Lab03 实验报告

1. 比较静态成员与非静态成员

无论将`lookahead`声明为static还是非static对程序的正确性均没有影响。

2. 比较消除尾递归前后程序的性能

数据选择

以操作数个数作为衡量标准将测试数据分为4组，每组10个测试用例。4组的操作数个数分别为10，100， 1000， 5000。

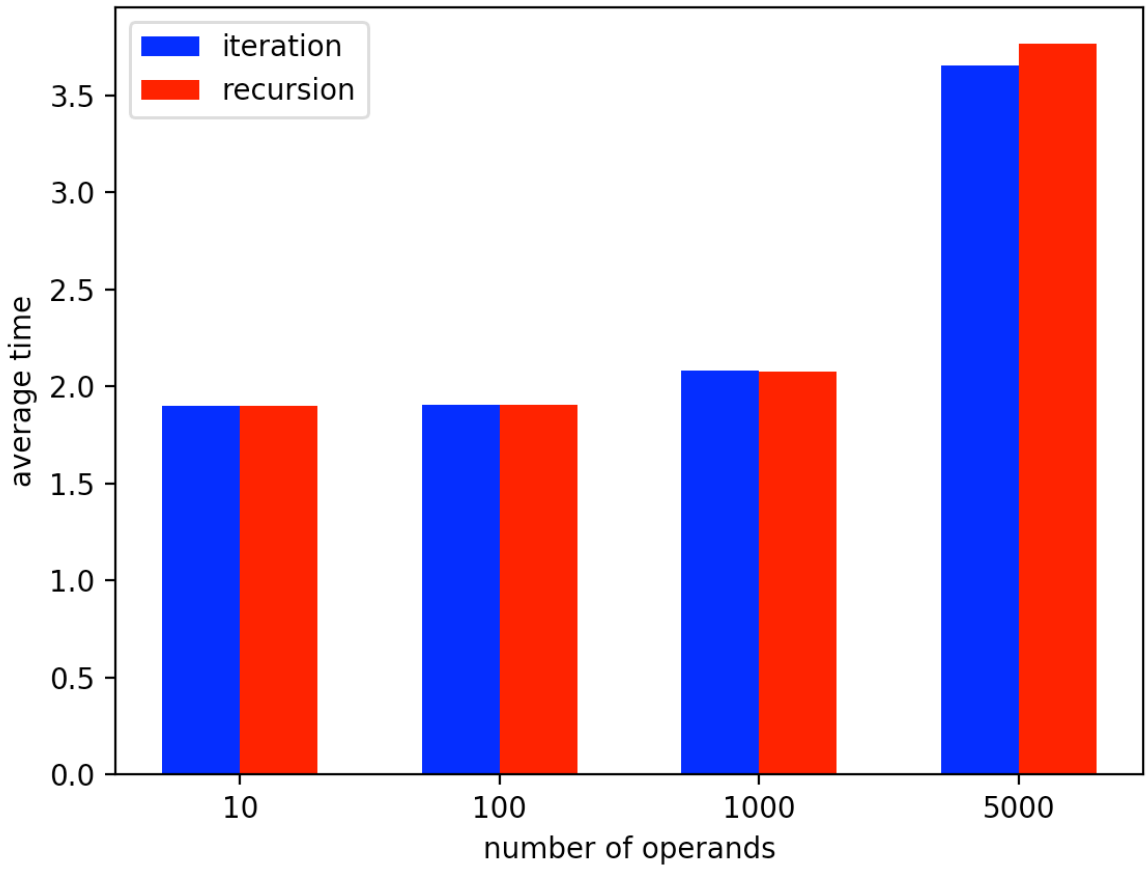
数据获取

通过python程序随机生成指定长度的表达式(生成程序为tools/performance.py中build函数)。

数据收集与分析

针对同一份benchmark分别运行代码的递归与循环版本，获得运行时间。为避免特例影响测试结果，因此4组数据分别有10个测试用例，最后计算10个用例的平均用时并生成柱状图以直观的比较程序效率。

结果



从上图中可以看到，对于小规模的数据两个版本的效率几乎一样，对于较大规模的数据循环版本效率高于递归版本，但是效率提升不明显。针对5000个操作数的测试，循环版本的效率只比递归版本高出3.09%。

3. 扩展错误处理功能

划分错误类型

为了能够将两种错误区分出来，我实现了一个简单的词法分析器`Scanner`，`Parser`通过调用`Scanner`的`getNextToken`来获取token。因此词法错误与语法错误的检查可以分别在`Scanner`与`Parser`中实现。`Scanner`的实现没有使用教材所介绍的双buffer设计，只是单纯的使用一个buffer将所有输入暂存。`Parser`每次通过`getNextToken`获取token时，`Scanner`首先进行词法检查，若遇到词法错误则立即输出错误信息，并将出错的token返回给`Parser`以便实现error recovery。`Parser`在接收到token后，会判断该token是否为预期的token，若不是则输出错误信息，并执行error recovery相关操作。

错误的定位

`Scanner`在生成token时会记录下当前是第几个token(在这个程序中也可以认为是第几个字符)，`Parser`可以通过`getCount`获得这一信息，以及`getBuffer`获得当前所处理的表达式。有了上述信息，无论是`Scanner`还是`Parser`都可以定位到错误的位置。

错误恢复

对于`Scanner`而言，是否遇到错误的字符并没有太大区别，如果遇到错误的字符则输出错误信息，其余的动作与正确的字符一致，因此即使遇到了错误的字符，程序也不会因此而停止运行。

对于`Parser`而言，如果lookahead所表示的token不是当前预期的，则会从错误的token开始不断丢弃后续的token直至遇到预期的token或是遇到结尾（在本程序中以\n作为结尾）。如果最后能够遇到预期的token，则从那个位置开始继续正常的语法分析。

测试截图及说明

figure 1

输入合法的表达式可以输出正确的后缀表达式

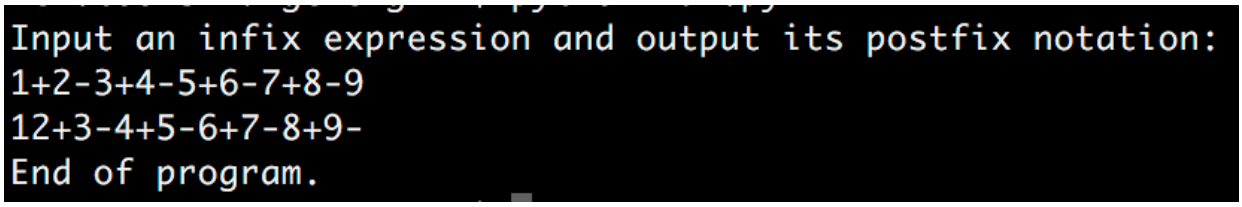
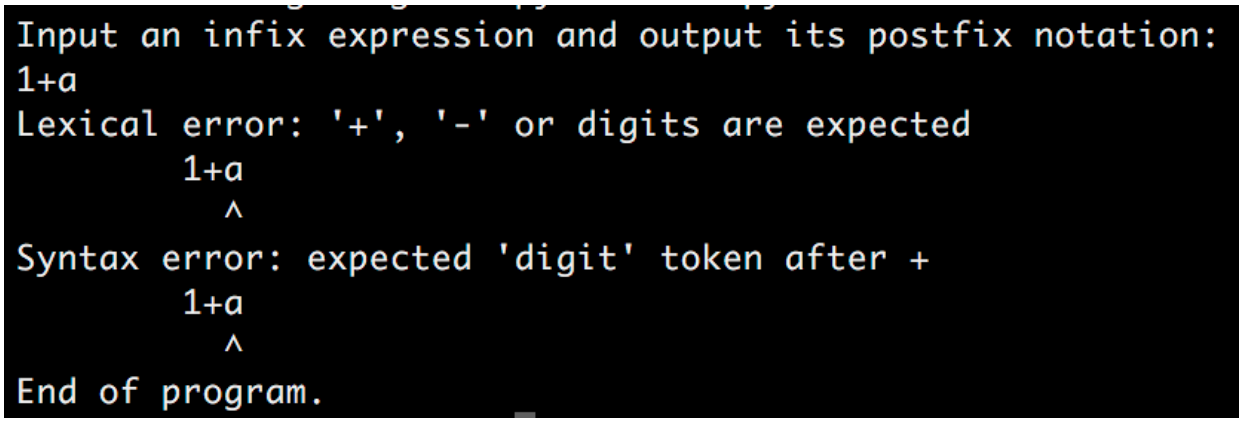


figure 2

输入错误的字符（包括空白符）会报错，并实现了错误分类，同时定位错误位置以及输出相关提示。



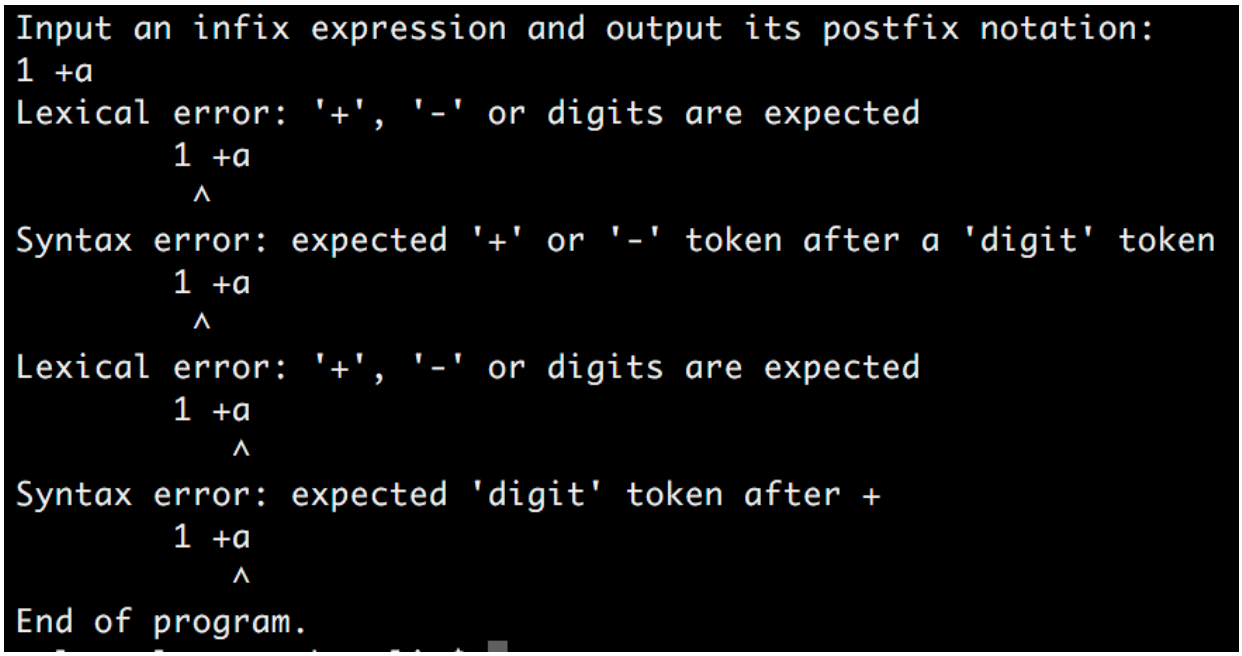
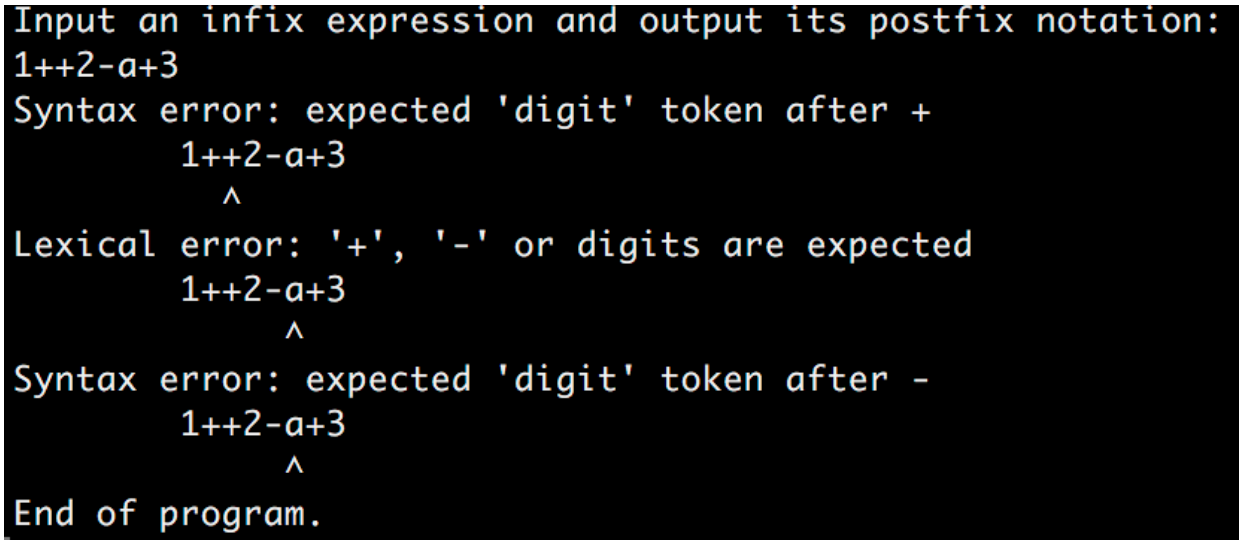


figure 3

表达式多处错误能够尽可能的检查出错误，而不是在第一处错误停止。



```
Input an infix expression and output its postfix notation:
1+2-abc+3-efg
Lexical error: '+', '-' or digits are expected
        1+2-abc+3-efg
            ^
Syntax error: expected 'digit' token after -
        1+2-abc+3-efg
            ^~~
Lexical error: '+', '-' or digits are expected
        1+2-abc+3-efg
             ^
Lexical error: '+', '-' or digits are expected
        1+2-abc+3-efg
              ^
Lexical error: '+', '-' or digits are expected
        1+2-abc+3-efg
                ^
Syntax error: expected 'digit' token after -
        1+2-abc+3-efg
                ^~
Lexical error: '+', '-' or digits are expected
        1+2-abc+3-efg
                 ^
Lexical error: '+', '-' or digits are expected
        1+2-abc+3-efg
                  ^
End of program.
```

```
Input an infix expression and output its postfix notation:
1+++a-2
Syntax error: expected 'digit' token after +
        1+++a-2
          ^~~
Lexical error: '+', '-' or digits are expected
        1+++a-2
            ^
End of program.
```