

实验报告

一、实验目标

借助 flex 工具实现一个词法分析器, 将 SysY 代码中的单词和符号进行分类, 然后按照单词符号出现顺序依次输出: 原始单词符号、种类、出现在源程序的位置 (行数和列数)。其中单词的符号共分为 K、I、C、O、D、T 共 6 类。还需要对注释进行处理, 包括单行注释和多行注释。

二、代码说明及思路

首先, 在声明部分, 除了需要的头文件之外, 还声明了两个全局变量, 其中 num_lines 用于统计当前行数, num_cols 用于统计当前列数。

```
%{  
#include <stdio.h>  
#include <stdlib.h>  
#include <string.h>  
int num_lines = 1;  
int num_cols = 1;  
%}
```

1. K 类, 关键字

关键字包括 int、main、if、else 等等, 直接在识别规则处以 “int” 的形式要求完全匹配即可。动作中的输出按照要求的格式, 种类为 K (已匹配的部分存放在 yytext 中); 此外动作中还要记得将 num_cols 加上已匹配内容的长度 (yyleng), 以统计当前所在列。(这一操作在以后的每次识别中都要进行, 此后不再赘述)

```

"int" {printf("%s: K, (%d, %d)\n", yytext, num_lines, num_cols); num_cols += yyleng;}
"main" {printf("%s: K, (%d, %d)\n", yytext, num_lines, num_cols); num_cols += yyleng;}
"continue" {printf("%s: K, (%d, %d)\n", yytext, num_lines, num_cols); num_cols += yyleng;}
"const" {printf("%s: K, (%d, %d)\n", yytext, num_lines, num_cols); num_cols += yyleng;}
"if" {printf("%s: K, (%d, %d)\n", yytext, num_lines, num_cols); num_cols += yyleng;}
"else" {printf("%s: K, (%d, %d)\n", yytext, num_lines, num_cols); num_cols += yyleng;}
"return" {printf("%s: K, (%d, %d)\n", yytext, num_lines, num_cols); num_cols += yyleng;}
"void" {printf("%s: K, (%d, %d)\n", yytext, num_lines, num_cols); num_cols += yyleng;}
"while" {printf("%s: K, (%d, %d)\n", yytext, num_lines, num_cols); num_cols += yyleng;}
"enum" {printf("%s: K, (%d, %d)\n", yytext, num_lines, num_cols); num_cols += yyleng;}
"switch" {printf("%s: K, (%d, %d)\n", yytext, num_lines, num_cols); num_cols += yyleng;}
"case" {printf("%s: K, (%d, %d)\n", yytext, num_lines, num_cols); num_cols += yyleng;}
"for" {printf("%s: K, (%d, %d)\n", yytext, num_lines, num_cols); num_cols += yyleng;}
"sizeof" {printf("%s: K, (%d, %d)\n", yytext, num_lines, num_cols); num_cols += yyleng;}
"static" {printf("%s: K, (%d, %d)\n", yytext, num_lines, num_cols); num_cols += yyleng;}
"typedef" {printf("%s: K, (%d, %d)\n", yytext, num_lines, num_cols); num_cols += yyleng;}
"break" {printf("%s: K, (%d, %d)\n", yytext, num_lines, num_cols); num_cols += yyleng;}
"do" {printf("%s: K, (%d, %d)\n", yytext, num_lines, num_cols); num_cols += yyleng;}
"struct" {printf("%s: K, (%d, %d)\n", yytext, num_lines, num_cols); num_cols += yyleng;}
"signed" {printf("%s: K, (%d, %d)\n", yytext, num_lines, num_cols); num_cols += yyleng;}
"unsigned" {printf("%s: K, (%d, %d)\n", yytext, num_lines, num_cols); num_cols += yyleng;}
"default" {printf("%s: K, (%d, %d)\n", yytext, num_lines, num_cols); num_cols += yyleng;}

```

2. I 类，标识符

标识符以字母或下划线开头，后接任意个字母数字下划线。采用辅助定义

IDENTIFIER，识别时输出为 I 类。

```

IDENTIFIER [_A-Za-z][_A-Za-z0-9]*
{IDENTIFIER} {printf("%s: I, (%d, %d)\n", yytext, num_lines, num_cols); num_cols += yyleng;}

```

3. C 类，常量

常量有三种表示方法：十进制，八进制，十六进制。十进制以非 0 的数开头，后接任意数字；八进制以 0 开头，后接 0-7 的任意数字；十六进制以 0x 或 0X 开头，后接任意个 0-F。分别进行辅助定义，并汇总为 INTCONST。

```

DECIMAL 0|[1-9][0-9]*
OCTALCONS 0[0-7]+
HEXCONS 0[xX][0-9a-fA-F]+
INTCONST {DECIMAL}|{OCTALCONS}|{HEXCONS}
{INTCONST} {printf("%s: C, (%d, %d)\n", yytext, num_lines, num_cols); num_cols += yyleng;}

```

4. O，算符

算符用辅助定义 OPERATOR 表示，将可能的算符全部列出即可。

```

OPERATOR ([\+\-\*\^%<\>\!\\=\&\|]|"+="|"-="|"*=|"|"/="|">="|"<="|"=="|"+="|"-="|"&&"|"||")

```

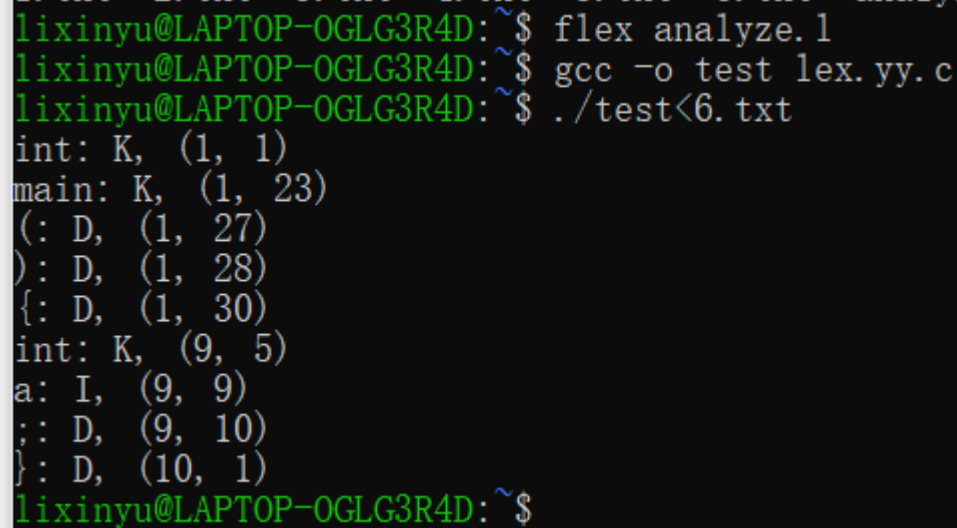

最后的用户子程序部分：

```
%%  
int yywrap(){  
    return 1;  
}  
int main(){  
    yylex();  
    return 0;  
}
```

三、实验过程

.l 源程序写好后，用 flex 运行生成 lex.yy.c 文件，再用 gcc 编译运行即可。图

中所示为输入样例 6 的输出结果，经检验符合要求。



```
lixinyu@LAPTOP-OGLG3R4D: ~$ flex analyze.l  
lixinyu@LAPTOP-OGLG3R4D: ~$ gcc -o test lex.yy.c  
lixinyu@LAPTOP-OGLG3R4D: ~$ ./test<6.txt  
int: K, (1, 1)  
main: K, (1, 23)  
(: D, (1, 27)  
): D, (1, 28)  
{: D, (1, 30)  
int: K, (9, 5)  
a: I, (9, 9)  
;: D, (9, 10)  
}: D, (10, 1)  
lixinyu@LAPTOP-OGLG3R4D: ~$
```

用同样方法测试了其他样例，输出均符合要求。受篇幅限制，在此只展示样例 1 的结果。

```

int: K, (1, 1)
main: K, (1, 5)
(: D, (1, 9)
): D, (1, 10)
{: D, (1, 11)
int: K, (2, 5)
b: I, (2, 9)
[: D, (2, 10)
10: C, (2, 11)
]: D, (2, 13)
[: D, (2, 14)
10: C, (2, 15)
]: D, (2, 17)
[: D, (2, 18)
10: C, (2, 19)
]: D, (2, 21)
: D, (2, 22)
a: I, (2, 23)
: D, (2, 24)
b: I, (3, 5)
[: D, (3, 6)
2: C, (3, 7)
]: D, (3, 8)
[: D, (3, 9)
2: C, (3, 10)
]: D, (3, 11)
[: D, (3, 12)
2: C, (3, 13)
]: D, (3, 14)
=: O, (3, 15)
010: C, (3, 16)
:: D, (3, 19)
int: K, (4, 5)
test: I, (4, 9)
=: O, (4, 13)
21474836472147483647: C, (4, 14)
:: D, (4, 34)
int: K, (5, 5)
test: I, (5, 9)
=: O, (5, 13)
0xabcd: C, (5, 14)
:: D, (5, 21)
int: K, (6, 5)
666bbb666: T, (6, 9)
:: D, (6, 18)
int: K, (7, 5)
aaalllaaa: I, (7, 9)
: D, (7, 18)
a: I, (8, 5)
=: O, (8, 6)
b: I, (8, 7)
[: D, (8, 8)
2: C, (8, 9)
]: D, (8, 10)
[: D, (8, 11)
2: C, (8, 12)
]: D, (8, 13)
[: D, (8, 14)
2: C, (8, 15)
]: D, (8, 16)
: D, (8, 17)
while: K, (9, 5)
(: D, (9, 10)
a: I, (9, 11)
>: O, (9, 12)
0: C, (9, 13)
): D, (9, 14)
{: D, (9, 15)
a: I, (10, 9)
=: O, (10, 10)
a: I, (10, 11)
=: O, (10, 12)
1: C, (10, 13)
: D, (10, 14)
if: K, (11, 9)
(: D, (11, 11)
a: I, (11, 12)
==: O, (11, 13)
5: C, (11, 15)
): D, (11, 16)
break: K, (12, 13)
: D, (12, 18)
}: D, (13, 5)
return: K, (14, 5)
a: I, (14, 12)
: D, (14, 13)
}: D, (15, 1)

```