Lab3 ExprEval Design

- **二义性分析：**

一元取负运算符和二元减法运算符在词法分析中已经将其区分开来，如果它前一个 Token 是数值或右括号时为减法运算符 MINUS，若‘-’是第一个字符，或者其他情况说明“-”为取负运算符 NEG。

- **词法分析的设计与实现**

根据所给出的正则表达式构造 DFA 然后进行编程实现。

对所给的输入生成一串 Token 流，然后交给 Parser 类进行语法和语义分析。

在其中会检查一些词法错误

- **算符优先关系定义**

算符优先级表大致如实验文档所给出的

级别	描述	算符	结合性质
1	括号	( )	
2	预定义函数	sin cos max min	
3	取负运算（一元运算符）	–	右结合
4	求幂运算	^	右结合
5	乘除运算	* /	
6	加减运算	+ –	
7	关系运算	= <> < <= > >=	
8	非运算	!	右结合
9	与运算	&	
10	或运算	|	
11	选择运算（三元运算符）	?:	右结合

对应的表格大致如下：

	decimal	TRUE	FALSE	sin	cos	max	min	neg	,	+	–	*	/	^	(	)	?	:	>	>=	<	<=	=	<>	&	|	!	$
decimal	1	1	1	1	1	1	1	>	>	>	>	>	>	>	1	>	>	>	>	>	>	>	>	>	>	>	>	>
TRUE	1	1	1	1	1	1	1	>	>	>	>	>	>	>	1	>	>	>	>	>	>	>	>	>	>	>	>	>
FALSE	1	1	1	1	1	1	1	>	>	>	>	>	>	>	1	>	>	>	>	>	>	>	>	>	>	>	>	>
sin	5	5	5	5	5	5	5	5	5	5	5	5	5	5	=	5	5	5	5	5	5	5	5	5	5	5	5	>
cos	5	5	5	5	5	5	5	5	5	5	5	5	5	5	=	5	5	5	5	5	5	5	5	5	5	5	5	>
max	5	5	5	5	5	5	5	5	5	5	5	5	5	5	=	5	5	5	5	5	5	5	5	5	5	5	5	>
min	5	5	5	5	5	5	5	5	5	5	5	5	5	5	=	5	5	5	5	5	5	5	5	5	5	5	5	>
neg	<	<	<	<	<	<	<	<	>	>	>	>	>	>	<	>	>	>	>	>	>	>	>	>	>	>	>	>
,	<	<	<	<	<	<	<	<	>	<	<	<	<	<	<	>	<	<	<	<	<	<	<	<	<	<	<	>
+	<	<	<	<	<	<	<	<	>	>	>	<	<	<	<	>	>	>	>	>	>	>	>	>	>	>	>	>
–	<	<	<	<	<	<	<	<	>	>	>	<	<	<	<	>	>	>	>	>	>	>	>	>	>	>	>	>
*	<	<	<	<	<	<	<	<	>	>	>	>	>	<	<	>	>	>	>	>	>	>	>	>	>	>	>	>
/	<	<	<	<	<	<	<	<	>	>	>	>	>	<	<	>	>	>	>	>	>	>	>	>	>	>	>	>
^	<	<	<	<	<	<	<	<	>	>	>	>	>	<	<	>	>	>	>	>	>	>	>	>	>	>	>	>
(	<	<	<	<	<	<	<	<	<	<	<	<	<	<	<	=	<	6	<	<	<	<	<	<	<	<	<	4
)	1	1	1	>	>	>	>	1	>	>	>	>	>	>	1	>	>	>	>	>	>	>	>	>	>	>	>	>
?	<	<	<	<	<	<	<	<	6	<	<	<	<	<	<	6	<	=	<	<	<	<	<	<	<	<	<	>
:	<	<	<	<	<	<	<	<	>	<	<	<	<	<	<	>	<	<	<	<	<	<	<	<	<	<	<	>
>	<	<	<	<	<	<	<	<	>	<	<	<	<	<	<	>	>	>	>	>	>	>	>	>	>	>	>	>
>=	<	<	<	<	<	<	<	<	>	<	<	<	<	<	<	>	>	>	>	>	>	>	>	>	>	>	>	>
<	<	<	<	<	<	<	<	<	>	<	<	<	<	<	<	>	>	>	>	>	>	>	>	>	>	>	>	>
<=	<	<	<	<	<	<	<	<	>	<	<	<	<	<	<	>	>	>	>	>	>	>	>	>	>	>	>	>
=	<	<	<	<	<	<	<	<	>	<	<	<	<	<	<	>	>	>	>	>	>	>	>	>	>	>	>	>
<>	<	<	<	<	<	<	<	<	>	<	<	<	<	<	<	>	>	>	>	>	>	>	>	>	>	>	>	>
&	<	<	<	<	<	<	<	<	>	<	<	<	<	<	<	>	>	>	<	<	<	<	<	<	<	>	>	>
|	<	<	<	<	<	<	<	<	>	<	<	<	<	<	<	>	>	>	<	<	<	<	<	<	<	<	>	>
!	<	<	<	<	<	<	<	<	>	<	<	<	<	<	<	>	>	>	<	<	<	<	<	<	<	<	>	>
$	<	<	<	<	<	<	<	<	<	<	<	<	<	<	<	3	<	<	<	<	<	<	<	<	<	<	<	

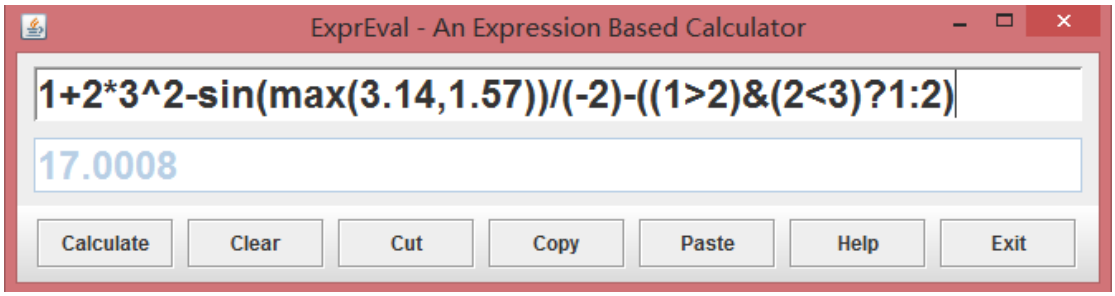
- **语法分析与语义处理的设计与实现**

根据 OPP 的 parser table 进行移近和归约操作。大致伪代码如下

```
initialize();
for (;;) {
if (top == $ && lookahead == $)
        accept();
topOp = stack.topMostTerminal();
if (topOp < lookahead || topOp ≡ lookahead) {
        // shift
        stack.push(lookahead);
        lookahead = scanner.getNextToken();
} else if (topOp ϕ lookahead) {
        // reduce
          do {
              topOp = stack.pop();
        } while (stack.topMostTerminal() > || ≡ topOp);
} else
        error(); }
```

- **程序运行的屏幕截图**

截图只上一张，其他的具体请看测试样例。



- **实验的心得体会**

这次的实验的问题不是在于实验本身的难度问题，主要是它把之前所学习的词法分析和语法分析的 OPP 结合在一起了。本身这两者的编程实现都需要一点时间，所以这次实验虽然有些艰难，但是真正完成后感觉自己学到了很多，也提升了很多。