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
1.1 JDK 版本.....3
1.2 开发环境.....3
2 实验过程&结果展示.....3
2.1 讨论语法定义的二义性.....3
2.2 设计并实现词法分析程序.....4
2.3 构造算符优先关系表.....8
2.4 设计并实现语法分析和语义处理程序.....9
2.5 测试你的实验结果.....12
3 实验心得.....12

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

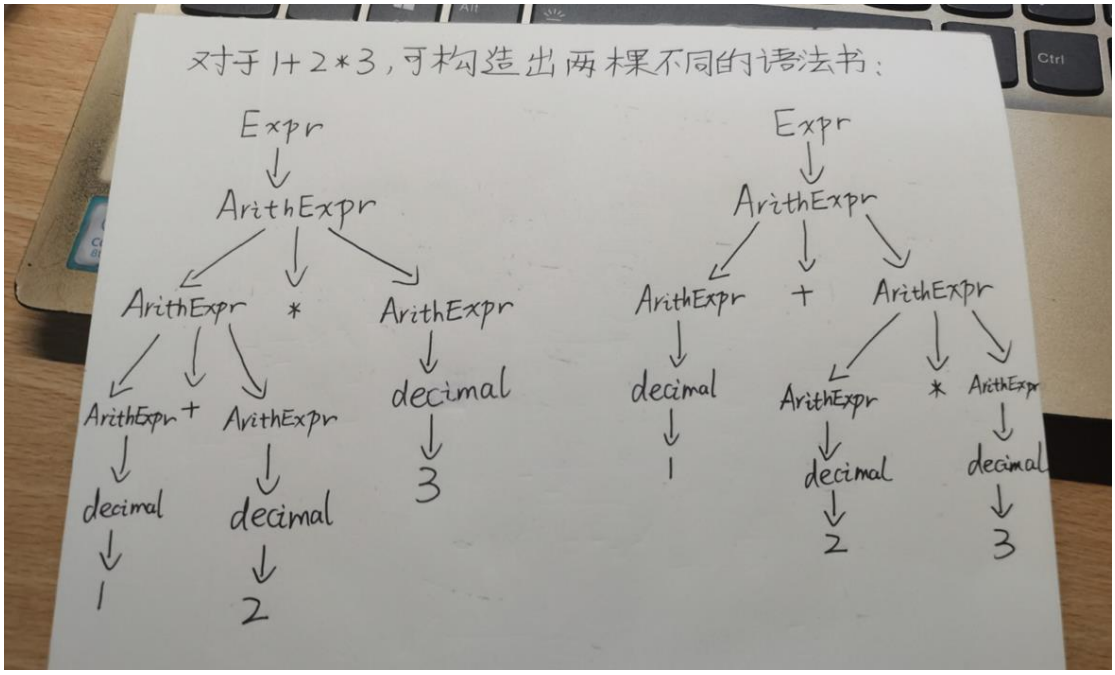
2.1 讨论语法定义的二义性

2.3.1 表达式的 BNF 定义

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*

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：

BooleanToken.java	2022/5/10
DecimalToken.java	2022/5/10
DollarToken.java	2022/5/10
FunctionToken.java	2022/5/10
OperatorToken.java	2022/5/10
Token.java	2022/5/10

（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：

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

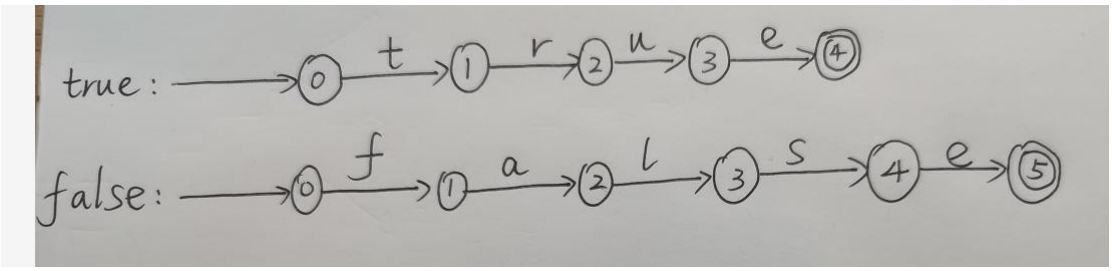
2，根据状态转换图将表达式转换为词法单元流

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：

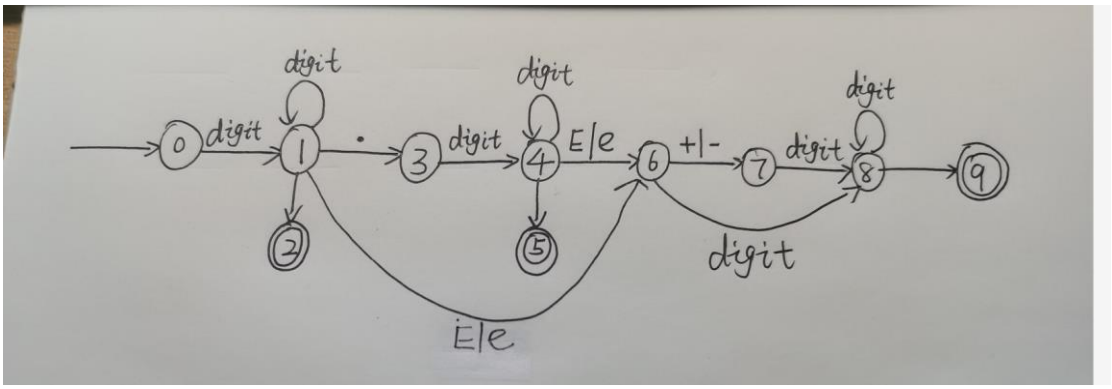
```
public Token booleanDFA(Character curChar)
```



（2）Decimal 十进制数值的 DFA 转换图：

调用函数为：

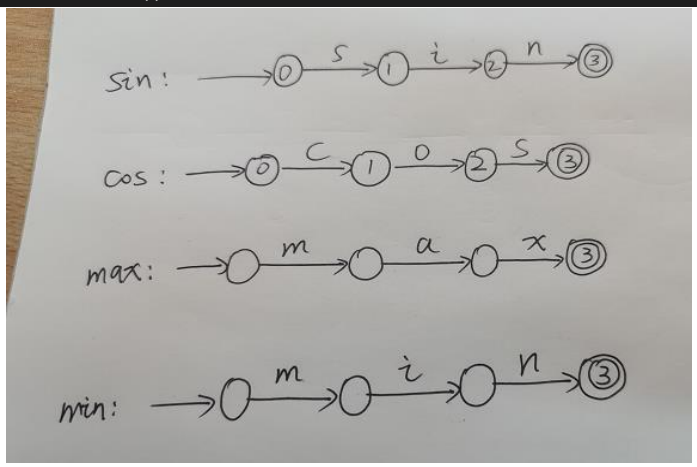
```
public Token decimalDFA()
```



（3）Function 预定义函数的 DFA 转换图：

调用函数为：

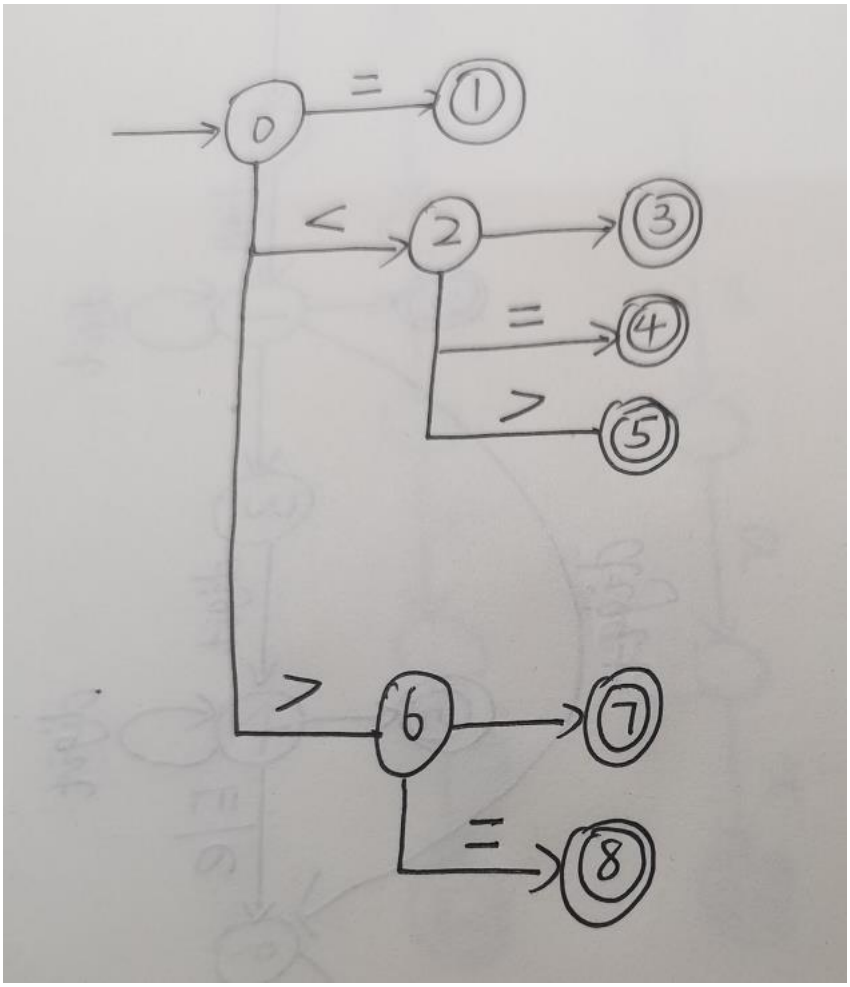
```
public Token functionDFA()
```



（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,-1, 0,-2},
    /*)*/    {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},
    /*-*/    {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,-1, 0,-2},
    /*)*/    {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},
    /*-*/    {0, 4, 0, 0, 1, 1, 1, 1,-6,-4,-4, 1, 1, 1},
    /*^*/    {0, 4, 0, 0, 0, 2, 2, 2,-6,-4,-4, 2, 2, 2},
    /*md*/   {0, 4, 0, 0, 0, 2, 2, 2,-6,-4,-4, 2, 2, 2},
    /*pm*/   {0, 4, 0, 0, 0, 0, 2, 2,-6,-4,-4, 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：

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

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

test_standard：

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

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

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。