

学生学号	0121710880415	成绩	
------	---------------	----	--

# 学 生 实 验 报 告 书

## 武汉理工大学

实验课程名称

编译原理

开 课 学 院

计算机科学与技术学院

指导教师姓名

王云华

学 生 姓 名

颜道江

学生专业班级

软件 1704

2019    --    2020    学 年    第   二   学 期

实验课程名称： 编译原理

实验项目名称	语义分析与中间代码生成			实验成绩	
实验者	颜道江	专业班级	软件 1704	组别	
同组者				实验日期	2019/12/22

## 第一部分：实验内容描述

### 1.设计题目

算术表达式从中缀式翻译成后缀式的程序实现。

### 2.设计目的及设计要求

设计一个语法制导翻译器，将算术表达式从中缀式翻译成后缀式。

先确定一个定义算术表达式的文法，为其设计一个语法分析程序，为每条产生式配备一个语义子程序，按照一遍扫描的语法制导翻译方法，实现翻译程序。对用户输入的任意一个正确的算术表达式，程序将其转换成后缀式输出。

### 3. 设计内容

对于将中缀表达式转换成后缀表达式，主要需要利用栈这个数据结构，具体的转换算法如下：

从左到右以此扫描每一个字符：

1. 如果遇到的是数字就直接输出；
2. 如果遇到左括号“(”，就直接压栈；
3. 如果遇到右括号“)”，弹出栈元素直到遇到左括号，并将左括号也弹出；
4. 扫描遇到其他的运算符：
  - 4.1 当前字符的优先级>栈顶字符优先级，直接入栈；
  - 4.2 当前字符的优先级<=栈顶字符优先级，出栈直到遇到左括号（左括号不弹

出),然后将当前的字符入栈;

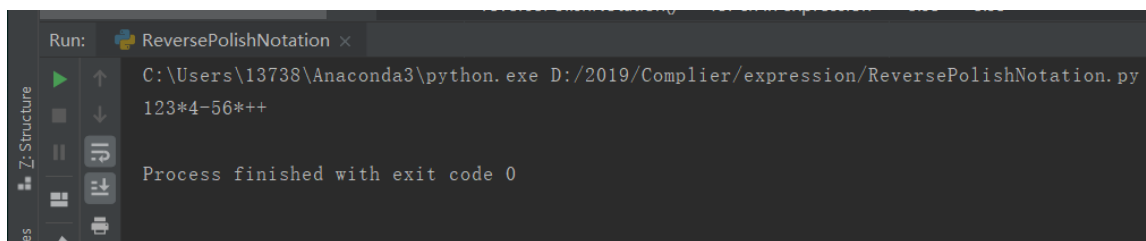
5. 扫描完成,如果栈内仍有元素,出栈直到栈为空。

## 4. 输入输出设计

本次实验比较的简单主要就是进行一个带括号的表达式的转换,因此程序的输入直接在内部进行表达式的定义,测试中使用的表达式为“ $1+(2*3-4)+5*6$ ”。

程序的输出在控制台进行显示。

## 5.运行结果



```
Run: ReversePolishNotation x
C:\Users\13738\Anaconda3\python.exe D:/2019/Compiler/expression/ReversePolishNotation.py
123*4-56*++
Process finished with exit code 0
```

## 6.总结

由于时间原因我选择了这组题目中实现起来比较简单的一个题目,中缀表达式转后缀表达式,并进行了一个基本的实现。实现的过程中主要运用到的就是之前数据结构这门课上学到的一些知识。本次虽然完成了一个基本的转换任务,但是在也还存在下面的一些问题:

1. “-”无法区分是操作数还是运算符;
2. 处理的操作数都是单个的操作数,并不涉及到两位数或者三位数这种操作数。

## 7.附录——核心代码

```
1. # -*-coding:utf-8 -*-
2. # file: ReversePolishNotation.py
3. def reversePolishNotation(expression):
4.     stack = [] # 初始化空栈
5.     result = []
6.     for ch in expression:
7.         if ch.isnumeric():
8.             result.append(ch)
9.         else:
10.            if len(stack) == 0: # 空栈就将符号直接推入
11.                stack.append(ch)
12.            elif ch in '*/':
13.                stack.append(ch)
14.            elif ch is ')': # 出栈直到遇到(
15.                c = stack.pop()
16.                while c is not '(':
17.                    result.append(c)
18.                    c = stack.pop()
19.            elif ch in '+-' and stack[-1] in '*/':
20.                # 如果栈中没有左括号就将所有出栈，加入结果
21.                if stack.count('(') == 0:
22.                    while stack:
23.                        result.append(stack.pop())
24.                else:
25.                    t = stack.pop()
26.                    while t != '(':
27.                        result.append(t)
28.                        t = stack.pop()
29.                    stack.append('(')
30.            stack.append(ch) # 出栈完成后将当前的入栈
31.        else:
32.            stack.append(ch)
33.    # 表达式遍历完成后将栈清空
34.    while stack:
35.        result.append(stack.pop())
36.    return "".join(result)
37. if __name__ == "__main__":
38.     expression = "1+(2*3-4)+5*6"
39.     result = reversePolishNotation(expression)
40.     print(result)
```