的用法，如：

```
diff -y #可以并列显示，进行对照
diff -w #可以忽略空格进行比较
```

这些都很好用。

(4) 自定义样例测试

注意到助教给定的样例未包括对注释部分的更多测试，故这里我主要给出关于注释的测试。

使用如下方法新建文件并测试。

```
touch my.cminus
nano my.cminus
写入要测试的文件
按照如上方式进行测试
nano my.token
查看测试
```

待测试代码如下。

```
/* *** */

int main(){
    int a = 5;int b[];int c[9];
    float d = .33;
    /*** COMMENT1 /
    ***/
    while(a) {
        a = a-1;
        d = d+1.5;
    }
    a = a + func()
    /*** /*COMMENT2 //
    ***/
    d = d+7.;
    d = d+6.0;
    return 1;
}
```

测试结果如下，经肉眼核对正确。

```
int 280 3 1 4
main 285 3 5 9
( 272 3 9 10
) 273 3 10 11
{ 276 3 11 12
int 280 4 5 8
a 285 4 9 10
= 269 4 11 12
5 286 4 13 14
; 270 4 14 15
int 280 4 15 18
```

```

b    285 4    19 20
[]   288 4    20 22
;    270 4    22 23
int  280 4    23 26
c    285 4    27 28
[    274 4    28 29
9    286 4    29 30
]    275 4    30 31
;    270 4    31 32
float 281 5    5   10
d    285 5    11 12
=    269 5    13 14
.33  287 5    15 18
;    270 5    18 19
while 284 8    5   10
(    272 8    10 11
a    285 8    11 12
)    273 8    12 13
{    276 8    14 15
a    285 9     9  10
=    269 9    11 12
a    285 9    13 14
-    260 9    14 15
1    286 9    15 16
;    270 9    16 17
d    285 10   9  10
=    269 10   11 12
d    285 10   13 14
+    259 10   14 15
1.5  287 10   15 18
;    270 10   18 19
}    277 11   5   6
a    285 12   5   6
=    269 12   7   8
a    285 12   9   10
+    259 12   11 12
func  285 12  13 17
(    272 12  17 18
)    273 12  18 19
d    285 15   5   6
=    269 15   7   8
d    285 15   9   10

```

```
+ 259 15 10 11
7. 287 15 11 13
; 270 15 13 14
d 285 16 5 6
= 269 16 7 8
d 285 16 9 10
+ 259 16 10 11
6.0 287 16 11 14
; 270 16 14 15
return 282 17 5 11
1 286 17 12 13
; 270 17 13 14
} 277 18 1 2
```

实验反馈

（1）了解gitee并做完成规定的操作花了一些时间，但这个跟github总体还是很相似的，之前有一点涉猎，了解起来也会轻松一些。

（2）配置环境花费了很多时间，之前使用的Linux虚拟机VitualBox，这学期重装之后没法开共享文件夹了，也是有很多bug没有解决，索性这次直接使用新的了，参照WSL的教程（前面有说明）配置了基于Win10的Linux ubuntu 20.04，然后一路上解决了一些bug

（3）关于实验内容，其实还是比较好理解的，就是一个非常简化的C语言，进行词法分析，输出词法分析信息以及5个感兴趣的参数，这些其实都比较简单，唯一有点难度的就是注释的实现（这个在前面有说明），实现还是比较顺利的。

（4）最最最折磨人的就是遇到的这个问题（在附录II里给出），由于我使用windows进行git clone，再将这个文件整体迁移到Win10下的WLS内，实际上它已经经过windows操作系统的存储了，存储时对于换行的处理是\r\n，而Linux实际上是没有\r的，其对于换行的描述只有\n。这就导致经过windows存储过的文件会多一个\r。这下进行词法分析的时候就要对这个多出来的\r进行处理，否则就会在运行词法分析时报错。如果仅仅是进行处理还没有结束，在token结果输出的时候，此时是在Linux下输出的，每行的结果实际上是只有\n没有\r，而参考的助教答案因为经过了windows操作系统，它保存的换行可都是\r\n。这下使用diff的结果可壮观了，每一行都是有问题的，但是打开文件细看，每一行都一模一样。因为这个隐藏的\r，导致这个真的很难看出来的问题。

这个问题耗费了我一整个晚上，直到我使用`diff -w`忽略空格时发现不报错了，联想到这方面可能存在问题，然后经过杨jh同学提醒可能是Linux和Win10对于文本的存储和换行的处理存在不同的地方。

之后在袁jh同学的建议下，直接使用Linux连接gitee进行git clone，将文件绕开windows直接存储到Linux下，这次测试就一切正常了。

这真是一个折磨人的问题，又掉了好多头发。

附录I 实验环境配置

https://blog.csdn.net/weixin_42705114/article/details/131106845

环境配置参照的所有可选项在这个文档下：

https://gitee.com/coderwym/cminus_compiler-2023-fall/blob/master/Documentations/environments.md

这里使用的是Win10的WSL，WSL2的参考文档在这里：（科学访问）

<https://iceyblacktea.vercel.app/blog/install-wsl2>

安装后使用命令行时出现问题

```
The attempted operation is not supported for the type of object
referenced. Press any key to continue...
```

解决方法，下载NoLSP.exe并使用这个进行修复。

报错：

```
root@LAPTOP-S8GDLRKI:/mnt/e/CP-exam/cminus_compiler-2023-
fall/build# cmake ../
-- The C compiler identification is GNU 9.4.0
-- The CXX compiler identification is unknown
-- Check for working C compiler: /usr/bin/cc
-- Check for working C compiler: /usr/bin/cc -- works
```



```
-- Detecting C compiler ABI info
-- Detecting C compiler ABI info - done
-- Detecting C compile features
-- Detecting C compile features - done
CMake Error at CMakeLists.txt:1 (project):
  No CMAKE_CXX_COMPILER could be found.

  Tell CMake where to find the compiler by setting either the
  environment
  variable "CXX" or the CMake cache entry CMAKE_CXX_COMPILER to the
  full path
  to the compiler, or to the compiler name if it is in the PATH.

-- Configuring incomplete, errors occurred!
See also "/mnt/e/CP-exam/cminus_compiler-2023-
fall/build/CMakeFiles/CMakeOutput.log".
See also "/mnt/e/CP-exam/cminus_compiler-2023-
fall/build/CMakeFiles/CMakeError.log".
```

原因：没有配置环境

```
sudo apt-get install build-essential
```

报错如下：

```
root@LAPTOP-S8GDLRKI:/mnt/e/CP-exam/cminus_compiler-2023-
fall/build# sudo apt-get install build essential
Reading package lists... Done
Building dependency tree
Reading state information... Done
E: Unable to locate package build
E: Unable to locate package essential
```

分别检查gcc和g++

```
gcc --version
g++ --version
```

发现没装g++哈哈哈哈

使用如下一键搞定

```
sudo apt-get install build-essential
```

现在成功了

附录II 由win与Linux文本存储区别引发的问题

★遇到问题如下：

现有2份代码，标记为X,Y。X是我的，Y是舍友的，两份差异较大。

2个环境：环境A是我的，Windows10下使用WSL配置Linux ubuntu20.04环境

环境B是舍友的，Linux系统（均为最新版本，非虚拟机）

出现情况如下：

代码X,Y在环境B下均正常运行并给出结果，经diff与标准代码比对完全一致。

代码X,Y在环境A下均无法正常运行，报错如下：（以6为范例）

```
[START]: Read from: ./tests/lab1/testcase/6.cminus
at 3 line, from 17 to 18e
at 4 line, from 1 to 2ize
at 5 line, from 17 to 18e
at 6 line, from 1 to 2ize
at 7 line, from 32 to 33e
at 8 line, from 1 to 2ize
at 11 line, from 28 to 29
at 12 line, from 1 to 2ze
at 13 line, from 21 to 22
at 14 line, from 1 to 2ze
at 15 line, from 23 to 24
at 16 line, from 1 to 2ze
[END]: Analysis completed.
```

其tokens如下（节选6部分作为范例）

```

[ERR]: unable to analyze
  at 3 line, from 17 to 18  258 3   17  18
[ERR]: unable to analyze
  at 4 line, from 1 to 2 258 4   1   2
void    283 5   1   5
main    285 5   6   10
(    272 5   10  11
void    283 5   11  15
)    273 5   15  16
{    276 5   16  17
[ERR]: unable to analyze
  at 5 line, from 17 to 18  258 5   17  18
[ERR]: unable to analyze
  at 6 line, from 1 to 2 258 6   1   2
int 280 7   5   8
x    285 7   9   10
;    270 7   10  11
int 280 7   12  15
y    285 7   16  17
;    270 7   17  18
int 280 7   19  22
Resultado 285 7   23  31
;    270 7   31  32
[ERR]: unable to analyze
  at 7 line, from 32 to 33  258 7   32  33
[ERR]: unable to analyze
  at 8 line, from 1 to 2 258 8   1   2
x    285 9   5   6
=    269 9   7   8

```

它无法读取每一行的最后一个（这个是不存在的）

推测是Linux与windows文件系统对于换行使用的\r\n不一致

改变代码，考虑\r的情况，并将\r作为空格读取处理，返回BLANK。

代码X,Y均可成功处理，不再报上述错误。

但是，但是经diff比对，显示每一行都不一致，

若使用

```
diff -w
```

取消空格比对，则完全一致。

解决方法：

直接使用Linux连接gitee进行git clone，将文件绕开windows直接存储到Linux下，这次测试就一切正常。

附录III 参考文献

https://blog.csdn.net/Coral_/article/details/128458671

<https://blog.csdn.net/Aaron503/article/details/128324923>