编译器构造实验

Lab4

基于表达式的计算器ExprEval

姓名：郝裕玮

班级：计科1班

学号：18329015

目录

[1 实验环境 3](#_Toc103182670)

[1.1 JDK版本 3](#_Toc103182671)

[1.2 开发环境 3](#_Toc103182672)

[2 实验过程&结果展示 3](#_Toc103182673)

[2.1 讨论语法定义的二义性 3](#_Toc103182674)

[2.2 设计并实现词法分析程序 4](#_Toc103182675)

[2.3 构造算符优先关系表 8](#_Toc103182676)

[2.4 设计并实现语法分析和语义处理程序 9](#_Toc103182677)

[2.5 测试你的实验结果 12](#_Toc103182678)

[3 实验心得 12](#_Toc103182679)

# 1 实验环境

## 1.1 JDK版本

JDK版本：11.0.14

## 1.2 开发环境

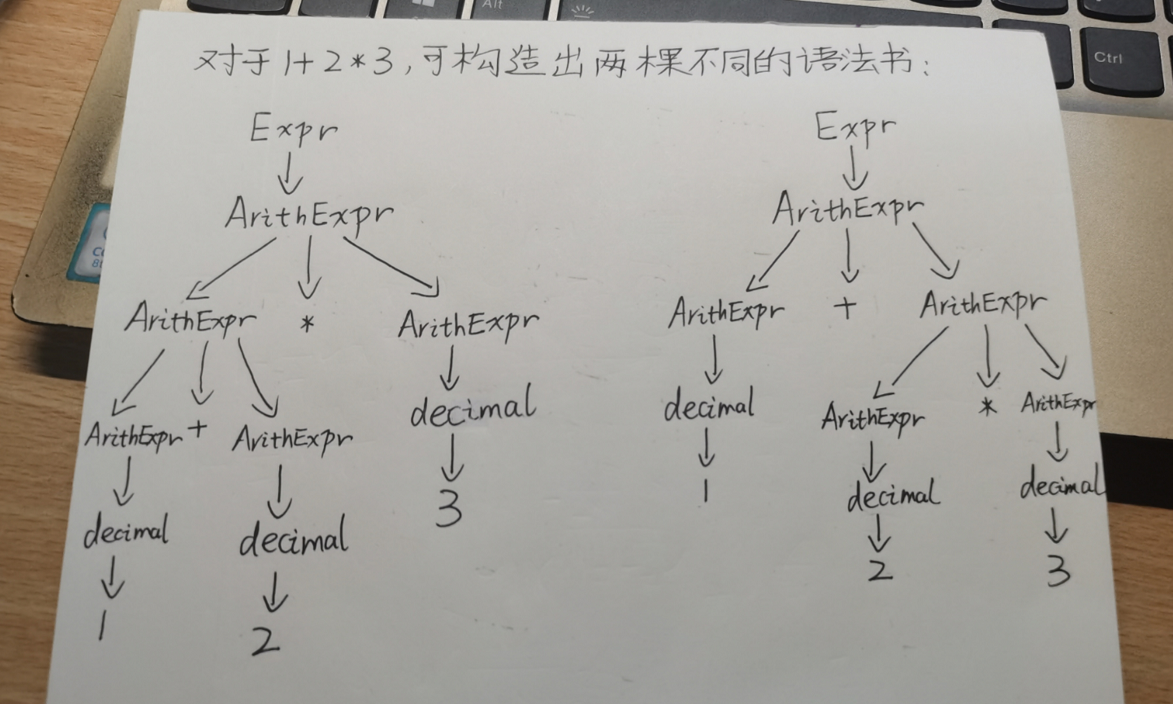
开发工具：Eclipse

# 2 实验过程&结果展示

## 2.1 讨论语法定义的二义性



该语法显然具有二义性，反例如下（见下页）：

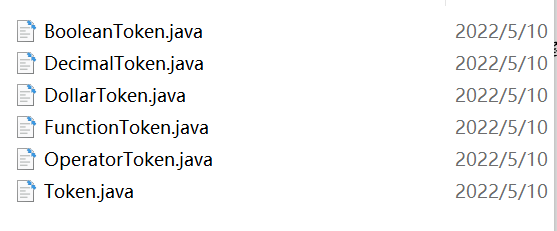


## 2.2 设计并实现词法分析程序

该部分程序共有2部分：词法单元分类，根据状态转换图DFA将表达式转换为词法单元流。

1，词法单元分类：

该部分主要由Token父类和其各种继承子类完成：



（1）Token：父类；

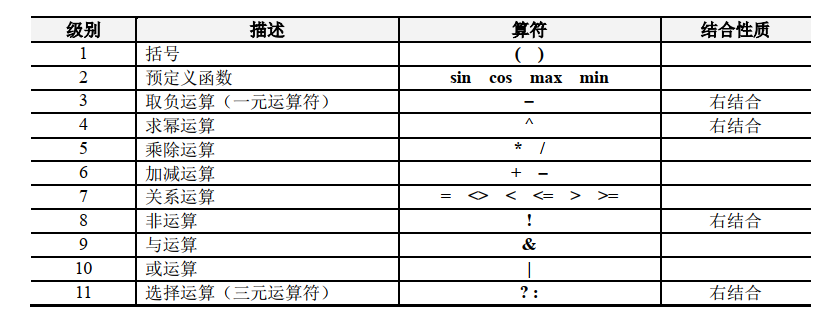
（2）BooleanToken：布尔类型的词法单元，类型为Boolean，值为true或false；

（3）DecimalToken：十进制数的词法单元，类型为Decimal，值为double类型的数值；

（4）DollarToken：终止符$的词法单元，类型为Dollar，值为$；

（5）FunctionToken： 预定义函数的词法单元，类型为Function，值为sin，cos，max，min这4个预定义函数string字符串；

（6）OperatorToken：操作符的词法单元，类型为Operator，包含的运算符如下图所示：



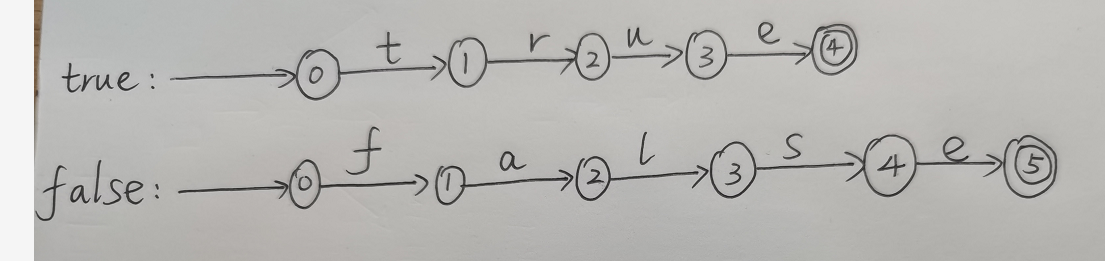
2，根据状态转换图将表达式转换为词法单元流

该部分功能主要由Scanner类实现（并借助了Token父类及其子类实现的词法单元分类功能）

（1）Boolean布尔类型的DFA转换图（见下页）

调用函数为：

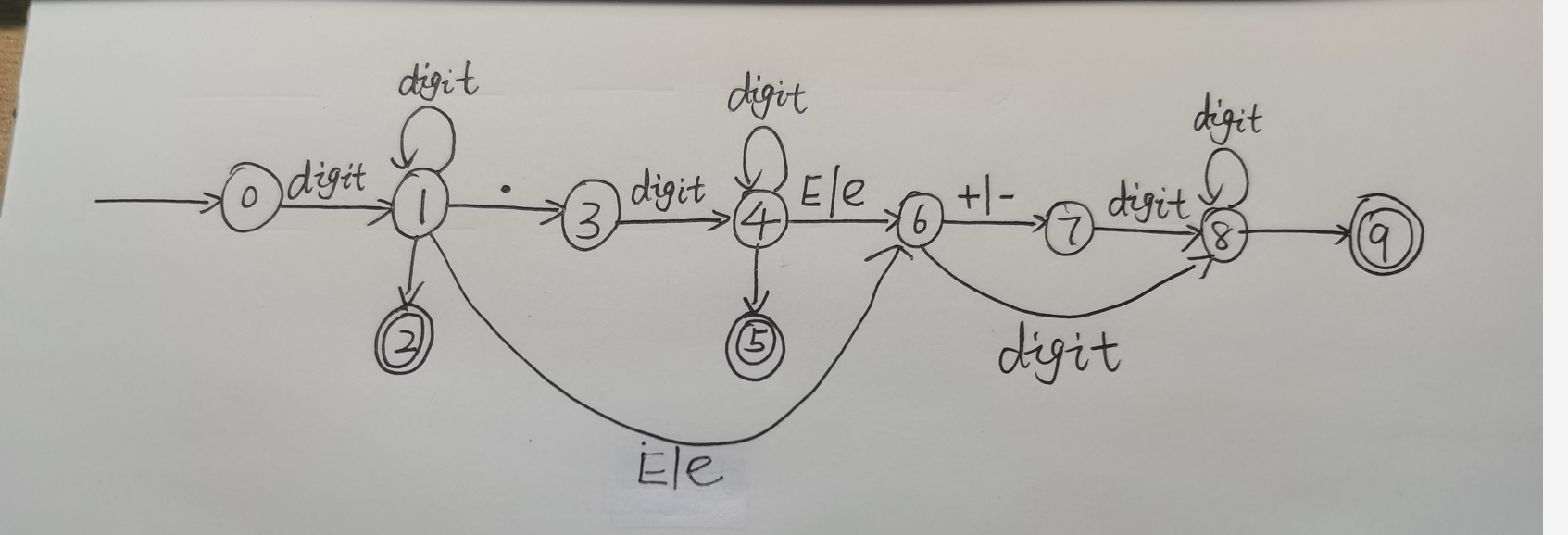
public Token booleanDFA(Character curChar)



（2）Decimal十进制数值的DFA转换图：

调用函数为：

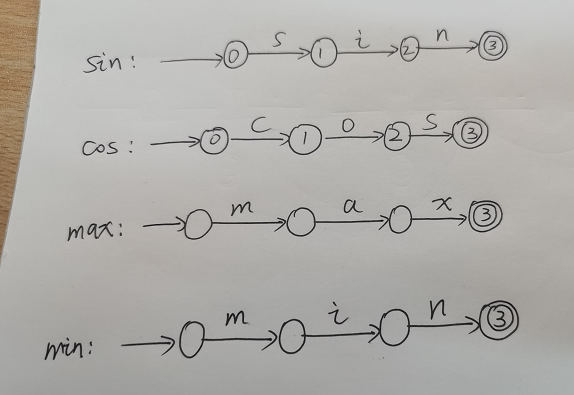
public Token decimalDFA()



（3）Function预定义函数的DFA转换图：

调用函数为：

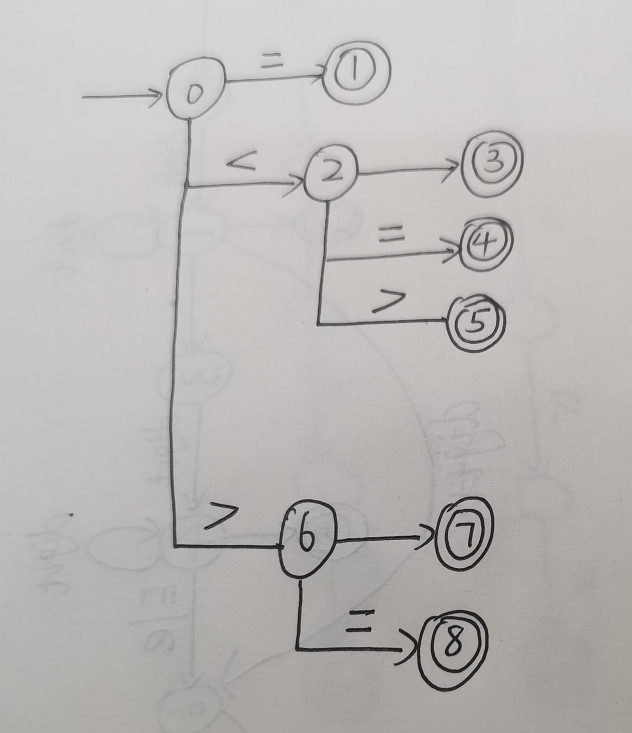
public Token functionDFA()



（4）操作符Operator的DFA转换图：

调用函数为：

public Token functionDFA()



词法分析的流程总结（需要Scanner类和Token类以及Token的子类共同作用）：先在构造函数中对输入字符串去除空格，再调用Scanner类中的getNextToken函数扫描字符串，并调用相应的DFA方法创建对应类型的词法单元（Token）。

## 2.3 构造算符优先关系表

表格如下述代码所示（具体分析已包含在代码注释中）：

package parser;

public class OPP {

    public static final int MISSING\_LEFT\_BRACKET = -7;  // 缺少左括号

    public static final int SYNTACTIC\_EXCEPTION = -6;   // 语法错误

    public static final int MISSING\_OPERAND = -5;   // 缺少操作数

    public static final int TYPE\_ERROR = -4;    // 类型错误

    public static final int FUNCTION\_ERROR = -3;    // 函数语法错误

    public static final int MISSING\_RIGHT\_BRACKET = -2; // 缺少右括号

    public static final int TRINARY\_OPERATION\_ERROR = -1;   // 三元运算符异常

    public static final int SHIFT = 0;  // 移入

    public static final int RD\_UNIQUE\_OPERATION = 1;    // 单元运算

    public static final int RD\_BINARY\_OPERATION = 2;    // 双元运算

    public static final int RD\_TRINARY\_OPERATION = 3;   // 三元运算

    public static final int RD\_BRACKET = 4; // 括号运算

    public static final int ACCEPT = 5; // 接受

    // md: 乘除运算 "multiple & divide"

    // pm: 加减运算 "plus & minus"

    // cmp: 关系运算 "compare"

    public static final int table[][] = {

           /\* (  ) fun -  ^ md pm cmp !  &  |  ?  :  ,  $ \*/

    /\*(\*/    {0, 4, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,-1, 0,-2},

    /\*)\*/    {4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4},

    /\*fun\*/  {0,-3,-3,-3,-3,-3,-3,-3,-3,-3,-3,-3,-3,-3,-3},

    /\*-\*/    {0, 4, 0, 0, 1, 1, 1, 1,-6,-4,-4, 1, 1, 1, 1},

    /\*^\*/    {0, 4, 0, 0, 0, 2, 2, 2,-6,-4,-4, 2, 2, 2, 2},

    /\*md\*/   {0, 4, 0, 0, 0, 2, 2, 2,-6,-4,-4, 2, 2, 2, 2},

    /\*pm\*/   {0, 4, 0, 0, 0, 0, 2, 2,-6,-4,-4, 2, 2, 2, 2},

    /\*cmp\*/  {0, 4, 0, 0, 0, 0, 0,-4,-6, 2, 2, 2,-1,-3, 2},

    /\*!\*/    {0, 4,-4,-4,-4,-4,-4, 0, 0, 1, 1, 1,-1,-3, 1},

    /\*&\*/    {0, 4,-4,-4,-4,-4,-4, 0, 0, 2, 2, 2,-1,-3, 2},

    /\*|\*/    {0, 4,-4,-4,-4,-4,-4, 0, 0, 0, 2, 2,-1,-3, 2},

    /\*?\*/    {0,-1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,-1,-1},

    /\*:\*/    {0, 4, 0, 0, 0, 0, 0,-1,-1,-1,-1, 0,-1,-1, 3},

    /\*,\*/    {0, 4, 0, 0, 0, 0, 0,-3,-3,-3,-3, 0,-1, 0,-3},

    /\*$\*/    {0,-7, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,-1,-3, 5}

    };

}

## 2.4 设计并实现语法分析和语义处理程序

该部分主要由Parser类实现：

重要参数如下：

    private final String[] TAG = {"(", ")", "func", "-", "^", "md", "pm", "cmp", "!", "&", "|", "?", ":", ",", "$"};

    private Stack<Token> operator = new Stack<Token>();

    private Stack<Token> operand = new Stack<Token>();

    private Token curToken = new Token();

    private Token topToken = new Token();

（1）String[] TAG 数组：运算符优先关系表的索引

（2）Stack<Token> operator ：用于存储运算符词法单元的栈

（3）Stack<Token> operand ：用于存储运算量词法单元的栈

（4）Token curToken ：当前扫描获取到的词法单元

（5）Token topToken ：当前运算符词法单元栈的栈顶词法单元

语法分析和语义处理过程主要是在 Parser 类中的 parsing(String expression) 函数中完成：

（1）根据输入字符串新建一个 Scanner 类对象；

（2）调用getNextToken() 函数获取 Token 对象，若Token对象的类型为Boolean或Decimal，则压入 operand操作量栈中，然后再次调用getNextToken() 函数获取下一个Token。若不是Boolean或Decimal，则根据curToken和topToken的tag，索引到OPP表，并根据表值执行相关动作：

public Double parsing(String expression)函数部分内容：

            else {

                lableReadIndex = getIndex(getTag(curToken));

                lableStackIndex = getIndex(getTag(topToken));

                action = OPP.table[lableStackIndex][lableReadIndex];

                switch (action) {

                    case OPP.ACCEPT:

                        completed = true;

                        break;

                    case OPP.SHIFT:

                        shift(curToken);

                        curToken = scanner.getNextToken();

                        break;

                    case OPP.RD\_UNIQUE\_OPERATION:

                        reduceUnary();

                        break;

                    case OPP.RD\_BINARY\_OPERATION:

                        reduceBinary();

                        break;

                    case OPP.RD\_TRINARY\_OPERATION:

                        reduceTrinary();

                        break;

                    case OPP.RD\_BRACKET:

                        matchReduce();

                        curToken = scanner.getNextToken();

                        break;

                    case OPP.MISSING\_LEFT\_BRACKET:

                        throw new MissingLeftParenthesisException();

                    case OPP.SYNTACTIC\_EXCEPTION:

                        throw new SyntacticException();

                    case OPP.MISSING\_OPERAND:

                        throw new MissingOperandException();

                    case OPP.TYPE\_ERROR:

                        throw new TypeMismatchedException();

                    case OPP.FUNCTION\_ERROR:

                        throw new FunctionCallException();

                    case OPP.MISSING\_RIGHT\_BRACKET:

                        throw new MissingRightParenthesisException();

                    case OPP.TRINARY\_OPERATION\_ERROR:

                        throw new TrinaryOperationException();

                    default:

                        break;

                }

            }

OPP.java（包含OPP表）：

public class OPP {

    public static final int MISSING\_LEFT\_BRACKET = -7;  // 缺少左括号

    public static final int SYNTACTIC\_EXCEPTION = -6;   // 语法错误

    public static final int MISSING\_OPERAND = -5;   // 缺少操作数

    public static final int TYPE\_ERROR = -4;    // 类型错误

    public static final int FUNCTION\_ERROR = -3;    // 函数语法错误

    public static final int MISSING\_RIGHT\_BRACKET = -2; // 缺少右括号

    public static final int TRINARY\_OPERATION\_ERROR = -1;   // 三元运算符异常

    public static final int SHIFT = 0;  // 移入

    public static final int RD\_UNIQUE\_OPERATION = 1;    // 单元运算

    public static final int RD\_BINARY\_OPERATION = 2;    // 双元运算

    public static final int RD\_TRINARY\_OPERATION = 3;   // 三元运算

    public static final int RD\_BRACKET = 4; // 括号运算

    public static final int ACCEPT = 5; // 接受

    // md: 乘除运算 "multiple & divide"

    // pm: 加减运算 "plus & minus"

    // cmp: 关系运算 "compare"

    public static final int table[][] = {

           /\* (  ) fun -  ^ md pm cmp !  &  |  ?  :  ,  $ \*/

    /\*(\*/    {0, 4, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,-1, 0,-2},

    /\*)\*/    {4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4},

    /\*fun\*/  {0,-3,-3,-3,-3,-3,-3,-3,-3,-3,-3,-3,-3,-3,-3},

    /\*-\*/    {0, 4, 0, 0, 1, 1, 1, 1,-6,-4,-4, 1, 1, 1, 1},

    /\*^\*/    {0, 4, 0, 0, 0, 2, 2, 2,-6,-4,-4, 2, 2, 2, 2},

    /\*md\*/   {0, 4, 0, 0, 0, 2, 2, 2,-6,-4,-4, 2, 2, 2, 2},

    /\*pm\*/   {0, 4, 0, 0, 0, 0, 2, 2,-6,-4,-4, 2, 2, 2, 2},

    /\*cmp\*/  {0, 4, 0, 0, 0, 0, 0,-4,-6, 2, 2, 2,-1,-3, 2},

    /\*!\*/    {0, 4,-4,-4,-4,-4,-4, 0, 0, 1, 1, 1,-1,-3, 1},

    /\*&\*/    {0, 4,-4,-4,-4,-4,-4, 0, 0, 2, 2, 2,-1,-3, 2},

    /\*|\*/    {0, 4,-4,-4,-4,-4,-4, 0, 0, 0, 2, 2,-1,-3, 2},

    /\*?\*/    {0,-1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,-1,-1},

    /\*:\*/    {0, 4, 0, 0, 0, 0, 0,-1,-1,-1,-1, 0,-1,-1, 3},

    /\*,\*/    {0, 4, 0, 0, 0, 0, 0,-3,-3,-3,-3, 0,-1, 0,-3},

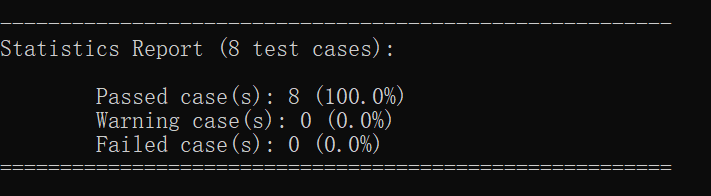
    /\*$\*/    {0,-7, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,-1,-3, 5}

    };

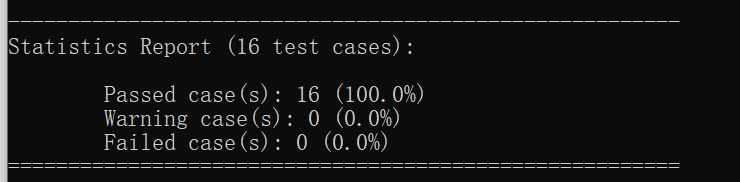
}

## 2.5 测试你的实验结果

test\_simple：



test\_standard：



# 3 实验心得

本次实验让我对编译原理中的词法分析、语法分析、语义分析等重要环节有了更深层次的理解。学会了自己手动创建算符优先关系表，并且根据词法规则定义来构造一个词法扫描程序的有限状态自动机。