实验报告

1. 比较静态成员与非静态成员

静态成员最主要的特点是它不属于任何一个类的对象，它不保存在任意一个对象的内存空间中，而是保存在类的公共区域中，所以任何一个对象都可以直接访问该类的静态成员，都能获得相同的数据值，修改时，也在类的公共区域进行修改。

将`lookahead`定位为非静态后，不会影响程序的正确性，因为`lookahead`的操作都在同一个类中进行。定义成`static`类型是为了避免其他对象产生多个`lookahead`副本，导致版本不统一的情况出现。因为在递归下降语法分析器中`lookahead`应该是唯一的。

2. 比较消除尾递归前后程序的性能

- 消除尾递归：把原来尾递归的`rest()`语句改成`continue`，然后在外面加一层死循环，当`lookahead`不为`+`或`-`运算符，即中缀表达式中所有的运算符都已输出完成时，`break`出死循环

```
    void rest() throws IOException {
        while(true) {
            if (lookahead == '+') {
                match('+');
                term();
                System.out.write('+');
                continue;
            } else if (lookahead == '-') {
                match('-');
                term();
                System.out.write('-');
                continue;
            } else {
                break;
            }
        }
    }
```

- 性能对比

	时间复杂度	空间复杂度
尾递归	$O(n)$	$O(n)$
非尾递归	$O(n)$	$O(1)$

3. 拓展错误处理功能

实验软装置的语法规则定义

实验软装置中的`Parser`类是一个递归下降语法分析器，根据代码可以写出其语法规则定义：

$Digits \rightarrow 0|1|2|3|4|5|6|7|8|9$

$Operators \rightarrow +|-|end$

$Expr \rightarrow TermRest$

$Term \rightarrow Digits$

$Rest \rightarrow OperatorsTermRest|\epsilon$

其中end为表达式的结束标志，在程序中为回车符。

词法错误

- 判断：输入字符不属于$Digits$和$Operators$的子集，具体代码参考`lexicalError`函数
- 错误恢复：读入字符时加一个`do-while`循环，直到有合法输出才跳出循环

语法错误

语法错误总共有三种：缺少左运算量、缺少右运算量和缺少运算符。为了判断语法错误，增加了两条语法规则：

$lackTerm \rightarrow Operators$

$lackOperator \rightarrow Term|\epsilon$

并对原来软装置中的部分语法规则进行了修改：

$Term \rightarrow Digits|lackTerm$

$Rest \rightarrow lackOperatorOperatorsTermRest|\epsilon$

- 这里假设缺少右运算量只会出现在表达式末尾，所以当`lackTerm`函数读取的`lookahead`为`+`或`-`时，程序提示缺少左运算量，否则提示缺少右运算量

4. 测试用例

- 输入：`-1+2a+3++4-5-6+`
- 该输入中存在多个错误：
  - 第一个`-`号缺少左运算量
  - `a`不是合法输入
  - `+`缺少右运算量
  - 最后一个`+`缺少右运算量
- 运行结果：从结果中可以看出，程序可以准确错误定位、划分错误类型与进行相应的错误处理

```
Input an infix expression and output its postfix notation:
-1+2a+3++4-5-6+
- lacks a left term!
a is an illegal input!
+ lacks a left term!
The final operator lacks a right term!
1-2+3++4+5-6-+
End of program.
```