中山大学计算机院本科生实验报告
（2023 学年春季学期）

课程名称：编译原理　　批改人：

实验	LAB1-后缀表达式 Postfix	专业（方向）	计算机科学与技术计科一班
学号	21307099	姓名	李英骏
Email	liyj323@mail2.sysu.edu.cn	完成日期	2024 年 4 月 17 日

目录

1 静态成员与非静态成员 **2**

2 比较消除尾递归前后程序的性能 **4**

3 扩展错误处理功能 **6**

3.1 识别和分类错误 6

3.2 具体实现 6

3.3 空格检测 7

3.4 连续运算量错误 7

3.5 连续运算符错误 8

3.6 出错恢复与位置 8

1 静态成员与非静态成员

这是一个单线程程序, 并且在整个程序中只存在一个 Paser 对象, 理论上, 是否将 lookahead 声明为 static 对于程序的正确性没有任何影响. 实验结果记录如下:

声明为 static 时, 运行./testcase.bat:

```
(base) PS C:\Users\liyj\Desktop\Compiler\LAB\LAB1\Lab_1_source> ./testcase.bat
Running Testcase 001: a correct input from DBv2.
================================================
The input is:
9-5+2
------------------------------------------------
Input an infix expression and output its postfix notation:
95-2+
End of program.
------------------------------------------------
The output should be:
95-2+
================================================
请按任意键继续. . .
C:\Users\liyj\Desktop\Compiler\LAB\LAB1\Lab_1_source>call testcase-003.bat
Running Testcase 003: missing an operator.
================================================
The input is:
95+2
------------------------------------------------
Input an infix expression and output its postfix notation:
9
End of program.
------------------------------------------------
The output should be:
9 (error)
================================================
请按任意键继续. . .

C:\Users\liyj\Desktop\Compiler\LAB\LAB1\Lab_1_source>call testcase-004.bat
Running Testcase 004: missing an operand.
================================================
The input is:
9-5+-2
------------------------------------------------
Input an infix expression and output its postfix notation:
Exception in thread "main" java.lang.Error: syntax error
        at Parser.term(Postfix.java:35)
        at Parser.rest(Postfix.java:18)
        at Parser.rest(Postfix.java:25)
        at Parser.expr(Postfix.java:12)
        at Postfix.main(Postfix.java:47)
------------------------------------------------
The output should be:
95- (error)
================================================
请按任意键继续. . .
```

```
C:\Users\liyj\Desktop\Compiler\LAB\LAB1\Lab_1_source>call testcase-002.b
Running Testcase 002: a correct long input.
================================================
The input is:
1-2+3-4+5-6+7-8+9-0
------------------------------------------------
Input an infix expression and output its postfix notation:
12-3+4-5+6-7+8-9+0-
End of program.
------------------------------------------------
The output should be:
12-3+4-5+6-7+8-9+0-
================================================
请按任意键继续. . .
```

图 1: static

去掉 static 时, 运行./testcase.bat:

```
(base) PS C:\Users\liyj\Desktop\Compiler\LAB\LAB1\Lab_1_source> ./testcase.bat
Running Testcase 001: a correct input from DBv2.
================================================
The input is:
9-5+2
------------------------------------------------
Input an infix expression and output its postfix notation:
95-2+
End of program.
------------------------------------------------
The output should be:
95-2+
================================================
请按任意键继续. . .
C:\Users\liyj\Desktop\Compiler\LAB\LAB1\Lab_1_source>call testcase-003.bat
Running Testcase 003: missing an operator.
================================================
The input is:
95+2
------------------------------------------------
Input an infix expression and output its postfix notation:
9
End of program.
------------------------------------------------
The output should be:
9 (error)
================================================
请按任意键继续. . .

C:\Users\liyj\Desktop\Compiler\LAB\LAB1\Lab_1_source>call testcase-004.bat
Running Testcase 004: missing an operand.
================================================
The input is:
9-5+-2
------------------------------------------------
Input an infix expression and output its postfix notation:
Exception in thread "main" java.lang.Error: syntax error
        at Parser.term(Postfix.java:35)
        at Parser.rest(Postfix.java:18)
        at Parser.rest(Postfix.java:25)
        at Parser.expr(Postfix.java:12)
        at Postfix.main(Postfix.java:47)
------------------------------------------------
The output should be:
95- (error)
================================================
请按任意键继续. . .
```

```
C:\Users\liyj\Desktop\Compiler\LAB\LAB1\Lab_1_source>call testcase-002.b
Running Testcase 002: a correct long input.
================================================
The input is:
1-2+3-4+5-6+7-8+9-0
------------------------------------------------
Input an infix expression and output its postfix notation:
12-3+4-5+6-7+8-9+0-
End of program.
------------------------------------------------
The output should be:
12-3+4-5+6-7+8-9+0-
================================================
请按任意键继续. . .
```

图 2: non static

可以看到没有任何区别. 我认为声明为非 static更合适, 因为

1. 在绝大多数需求中, 都不会遇到需要多个 paser 实例解析同一个字符流的情形.
2. 在单 paser 实例的应用程序中, 是否声明为 static 影响不大. 但 static 显然会影响代码的扩展性和复用性.
3. 在可能的多实例或多线程中,lookahead 设置为 static 会导致错误或竞态等线程安全问题.

2 比较消除尾递归前后程序的性能

消除尾递归如图:

```
1 usage
15      void rest() throws IOException {
16          while (lookahead == '+' || lookahead == '-') {
17              if (lookahead == '+') {
18                  match( t: '+');
19                  term();
20                  System.out.write( b: '+');
21              } else if (lookahead == '-') {
22                  match( t: '-');
23                  term();
24                  System.out.write( b: '-');
25              } else{
26                  // do nothing
27              }
28          }
29      }
```

图 3: loop version

我们使用如下脚本生成测试数据

```java
public class GenerateExpressions {
    public static void main(String[] args) throws IOException {
        File directory = new File( pathname: "./testcases");
        if (!directory.exists()) {...}

        int maxN = 10000; // 表达式的最大长度
        int trials = 2; // 每个长度生成的表达式数
        Random rand = new Random();
        int fileIndex = 1;

        for (int N = 1; N <= maxN; N+=10) {
            for (int trial = 0; trial < trials; trial++) {
                String fileName = String.format("tc-%03d.infix", fileIndex);
                File file = new File(directory, fileName);
                try (PrintWriter writer = new PrintWriter(file)) {
                    String expression = generateExpression(N, rand);
                    System.out.println(expression);
                    writer.println(expression);
                }
                fileIndex++;
            }
        }
    }

    1 usage
    static String generateExpression(int N, Random rand) {
        StringBuilder sb = new StringBuilder();
        for (int i = 0; i < N; i++) {
            sb.append(rand.nextInt( bound: 10));  // Random digit
            if (i < N - 1) {  // Append operator if not the last character
                sb.append(rand.nextBoolean() ? '+' : '-');
            }
        }
        return sb.toString();
    }
```

图 4: loop version

如图:

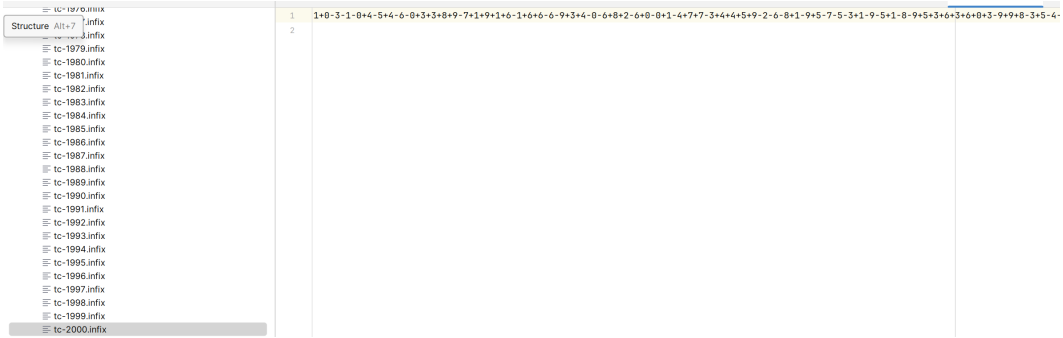

图 5: loop version

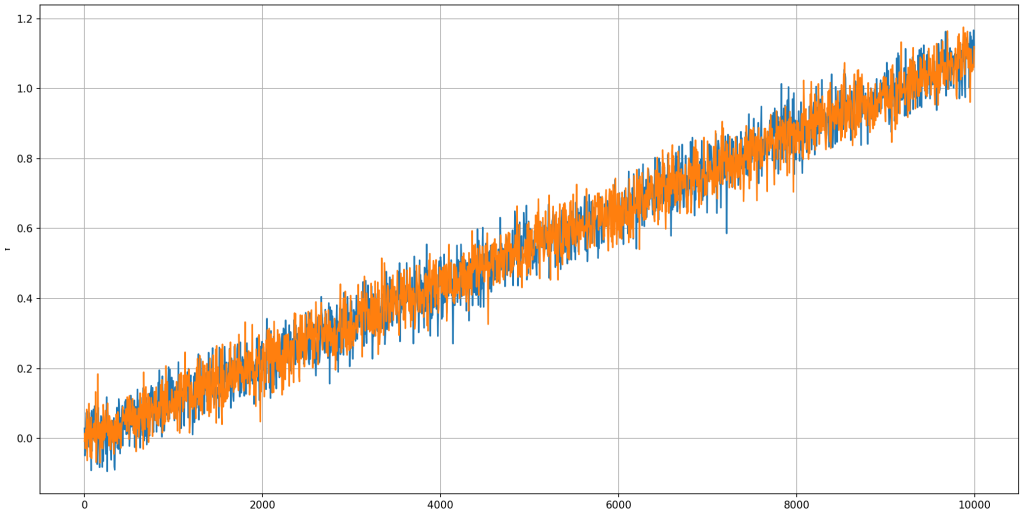

图 6: loop version

发现性能基本没有差异, 经搜索发现是 jvm 在优化时会自动消除尾递归, 导致性能没有差异. 暂时没有找到关闭该优化的方法.

3 扩展错误处理功能

3.1 识别和分类错误

词法错误：这些错误涉及输入中的非法字符或结构，例如空格、非法字符等。

语法错误：这些错误发生在字符组合违反了语法规则时，例如两个数字之间缺少运算符，或者运算符前缺少操作数。

3.2 具体实现

- **空格处理：**通过特定函数 `skipWhitespace()` 检查和跳过空格，并标记为词法错误。

- **两个数字之间缺少运算符：**通过状态变量 `expectingOperator` 来检测。如果期望一个运算符（例如，读取一个数字后）但是遇到另一个数字，则报告语法错误。

- **运算符前缺少操作数：**同样使用 `expectingOperator` 状态。如果期望一个操作数（例如，读取一个运算符后）但是遇到另一个运算符或非法字符，则报告语法错误。

3.3 空格检测

```
Input an infix expression and output its postfix notation:
1 +2
1+2
End of program.
Error found:
position 2: blank is not allowed [lexical error]


Process finished with exit code 0
```

图 7: loop version

3.4 连续运算量错误

```
Input an infix expression and output its postfix notation:
1+22+3
1+22+3
End of program.
Error found:
position 4: Missing operator between numbers [Syntax Error]


Process finished with exit code 0
```

图 8: loop version

3.5 连续运算符错误

```
Input an infix expression and output its postfix notation:
1+2++3
1+2++3
End of program.
Error found:
position 5: Missing operand before operator [Syntax Error]


Process finished with exit code 0
```

图 9: loop version

3.6 出错恢复与位置

- 在解析器中，使用 `position` 变量来跟踪当前处理字符的位置。每当读取一个新字符时，更新 `position`。当报告错误时，使用 `position` 来指出错误发生的具体位置

- 在遇到错误时，通过 `recover()` 函数尝试恢复。此函数的目的是跳过当前的错误点，寻找下一个可能的合法输入点。例如跳过直到找到下一个数字或运算符，从而尝试从新的位置重新开始解析过程。

```
Input an infix expression and output its postfix notation:
1 ++2+333+4+%+6
1++2+333+4++6
End of program.
Error found:
position 2: blank is not allowed [lexical error]
position 4: Missing operand before operator [Syntax Error]
position 8: Missing operator between numbers [Syntax Error]
position 9: Missing operator between numbers [Syntax Error]
position 13: Illegal character [Lexical Error]
position 14: Missing operand before operator [Syntax Error]


Process finished with exit code 0
```

图 10: loop version