编译原理实验

基于表达式的计算器 ExprEval

设计文档

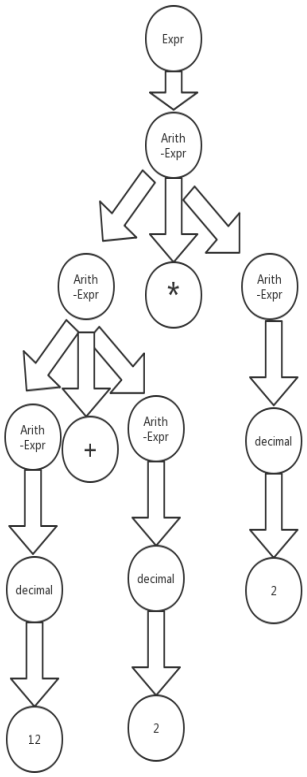
目 录

1. 讨论语法的二义性

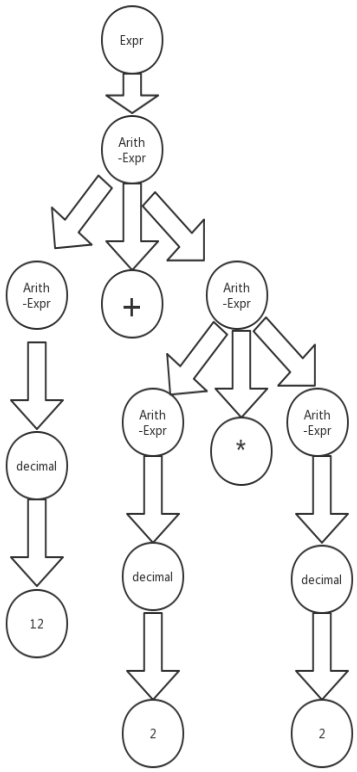
BNF 语法存在二义性。举例：

ArithExpr → **decimal** | **(** *ArithExpr* **)**
| *ArithExpr* **+** *ArithExpr* | *ArithExpr* **–** *ArithExpr*
| *ArithExpr* ***** *ArithExpr* | *ArithExpr* **/** *ArithExpr*
| *ArithExpr* **^** *ArithExpr*
| **–** *ArithExpr*
| *BoolExpr* **?** *ArithExpr* **:** *ArithExpr*
| *UnaryFunc* | *VariablFunc*

12+2*2



可以解析为



和

此程序消除二义性的方法，是通过建立算符优先关系表，确定优先关系来消除二义性。

2. 设计并实现词法分析程序

1. 定义 token

实现词法分析程序，首先要列出所有的 token：
操作符类 token: + - * / = <= >= < > <> & | ? : ! ^
符号类 token: ( ) ,
函数名类 token(大小写不限): cos sin max min
布尔常量 token(大小写不限): true false
数字 token:

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)

终结 token: $

首先定义 token 类,在网上参考资料的启发下，我为每种类型的 token 定下了一个 int 类型的标签：

```
public class Tag {
    /**
     * 给每个token设置一个字符标签，方便分辨token，只是记号并无实际意义，如小于等于为'['。
     */
    public final static int
        ADD='+', AND='&',
        COS='c', COLON=':', COMMA=',',
        DIVIDE='/',DOLLAR='$',
        EQ='=',
        FALSE='F',
        GE=']', GT='>',
        LE='[', LT='<', LP='(',
        MAX='a', MIN='i', MINUS='-', MULTIPLY='*',
        NE='N', NEG='n', NOT='!', NUM='d',
        OR='|',
        POWER='^',
        QM='?',
        RP=')',
        SIN='s',
        TRUE='T';
}
```

Token 类只需记录此 token 的标签即可。

当扫描到不同 token 构建 token 类时，这个类就会带上这个标签。因为 num 和 true,false 是会额外带有一个值的，但 true,false 可用标签本色标记自己的值，所以额外写一个 num 类继承 token 类，新增一个参数 public double value，用于记录 num 类的值。

2.编写 scanner

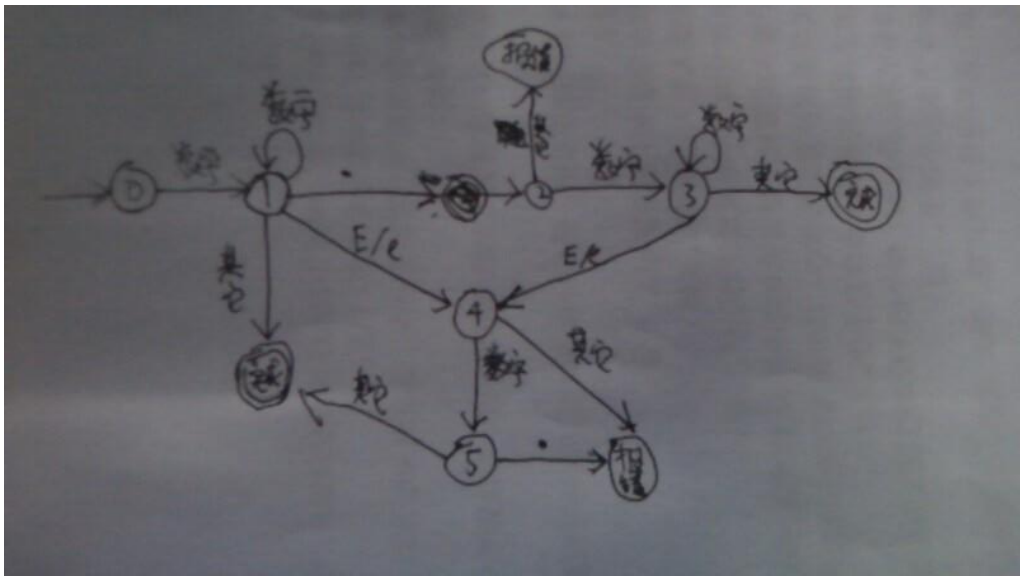
而 scanner 也很好写，因为大多数 token 只有一个字符，所以直接扫描到构造 token 类然后返回便可。

特殊情况有

1.<= ，<>这种类型的，需在匹配到第一个字符之后<扫描下一个字符，若下一个字符为对应的下一个字符，则返回对应的<=或者<>，否则则返回<。

2.函数类型和布尔常量，构建一个哈希表，储存字符串对应的标签，由于这种整个 token 都是以字母构成的，所以一直扫描，直到扫描到非字母字符，再把扫描到的字符串在哈希表中寻找，若返回空则报错，否则返回对应 token。注意，**字符串是不分大小写的，所以在 readch 函数中，需要把大写字母全部转化为小写。**

3.数字常量，当扫描到数字时，按下面的自动机运行：



3. 构建算符优先关系表

要构建优先关系表，首先我是想到用一个矩阵来做的，但是怎么实现这个矩阵一开始我毫无头绪。在同学的提示下，我决定为每种优先级类型的终结符设置一个编号 0-16，对应矩阵的行和列。

```
public final static int
        num = 0,
        bool = 1,
        addminus=2,
        muldiv=3,
        neg=4,
        power=5,
        func=6,
        lp=7,
        comma=8,
        rp=9,
        relation=10,
        not=11,
        and=12,
        or=13,
        qm=14,
        colon=15,
        dollar=16;
```

在课件中，我学会了 shift()方法来处理运算符优先级，那么只需对需要跳过的优先级进行标记即可。但是有些终结符是不能相邻的，这种终结符将会被标记为对应的报错。还有需要标记分析结束的 accept()。

```
public final static int[][] table= new int[][]
{
// num,bool,+-,*/,-,^,fucn,(,,,),Relation,!,&,|,?,:,$

    {-1,-1, 1, 1, 1, 1, 1,-1, 1, 1, 1,-5, 1, 1, 1, 1, 1},
    {-1,-1,-5,-5,-5,-5,-1,-1,-5, 1,-5,-1, 1, 1, 1, 1, 1},
    { 0,-5, 1, 0, 0, 0, 0, 0, 1, 1, 1,-5,-5,-5,-5, 1, 1},
    { 0,-5, 1, 1, 0, 0, 0, 0, 1, 1, 1,-5,-5,-5,-5, 1, 1},
    { 0,-5, 1, 1, 0, 1, 0, 0, 1, 1, 1,-5,-5,-5,-5, 1, 1},
    { 0,-5, 1, 1, 0, 0, 0, 0, 1, 1, 1,-5,-5,-5,-5, 1, 1},
    {-3,-3,-3,-3,-3,-3,-3, 0,-3, 1,-3,-3,-3,-3,-3, 1,-3},
    { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,-2},
    { 0,-5, 0, 0, 0, 0, 0, 0, 0, 0,-5,-5,-5,-5, 0, 0,-2},
    {-1,-1, 1, 1, 1, 1,-1,-1, 1, 1, 1,-1, 1, 1, 1, 1, 1},
    { 0,-5, 0, 0, 0, 0, 0, 0,-5, 1, 1, 1, 1, 1, 1, 1, 1},
    { 0, 0,-5,-5,-5,-5,-5, 0,-5, 1, 0, 0, 1, 1, 1, 1, 1},
    { 0, 0,-5,-5,-5,-5,-5, 0,-5, 1, 0, 0, 1, 1, 1, 1, 1},
    { 0, 0,-5,-5,-5,-5,-5, 0,-5, 1, 0, 0, 0, 1, 1, 1, 1},
    { 0,-7, 0, 0, 0, 0, 0, 0,-7,-6, 0,-7,-7,-7, 0, 0,-7},
    { 0,-7, 0, 0, 0, 0, 0, 0, 1, 1, 0,-7,-7,-7, 1, 1, 1},
    { 0, 0, 0, 0, 0, 0, 0, 0,-3,-3, 0, 0, 0, 0, 0,-7, 2}
};
/* 0 for shift
 * 1 for reduce
 * 2 for accept
 * -1 for MissingOperatorError
 * -2 for MissingRightParenthesisException
 * -3 for MissingLeftParenthesisException
 * -4 for FunctionCallException
 * -5 for TypeMismatchedException
 * -6 for MissingOperandError
 * -7 for TrinaryOperationError
 */
```

4. 设计并实现语法分析和语义处理程序

按照第四第五章的内容，利用堆栈构建语法分析程序。

构建 terminal 类，在其中标记表达式的类型，若已经到达终结符，则标记终结符的类型。

其中类型类为：

```
public class Type {
        /**
         * 非终结符
         */
        public final static int
                Expr=300,
                ArithExpr=301,
                BoolExpr=302,
                UnaryFunc=303,
                VariablFunc=304,
                ArithExprList=305;
        /**
         * 终结符
         */
        public final static int
                num = 0,
                bool = 1,
                addminus=2,
                muldiv=3,
                neg=4,
                power=5,
                func=6,
                lp=7,
                comma=8,
                rp=9,
                relation=10,
                not=11,
                and=12,
                or=13,
                qm=14,
                colon=15,
                dollar=16;
}
```

具体操作如下，
重复下列操作。
按照顺序扫描 token，然后构建 terminal，然后利用优先表对比此 terminal 与栈顶 terminal，若是报错则报错，若不是 shift 则入栈，若是 shift()入栈，若是 reduce()则调用 reduce()函数对表达式进行匹配和计算，并缩减栈，使多个 terminal 合并为一个。
直到栈顶为$，调用 accept()结束编译，返回之前记录的最后一个 terminal 的值。
Reduce 方法封装在 terminalReduce 类里，代码就不贴了。

5. 测试样例

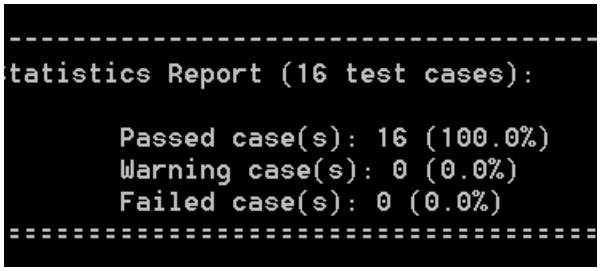
个人加的几个测试样例:

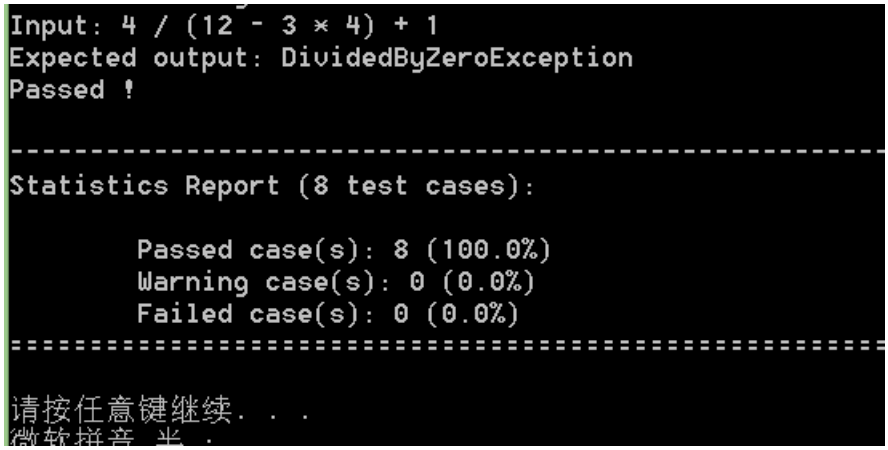
```
<test-case>
    <id>M001</id>
    <description>Very much minus sign.</description>
    <input>9 - ------------3</input>
    <output>6</output>
</test-case>

<test-case>
    <id>M002</id>
    <description>Complicated judge.</description>
    <input>true?((2*3=6)?(42-2=42?1:0):(54/6=8?2:3)):33</input>
    <output>0</output>
</test-case>

<test-case>
    <id>ME001</id>
    <description>Wrong function error</description>
    <input>1+ aaa(1,1)</input>
    <exception>IllegalIdentifierException</exception>
</test-case>

<test-case>
    <id>ME002</id>
    <description>Scientific Notation Error.</description>
    <input>4 + 10.E+1.5 + 1</input>
    <exception>IllegalDecimalException</exception>
</test-case>


<test-case>
    <id>ME003</id>
    <description>Null INPUT.</description>
    <input><![CDATA[]]></input>
    <exception>EmptyExpressionException</exception>
</test-case>

</test-case-definitions>
```

1 多个减号，2 复杂判断式，3 不合法函数名，4 指数部分有小数，5 空输入

```
Statistics Report (5 test cases):

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

标准和简单测试样例：


tatistics Report (16 test cases):
Passed case(s): 16 (100.0%)
Warning case(s): 0 (0.0%)
Failed case(s): 0 (0.0%)


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%)
请按任意键继续. . .