

# 用 lex 和 yacc 写一个计算布尔表达式真值的计算器

## 一、实验目的：

熟悉语法分析器生成工具 Yacc 的使用，并学会在 cygwin 下使用 bison 工具编译 Yacc 语法说明文件。学习如何使用 lex 和 yacc 合作进行语法分析。

## 二、实验内容：

根据给出的 calculator 例子 (calculator0, calculator1, calculator2, calculator3) 完成下面题目：用 lex 和 yacc 写一个计算布尔表达式真值的计算器。

## 三、实验要求：

1. 输入为一个布尔表达式，以换行结束。输出为这个布尔表达式的真值 (true 或 false)。
2. 必须用二义文法实现。布尔表达式二义文法为： $S \rightarrow S \text{ or } S \mid S \text{ and } S \mid \text{not } S \mid (S) \mid \text{true} \mid \text{false}$ ，其中优先级  $\text{or} < \text{and} < \text{not}$ ，or 和 and 左结合，not 右结合。
3. 用非二义文法实现作为选作内容，非二义文法请参照表达式非二义文法自己写出来。
4. 在 cygwin 下用 flex, bison 和 gcc 工具将实验调试通过，并写出测试例测试正确性。

## 四、具体实现

### Yacc 文件

#### 1. 定义段部分：

```
1 %{\n2 #include <ctype.h>\n3 #include <stdio.h>\n4 int yylex();\n5 int yyerror();\n6 %}\n7\n8 %token T F LPAREN RPAREN ENTER\n9 %left OR\n10 %left AND\n11 %right NOT
```

- int yylex()是词法分析程序，它返回记号。语法分析驱动程序 yyparse()将会调用 yylex()获取记号。如果不使用 lex 生成这个函数，则必须在辅助函数段用 C 语言写这个程序。
- yyerror()函数用于输出错误信息。
- 使用默认属性值栈的元素类型，即 int 型，直接用 0 和 1 表示 false 和 true。
- 记号 T 和 F 分别表示文法符号 true 和 false，之后的三个分别是左右括号和回车，用于改变优先级和结合性以及识别输入是否结束。
- 接下来的是三个逻辑运算，OR 和 AND 是左结合，NOT 是右结合。对应的文法符号就是 C 语言中的逻辑预算 ||、&&、!。
- 结合性：由定义出现的顺序决定的，先定义的优先级低，最后定义的优先级最高，同时定义的优先级相同。即  $\text{OR} < \text{AND} < \text{NOT}$ ，符合要求。
- 当然这是带二义文法的定义，如果是无二义文法，则可以把 AND 和 OR 合在一行，并且 NOT 符号可以提前写。

#### 2. 有二义的文法：

```

16 prog      : prog exprp
17           | exprp
18           ;
19
20 exprp     : expr ENTER {
21           if($1) printf("true\n");
22           else printf("false\n");}
23           ;
24
25 expr      : expr AND expr {$$ = $1 && $3;}
26           | expr OR expr  {$$ = $1 || $3;}
27           | NOT expr %prec NOT {$$ = ! $2;}
28           | LPAREN expr RPAREN {$$ = $2;}
29           | T {$$ = 1;}
30           | F {$$ = 0;}
31           ;

```

- 直接使用题目给的文法，具有二义性，通过对优先级和结合性消除二义。
- 紧接着文法的时候语义动作，引用存放在属性值栈中的文法符号的属性值，模拟移进-规约过程。
- 第二条文法是为了识别输入是否结束，其语义动作是通过判断输出输出式的布尔值。
- 用%prec NOT 强制定义了其优先级与结合性跟 NOT 相同，使得非运算优先级最高。

### 3. 无二义的文法:

```

16 prog      : prog exprp
17           | exprp
18           ;
19
20 exprp     : expr ENTER {
21           if($1) printf("true\n");
22           else printf("false\n");}
23           ;
24
25 expr      : expr OR expr1 {$$ = $1 || $3;}
26           | expr1
27           ;
28
29 expr1     : expr1 AND expr2 {$$ = $1 && $3;}
30           | expr2
31           ;
32
33 expr2     : NOT expr2 {$$ = ! $2;}
34           | LPAREN expr RPAREN {$$ = $2;}
35           | T {$$ = 1;}
36           | F {$$ = 0;}
37           ;

```

直观来看就是:

- $E \rightarrow E \text{ or } T \mid T$
- $T \rightarrow T \text{ and } F \mid F$
- $F \rightarrow \text{not } F \mid (E) \mid \text{true} \mid \text{false}$

### 4. 辅助函数段:

```

42 int main()
43 {
44     yyparse();
45     return 0;
46 }

```

- yyparse()是语法分析驱动程序，它会调用 yylex()获取记号。

## Lex 文件

```
1  %{
2  #include "cal.tab.h"
3  int yywrap(void){ return 1;}
4  %}
5
6  delim      [ \t]
7  ws         {delim}+
8
9
10 %%
11 false      {return F;}
12 true       {return T;}
13 "||"       {return OR;}
14 "&&"       {return AND;}
15 "!"        {return NOT;}
16 "("        {return LPAREN;}
17 ")"        {return RPAREN;}
18 {ws}       {;}
19 "\n"       {return ENTER;}
20 .          {printf("\nLEX:ERROR! c=%s\n", yytext);}
21
22 %%
```

- 用 yacc 编译器对 cal.y 文件进行编译，编译时带上参数-d，此时编译器除生成 cal.tab.c 以外，还将生成名为 cal.tab.h 的头文件。该头文件中包含 cal.y 中定义的所有终结符的常量定义，属性值栈的类型定义，以及变量 yylval 的外部引用定义。用 Lex 写的词法分析规则文件为 cal.l，则在 cal.l 的声明部分应包含头文件 cal.tab.h，即，在 cal.l 声明部分应包含如下语句：#include "cal.tab.h"。并且，cal.l 文件中凡涉及返回记号名的部分，都返回 cal.y 中定义的终结符名。
- 其他的和之前的 Lex 文件是一样的。

## Makefile 文件

1. 在 cygwin 下用 LEX 定义词法分析器并把它和 YACC 写的语法分析器链接起来的命令相对较多，使用 bison 和 flex 联合写一个语法分析器时，需要的编译步骤稍显复杂，用 makefile 可以将这个复杂的编译步骤简化。
2. Makefile 告诉我们如何对一个包含若干源文件的工程进行编译，比如，先编译什么，后编译什么，怎样链接等。Makefile 文件直接使用例子给出的 makefile 文件作出部分修改：

```

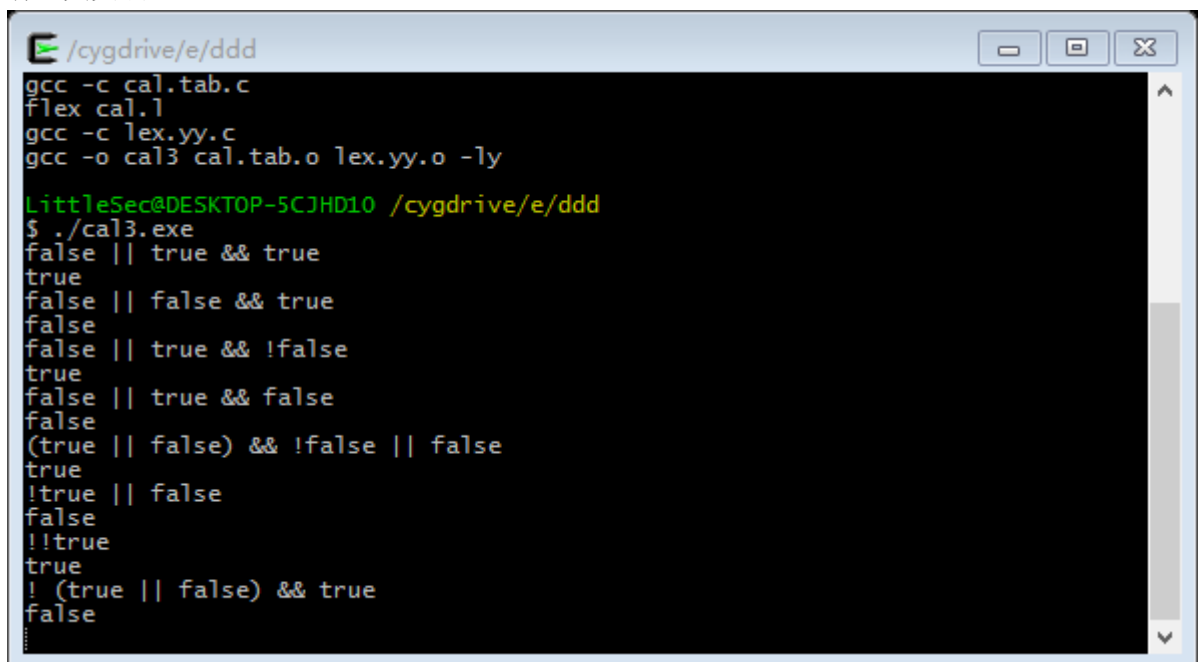
1  cal3: cal.tab.o lex.yy.o
2      gcc -o cal3 cal.tab.o lex.yy.o -ly
3
4  lex.yy.o: lex.yy.c cal.tab.h
5      gcc -c lex.yy.c
6
7  cal.tab.o: cal.tab.c
8      gcc -c cal.tab.c
9
10 lex.yy.c: cal.l
11     flex cal.l
12
13 cal.tab.c: cal.y
14     bison -dv cal.y
15
16 cal.tab.h: cal.y
17     echo "cal.tab.h was created at the same time as cal.tab.c."
18
19 clean:
20     rm -f cal3.exe lex.yy.o cal.tab.o lex.yy.c cal.tab.c cal.tab.h cal3.exe.stackdump cal.output

```

3. 编译时，在 cygwin 下进入文件路径，直接输入 make 即可正确编译。

## 实验结果

1. 有二义文法:



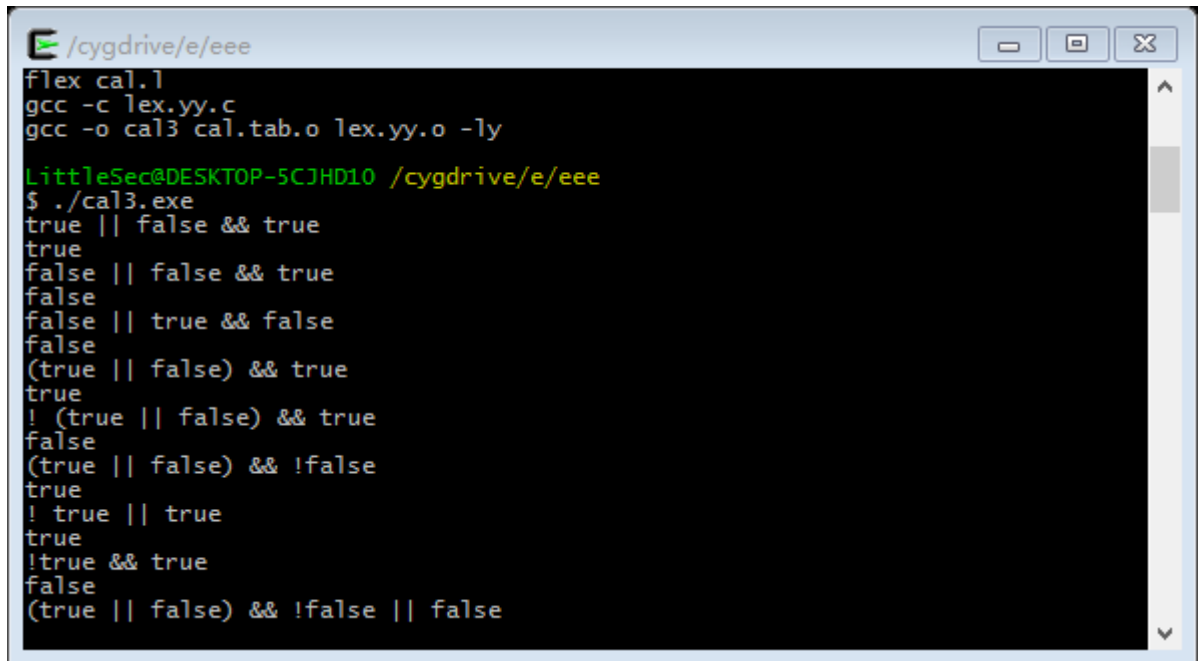
```

/cygdrive/e/ddd
gcc -c cal.tab.c
flex cal.l
gcc -c lex.yy.c
gcc -o cal3 cal.tab.o lex.yy.o -ly

LittleSec@DESKTOP-5CJHD10 /cygdrive/e/ddd
$ ./cal3.exe
false || true && true
true
false || false && true
false
false || true && !false
true
false || true && false
false
(true || false) && !false || false
true
!true || false
false
!!true
true
! (true || false) && true
false
...

```

2. 无二义文法:



```
/cygdrive/e/eee
flex cal.l
gcc -c lex.yy.c
gcc -o cal3 cal.tab.o lex.yy.o -ly

LittleSec@DESKTOP-5CJHD10 /cygdrive/e/eee
$ ./cal3.exe
true || false && true
true
false || false && true
false
false || true && false
false
(true || false) && true
true
!(true || false) && true
false
(true || false) && !false
true
! true || true
true
!true && true
false
(true || false) && !false || false
```

- 输入 make 会自动执行 makefile 文件中的指令，对各个文件进行编译连接。
- 输入表达式，回车后会输出表达式的布尔值。
- 经过多组测试，无论是有二义还是无二义的文法，结果均正确。

## 五、心得与体会

1. 了解和熟悉语法分析器生成工具 Yacc 的使用，并学会在 cygwin 下使用 bison 工具编译 Yacc 文法说明文件。学习了使用 Lex 和 Yacc 合作进行语法分析还有 makefile 文件的使用。
2. Yacc 和 Lex 的链接其实就是通过编译 Yacc 源程序得到的 .tab.h 头文件连接的，在 Lex 源程序中包含这个 Yacc 生成的头文件即可。
3. 无二义文法和带优先级的二义文法的效果大致相同。两者比较来看，带优先级的二义文法的书写更加简便一些，只需为相应的运算符指定优先级就可以了。而改造后的无二义的文法结构更加清晰，容易理解。