中山大學 SUN YAT-SEN UNIVERSITY

编译原理

作业名称	基于表达式的计算器 ExprEval		任课老师	万老师		
院系	软件学院		方向	通信软件	姓名	李明宽
学号	11331173		完成日期	2013 年 11 月 24 日		
QQ	736459905		E-mail	limkuan@mail2.sysu.edu.cn		

实验报告目录

一、 实验目的

本实验借助一个基于表达式的计算器 ExprEval 的设计与实现过程，帮助我们深入了解和牢固掌握编译原理中的词法分析、语法分析、语义分析等重要环节。本实验的核心知识是算符优先分析（Operator Precedence Parsing，简称 OPP）技术，重点是算符优先关系表的构造。

本实验的主要目标包括：

1. 掌握词法分析程序的工作原理与构造方法，包括较复杂的浮点数常量的词法规则定义及其识别程序的构造。学习如何根据词法规则定义（例如正则表达式或正则文法）来构造一个词法扫描程序的程序蓝图（即有限状态机），并利用高级程序语言实现词法分析过程。
2. 掌握算符优先分析技术，除基础的算术运算符外，还包括重载的（Overloading）一元运算符、三元运算符、关系运算符、逻辑运算符、预定义函数等运算符的处理。
3. 掌握基本的语义处理技术，能够正确地处理表达式计算中的类型兼容检测和类型自动推导。
4. 通过加强软件设计方面的交流与讨论，并在面向对象编程风格的大量实践，提高对面向对象设计的认识，养成良好的编程习惯，并了解大型工程文档的组织与提交。
5. 加深了解软件测试的工作原理与使用方法，初步体会软件测试自动化基本思路。

二、 实验过程

1. 讨论语法定义的二义性

这个文法的定义在实验说明文档中 2.3.1 小节给出。如果只看这个文法来说，它是有二义性的，因为对于文法 $decimal + decimal * decimal$，可能产生如下两种不同的最左推导（见下一页）：

$$
\begin{aligned}
Expr &\rightarrow ArithExpr \\
&\rightarrow ArithExpr + ArithExpr \\
&\rightarrow decimal + ArithExpr \\
&\rightarrow decimal + ArithExpr * ArithExpr \\
&\rightarrow decimal + decimal * ArithExpr \\
&\rightarrow decimal + decimal * decimal
\end{aligned}
$$

另一种推导如下：

$$
\begin{aligned}
Expr &\rightarrow ArithExpr \\
&\rightarrow ArithExpr * ArithExpr \\
&\rightarrow ArithExpr + ArithExpr * ArithExpr \\
&\rightarrow decimal + ArithExpr * ArithExpr \\
&\rightarrow decimal + decimal * ArithExpr \\
&\rightarrow decimal + decimal * decimal
\end{aligned}
$$

对于同样一个文法 $decimal + decimal * decimal$，却存在两棵不同的语法树，显然如果单单是 BNF 定义的语法是存在二义性的。

而 ExprEval 是不存在二义性的，因为在定义了语法推导之后，ExprEval 有定义对于每一个终结符的优先级，通过这样消除了文法的二义性，因为可以在语法分析过程中，对于一个给定的文法，由于有优先级的存在，每次只能使用唯一推导式进行推导，最后也就消除了二义性。对于上面给出的例子，由于加法运算符+以及乘法运算符*有优先级的差别，因此第二种推导方法是不会出现的，只能出现第一种语法推导，因此它是无二义性的。

2. 设计并实现词法分析程序

在词法分析中，我对符号的分类比较具体，基本完成了实验后才体会到其实很多符号都是可以合并的，例如布尔的常量 true 和 false，以及预定义的函数等等，但在实现的过程中，我把每一个符号都进行了划分处理。

Token 类描述的是一个词法单元，里面包含了这个词法单元的类型以及这个词法单元的字符串表示。Scanner 类有一个成员函数是 getNextToken，可以通过这个函数获得当前指针的下一个 Token。

对于表达式的词法分析比较简单，就没有画出优先状态机，下面分别对一个个词法单元符号进行说明：

（首先对表达式字符串进行了处理，所有的字母都变成了小写）

- 判断下一个 Token 是不是数值常量的方法：如果扫描到当前位置的字符是一个数字，那接着往下扫描，直到遇到的字符不是[.e0-9+-]，其中扫描到加减号的时候进行判断前一个字符是不是 e，如果是那就说明这个运算符是在数值常量里的，如果不是那就说明数值常量已经结束，这个加减号是运算符。最后获得一个字符串，使用正则表达式"[0-9]+(.[0-9]+)?(e[+-]?[0-9]+)?"判断他是不是一个合法的 Decimal，如果是就返回这个数值常量的 Token，如果不是抛出 IllegalDecimalException 异常。

- 判断下一个 Token 是取负运算符还是减法运算符的方法：查看前一个 Token（这里使用的方法是看前一个除空格之外的字符），如果前一个 Token 是数值常量或者右括号，那这个-就是减法运算符，否则它就是取负运算符。
- 判断下一个 Token 是不是预定义的函数：如果读到一个字符是 s，那接着往下看是不是 i 和 n，如果是那就是 sin，否则报出 IllegalIdentifierException 的异常，同理如果读取到的字符是 c，就判断下两个字符是不是 o 和 s，是就返回预定义函数符号 cos 的 Token。如果到扫描到字母 m，继续判断下一位是 i 还是 a，如果是 i，判断在下一位是不是 n，如果是那就返回词法单元 min，如果是 a，判断下一位是不是 x，如果是就返回词法单元 max，否则抛出 IllegalIdentifierException 异常。
- 对于 true 和 false 的布尔常量的判断与上面判断函数的方法类似，当读取到一个 t 的时候,只有接下来的字符是 ure,才返回 true 这一布尔常量的 Token，如果读取 f 的时候，接下来的字符是 alse 才返回布尔常量的 Token，否则都抛出 IllegalIdentifierException 异常
- 对于其他一些只有一个字符的运算符，如+*/^&|!等，只要读取到这些字符，就把字符对应的操作符 Token 返回。
- 特别注意当读取到空格的时候，与要忽略空格，就把指针往下一个字符，重复上述过程直到遇到的字符不是空格。
- 结束的时候如果都没有匹配的词法单元，如果当前指针的位置是一个字母，那就会让词法分析器出现 IllegalIdentifierException 这个异常，否则就会抛出 IllegalSymbolException。
- 当指针的位置是表达式字符串结尾的时候，词法分析结束，并把终止符$返回语法分析器。

3. 构造算符优先关系表

经过长时间仔细构造,得到一个比较完整的 OPP 表,详见同文件夹<OPP.xlsx>。

在算符优先分析表中，通过合并得到 17 种不同的符号，其中所有的预定义函数如 sin、cos、max、min 合并成 Function 关键字，加法运算符与减法运算符可以合并，乘法和除法运算符也可以合并，所有的关系运算符由于是相同的优先级因此也可以进行合并成 Com 关键字。

建 OPP 表的过程中一定要对比清楚两个相应的运算符之间的优先级，横坐标是下一个 Token 的符号，纵坐标是当前符号栈的栈顶符号

PS：在建表格的过程中，由于疏忽，完成表格后半部分的时候忘记有关缺少运算符的异常在前面已经定义，于是又重新用一个新的符号。直到语法分析程序 Parser 快完成的时候才发现，改动涉及的工程量较大就没改它。这个警告对程序运行的正确性没有响应。

	+/-	*/	^	-	(	)	?	:	&	|	!	Func	,	Com	Dec	Bool	$
+/-	2	1	1	1	1	2	2	2	2	2	-1	1	2	2	1	-2	2
*/	2	2	1	1	1	2	2	2	2	2	-1	1	2	2	1	-2	2
^	2	2	1	1	1	2	2	2	2	2	-1	1	2	2	1	-2	2
-	3	3	3	1	1	3	3	3	3	3	-1	1	3	3	1	-2	3
(	1	1	1	1	1	1	1	-3	1	1	1	1	1	1	1	1	-4
)	4	4	4	-5	-5	4	4	4	4	4	-5	-5	4	4	-5	-5	4
?	1	1	1	1	1	-3	1	1	1	1	1	1	-3	1	1	1	-3
:	1	1	1	1	1	5	1	1	1	1	1	1	-5	1	1	1	5
&	1	1	1	1	1	6	6	6	6	6	6	1	6	1	1	1	6
|	1	1	1	1	1	6	6	6	1	6	6	1	6	1	1	1	6
!	1	1	1	1	1	7	7	7	7	7	1	-5	-2	1	-2	1	7
Func	-6	-6	-6	-6	1	-6	-6	-6	-6	-6	-6	-6	-6	-6	-6	-6	-6
,	1	1	1	1	1	1	1	-3	1	1	1	1	1	1	1	1	-6
Com	1	1	1	1	1	8	8	8	8	8	8	1	8	8	1	-2	8
Dec	9	9	9	9	-5	9	9	9	9	9	-5	-5	9	9	-5	-5	9
Bool	-2	-2	-2	-2	-5	9	9	-2	9	9	-5	-5	-2	-2	-5	-5	9
$	1	1	1	1	1	1	1	1	1	1	1	1	-7	1	1	1	0

-7	未知错误
-6	函数语法形式错误
-5	缺少运算符
-4	缺少右括号
-3	三目运算符异常
-2	类型不匹配异常
-1	缺少运算符
0	Accept
1	移入
2	二元运算
3	单目运算（取负）
4	括号运算(函数)
5	三目运算
6	与或运算
7	布尔运算
8	比较运算
9	操作数压栈

对于处理一元运算符取负运算以及二元运算符减法的区分：

- 由于我在词法分析过程已经把取负运算以及减法运算相区分，因此在建立 OPP 表的时候，两者是分开的。取负运算的优先级比较高，而且是右结合性的。因此在遇到它本身，或者左括号，或者函数，或者数值型变量的时候需要压栈，其他情况都是进行计算规约。
- 而减法运算的运算优先级较低，因此当遇到比它更低的关系运算，与或非运算以及选择运算，它本身，左括号，字符串终结符的时候才进行计算规约，而其他进行移入。（在没有错误的情况下）

4. 设计并实现语法分析和语义分析程序

- 对于 OPP 语法分析的核心控制程序是函数 parser，所有的操作都包含在一个循环当中，每进行一次判断的时候，得到当前符号栈栈顶的 Token 以及下一个要读取的 Token，对比这两个 Token 表示的符号的优先级，也就是查 OPP 表看这一步需要进行什么操作，如果是接收状态，那就退出循环，并判断操作数栈顶是不是数值常量，如果是，就语法分析结束，那个值就是整个表达式运算的结果；如果是移入操作，就调用 shift()函数进行移入，如果是 2-9 就调用 reduce()函数进行规约；如果得到的 action 是负数，说明出现了错误，就要在程序中抛出相应的错误。核心代码如下：

```
while (true) {
            printState();
            topToken = opStack.peek();
            int action =
            OPP_TABLE[getTokenIndex(topToken)][getTokenIndex(nextToken)];
            if (action == 0) {
                break;
            } else if (action == 1) {
                shift(nextToken);
            } else if (action == 2 || action == 3 || action == 4
                    || action == 5 || action == 6 || action == 7
                     || action == 8 || action == 9) {
                reduce(action);
            } else if (action == -1 || action == -2 || action == -3
                    || action == -4 || action == -5 || action == -6
                    || action == -7) {
                throw new Exception();
            }
        }
        if (valStack.isEmpty()) {
            throw new MissingOperandException();
        } else if (valStack.peek().symbol == Symbol.DECIMAL) {
            result = Double.parseDouble(valStack.peek().valueString);
        } else {
            throw new TypeMismatchedException();
        }
    }
```

- 对于 shift 函数，主要执行的操作是把 Token 压入运算符栈，然后 nextToken 往前读取一个。
- 对于 reduce 函数，由于按照不同的文法推导式进行规约使用的是不同的方法，因此，根据 OPP 表中每一种规约类型有：

1) 对于双目运算，从操作数栈中取出两个 Token，判断它是不是数值类型的，如果不是就抛出类型不匹配的语义错误，如果是数值类型，使用 Double 的 parseDouble 接口对这个进行字符串转换成浮点数，然后从符号栈中得到栈顶的二元运算符，判断相应的进行计算。要注意的是计算除法的时候要先判断除数是否为零，如果是就抛出语义错误。最后运算结果变成 Token 重新压进符号栈中。

2) 对于取负运算，操作数栈中弹出一个 Token 判断这个 Token 是不是数值类型，如果正常就把取负后得到的结果变成 Token 压入符号栈中。

3) 对于括号操作，这个比较麻烦，不同的处理方法可能会导致后面异常判断的不一样。我使用的方法是首先弹出符号栈中栈顶的右括号，然后接着弹出符号栈的第二个 Token 并记录，如果这个 Token 是不是左括号，如果是，那看新的符号栈栈顶是不是预定义的 sin 运算，如果是，取出操作数栈中的栈顶数值，然后进行计算，最后把计算结果重新压进符号栈中。Cos 的处理方法与 sin 类似。

   如果刚刚所说的符号栈的栈顶不是左括号，那有可能是 min 或者 max 函数，然后通过迭代器从后往前遍历左括号，根据找到左括号后再前面一个 Token 判断它是 min 还是 max，如果都不是那就标识符错误了；知道是哪个函数后，从后往前取出操作数栈栈顶的数值，并对比前一个，根据函数的需求保留，然后弹出符号栈栈顶的逗号，直到符号栈栈顶为左括号，这样最后保留下来的那个数就是函数的结果，相当于 max 函数中的最大值。要注意的是取出的每一个操作数都要进行判断，如果不是数值类型的，那就抛出类型不匹配的语义错误。

4) 对于三目运算符的运算，取出操作数栈栈顶的三个 Token，其中前两个都是数值类型，最后一个是布尔类型，然后使用？：选择运算得到的结果压入符号栈中。要注意的错误类型会如果取到的第二个操作数是布尔型的，可能不是类型错误，而是缺少操作数，因为如果对于？：这个选择运算来说，如果问号后面识别到一个布尔型的值，那在 OPP 表里已经能够判断出示类型错误。

5) 对于接下来的与或二元运算，逻辑非一元运算，以及比较操作符的运算都比较类似，都是取出操作数栈顶的数值进行计算，然后根据不同的运算符把相应计算结果变成 Token 压入符号栈中。一样要注意的是取出操作数 Token 的时候要进行类型判断，可以适当抛出类型不匹配异常的同时，也能避免出现字符串转换成 double 类型或者布尔型的时候出错。

6) 还有一个操作就是把操作数从符号栈中弹出并压入操作数栈中，进行这一步的目的是能够检测出两个操作数之间缺少运算符等一些异常。

5. 测试你的实验结果

对于测试样例，我自己根据每一个错误类型都写了相应的测试样例去检测程序是否能够正确识别各种错误类型，以及使用一些比较复杂的正确的测试样例来检测程序运行的正确性，实验结果的测试见下一节成果展示。

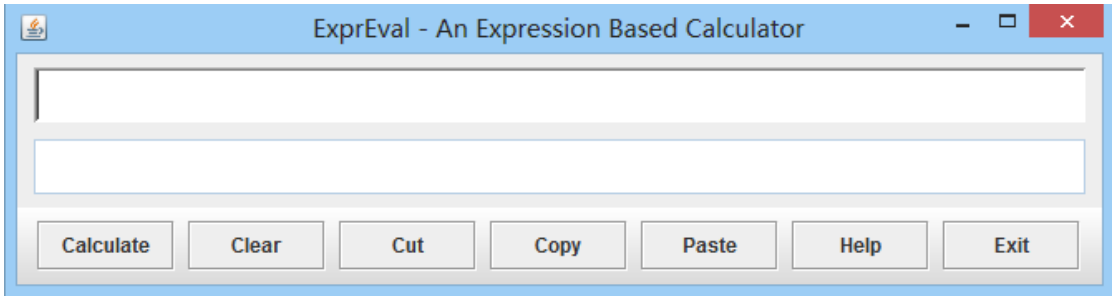
三、 实验成果展示

1. 目录结构与文件说明

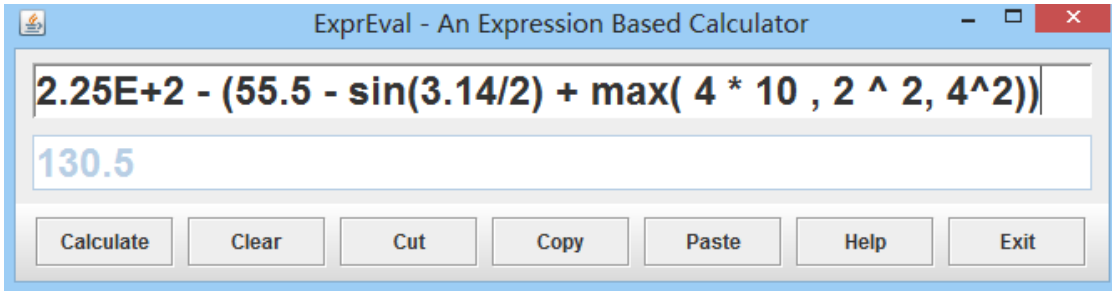
```
├─── bin……编译生成的.class 文件所在目录
├─── build.bat……编译构建代码的脚本
├─── clean.bat……清理构建出来的文件（包括 doc）的脚本
├─── design
│    ├─── design.docx……该实验的实验报告
│    └─── OPP.xlsx……实验中对应的算符优先表
├─── doc……生成的 JavaDoc 存放的目录
├─── doc.bat……生成 JavaDoc 的脚本文件
├─── ref……实验软装置中所有源程序的文档
├─── run.bat……程序的运行脚本
├─── readme.txt……项目自述文件，包括我的详细信息
├─── src……程序的源代码文件夹
│    ├─── exceptions
│    │    ├─── DividedByZeroException.java
│    │    ├─── EmptyExpressionException.java
│    │    ├─── ……共 16 个 Exception.java
│    │    └─── TypeMismatchedException.java……一系列共 16 个错误类型的定义
│    ├─── ExprEval.java……启动 ExprEval 计算器的主程序
│    ├─── gui
│    │    └─── MainWindow.java……GUI 界面的主程序
│    ├─── parser
│    │    ├─── Calculator.java……简单的计算器程序，给上层提供接口
│    │    ├─── Parser.java……表达式求值中用到的语法分析器
│    │    ├─── Scanner.java……表达式求值中用到的词法分析器
│    │    ├─── Symbol.java……表达式字符串会出现的所有符号定义
│    │    └─── Token.java……词法单元的定义
│    └─── test
│         ├─── ExprEvalTest.java……回归测试的主程序
│         └─── TestCase.java……一个测试用例的抽象
├─── testcases
│    ├─── Mytestcases.xml……我自己编写的测试文件
│    ├─── reports……运行回归测试生成的测试报告
│    ├─── simple.xml……软装置中给出的一个测试样例
│    └─── standard.xml……软装置中给出的另一个测试样例
├─── test_myself.bat……运行我的测试样例的脚本
├─── test_simple.bat……运行软装置的测试样例
└─── test_standard.bat……运行软装置中的测试样例
```

2. 主程序运行截图：

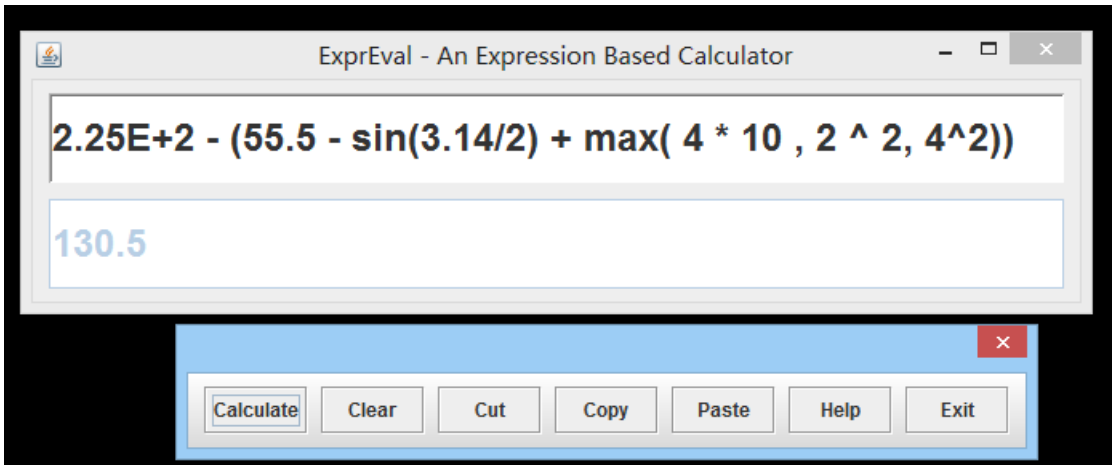
- 打开主程序



- 进行字符串求值：

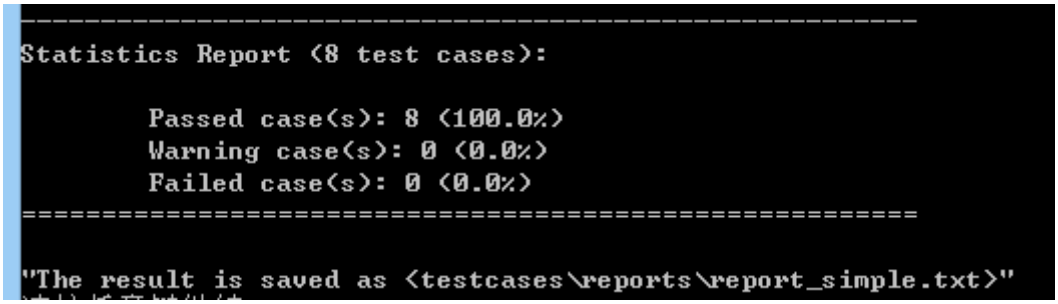


- 工具栏的浮动功能（软装置中已实现）：

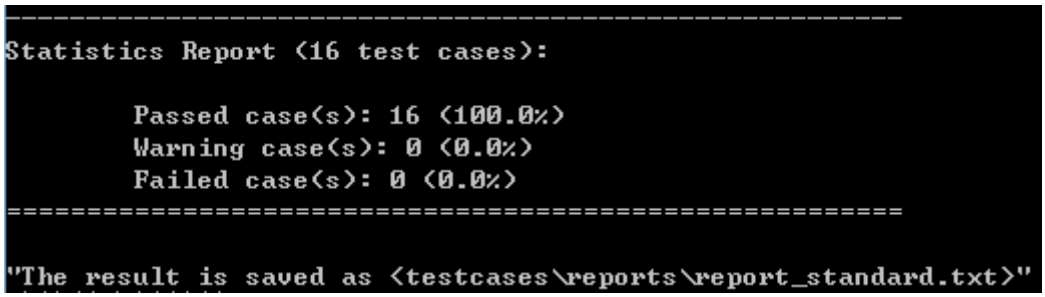


3. 回归测试的运行结果：

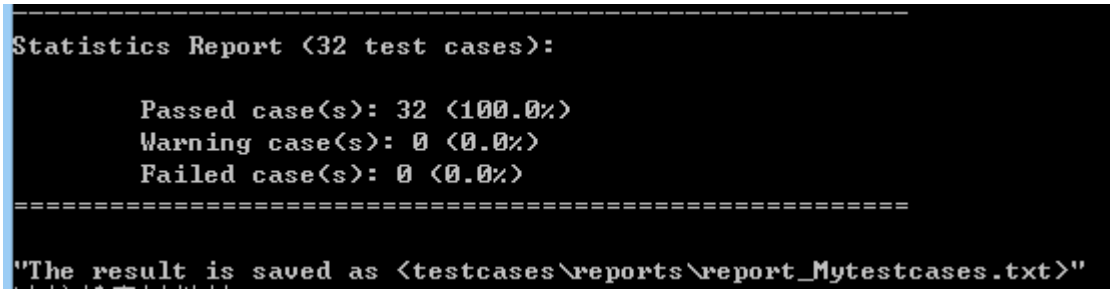
- Simple 回归测试样例的结果



- Standard 回归测试样例的结果：



- 我自己设计的回归测试样例的结果：



回归测试样例文件见 testcases 文件夹下的 Mytestcases.xml

四、 遇到的问题及解决办法

1. 已解决的问题

在实验过程中由于细节出现了很多比较微小的错误，但都在完成整个项目前已经解决，一下列出的问题是进行回归测试的时候发现并解决的问题：

- 处理三元运算符的时候，如果缺少操作数会报出类型不匹配的错误

  原因：处理三元运算符的时候，只管取出操作数栈中的值，然后去到第二个操作数的时候，本来应该是一个数值型的值，但由于缺少操作数取到了本来应该是第一个操作数的布尔型的值，因此出现了类型不匹配。

  解决办法：上面已经提到，对第二个操作数进行类型判断如果是布尔型，那就抛出缺少运算符异常而不是类型错误，如果真的是类型错误能在 OPP 表中通过算符的优先级判断检测出来。

- 只输入多个空格的时候报出错误缺少运算符

  原因：判断是否为空字符串的时候只用了输入字符串长度为零判断

  解决办法：使用 inputString.replaceAll(“ ”, “”)去掉所有的空格后再判断字符串长度是不是为零。

2. 未解决的问题

对于输入 cos(3.14,)， 在文档中是提示输出了缺少操作数异常，但是我的程序中出现了函数形式调用语法错误。出现这个问题的原因是我处理括号函数运算的时候，是默认 cos 只有一个参数，以及对 FunctionCallException 的理解可能不太一样导致的。

五、 实验心得

这次 Project 相对来说比较难，使用的时间也是很久的，但是做完之后自我感觉收获也是很大的。

首先在实验中深入理解了 OPP 算符优先算法，以及对一个编译器进行语法分析和词法分析的过程进行学习，巩固了编译原理理论的知识。还有，对错误检测以及错误恢复的过程有了一定的了解与认识（好像实验中没有错误恢复），在测试方面，使用回归测试的方法，通过编写 xml 文件进行设计测试样例，方便了对后期代码修改之后对程序正确性以及健壮性的检查。

时间不多了(“￣□￣)，希望接下来能够通过实践与理论的相结合的方法，深入学习好编译原理这门课程~

Thanks\(^o^)/~