二、《编译原理》实验

后缀表达式 *Postfix*[1]

1. 实验目标

在完成帮助掌握 Java 语言开发过程和面向对象设计的预备实验后，本实验是进入编译原理课程实验的第一个入门实验。

2. 实验环境

编程语言： Java JDK 1.5 或以上版本

开发工具： 可自由选择 Eclipse、JBuilder 等 IDE 环境，也可直接采用 UltraEdit、EditPlus 等编辑器在命令行工作。但提交的实验结果必须独立于特定的 IDE，可直接运行在 JDK 上。

编码规范： 要采用面向对象风格来设计实验中的所有类，并遵循一种良好的程序设计习惯。例如，如果你将一个程序的很多功能全部放在一个长长的 main()方法中实现，这样的设计与编码风格会被扣分。

在实验过程中应注意培养规范的编码风格。本实验要求所有源代码严格遵循 Sun 公司（现为 Oracle 公司）关于 Java 程序设计语言的编码规范（Code Conventions for the Java Programming Language，Revised April 1999），详细内容请参见：Code Conventions for the Java Programming Language: Contents (oracle.com)。

完成项目代码后，应使用 JDK 附带的文档工具 javadoc，根据源程序中的文档化注释，自动生成相应的说明性文档。

1. Credit: Prof. Wenjun Li @ SYSU. 若对实验内容有任何疑问或改进意见，请联系任课教师。

3. 实验内容

本实验提供了一个完整的 Java 程序，它处理的对象是一种简单的中缀表达式：表达式中的运算量只能是 0～9 的单个数字，仅支持加、减运算，且表达式中不允许任何空格、制表符等空白符号。该程序的唯一功能是将终端用户输入的一个中缀表达式转换为等价的后缀表达式后输出，例如输入中缀表达式 1+2，将输入其等价的后缀表达式 12+。若用户输入的表达式有误，则程序的输出是无意义的，但该程序不会提示用户输入有误。

本文档的第 6 章详细给出了该程序的源代码清单，并介绍了为便于学生开展本实验而提供的一个简单的实验软装置。

3.1 实验步骤

请按以下步骤改进在本实验的实验软装置中提供的源程序代码。

步骤一 静态成员与非静态成员

实验软装置中将类 Parser 中的数据成员（又称域，Field）**int** lookahead 定义为静态的（**static**），为什么要这样定义？尝试将 lookahead 定义为非静态的（即删除声明之前的修饰符 **static**），看看这是否会影响程序的正确性。

步骤二 消除程序中的尾递归

所谓尾递归（Tail Recursion），是指在方法体（method body）的最后一条语句是一条直接递归调用语句，即在方法体最后直接调用了方法自身。原程序中的 rest()方法显然是一个含尾递归的递归程序。

试消除原程序中 rest() 方法的尾递归，意即将该程序转换为一个不带尾递归的等价程序，因为将递归转换为循环通常都会提高程序的执行效率。将一个任意的递归程序转换为循环程序并不简单，但尾递归的消除往往是很容易的。

步骤三 为程序扩展错误处理功能

实验软装置中的程序未处理用户输入的中缀表达式出错的情况，譬如表达式中含有空格、两个运算量之间缺少运算符、或运算符缺少左（或右）运算量等。

请为该程序扩展错误处理功能，最低要求是当输入表达式有错时，给出一个报错信息。一个做得更好的错误处理功能应该能够准确地给出错误的位置和类别（属词法错误亦或属语法错误）。

学有余力的学生还可考虑尝试如何实现出错恢复（Error Recovery），即当程序发现一个错误时不是立马停下来，而是能够从跌倒的地方爬起来，继续分析下去，从而一次运行即可发现更多的错误。

步骤四 为程序增加文档化注释

根据本文档规定的 **Java** 编码风格，为你的源代码撰写文档化注释（Documentation Comments），并利用 **JDK** 提供的 **javadoc** 工具生成源程序的 **HTML** 文档，并存放在一个名为 **doc** 的目录中。

步骤五 程序的单元测试（可选）

学有余力的同学可下载 **Java** 开发平台上使用最广泛的开源（Open Source）单元测试工具 **JUnit**，并使用该工具完成对改进后程序的单元测试。

可从 http://www.junit.org/ 下载 **JUnit** 的最新版本。

3.2 实验结论

请在你提交的文档中描述以下内容。

1. 比较静态成员与非静态成员

记录你在实验中观察到的结果，分析将类 Parser 的成员 lookahead 声明为 **static** 和非 **static** 是否对程序的正确性造成影响，并说明你对此问题的看法，即到底是声明为静态成员为好，还是声明为非静态成员为好。

2. 比较消除尾递归前后程序的性能（可选）

按常理，将递归程序转换为等价的循环程序通常会提高程序的效率，但真的是这样吗？通常有两种途径验证：一种理论途径，通过理论（通常基于数学的理论）上的建模与分析得出结论（例如，在我们熟悉的算法分析中的 $O(n)$、$O(n\log n)$、$O(n^2)$ 等结论）；另一种是 Empirical Software Engineering 中的实验途径，通过设计实验、执行实验、并收集数据加以分析，然后得出结论。

为采用实验手段验证结论，我们该如何对改进前和改进后的两个程序进行实验？请给出你设计的实验方案，包括测试数据如何选择及获取（譬如一个很长的表达式当然不适合以手工方式来输入）、测试后我们该收集哪些数据、对数据如何进行分析、数据分析结果如何展示、数据分析的预期结果如何等。

学有余力的同学可尝试完成这一实验过程，并展示你的实验结果。注意，实验结果的最佳展示方式是同时绘有改进前后两个程序各自性能的统计图形，可使用 Excel 辅助你的工作。

3. 扩展错误处理功能

描述你对扩展程序的错误处理或错误恢复功能的思路，重点是如何设计与实现以下 3 个目标：

① 如何准确地划分错误类型，指出错误是词法错误亦或语法错误。在语法错误中，如何更精细地给出明确的出错信息，譬如缺少运算符、或缺少左运算量等。

② 如何实现错误的定位。

③ 如何实现出错后的恢复。

可在文档中给出运行各类测试用例的屏幕截图。测试用例需能够展示扩展的错误处理功能在错误分类、错误定位、出错恢复等方面的所有特性。

4. 结果提交

提交的实验结果中应包含：

① 改进的实验软装置

② 自述文件 readme.pdf，描述姓名、学号等基本信息，并以图文形式分点描述实验结论中要求的内容。

实验结果全部存放在一个名为“学号_姓名_lab1”（中间不要任何空格）的文件夹中，压缩成单个文件后（名称保持一致），在 **2025/4/16 23:59** 前提交至以下链接： https://yunbiz.wps.cn/c/collect/c2D4G1z7a4N .

5. 实验交流

在本实验过程中完成了指定项目之外的实验内容（譬如完成了以实验手段比较消除尾递归前后两个程序的性能，或在扩展错误处理功能中对错误定位或出错恢复等有特别心得）的同学可主动申请在课堂交流自己的实验心得，这些主动交流的同学将获得额外的加分。

建议你按以下次序展示你的实验结果：

① 介绍自己对问题的理解以及解决问题的方案。
② 向大家展示你的源代码或 javadoc 文档（HTML 格式）。
③ 向大家展示你的实验结果。
④ 接受老师和其他同学的提问和建议。

6. 实验软装置介绍

为便于同学们开展实验，本实验提供了实验软装置。每位同学应下载该实验软装置，并基于它开展实验过程。

6.1 目录和文件组织

这是一个非常简单的实验软装置，包括：

- 空子目录 bin，用于存放编译后生成的可运行 Java 字节码。
- 空子目录 doc，用于存放根据你撰写的文档化注释生成的 HTML 文档。
- 子目录 src，其中包含本实验的处理对象 Postfix.java 的源程序。
- 子目录 testcases，其中包含 4 个预先提供的测试用例及其预期输出；*.infix 中各包含了一个以中缀表示的输入表达式，*.postfix 中保存了对应的后缀表达式预期输出。
- 脚本 build.bat，对 src 中的源程序进行编译，生成的可执行结果存放在 bin 中。
- 脚本 doc.bat，根据 src 中的源程序生成 HTML 文档，生成结果存放在 doc 中。
- 脚本 run.bat，运行 bin 中的可执行程序（Java 字节码）。

- 脚本 testcase*.bat，运行各个测试用例；脚本 testcase-00n 表示运行编号为 00n 的测试用例，脚本 testcase.bat 直接运行所有的测试用例。在这些脚本中使用了命令行的输入重定向机制（Input Redirection），以直接从文件中读取测试用例，避免了手工键盘操作方式输入测试用例。

6.2 源程序清单

源程序 src\Postfix.java 的代码清单如下：

```
import java.io.*;

class Parser {
    static int lookahead;
    public Parser() throws IOException {
        lookahead = System.in.read();
    }
    void expr() throws IOException {
        term();
        rest();
    }
    void rest() throws IOException {
        if (lookahead == '+') {
            match('+');
            term();
            System.out.write('+');
            rest();
        } else if (lookahead == '-') {
            match('-');
            term();
            System.out.write('-');
            rest();
        } else {
            // do nothing with the input
        }
    }
    void term() throws IOException {
        if (Character.isDigit((char)lookahead)) {
```

```
            System.out.write((char)lookahead);
            match(lookahead);
        } else    throw new Error("syntax error");
    }

    void match(int t) throws IOException {
        if (lookahead == t)    lookahead = System.in.read();
        else    throw new Error("syntax error");
    }
}
public class Postfix {
    public static void main(String[] args) throws IOException {
        System.out.println("Input an infix expression and output its postfix notation:");
        new Parser().expr();
        System.out.println("\nEnd of program.");
    }
}
```