编译器构造实验

实验 1：词法分析器

单招文

7/3/2024

实验任务

1. 是否提取出正确的 token（60 分）

2. 是否提取出正确的 token location（30 分）

3. 是否识别其他无关字符（10 分）

补充说明：eof 这个token 对应的行号和列号不计入评分

输入词法分析器的文件

build > test > task0 > functional-0 > C 000_main.sysu.c > ...

```
1  # 1 "./functional-0/000_main.sysu.c"
2  # 1 "<built-in>" 1
3  # 1 "<built-in>" 3
4  # 384 "<built-in>" 3
5  # 1 "<command line>" 1
6  # 1 "<built-in>" 2
7  # 1 "./functional-0/000_main.sysu.c" 2
8  int main(){
9      return 3;
10 }
```

同学们拿到的残缺词法分析器的输出

build > test > task1 > functional-0 > 000_main.sysu.c > output.txt

```
1  int 'int' Loc=<0:0>
2  identifier 'main' Loc=<0:0>
3  l_paren '(' Loc=<0:0>
4  r_paren ')' Loc=<0:0>
5  l_brace '{' Loc=<0:0>
6  return 'return' Loc=<0:0>
7  numeric_constant '3' Loc=<0:0>
8  semi ';' Loc=<0:0>
9  r_brace '}' Loc=<0:0>
10 eof '' Loc=<0:0>
```

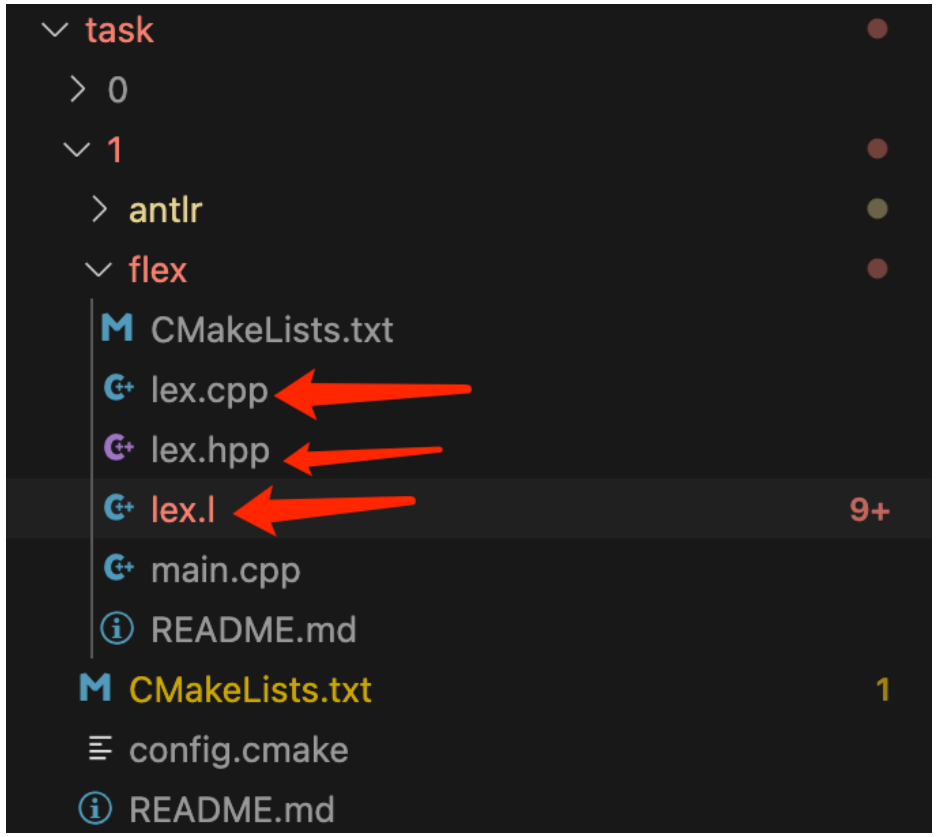
clang词法分析器输出的标准答案

build > test > task1 > functional-0 > 000_main.sysu.c > answer.txt

```
1  int 'int'    [StartOfLine]  Loc=<./functional-0/000_main.sysu.c:1:1>
2  identifier 'main'    [LeadingSpace] Loc=<./functional-0/000_main.sysu.c:1:5>
3  l_paren '('     Loc=<./functional-0/000_main.sysu.c:1:9>
4  r_paren ')'     Loc=<./functional-0/000_main.sysu.c:1:10>
5  l_brace '{'     Loc=<./functional-0/000_main.sysu.c:1:11>
6  return 'return'  [StartOfLine] [LeadingSpace]   Loc=<./functional-0/000_main.sysu.c:2:5>
7  numeric_constant '3'     [LeadingSpace] Loc=<./functional-0/000_main.sysu.c:2:12>
8  semi ';'        Loc=<./functional-0/000_main.sysu.c:2:13>
9  r_brace '}'  [StartOfLine]  Loc=<./functional-0/000_main.sysu.c:3:1>
10 eof ''      Loc=<./functional-0/000_main.sysu.c:3:2>
```

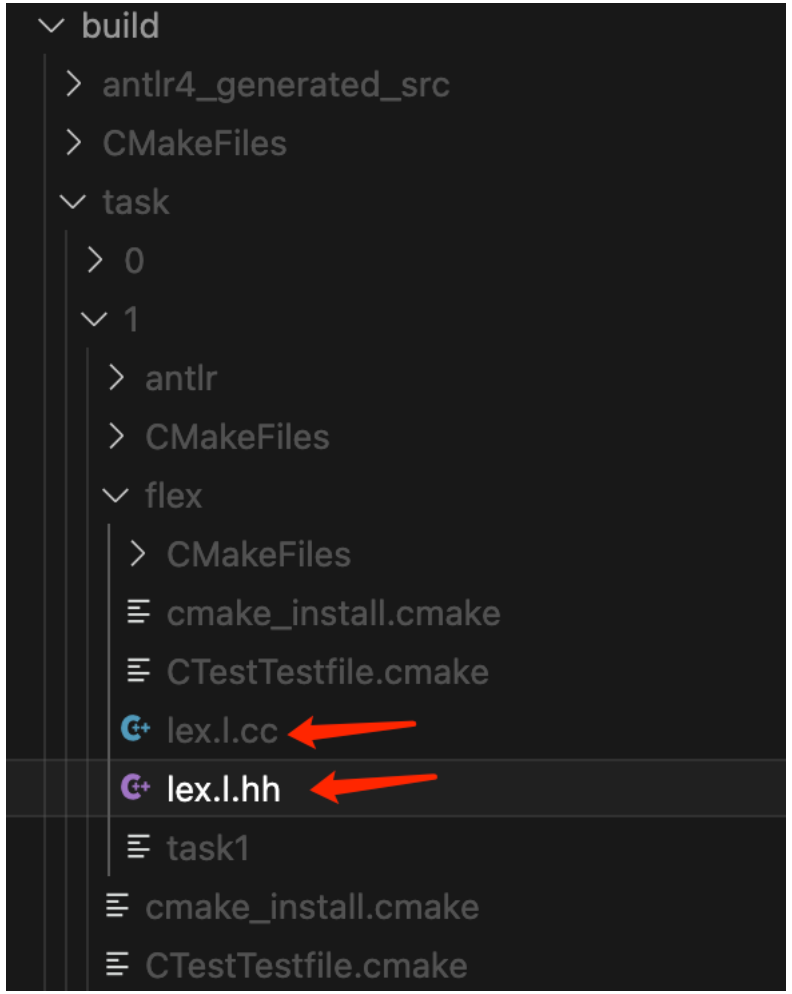
使用 flex 完成词法分析器实验

Flex: fast lexical analyzer generator（快速词法分析器生成器）



300行左右代码

生成



2500行左右代码

使用 flex 完成词法分析器实验

识别 token

1.简单token的规则

```
"int"       { ADDCOL(); COME(INT); }
"return"    { ADDCOL(); COME(RETURN); }
```

2.复杂 token 的规则

①定义正则表达式

```
D       [0-9]
L       [a-zA-Z_]
IS      ((u|U)|(u|U)?(l|L|ll|LL)|(l|L|ll|LL)(u|U))
```

② 利用正则表达式定义复杂 token 的规则

```
0[0-7]*{IS}?            { ADDCOL(); COME(CONSTANT); }
[1-9]{D}*{IS}?          { ADDCOL(); COME(CONSTANT); }
```

识别 token的 Loc

几个你可能会用到的变量

```
#define ADDCOL() g.mColumn += yyleng;
```

函数和全局变量

- yylex(): 词法分析器的主要入口点，每次调用返回下一个词法单元。
- yy_scan_string(const char *str): 使词法分析器从一个字符串而不是标准输入或文件中读取输入。
- yy_switch_to_buffer(YY_BUFFER_STATE new_buffer): 切换当前的输入缓冲区。
- yy_create_buffer(FILE *file, int size): 为给定的文件创建一个新的输入缓冲区。
- yy_delete_buffer(YY_BUFFER_STATE b): 删除一个输入缓冲区。
- yyrestart(FILE *file): 重置词法分析器的状态并从新的文件开始读取输入。
- YY_BUFFER_STATE: 表示输入缓冲区的状态的类型。
- yylval: 在与 yacc/bison 配合使用时，用于传递词法单元的值。
- yytext: 包含当前匹配的文本。
- yyleng: 包含yytext的长度。
- yylineno: 跟踪当前的行号（如果%option yylineno被使用）。

使用 antlr 完成词法分析器实验

ANTLR : Another Tool for Language Recognition
是一个强大的工具，用于生成词法分析器、解析器以及遍历代码生成树的代码。

task > 1 > antlr > SYsU_lang.g4 > ...

```
1   lexer grammar SYsU_lang;
2
3   Int : 'int';
4   Return : 'return';
5
6   LeftParen : '(';
7   RightParen : ')';
8   LeftBracket : '[';
9   RightBracket : ']';
10  LeftBrace : '{';
11  RightBrace : '}';
```

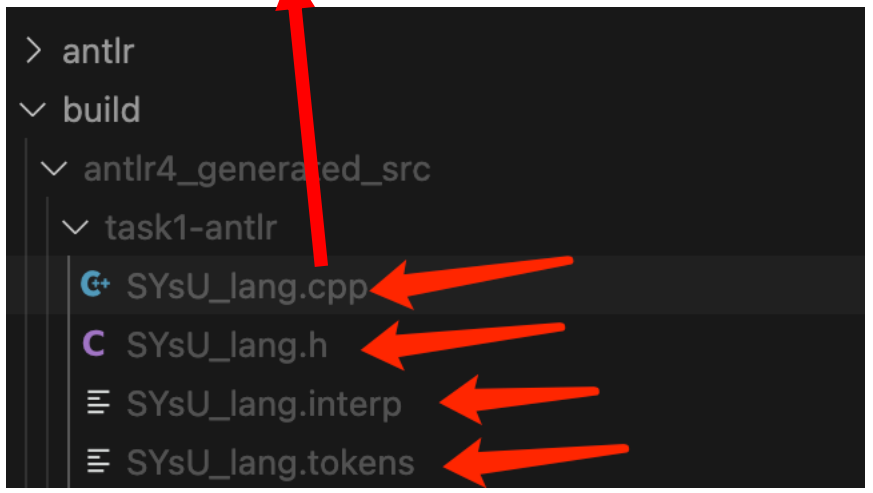
300行左右代码（.g4文件）

生成

main.cpp M

task > 1 > antlr > main.cpp > main(int, char * [])

```
81  std::cout << "程序 '" << argv[0] << std::endl;
82  std::cout << "输入 '" << argv[1] << std::endl;
83  std::cout << "输出 '" << argv[2] << std::endl;
84
85  antlr4::ANTLRInputStream input(inFile);
86  SYsU_lang lexer(&input);
```



500行左右代码

使用 antlr 完成词法分析器实验

1.简单token的规则

词法规则的定义要大写字母开头

task > 1 > antlr > SYsU_lang.g4 >

```
1  lexer grammar SYsU_lang;
2
3  Int : 'int';
4  Return : 'return';
```

2.复杂 token 的规则

识别 token

①定义正则表达式

```
fragment
Nondigit
    :   [a-zA-Z_]
    ;
```

② 利用正则表达式定义复杂 token 的规则

```
Identifier
    :   IdentifierNondigit
        (   IdentifierNondigit
        |   Digit
        )*
    ;
```

识别 token的 Loc

几个你可能会用到的变量

Token（词法单元接口）

Token接口表示由词法分析器生成的一个词法单元，包含了关于该词法单元的所有

主要属性：

- getType()：获取词法单元的类型，类型通常由词法分析器的规则定义。
- getText()：获取词法单元的文本内容。
- getLine()：获取词法单元出现的行号。
- getCharPositionInLine()：获取词法单元在其所在行的位置（字符偏移量）。