编译原理实验4

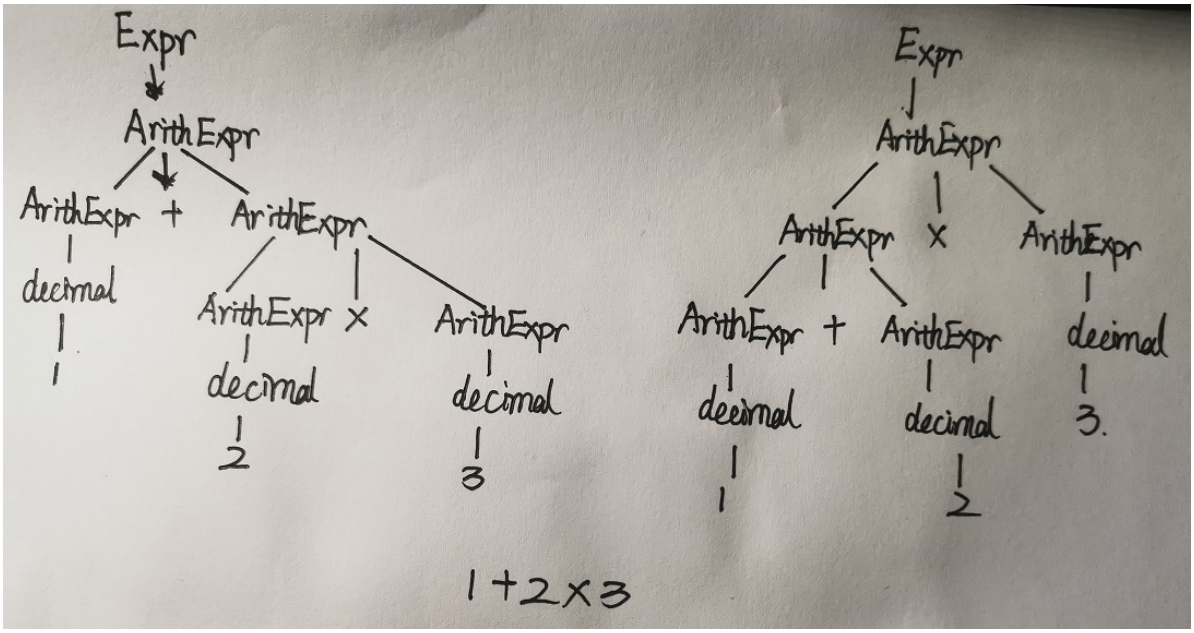
> 19335019 陈泷

1.讨论语法定义的二义性

ExprEval 的表达式规格说明如下列 BNF 所示：

Expr	→	*ArithExpr*
ArithExpr	→	**decimal** | **(** *ArithExpr* **)**
	|	*ArithExpr* **+** *ArithExpr* | *ArithExpr* **–** *ArithExpr*
	|	*ArithExpr* ***** *ArithExpr* | *ArithExpr* **/** *ArithExpr*
	|	*ArithExpr* **^** *ArithExpr*
	|	**–** *ArithExpr*
	|	*BoolExpr* **?** *ArithExpr* **:** *ArithExpr*
	|	*UnaryFunc* | *VariablFunc*
UnaryFunc	→	**sin** **(** *ArithExpr* **)** | **cos** **(** *ArithExpr* **)**
VariablFunc	→	**max** **(** *ArithExpr* **,** *ArithExprList* **)**
	|	**min** **(** *ArithExpr* **,** *ArithExprList* **)**
ArithExprList	→	*ArithExpr* | *ArithExpr* **,** *ArithExprList*
BoolExpr	→	**true** | **false** | **(** *BoolExpr* **)**
	|	*ArithExpr* **>** *ArithExpr*
	|	*ArithExpr* **>=** *ArithExpr*
	|	*ArithExpr* **<** *ArithExpr*
	|	*ArithExpr* **<=** *ArithExpr*
	|	*ArithExpr* **=** *ArithExpr*
	|	*ArithExpr* **<>** *ArithExpr*
	|	*BoolExpr* **&** *BoolExpr*
	|	*BoolExpr* | *BoolExpr*
	|	**!** *BoolExpr*

（1）该语法定义具有二义性。例如对于表达式 `1+2*3` 可以构建出两棵不同的语法树。



（2）这种二义性可以通过构造算符优先关系表进行解决。

2.设计并实现词法分析程序

2.1 如何对单词进行分类

本实验的单词分类如下：

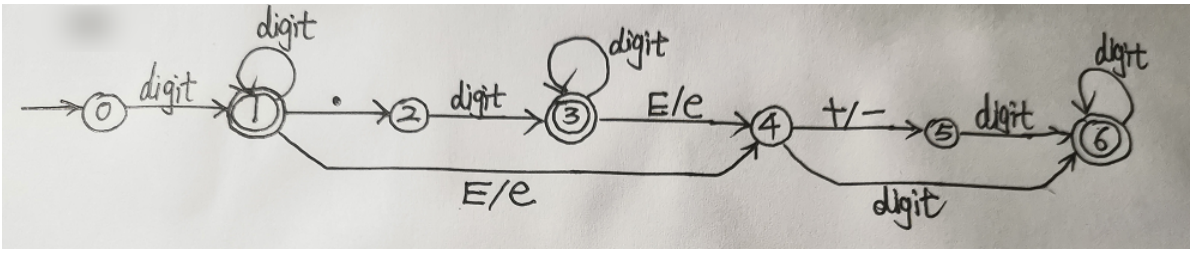
- 数值类型：包含整数、小数、科学计数法表示的数值
- 运算类型：运算符及预定义函数
- 布尔类型
- 辅助符号类型：，和$

2.2 数值类型的处理

数值类型的正则定义式如下：

digit	→	**0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9**
integral	→	*digit* $^+$
fraction	→	**.** *integral*
exponent	→	(**E** | **e**) (**+** | **–** | ε) *integral*
decimal	→	*integral* (*fraction* | ε) (*exponent* | ε)

状态转换图：



2.3 其他类型的处理

根据识别到的token内容，传输给各自对应的处理函数，预定义函数名、布尔常量等其他的数据类型之间不存在二义性。

2.4 如何处理字符串的边界

对于数字类型的字符串，从当前lookahead的位置开始向后读取字符串，直到读取到一个非数字类型的字符串时停止并记录此时的坐标索引，将lookahead开始到当前位置的字串复制并进行数值数据类型的处理。

对于sin、cos、true等类型的字符串，根据其首字母判断它们的类别，从而得知对应的长度，截取相应长度的字串与对应的预期字符串进行匹配，匹配成功即识别成功，否则进行报错。在这一过程中需要考虑索引越界的问题。

对于其他类型的字符串，由于都是当个字符，所以直接进行处理。

3.构造算符优先关系表

各运算符的优先级与结合性质定义如下：

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

3.1 算符优先关系表

	num	bool	(	)	func	-	^	*/	+-	Relation	!	&	|	?	:	,	$
num	Miss Operator	Miss Operator	Miss Operator	reduce	reduce	reduce	reduce	reduce	reduce	reduce	Type Mismatch	reduce	reduce	reduce	reduce	reduce	reduce
bool	Miss Operator	Miss Operator	Miss Operator	reduce	Miss Operator	Type Mismatch	Type Mismatch	Type Mismatch	Type Mismatch	Type Mismatch	Miss Operator	reduce	reduce	reduce	reduce	Type Mismatch	reduce
(	shift	shift	shift	shift	shift	shift	shift	shift	shift	shift	shift	shift	shift	shift	shift	shift	Miss Right Parenthesis
)	Miss Operator	Miss Operator	Miss Operator	reduce	Miss Operator	reduce	reduce	reduce	reduce	reduce	Miss Operator	reduce	reduce	reduce	reduce	reduce	reduce
func	Miss Left Parenthesis	Miss Left Parenthesis	shift	reduce	Miss Left Parenthesis	Miss Left Parenthesis	Miss Left Parenthesis	Miss Left Parenthesis	Miss Left Parenthesis	Miss Left Parenthesis	Miss Left Parenthesis	Miss Left Parenthesis	Miss Left Parenthesis	Miss Left Parenthesis	reduce	Miss Left Parenthesis	Miss Left Parenthesis
-	shift	Type Mismatch	shift	reduce	shift	shift	reduce	reduce	reduce	reduce	Type Mismatch	Type Mismatch	Type Mismatch	Type Mismatch	reduce	reduce	reduce
^	shift	Type Mismatch	shift	reduce	shift	shift	shift	reduce	reduce	reduce	Type Mismatch	Type Mismatch	Type Mismatch	Type Mismatch	reduce	reduce	reduce
*/	shift	Type Mismatch	shift	reduce	shift	shift	shift	reduce	reduce	reduce	Type Mismatch	Type Mismatch	Type Mismatch	Type Mismatch	reduce	reduce	reduce
+-	shift	Type Mismatch	shift	reduce	shift	shift	shift	shift	reduce	reduce	Type Mismatch	Type Mismatch	Type Mismatch	Type Mismatch	reduce	reduce	reduce
Relation	shift	Type Mismatch	shift	reduce	shift	shift	shift	shift	shift	reduce	reduce	reduce	reduce	reduce	reduce	Type Mismatch	reduce
!	shift	shift	shift	reduce	Type Mismatch	Type Mismatch	Type Mismatch	Type Mismatch	Type Mismatch	shift	shift	reduce	reduce	reduce	reduce	Type Mismatch	reduce
&	shift	shift	shift	reduce	Type Mismatch	Type Mismatch	Type Mismatch	Type Mismatch	Type Mismatch	shift	shift	reduce	reduce	reduce	reduce	Type Mismatch	reduce
|	shift	shift	shift	reduce	Type Mismatch	Type Mismatch	Type Mismatch	Type Mismatch	Type Mismatch	shift	shift	shift	reduce	reduce	reduce	Type Mismatch	reduce
?	shift	Trinary Operation Exception	shift	Miss Operator	shift	shift	shift	shift	shift	shift	Trinary Operation Exception	Trinary Operation Exception	Trinary Operation Exception	shift	shift	Trinary Operation Exception	Trinary Operation Exception
:	shift	Trinary Operation Exception	shift	reduce	shift	shift	shift	shift	shift	shift	Trinary Operation Exception	Trinary Operation Exception	Trinary Operation Exception	reduce	reduce	reduce	reduce
,	shift	Type Mismatch	shift	shift	shift	shift	shift	shift	shift	Type Mismatch	Type Mismatch	Type Mismatch	Type Mismatch	shift	shift	shift	Miss Right Parenthesis
$	shift	shift	shift	Miss Left Parenthesis	shift	shift	shift	shift	shift	shift	shift	shift	shift	shift	Trinary Operation Exception	Miss Left Parenthesis	accept

3.2 敏感关系的处理

3.2.1 如何处理表达式中两个重载（Overloading）的运算符：一元取负运算符“-”和二元减法运算符“-”？

> 在词法分析的过程中，根据前一个token的类型判断”-“所对应的运算符：如果”-“的前一个为`数值类型`或`)`则该”-“为二元减法运算符，否则为一元取负运算符。

3.2.2 如何处理三元运算符与其他运算符之间的关系、预定义函数与其他运算符之间的关系？

> 如3.1中的算符优先关系表所示，只要根据表格中相应的操作进行处理。

4.设计并实现语法分析和语义处理程序

4.1 如何实现 OPP 的核心控制程序， 请以伪码或 Java 代码给出核心的算法

将算符优先关系表中各种操作进行编号：0代表shift，1代表reduce，2代表accept，其余不同的负值代表不同类型错误。每次读取一个lookahead与当前操作符栈顶的操作符，通过查表获取当前应执行的操作，从而调用对应函数执行相关操作。

Java代码如下所示：

```
1    /**
```

```
     * 进行parse动作的函数,由calculator进行调用
     * 当值为0时代表shift，1代表reduce，2代表accept，其余不同的负值代表不同类型错误
     *
     * @return 计算结果
     * @throws ExpressionException 不同错误的集合
     */
    public double parse() throws ExpressionException {
        lookahead = new Terminal(scanner.getNextToken());
        while (true) {
            Expr topMostTerminal = stack.get(stack.size() - 1);  // 获取操作符栈顶的操作符
            /* 查询算符优先关系表，执行相应的操作 */
            switch (OPPTable.table[topMostTerminal.getTag()][lookahead.getTag()]) {
                case 0:
                    shift();
                    break;
                case 1:
                    reduce();
                    break;
                case 2:
                    return accept();
                case -1:
                    throw new MissingOperatorException();
                case -2:
                    throw new MissingRightParenthesisException();
                case -3:
                    throw new MissingLeftParenthesisException();
                case -4:
                    throw new FunctionCallException();
                case -5:
                    throw new TypeMismatchedException();
                case -6:
                    throw new MissingOperandException();
                case -7:
                    throw new TrinaryOperationException();
            }
        }
    }
```

对于各个操作的说明：

- reduce( )： 见 `4.2`
- shift( )：直接将新的lookahead加入操作符栈内
- accept( )：对于算术表达式，直接返回操作数栈（辅助栈）内的操作数即为最终的运算结果；对于布尔表达式，则按照实验要求，在屏幕显示判断结果，并报出 `TypeMismatchedException( )` 的错误。

4.2 如何实现各种运算符的归约（Reduce）动作

对于操作符栈内的各个操作符，在scanner形成token时都会对其归属的类别进行标注。因此在reduce时，通过获取操作符的标签即可知道其类别，从而调用对应的归约函数。另外，不同的运算符需要不同个数的操作数，这些区别在其对应的归约函数中实现。总之，每次需要操作数时都直接从操作数栈（辅助栈）中直接获取对应数目的操作数，如果不能满足需求则进行报错。

Java代码如下所示：

```
private void reduce() throws ExpressionException {
        Expr topMostTerminal = stack.get(stack.size() - 1);  // 获取操作符栈顶的操作符
        switch (topMostTerminal.getTag()) {                  // 根据该操作符的类型执行对应的规约操作
            case typeofToken.NUM:
                reducer.numReducer(stack,auxiliaryStack);
                break;
            case typeofToken.BOOL:
                reducer.boolReducer(stack,auxiliaryStack);
                break;
            case typeofToken.ADDSUB:
                reducer.add_subReducer(stack,auxiliaryStack);
                break;
            case typeofToken.MULDIV:
                reducer.mul_divReducer(stack,auxiliaryStack);
                break;
            case typeofToken.NOT:
                reducer.notReducer(stack,auxiliaryStack);
                break;
            case typeofToken.NEG:
                reducer.negReducer(stack,auxiliaryStack);
                break;
            case typeofToken.POWER:
                reducer.powReducer(stack,auxiliaryStack);
                break;
            case typeofToken.AND:
            case typeofToken.OR:
                reducer.and_orReducer(stack,auxiliaryStack);
                break;
            case typeofToken.RE:
                reducer.relationReducer(stack,auxiliaryStack);
                break;
            case typeofToken.COLON:
                reducer.colonReducer(stack,auxiliaryStack);
                break;
            case typeofToken.RP:
                reducer.rpReducer(stack,auxiliaryStack);
                break;
            default:
                System.out.println(topMostTerminal.getTag());
                break;
        }
    }
```

4.3 如何对语义进行处理， 主要是类型的兼容性检查与类型的推导

兼容性检查：执行reduce时，根据操作符的标签即可知道其类别，从而调用reducer中对应的归约函数进行归约，因此在进行归约时已经完成了类型的兼容性检查。

类型的推导：类型的推导主要根据BNF进行，每个操作数都有对应的标签，记录经过归约后该操作数的当前类型（如，ArithExpr等），因此根据该标签以及BNF的内容，即可进行类型的推导。

语义处理：语义的处理主要在reducer类中完成。在进行语义处理时，根据推算各个操作符此时在栈内对应的位置，同时可以预测对应的操作数的类型。如果调取相关数据后符合预期，则进行语义处理；否则根据出错的内容抛出对应的错误类型。类型的推导主要根据BNF进行，每个操作数都有对应的

标签，记录经过归约后该操作数的当前类型（如，ArithExpr等），因此根据该标签以及BNF的内容，即可进行类型的推导。

5.实验结果

5.1 test_simple.bat测试结果

```
Testing Procedure
=====================================================

C001 A simple expression.
Input: 9 - 3 * 2
Expected output: 3
Passed !

C002 Expression with arithmetic operations.
Input: 2.25E+2 - (55.5 + 4 * (10 / 2) ^ 2)
Expected output: 69.5
Passed !

C003 Expression with arithmetic operations.
Input: 65 / 5 - 130e-1
Expected output: 0.0
Passed !

E001 Left parenthesis expected.
Input: (2 + 3) ^ 3) - ((1 + 1)
Expected output: MissingLeftParenthesisException
Passed !

E002 Right parenthesis expected.
Input: ((2 + 3) ^ ((3 - 1) + 1)
Expected output: MissingRightParenthesisException
Passed !

E003 Operator expected.
Input: (1 + 2) ^ (3 - 4)  5
Expected output: MissingOperatorException
Passed !

E004 Operand expected.
Input: (1 + 2) ^ (3 - ) + 5
Expected output: MissingOperandException
Passed !

E005 Divided by 0.
Input: 4 / (12 - 3 * 4) + 1
Expected output: DividedByZeroException
Passed !

-----------------------------------------------------
Statistics Report (8 test cases):

        Passed case(s): 8 (100.0%)
        Warning case(s): 0 (0.0%)
        Failed case(s): 0 (0.0%)
```

5.2 test_standard.bat测试结果

```
Testing Procedure
========================================================

C001 A simple expression.
Input: 9 - 3 * 2
Expected output: 3
Passed !

C002 Expression with arithmetic operations.
Input: 2.25E+2 - (55.5 + 4 * (10 / 2) ^ 2)
Expected output: 69.5
Passed !

C003 Expression with arithmetic operations.
Input: 65 / 5 - 130e-1
Expected output: 0.0
Passed !

C004 Expression with relational and logical operations.
Input: (5 > 3) & (4 < 8) ? 15 : 16
Expected output: 15
Passed !

C005 Predefined functions.
Input: max(sin(0.15), cos(0.15), sin(cos(0.15)))
Expected output: 0.98877
Passed !

C006 Predefined functions.
Input: sin(min(12, 3 * 5, 2 + 3 ^ 2, 3.14E2))
Expected output: -0.99999
Passed !

E001 Left parenthesis expected.
Input: (2 + 3) ^ 3) - ((1 + 1)
Expected output: MissingLeftParenthesisException
Passed !

E002 Right parenthesis expected.
Input: ((2 + 3) ^ ((3 - 1) + 1)
Expected output: MissingRightParenthesisException
Passed !

E003 Operator expected.
Input: (1 + 2) ^ (3 - 4)  5
Expected output: MissingOperatorException
Passed !

E004 Operand expected.
Input: (1 + 2) ^ (3 - ) + 5
Expected output: MissingOperandException
Passed !

E005 Divided by 0.
Input: 4 / (12 - 3 * 4) + 1
Expected output: DividedByZeroException
Passed !

E006 Type mismatched.
Input: (13 < 2 * 5) + 12
Expected output: TypeMismatchedException
Passed !

E007 Scientific Notation Error.
Input: 4 + 10.E+5 + 1
Expected output: IllegalDecimalException
Passed !

E008 Not a predefined identifier.
Input: 4 + mix(5, 2) + 1
Expected output: IllegalIdentifierException
Passed !

E009 Function call error.
Input: sin(2, 1)
Expected output: FunctionCallException
Passed !

E010 Function call error.
Input: min(2.5)
Expected output: MissingOperandException
Passed !

--------------------------------------------------------
Statistics Report (16 test cases):

        Passed case(s): 16 (100.0%)
        Warning case(s): 0 (0.0%)
        Failed case(s): 0 (0.0%)
========================================================
```

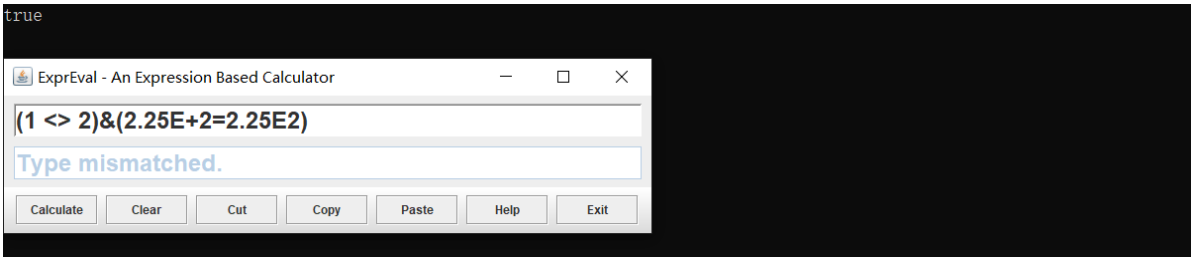
5.3 自定义测试

```
Testing Procedure
=========================================================

C001 Expression with relational and logical operations.
Input:  (1 <> 2)&(2.25E+2=2.25E2)? 1:0
Expected output: 1
Passed !

C002 Expression with relational and logical operations.
Input:  (!(4>=4))|(!(-4<=-4))? 1:0
Expected output: 0
Passed !

E001 IllegalSymbolException
Input:  ((2 + 3) # ({3 @ 1} + 1)
Expected output: IllegalSymbolException
Passed !

E002 Empty Expression Exception.
Input:
Expected output: EmptyExpressionException
Passed !

E003 Illegal Identifier Exception
Input: s*d/d-a+cos(s,c)
Expected output: IllegalIdentifierException
Passed !

---------------------------------------------------------
Statistics Report (5 test cases):

        Passed case(s): 5 (100.0%)
        Warning case(s): 0 (0.0%)
        Failed case(s): 0 (0.0%)
=========================================================

请按任意键继续. . .
```

根据前两个测试中未涉及的测试内容进行补充的自定义测试，主要涉及布尔表达式、关系运算、逻辑运算、取负运算、科学计数法等。另外对于未涉及的错误类型也进行补充测试，测试的具体内容如上所示。

5.4 补充测试

补充测试主要是因为由于原程序的限制，对布尔表达式的结果返回是type mismatched，因此在进行设计时将判断结果返回到命令窗口中，测试结果如下，可以看到符合预期。



结论：可以看到，各项测试内容都成功通过，达到实验预期。

6.实验感想

本次实验对OPP的具体过程进行了实现，从词法分析到语法分析再到语义分析，对于编译器的完整实现过程有了深刻的理解。另外学习了XML测试，并对实验软装置内testcase未测试内容进行了补充测试，进一步检验了程序的鲁棒性。虽然耗费了大量的时间，但是整体上还是比较有收获的。