## **青岛科技大学实验报告**

姓名： 刘洋 专业： 计算机科学与技术 班级： 计算213

学号： 2108010316 课程： 编译原理 时间： 2024.05.19

实验项目： 实验4 逆波兰式的生成

**实验名称：逆波兰式分析**

实验目的和要求

将非后缀式用来表示的算术表达式转换为用逆波兰式来表示的算术表达式，并计算用逆波兰式来表示的算术表达式的值。

实验内容和步骤：

1. **实验内容**

实现一个程序，能够接收包含加减乘除和括号的算术表达式，将其转换为后缀表达式，并计算出表达式的值。

**二、实验步骤**

1.定义一个堆栈和一个字符串来保存逆波兰式。

2.遍历中缀表达式的每个字符：

如果是数字，则直接加入逆波兰式字符串中。

如果是运算符：

如果是左括号，直接入栈。

如果是右括号，将栈顶元素弹出并加入逆波兰式字符串，直到遇到左括号。

如果是其他运算符，比较其与栈顶运算符的优先级，如果优先级低于或等于栈顶运算符，则将栈顶元素弹出并加入逆波兰式字符串，直到栈顶元素的优先级低于该运算符，然后将该运算符入栈。

3.遍历完整个表达式后，将栈中剩余的运算符依次弹出并加入逆波兰式字符串。

4.使用逆波兰式计算器计算逆波兰式的值。

实验代码如下：

package org.example;  
  
import java.util.Stack;  
  
public class RPNCalculator {  
  
 public static String infixToRPN(String expression) {  
 StringBuilder rpn = new StringBuilder();  
 Stack<Character> stack = new Stack<>();  
  
 for (char c : expression.toCharArray()) {  
 if (Character.*isDigit*(c)) {  
 rpn.append(c);  
 } else if (c == '(') {  
 stack.push(c);  
 } else if (c == ')') {  
 while (!stack.isEmpty() && stack.peek() != '(') {  
 rpn.append(stack.pop());  
 }  
 stack.pop(); // Discard '('  
 } else {  
 while (!stack.isEmpty() && *precedence*(stack.peek()) >= *precedence*(c)) {  
 rpn.append(stack.pop());  
 }  
 stack.push(c);  
 }  
 }  
  
 while (!stack.isEmpty()) {  
 rpn.append(stack.pop());  
 }  
  
 return rpn.toString();  
 }  
  
 public static double evaluateRPN(String rpnExpression) {  
 Stack<Double> stack = new Stack<>();  
  
 for (char c : rpnExpression.toCharArray()) {  
 if (Character.*isDigit*(c)) {  
 stack.push((double)(c - '0'));  
 } else {  
 double b = stack.pop();  
 double a = stack.pop();  
 switch (c) {  
 case '+':  
 stack.push(a + b);  
 break;  
 case '-':  
 stack.push(a - b);  
 break;  
 case '\*':  
 stack.push(a \* b);  
 break;  
 case '/':  
 if (b == 0) {  
 throw new ArithmeticException("Division by zero");  
 }  
 stack.push(a / b);  
 break;  
 case '^':  
 stack.push(Math.*pow*(a, b));  
 break;  
 }  
 }  
 }  
  
 return stack.pop();  
 }  
  
 private static int precedence(char operator) {  
 if (operator == '+' || operator == '-') {  
 return 1;  
 } else if (operator == '\*' || operator == '/') {  
 return 2;  
 } else if (operator == '^') {  
 return 3;  
 }  
 return 0;  
 }  
  
 public static void main(String[] args) {  
 String infixExpr = "3+4\*2/(1-5)^2/0";  
 String rpnExpr = *infixToRPN*(infixExpr);  
 System.*out*.println("中缀表达式: " + infixExpr);  
 System.*out*.println("逆波兰式: " + rpnExpr);  
 System.*out*.println("计算结果: " + *evaluateRPN*(rpnExpr));  
 }  
}

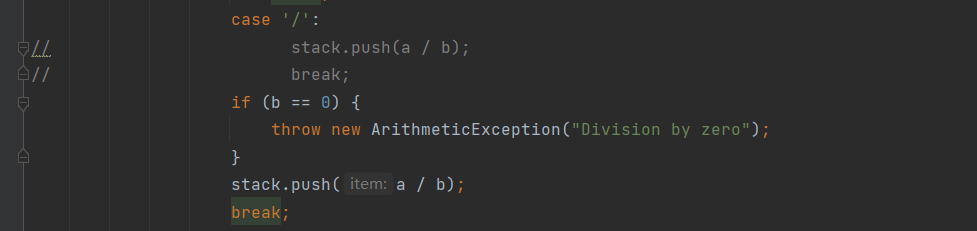
**三、实验过程记录：**

1.在实现逆波兰式转换的过程中，一开始没有考虑到运算符的优先级，导致转换出的逆波兰式不正确。经过分析发现，需要在入栈时判断运算符的优先级，如果当前运算符优先级低于栈顶运算符，则需要将栈顶元素弹出，直到栈顶元素的优先级低于当前运算符，然后再将当前运算符入栈。

2.在处理右括号时，没有考虑到右括号与左括号之间可能存在多个运算符，需要将这些运算符全部弹出并加入到逆波兰式中。解决方法是在遇到右括号时，循环弹出栈顶元素，直到遇到左括号为止。

3.在计算逆波兰式值时，对于操作数的处理需要注意，需要将字符型的数字转换为实际的数字。这里使用了字符型数字减去字符'0'来实现转换。

4.在编写代码时，需要考虑异常情况的处理，如除数为0等情况，避免程序崩溃。

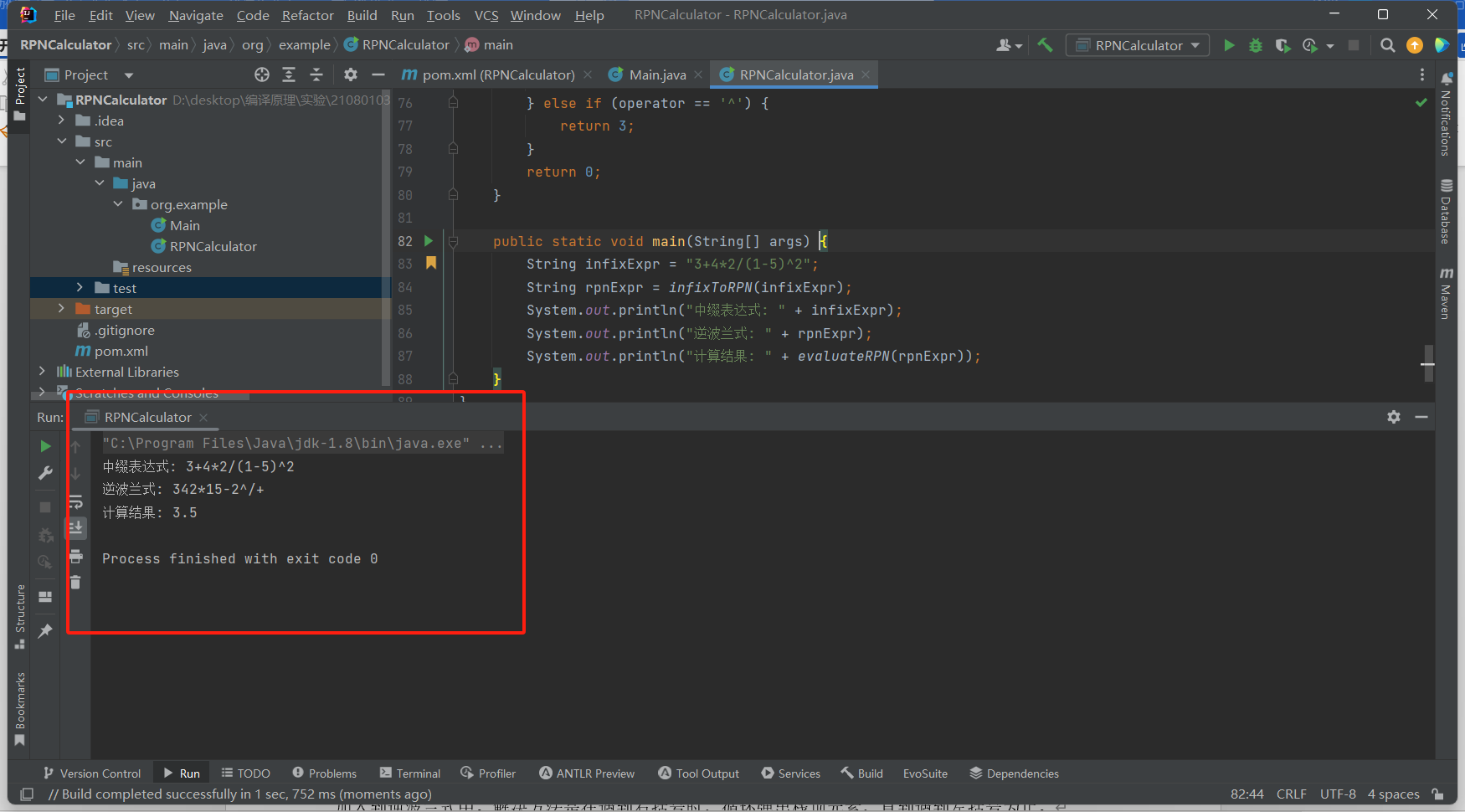


**四、实验结果：**

中缀表达式：3+4\*2/(1-5)^2

转换后的逆波兰式：342\*15-2^/+

计算结果：3.5



**五、实验总结：**

本次实验通过实现逆波兰式生成和计算的算法，加深了我对堆栈数据结构和中缀表达式转换的理解。在实验过程中，我遇到了一些问题，如运算符优先级判断、括号处理等，通过查阅资料和思考，最终成功解决了这些问题。堆栈是处理中缀表达式转换为后缀表达式以及计算后缀表达式的重要数据结构，在实验中起到了关键作用。在将中缀表达式转换为后缀表达式时，需要考虑运算符的优先级，这需要仔细设计算法来确保转换的正确性。