实验一 词法分析器

一、实验目的

1. 理解编译器的工作机制，掌握编译器的构造方法
2. 掌握词法分析器的生成工具 LEX 的用法

二、实验内容

1.PL/0 语言简介

a. PL/0 语言是 Pascal 语言的子集
   - 数据类型只有整型
   - 标识符的有效长度是 10，以字母开头的字母数字串
   - 数最多 14 位
   - 过程无参，可嵌套（最多三层），可递归调用
   - 变量的作用域同 Pascal，常量为全局的

b. 语句类型：
   - 赋值语句：if…then…, while…do…, read, write, call
   - 复合语句：begin…end
   - 说明语句：const…, var…, procedure…

c. 13 个保留字：
   - If, then, while, do, read, write, call, begin, end, const, var, procedure, odd

2.实验内容

1）用 flex 工具生成一个 PL/0 语言的词法分析程序，对 PL/0 语言的源程序进行扫描，识别出单词符号的类别，输出各种符号的信息。

2）输入：PL/0 源程序

3）输出：把单词符号分为下面六类，然后按单词符号出现顺序依次输出各单词符号的种类和出现在源程序中的位置（行数和列数）
   - K 类（关键字）
   - I 类（标识符）
   - C 类（常量）
   - O 类（算符）
   - D 类（界符）
   - T 类（其他）

4）LEX 概述及源程序的格式：见实验文档

5）实验环境
   - Windows & C 或 C++
   - 词法分析器生成工具：flex

三、实验步骤

1. 实验步骤：

a. 将 flex.exe 文件、写好的 lex 程序和待输入的 PL/0 程序源代码放到同一个文件夹

b. 在 Windows 系统下打开 cmd，进行入当前文件夹

c. 在 cmd 下输入命令 flex lex.l 将 lex 程序用 flex 工具生成 lex.yy.c 文件，在 C 编译环境下对该文件进行编译，生成 lex.yy.exe 文件

d. 输入命令 lex.yy.exe < max.pl0 对 max.pl0 文件进行词法分析

2. 实验源代码如下：

```
%{
/*
Date: 2019.3.21
Author: Aman
ALL RIGHTS DESERVED
 */
int row_num = 1, col_num = 1;  //行数和列数
FILE *fp;  //指向输出文件的指针 Constant [-]?([0-9])|([1-9][0-9]+[\.]?[0-9]*)
%}

KeyWord [iI][fF]|[tT][hH][eE][nN]|[wW][hH][iI][lL][eE]|[dD][oO]|[rR][eE][aA][dD]|[wW][rR][iI][tT][eE]|[cC][aA][lL][lL]|[bB][eE][gG][iI][nN]|[eE][nN][dD]|[cC][oO][nN][sS][tT]|[vV][aA][rR]|[pP][rR][oO][cC][eE][dD][uU][rR][eE]|[oO][dD][dD]|[cC][aA][sS][eE]|[eE][nN][dD][cC][aA][sS][eE]
Identifier [A-Za-z][A-Za-z0-9]*
Zero [0]
PossiInt ([+]?[1-9][0-9]*)
NegatiInt ([-]?[0-9]+)
Float {Zero}|{PossiInt}|{NegatiInt}(.[0-9]+)
Constant {PossiInt}|{NegatiInt}|{Zero}
Operator [\.\[\]\^\-\*\+\?\{\}\"\\\(\)\|\/\$\<\>]|:=
Delimiter  [\,\;\.]
Space (\ )
Other [^{KeyWord}{Identifier}{Constant}{Operator}{Delimiter}]

%%

 //这部分注释要有前导空格，否则 lex 编译出错

{KeyWord} {fprintf(fp, "%s : K, (%d, %d)\n", yytext, row_num, col_num); col_num += yyleng;}
{Identifier} {
 if(yyleng > 10){
      fprintf(fp, "Error!Expected a shorter IDENTIFIER!(%d, %d)\n", row_num, col_num);
```

```
31.      col_num += yyleng;
32.  }
33.  else{
34.      fprintf(fp, "%s : I, (%d, %d)\n", yytext, row_num, col_num); col_num +=
    yyleng;
35.  }
36. }
37. {Constant} {
38.  if(yyleng > 14){
39.      fprintf(fp, "Error!Expected a shorter CONSTANT!(%d, %d)\n", row_num,
    col_num);
40.      col_num += yyleng;
41.  }
42.  else{
43.      fprintf(fp, "%s : C, (%d, %d)\n", yytext, row_num, col_num);
44.      col_num += yyleng;
45.  }
46. }
47. {Operator} {fprintf(fp, "%s : O, (%d, %d)\n", yytext, row_num, col_num);
    col_num += yyleng;}
48. {Delimiter} {fprintf(fp, "%s : D, (%d, %d)\n", yytext, row_num, col_num);
    ++col_num;}
49. \n {++row_num; col_num = 1;}  //换行符计数
50. {Space} {;}
51. {Other} {fprintf(fp, "%s : T, (%d, %d)\n", yytext, row_num, col_num);
    col_num += yyleng;}
52.
53. %%
54. main(){
55. fp = fopen("E:/…/…/Compiling Principle/Lab1/Output.txt", "w+");
56. yylex();
57.
58. }
59. int yywrap(){
60.  return 1;
61. }
```

3.对代码的说明

代码分为四个部分：声明、辅助定义、识别规则和用户子程序

- 声明部分定义了要用到的全局变量；
- 辅助定义定义了 Keyword, Identifier 等正规式，以便后续的代码更加易读。(尽管 PL/0 语言中数据类型只有整型，也同时定义了识别正负浮点数的正规式并通过了测试，可供功能拓展)；
- 识别规则则是给出了对相关正规式识别后的一系列操作，包括输

出到 txt 文件和错误处理等；

- 用户子程序主要是调用 yylex()函数，并增加 yywrap()函数（由于不增加时在编译 lex.yy.c 文件过程中会报错，故增加一直接返回 1 的 yylex()函数）；
- 对于未匹配的字符，程序会识别为 T 类，同时对于长度超过限定的标识符和常量会进行报错。

四、实验结果

执行 flex test.l 命令后未报错如下：

```
Compiling Principle\Lab1>flex test.l

Compiling Principle\Lab1>
```

对 lex.yy.c 进行编译，并执行 lex.yy.exe < case_test.pl0:

```
Compiling Principle\Lab1>flex test.l

Compiling Principle\Lab1>lex.yy.exe < case_test.pl0

Compiling Principle\Lab1>
```

case_test.pl0 作为样例 pl0 代码所示如下，在其末尾人为添加了不符合规定的标识符和各种常量和符号用于测试不同情况下的输出情况。

```
1
2  var n, m, i, j, digit(1:3);
3
4  procedure init;
5  begin
6      i := 1;
7      while i < 4 do
8          begin
9              digit(i) := 0;
10             i := i + 1;
11         end;
12 end;
13
14 procedure parse;
15 var tmp;
16 begin
17   tmp := n;
18     i := 0;
19     while n != 0 do
20         begin
21             m := n / 10;
22           i := i + 1;
23           digit(i) := n - m * 10;
24             n := m;
25         end;
26   n := tmp;
27 end;
28
29 procedure sumdigit;
30 begin
31     m := digit(1) * digit(1) * digit(1) +
32         digit(2) * digit(2) * digit(2) +
33         digit(3) * digit(3) * digit(3);
34 end;
```

```
53 var input, a;
54
55 procedure P;
56 begin
57     read(input);
58     case input:
59         1: a := 0
60         2: a := 1
61         3: a := 2;
62     endcase;
63 end;
64
65 call P.
66
67
68 +1321
69 a:=+2-5
70 a:=2-6
71 a:= 2 - 6
72 +2 + 6
73 -2 + 6
74 -5 - 2
75 +1 + +6
76 0
77 1
78 -1
79 123
80 -123
81 &
82 _
83 @
84 :::
85 a12121
86 a12132131321321
87 9adsasdgasdgasdga454adsg2sd
88 adsasdg3asdgasdga454adsg2sd
89 123456789123456789
90 1/0
```

其最终输出到 txt 文件中的结果如下：

```
var : K, (2, 1)
n : I, (2, 4)
, : D, (2, 5)
m : I, (2, 6)
, : D, (2, 7)
i : I, (2, 8)
, : D, (2, 9)
j : I, (2, 10)
, : D, (2, 11)
digit : I, (2, 12)
( : O, (2, 17)
1 : C, (2, 18)
: : T, (2, 19)
3 : C, (2, 20)
) : O, (2, 21)
; : D, (2, 22)
procedure : K, (4, 1)
init : I, (4, 10)
; : D, (4, 14)
begin : K, (5, 1)
i : I, (6, 1)
:= : O, (6, 2)
1 : C, (6, 4)
; : D, (6, 5)
while : K, (7, 1)
i : I, (7, 6)
< : O, (7, 7)
4 : C, (7, 8)
do : K, (7, 9)
begin : K, (8, 1)
digit : I, (9, 1)
( : O, (9, 6)
i : I, (9, 7)
) : O, (9, 8)
:= : O, (9, 9)
0 : C, (9, 11)
; : D, (9, 12)
; : D, (55, 11)
begin : K, (56, 1)
read : K, (57, 1)
( : O, (57, 5)
input : I, (57, 6)
) : O, (57, 11)
; : D, (57, 12)
case : K, (58, 1)
input : I, (58, 5)
: : T, (58, 10)
1 : C, (59, 1)
: : T, (59, 2)
a : I, (59, 3)
:= : O, (59, 4)
0 : C, (59, 6)
2 : C, (60, 1)
: : T, (60, 2)
a : I, (60, 3)
:= : O, (60, 4)
1 : C, (60, 6)
3 : C, (61, 1)
: : T, (61, 2)
a : I, (61, 3)
:= : O, (61, 4)
2 : C, (61, 6)
; : D, (61, 7)
endcase : K, (62, 1)
; : D, (62, 8)
end : K, (63, 1)
; : D, (63, 4)
call : K, (65, 1)
P : I, (65, 5)
. : O, (65, 6)
+1321 : C, (68, 1)
a : I, (69, 1)
:= : O, (69, 2)
+2 : C, (69, 4)
:= : O, (71, 2)
2 : C, (71, 4)
- : O, (71, 5)
6 : C, (71, 6)
+2 : C, (72, 1)
+ : O, (72, 3)
6 : C, (72, 4)
-2 : C, (73, 1)
+ : O, (73, 3)
6 : C, (73, 4)
-5 : C, (74, 1)
- : O, (74, 3)
2 : C, (74, 4)
+1 : C, (75, 1)
+ : O, (75, 3)
+6 : C, (75, 4)
0 : C, (76, 1)
1 : C, (77, 1)
-1 : C, (78, 1)
123 : C, (79, 1)
-123 : C, (80, 1)
& : T, (81, 1)
_ : T, (82, 1)
@ : T, (83, 1)
: : T, (84, 1)
: : T, (84, 2)
: : T, (84, 3)
a12121 : I, (85, 1)
Error!Expected a shorter IDENTIFIER!(86, 1)
9 : C, (87, 1)
Error!Expected a shorter IDENTIFIER!(87, 2)
Error!Expected a shorter IDENTIFIER!(88, 1)
Error!Expected a shorter CONSTANT!(89, 1)
1 : C, (90, 1)
/ : O, (90, 2)
0 : C, (90, 3)
```

从实验结果看来，效果还是比较理想的。

五、实验体会

本次实验是关于编译器制作中的第一步，即词法分析器。通过本次实验，理解了编译器的工作机制，掌握了词法分析器的生成工具 LEX 的用法，熟悉了正规式的匹配规则和相关指令，有待在后续实验中进一步熟悉和掌握。