**Version 1.00; Dec. 20th, 2015**

**编译器实验二**

基于表达式的计算器ExprEval

陈跃东

12330054

中山大学数据科学与计算机学院

目录

[一． 讨论语法定义的二义性 2](#_Toc438392201)

[二． 设计并实现词法分析程序 2](#_Toc438392202)

[1. 如何对单词进行分类 3](#_Toc438392203)

[2. 如何处理对预定义函数名和布尔常量的识别 3](#_Toc438392204)

[3. 如何处理科学计数法表示的数值常量 3](#_Toc438392205)

[4. 如何处理字符串的边界 3](#_Toc438392206)

[三． 构造算符优先关系表 3](#_Toc438392207)

[1. 一元取负和二元减号 3](#_Toc438392208)

[2. 三元运算符和其他操作符 4](#_Toc438392209)

[3. 预定义函数 4](#_Toc438392210)

[四． 设计并构造语法分析和语义处理程序 4](#_Toc438392211)

[1. 如何实现OPP的核心控制程序 4](#_Toc438392212)

[2. 如何实现运算符的各种归约动作 5](#_Toc438392213)

[3. 如何对语义进行处理 5](#_Toc438392214)

[五． 测试实验结果 5](#_Toc438392215)

[1. 手工输入测试 6](#_Toc438392216)

[2. 回归测试 6](#_Toc438392217)

[六． 实验心得体会 7](#_Toc438392218)

# 讨论语法定义的二义性

该BNF是存在二义性的。

证明：

该文法允许句子 decimal+decimal\*decimal存在两个最左推导，如下

|  |
| --- |
| Expr ->ArithExpr  ->ArithExpr + ArithExpr  ->decimal + ArithExpr  ->decimal + ArithExpr \* ArithExpr  ->decimal + decimal \* ArithExpr  ->decimal + decimal \* decimal |

或者是

|  |
| --- |
| Expr ->ArithExpr  ->ArithExpr \* ArithExpr  ->ArithExpr + ArithExpr \* ArithExpr  ->decimal + ArithExpr \* ArithExpr  ->decimal + decimal \* ArithExpr  ->decimal + decimal \* decimal |

如上，同一个句子根据该文法可以得到两棵不通的语法树，可见该BNF存在二义性。

而由实验文档可知，ExprEval通过定义算符（即终结符）的优先级和结合性来解决二义性冲突。

# 设计并实现词法分析程序

本实验中，程序定义了一个主类Token，用来做为其他词法单元的父类。再此基础上，将词法单元细分为五大类。它们的关系如下图：

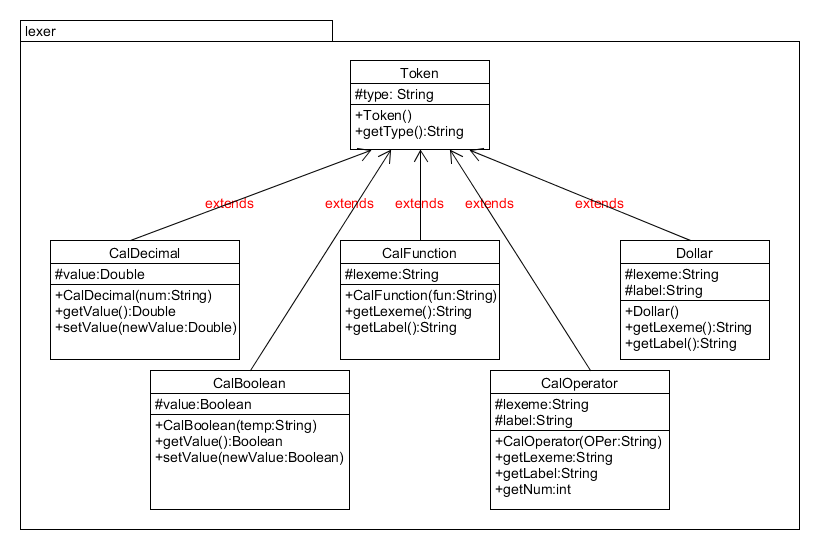
****

Figure 1 lexer的UML图，使用umlet作图

|  |  |
| --- | --- |
| **CalBoolean** | 存储布尔值的词法单元，包括true和false |
| **CalFunction** | 存储函数名的词法单元，包括sin，cos，max和min |
| **CalOperator** | 存储操作符的词法单元，包括算术运算符和关系运算符等所有的操作符 |
| **CalDecimal** | 存储数值常量的词法单元，包括普通十进制数和科学计算法十进制数。 |
| **Dollar** | 存储终结符的词法单元。即$。 |

注：在读入输入的表达式字符串之后，程序会对字符串进行去空格和变为小写的处理，所以以下处理均不再考虑大小写和空格这两个问题。

## 如何对单词进行分类？

具体的分类已经在上述词法单元类的描述中给出。本人将算术运算符和关系运算符划分为同一种词法单元CalOperator，因为他们本质上都是操作符，只是操作数的数量可能存在差异，而那应该是执行语义动作应该解决的问题，此处划分为同一个类比较合理。

## 如何处理对预定义函数名和布尔常量的识别？

在词法分析器lexer中，当程序扫描到字符s，c或者m时，便会向后预读两个字符，以此来判断是否是预定义函数名，如果是，将该字符划分为CalFunction并返回该词法单元，否则，抛出错误。

同理，当程序扫描到字符t（或者f）时，便会向后预读三个（或者四个）字符，以此来判断是否是布尔常量。如果是，将该字符划分为CalBoolean并返回该词法单元，否则，抛出异常。

## 如何处理科学计数法表示的数值常量？

科学计数法和普通数值常量都一样，必定是以数字开头的。所以，当程序读取到数字时，就会向后扫描，通过判断小数点以及E等的关系，直至判断到读完整个数值字符串，就将其新建为一个CalDecimal的词法单元。在CalDecimal类的构造函数中，会根据E的位置，将科学计数法常量划分为数值和指数部分，并将其转化为对应的普通数值常量后存储起来。

## 如何处理字符串的边界？

当程序读取完整个字符串后，再读取下一个词法单元时，便会返回一个Dollar类的词法单元，用来标记以及读完整个表达式字符串。

# 构造算符优先关系表

由于本实验提供的BNF存在二义性，所以必须使用一种算法来使得文法满足无二义性的条件，而按照实验要求，通过定义算符（即终结符）的优先级和结合性来解决二义性冲突。故此步骤要构造算符优先关系表Operator Priority Relation Table（OPP）。

OPP表在此处的具体实现，可以理解为就是一张移入归约表，通过比较两个运算符之间的优先级，来决定是执行移入，还是归约。规则就是，维护一个操作符堆栈，然后：

1. 如果栈顶符号优先级大于读入符号，或者两者优先级相同但符号是左结合的，那么就对栈顶符号进行归约。
2. 如果栈顶符号优先级小于读入符号，或者两者优先级相同但符号是右结合的，那么就将读入符号进行移入。

为了使函数功能单一高效，本程序将归约进一步细分为单目运算，二目运算，三目运算，以及括号匹配这四种。

另外，如果两个符号之间的相遇不符合正常情况，就抛出相应的错误。

敏感关系处理：

## 一元取负和二元减号

在OPP表中，本程序又定义了一些label用来标记不同的操作符，而其中一元取负的标记为‘-’，二元减号和加号合起来标记为‘pm’（意为plus，minus），所以两者在OPP表中是不同的。

实现的方法很简单，就是在词法分析器读取到‘-’时，判断其前一个字符是否为‘）’或者数字，若是，那么它便是减号，设置其label为‘pm’；否则，它便是负号，设置其label为‘-’。

## 三元运算符和其他操作符

三元运算符由于优先级最低，所以，只有在当‘：’遇到‘$’时，才会进行归约，其他时候分情况移入或者抛出异常。

## 预定义函数

预定义函数名之后必须紧跟着括号，所以，只有遇到‘（’才是合法的，将其移入。而遇到其他运算符都应该抛出异常。

# 设计并构造语法分析和语义处理程序

在算符优先分析法中，比较的只是两个操作符之间的关系，不比较操作数。所以，本程序使用一个堆栈来存储操作符，而使用另一个堆栈来存储操作数。

## 如何实现OPP的核心控制程序？

在OPP表中，已经定义了相应的符号遇到另一个符号应该进行移入，归约或者抛错动作，所以，在这一步中，程序会读取符号堆栈栈顶元素的OPP表中的label，以及读入符号的OPP表的label，然后根据两者相遇在OPP表中定义好的语义动作来执行相应的动作。

|  |
| --- |
| lableReadIndex = getIndex(getLabel(curToken));  lableStackIndex = getIndex(getLabel(topToken));  action = OPP.table[lableStackIndex][lableReadIndex];  switch (action) {  case OPP.ACCEPT:  completed = true;  break;  case OPP.SHIFT:  shift(curToken);  curToken = lexer.getNextToken();  break;  case OPP.RDUNAOPER:  unaryReduce();  break;  case OPP.RDBINAOPER:  binaryReduce();  break;  case OPP.RDTRINAOPER:  trinaryReduce();  break;  case OPP.RDMATCH:  matchReduce();  curToken = lexer.getNextToken();  break;  case OPP.ERRLEFTPAR:  throw new MissingLeftParenthesisException();  case OPP.ERRSYN:  throw new SyntacticException();  case OPP.ERROPERAND:  throw new MissingOperandException();  case OPP.ERRTYPE:  throw new TypeMismatchedException();  case OPP.ERRFUNCSYN:  throw new FunctionCallException();  case OPP.ERRRIGHTPAR:  throw new MissingRightParenthesisException();  case OPP.ERRTRINA:  throw new TrinaryOperationException();  default:  break;  } |

Table 1 语法分析器核心代码

如上核心代码所示，函数shift()执行移入操作，函数unaryReduce()执行单目运算归约，binaryReduce()执行双目运算归约，函数trinaryReduce()执行三目运算归约，而函数matchReduce()执行括号匹配以及函数运算归约。其他的不合法的情况则抛出相应的异常。

另外，如果遇到的是十进制数或者布尔值，那么就直接压入操作数堆栈即可。

## 如何实现运算符的各种归约动作？

如上所述，本实验将归约动作细分为单目运算归约，双目运算归约，三目运算归约以及括号匹配及函数运算归约。

对于单目运算归约，程序会从操作数堆栈中pop出来一个操作数词法单元，然后根据从操作符堆栈中pop出来的单目操作符来执行相应的运算，比如负号操作符就是执行用0减去读取出来的操作数的动作，最后将运算得到的结果重新压入操作数堆栈中。而双目运算归约以及三目运算归约也与此相似，不再累赘。

对于括号匹配以及函数运算归约，是在读取到右括号“）”时执行的。它会不断地从操作符堆栈中pop出操作符，然后执行相应的归约动作，直到遇到左括号“（”为止。当pop掉左括号后，会从操作符堆栈中再读取一个操作符，如果该词法单元的类型是函数，那么就执行相应的函数动作，否则结束操作。结果同样也是压入操作数堆栈中。

## 如何对语义进行处理？

本程序在lexer获取到词法单元时，会给每个词法单元一个类型属性，所以此次便可以通过获取词法单元的类型属性来得到它的类型。在执行归约运算时，会根据操作符得知它所需要的操作数的类型，然后比对从操作数堆栈中读取出来的操作数的类型，如果符合就继续执行归约动作，否则，抛出异常。

# 测试实验结果

以下附上程序运行的截图，包括手工输入测试以及回归测试。但由于篇幅有限，不可能给出所有情况的截图，如有必要清自行进行相关测试。

## 手工输入测试

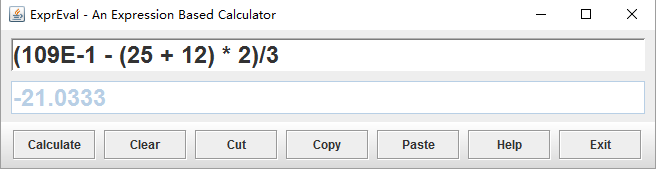


Figure 2 正确输入

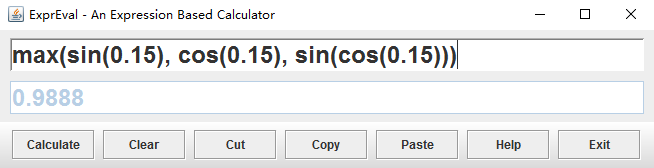


Figure 3 函数运算

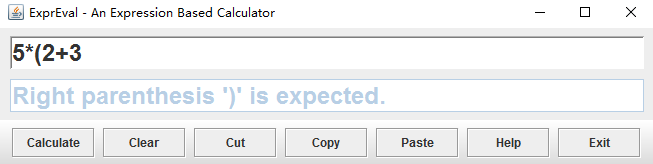


Figure 4 缺少右括号

## 回归测试

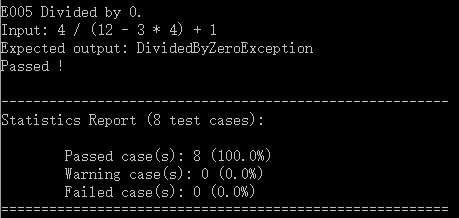


Figure 5 简单回归测试全部通过

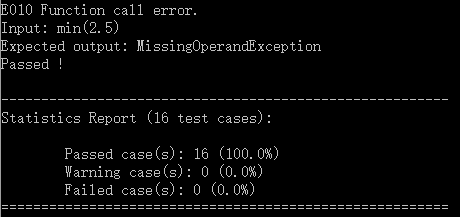


Figure 6 标准回归测试全部通过

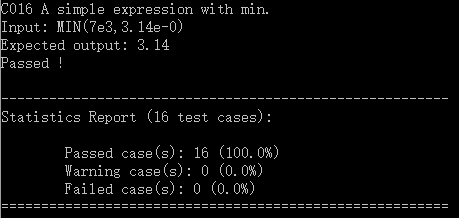


Figure 7 个人回归测试全部通过

# 实验心得体会

本次实验给出的文法具有二义性，咋一看以为是要通过消除左递归来解决二义性问题，但通过阅读实验要求的后半部分可知，实验的目的是通过定义运算符的结合性以及优先级来解决二义性冲突。这样，实现的思路就非常清晰明了，和普通的中缀转后缀问题相似，通过堆栈便可以合理的解决问题，技术难度不大。但由于涉及的运算操作类型相对较多，所以实现起来仍旧颇费功夫。

在词法分析器lexer中，难度最大的就是对十进制数的读取。因为它不仅包含整数，还包括小数以及科学计数法，这样，就要谨慎地判断小数点以及E的数量和位置来决定是否是合法输入等等。而当成功判断某段字符是数值之后，就没有必要再保留其原型了，因为程序后面关心的只是数值本身。所以，在创建数值词法单元时，程序通过调用Double.toParser()函数，并结合E来判断区分数值和指数部分，将字符串转换为对应的数值。这样，便极大地方便了后续语法分析和语义动作的执行。

程序最核心的部分，就是算符优先关系表的构建。应该依据算符优先关系以及相应的移入归约规则，细心地判断两者相遇应该执行什么样的动作。

在parser部分，由于算符优先分析法只是比较两个操作符之间的关系，所以使用两个堆栈来分别维护操作符和操作数，可以节省大量的无用的将操作数移入堆栈与移出堆栈操作，既提高代码可读性，又提高程序执行效率。

最后，再提几点Java代码实现过程中遇到的应该注意的细节：

1. 对于字符串的比较，不要使用“==”，而应该使用函数string.equals()。

“==”比较的是两个字符串对象的引用是否是相同的，而string.equals()比较的才是两个字符串包含的内容是否是一样的。

1. 不要直接比较Double类型等浮点数之间的相等性。

浮点数实现的机制决定了它不是一个固定的数值，可能会有很微小的波动，所以，直接比较两个浮点数之间的相等性，几乎永远是不相等的。而应该通过判断两个浮点数之间的差的绝对值是否小于某个特别小的数值来判断两者是否相等。

1. 关于函数sin()以及cos()的求值问题。

一般来讲，括号中的值的单位应该默认指的度，所以运算时应该先将角度的度数表示转化为相应的弧度表示，再进行求值。在Java中，角度转化为弧度可以通过函数Math.toRadians()来实现。

但从实验软装置提供的回归测试来看，本实验应该默认括号中的值的单位是弧度，所以本程序中就是将括号中的数值直接求sin和cos的。