用 Lex 设计词法分析器 1

一、实验目的：

学会用 lex 设计一个词法分析器。

二、实验内容：

使用 lex 为下述文法语言写一个词法分析器：

语言文法：

- <程序>→PROGRAM <标识符> ; <分程序>
- <分程序>→<变量说明> BEGIN <语句表> END.
- <变量说明>→VAR <变量说明表>;
- <变量说明表>→<变量表>: <类型> | <变量表>: <类型>; <变量说明表>
- <类型>→INTEGER | REAL
- <变量表>→<变量> | <变量>, <变量表>
- <语句表>→<语句> | <语句>; <语句表>
- <语句>→<赋值语句> | <条件语句> | <WHILE 语句> | <复合语句>
- <赋值语句>→<变量> := <算术表达式>
- <条件语句>→IF <关系表达式> THEN <语句> ELSE <语句>
- <WHILE 语句>→WHILE <关系表达式> DO <语句>
- <复合语句>→BEGIN <语句表> END
- <算术表达式>→<项> | <算术表达式> + <项> | <算术表达式> - <项>
- <项>→<因式> | <项> * <因式> | <项> / <因式>
- <因式>→<变量> | <常数> | (<算术表达式>)
- <关系表达式>→<算术表达式> <关系符> <算术表达式>
- <变量>→<标识符>
- <标识符>→<标识符><字母> | <标识符><数字> | <字母>
- <常数>→<整数> | <浮点数>
- <整数>→<数字> | <数字> <整数>
- <浮点数>→.<整数> | <整数>.<整数>
- <关系符>→< | <= | = | > | >=| <>
- <字母>→A | B | …| X | Y | Z | a | b | …| x | y | z
- <数字>→0|1|2|…|9

三、实验要求：

输入为用该语言所写的源程序文件；输出为记号序列，每个记号显示为二元组(记号名，记号属性值)的形式。输出可以在屏幕上，也可以输出到文件中。不要求建立符号表。

在 cygwin 下用 flex 和 gcc 工具将实验调试通过，并能通过例子 parser0 中 testcases 目录下的 test1.p 测试例的测试。

四、具体实现（文件 exam2.l 的展示）

1. 定义和声明：

```
%{

#include <stdio.h>
#define LT          1
#define LE          2
#define GT          3
#define GE          4
#define EQ          5
#define NE          6
#define PROGRAM     7
#define END         9
#define VAR         10
#define INTEGER     11
#define REAL        12
#define INT         13
#define FLOAT       14

#define IF          15
#define THEN        16
#define ELSE        17
#define WHILE       18
#define DO          19
#define ID          20
#define RELOP       22

#define NEWLINE     23
#define ERRORCHAR   24

#define COPY        25
#define SEMICOLON   26
#define COLON       27
#define OP          28
#define COMMA       29

%}
```

对应文法产生式：<程序>和<分程序>的产生式

- COPY 是赋值符号“=”的记号名；
- SEMICOLON 是分号“;”的记号名；
- COLON 是冒号“:”的记号名；
- OP 是运算符号的记号名，包括加减乘除：”+” ”-” ”*” ”/”
- COMMA 是逗号“,”的记号名

2. 正规定义：

```
delim       [ \t \n]
ws          {delim}+
letter_     [A-Za-z_]
digit       [0-9]
id          {letter_}({letter_}|{digit})*
Int         {digit}+
Float       {Int}?\.{Int}
```

- ws 是空格制表符回车的组合的正规定义；
- Int 和 Float 分别对应整形数和浮点型数的正规定义；
- 允许 0~1 之间的浮点数省略个位数的 0 且.在 Lex 源程序的规则中有定义，所以需要用\转义

3. 翻译规则：

```
62  /* ECHO是一个宏，相当于 fprintf(yyout, "%s", yytext)*/
63
64  <INITIAL>{ws}           {;}
65  <INITIAL>WHILE          {return (WHILE);}
66  <INITIAL>DO             {return (DO);}
67  <INITIAL>IF             {return (IF);}
68  <INITIAL>THEN           {return (THEN);}
69  <INITIAL>ELSE           {return (ELSE);}
70  <INITIAL>PROGRAM        {return (PROGRAM);}
71  <INITIAL>VAR            {return (VAR);}
72  <INITIAL>INTEGER        {return (INTEGER);}
73  <INITIAL>REAL           {return (REAL);}
74  <INITIAL>END            {return (END);}
75  <INITIAL>{id}           {return (ID);}
76  <INITIAL>{Int}          {return (INT);}
77  <INITIAL>{Float}        {return (FLOAT);}
78
79  <INITIAL>"<"            {return (RELOP);}
80  <INITIAL>"<="           {return (RELOP);}
81  <INITIAL>"=="           {return (RELOP);}
82  <INITIAL>"!="           {return (RELOP);}
83  <INITIAL>">"            {return (RELOP);}
84  <INITIAL>">="           {return (RELOP);}
85
86  <INITIAL>":="           {return (COPY);}
87  <INITIAL>";"            {return (SEMICOLON);}
88  <INITIAL>":"            {return (COLON);}
89  <INITIAL>","            {return (COMMA);}
90
91  <INITIAL>"+"            {return (OP);}
92  <INITIAL>"-"            {return (OP);}
93  <INITIAL>"*"            {return (OP);}
94  <INITIAL>"/"            {return (OP);}
95
96  <INITIAL>.              {return ERRORCHAR;}
```

关键字的翻译规则

比较运算符的翻译规则

赋值符号、分号、冒号、逗号的翻译规则

运算符的翻译规则

4． 输出函数：

```
104 void writeout(int c){
105   switch(c){
106     case ERRORCHAR: fprintf(yyout, "(ERRORCHAR, \"%s\") ", yytext);break;
107     case PROGRAM: fprintf(yyout, "(PROGRAM, \"%s\") ", yytext);break;
108
109     case END: fprintf(yyout, "(END, \"%s\") ", yytext);break;
110     case INTEGER: fprintf(yyout, "(INTEGER, \"%s\") ", yytext);break;
111     case REAL: fprintf(yyout, "(REAL, \"%s\") ", yytext);break;
112     case RELOP: fprintf(yyout, "(RELOP, \"%s\") ", yytext);break;
113     case WHILE: fprintf(yyout, "(WHILE, \"%s\") ", yytext);break;
114     case DO: fprintf(yyout, "(DO, \"%s\") ", yytext);break;
115     case IF: fprintf(yyout, "(IF, \"%s\") ", yytext);break;
116     case THEN: fprintf(yyout, "(THEN, \"%s\") ", yytext);break;
117     case ELSE: fprintf(yyout, "(ELSE, \"%s\") ", yytext);break;
118     case INT: fprintf(yyout, "(INT, \"%s\") ", yytext);break;
119     case FLOAT: fprintf(yyout, "(FLOAT, \"%s\") ", yytext);break;
120     case ID: fprintf(yyout, "(ID, \"%s\") ", yytext);break;
121     case NEWLINE: fprintf(yyout, "\n");break;
122     case COPY: fprintf(yyout, "(COPY, \"%s\") ", yytext);break;
123     case SEMICOLON: fprintf(yyout, "(SEMICOLON, \"%s\") ", yytext);break;
124     case COLON: fprintf(yyout, "(COLON, \"%s\") ", yytext);break;
125     case COMMA: fprintf(yyout, "(COMMA, \"%s\") ", yytext);break;
126     case OP: fprintf(yyout, "(OP, \"%s\") ", yytext);break;
127     default:break;
128   }
129   return;
130 }
```

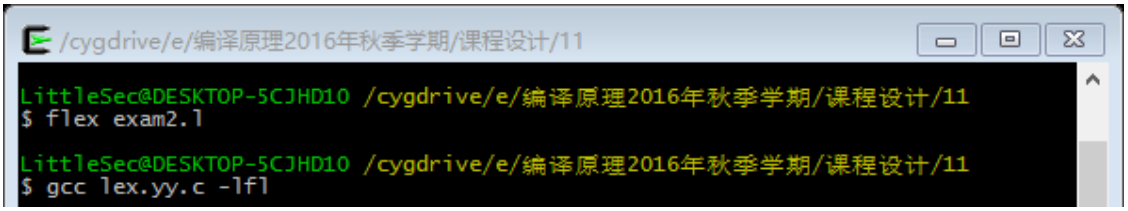
5. 测试文件 test1.p 内容如下：

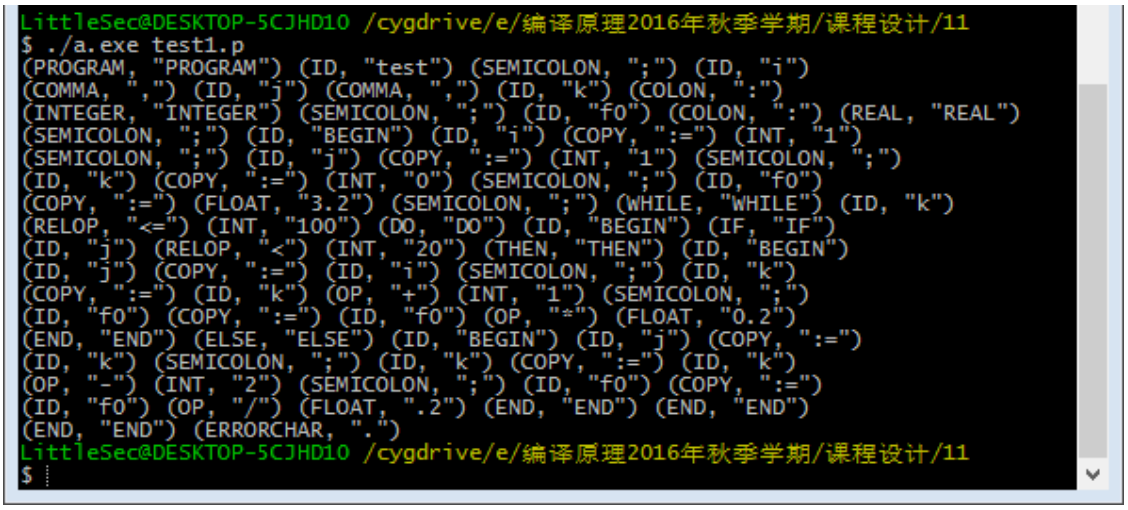
```
1  PROGRAM test;
2  VAR i, j, k: INTEGER;
3      f0: REAL;
4  BEGIN
5    i := 1;
6    j := 1;
7    k := 0;
8    f0 := 3.2;
9    WHILE k<=100 DO
10     BEGIN
11       IF j <20 THEN
12         BEGIN
13           j := i;
14           k := k+1;
15           f0 := f0*0.2
16         END
17       ELSE
18         BEGIN
19           j := k;
20           k := k-2;
21           f0 := f0/.2
22         END
23     END
24 END.
```

测试文件中包含：

- ✓ PROGRAM、VAR、BEGIN、END、IF、THEN、ELSE、WHILE、INTEGER、REAL 关键字
- ✓ 4 个变量名
- ✓ 赋值语句若干
- ✓ 整型数和浮点数若干
- ✓ 比较表达式若干
- ✓ 运算表达式若干

6. 在 cygwin 下用 flex 编译器对新的 exam2.l 进行编译，生成 lex.yy.c 文件，用 gcc 编译器对该 C 源程序进行编译得到 a.exe 文件，用测试文件 test1.p 进行测试，如图所示：





经核对，结果正确。

五、 心得与体会

1. 基本了解和掌握了 flex 词法分析器生成工具的使用，lex 的语法规则和组织方式。熟悉了 cygwin 环境的使用，并能在 cygwin 下使用 flex 编译器调试 lex 程序，利用 gcc 编译器调试 lex.yy.c 文件，以及执行测试文件（.p）。
2. 状态的定义和使用，有时候通过状态可有效的解决一些复杂问题，比如对注释的操作等。不同的状态之间可以使用“BEGIN”进行切换，在不同的状态中对同一正规式表示的句子可以执行不同的动作。
3. 动作可以是一个空语句，相当于过滤掉某些输入。例如对空格、制表符、换行的组合。
4. 动作中有返回语句时，返回值应当事先在定义段的第一部分用 C 语言的语法进行定义，以便于后面的操作和使用。
5. 在词法规则段，lex 总是尽可能匹配较长的句子。当发生冲突时，在词法规则段中首先出现的正规式将被匹配。
6. 掌握了一些具体的 lex 元字符的使用方法，可以简化正规定义。
7. 当想要将元字符作为正文字符使用时，可以使用转义字符\或“”。
8. 理解了用 C 语言写的辅助函数对于词法分析的重要作用。这些辅助函数一方面可以作为语义动作的一部分，另一方面可以为词法分析提供依据。而且可以将 main 函数作为辅助函数放在这里，当单独使用词法分析器的时候可以在 main 函数里实现对词法分析器的调度和使用。