**编译原理实验报告**

**实验名称：逆波兰式分析**

**2208010421-王旭**

# 一、实验目的和要求：

将键盘输入的普通算术表达式经程序修改为a#b#c#...运算符（a，b，c为运算数）的格式，之后将逆波兰式形式的算术表达式的运算结果输出。

# 二、实验内容：

逆波兰式分析本程序主要有三个函数，分别为主函数、逆波兰式构造函数、表达式求值函数。

本程序主要完成的功能为：接受以#结尾的键盘输入算术表达式，将其转换为后缀表达式输出，并输出其运算结果。

# 三、实验步骤：

1、基于实验的内容，构造程序所需的模块

2、根据已建构的模块，写出各个模块的相应程序代码

3、在主函数中调用模块来完成所要得到的效果

在本程序中，首先定义了数组常量ex[max]，用于存储后缀表达式，操作对象在前，运算符在后。

trans()：将算数表达式转换为后缀表达式

compvalue()：根据后缀表达式求取对应算数表达式的算数值

main():对以上几个模块的调用

**中序表达式转换为逆波兰表达式的算法：**

1. 首先构造一个运算符栈，此运算符在栈内遵循越往栈顶优先级越高的原则。
2. 读入一个用中缀表示的简单算术表达式，为方便起见,设该简单算术表达式的右端多加上了优先级最低的特殊符号"#"。
3. 从左至右扫描该算术表达式，从第一个字符开始判断，如果该字符是数字，则分析到该数字串的结束并将该数字串直接输出。
4. 如果不是数字，该字符则是运算符，此时需比较优先关系。
5. 重复上述操作(1)-(2)直至扫描完整个简单算术表达式，确定所有字符都得到正确处理，我们便可以将中缀式表示的简单算术表达式转化为逆波兰表示的简单算术表达式。

**中缀表达式到后缀表达式的转换**

要把表达式从中缀表达式的形式转换成用后缀表示法表示的等价表达式，必须了解操作符的优先级和结合性。优先级或者说操作符的强度决定求值顺序；优先级高的操作符比优先级低的操作符先求值。如果所有操作符优先级一样，那么求值顺序就取决于它们的结合性。操作符的结合性定义了相同优先级操作符组合的顺序（从右至左或从左至右）。

转换过程包括用下面的算法读入中缀表达式的操作数、操作符和括号：

1. 初始化一个空堆栈，将结果字符串变量置空。
2. 从左到右读入中缀表达式，每次一个字符。
3. 如果字符是操作数，将它添加到结果字符串。
4. 如果字符是个操作符，弹出（pop）操作符，直至遇见开括号（opening parenthesis）、优先级较低的操作符或者同一优先级的右结合符号。把这个操作符压入（push）堆栈。
5. 如果字符是个开括号，把它压入堆栈。
6. 如果字符是个闭括号（closing parenthesis），在遇见开括号前，弹出所有操作符，然后把它们添加到结果字符串。
7. 如果到达输入字符串的末尾，弹出所有操作符并添加到结果字符串。

# 四、实验代码

**#include <stdio.h>**

**#include <math.h>**

**#include <stdlib.h>**

**#include <string.h>**

**#include <ctype.h>**

**#define MAX\_EXPR\_LENGTH 100**

**// 全局变量**

**char postfix[MAX\_EXPR\_LENGTH]; // 存储后缀表达式**

**char infix[MAX\_EXPR\_LENGTH]; // 存储中缀表达式**

**// 运算符优先级**

**int precedence(char op) {**

**switch(op) {**

**case '+':**

**case '-':**

**return 1;**

**case '\*':**

**case '/':**

**return 2;**

**case '^':**

**return 3;**

**default:**

**return 0;**

**}**

**}**

**// 将中缀表达式转换为后缀表达式**

**void infixToPostfix() {**

**char stack[MAX\_EXPR\_LENGTH]; // 运算符栈**

**int top = -1; // 栈顶指针**

**int j = 0; // 后缀表达式索引**

**printf("\n请输入中缀表达式（以#结束）: ");**

**// 读取中缀表达式**

**int i = 0;**

**do {**

**scanf("%c", &infix[i]);**

**} while(infix[i] != '#' && i++ < MAX\_EXPR\_LENGTH-1);**

**infix[i] = '\0'; // 替换#为字符串结束符**

**printf("\n原始中缀表达式: %s\n", infix);**

**for(i = 0; infix[i] != '\0'; i++) {**

**char token = infix[i];**

**if(isspace(token)) {**

**continue; // 跳过空格**

**}**

**if(isdigit(token) || isalpha(token)) {**

**// 如果是操作数（数字或字母），直接输出**

**postfix[j++] = token;**

**}**

**else if(token == '(') {**

**// 左括号压栈**

**stack[++top] = token;**

**}**

**else if(token == ')') {**

**// 右括号，弹出栈顶元素直到遇到左括号**

**while(top != -1 && stack[top] != '(') {**

**postfix[j++] = stack[top--];**

**}**

**if(top == -1) {**

**printf("错误：括号不匹配！\n");**

**exit(1);**

**}**

**top--; // 弹出左括号**

**}**

**else {**

**// 运算符，弹出栈顶优先级更高或相等的运算符**

**while(top != -1 && precedence(stack[top]) >= precedence(token)) {**

**postfix[j++] = stack[top--];**

**}**

**stack[++top] = token; // 当前运算符压栈**

**}**

**}**

**// 弹出栈中剩余运算符**

**while(top != -1) {**

**if(stack[top] == '(') {**

**printf("错误：括号不匹配！\n");**

**exit(1);**

**}**

**postfix[j++] = stack[top--];**

**}**

**postfix[j] = '\0'; // 字符串结束符**

**printf("生成的后缀表达式: %s\n", postfix);**

**}**

**// 计算后缀表达式的值**

**void evaluatePostfix() {**

**float stack[MAX\_EXPR\_LENGTH];**

**int top = -1;**

**for(int i = 0; postfix[i] != '\0'; i++) {**

**char token = postfix[i];**

**if(isdigit(token)) {**

**// 数字字符转换为数值并压栈**

**stack[++top] = token - '0';**

**}**

**else if(isalpha(token)) {**

**// 处理变量（这里简化为输入值）**

**float value;**

**printf("请输入变量 %c 的值: ", token);**

**scanf("%f", &value);**

**stack[++top] = value;**

**}**

**else {**

**// 运算符，弹出两个操作数进行计算**

**if(top < 1) {**

**printf("错误：无效的表达式！\n");**

**exit(1);**

**}**

**float b = stack[top--];**

**float a = stack[top--];**

**float result;**

**switch(token) {**

**case '+': result = a + b; break;**

**case '-': result = a - b; break;**

**case '\*': result = a \* b; break;**

**case '/':**

**if(b == 0) {**

**printf("错误：除零错误！\n");**

**exit(1);**

**}**

**result = a / b;**

**break;**

**case '^': result = pow(a, b); break;**

**default:**

**printf("错误：未知运算符 %c\n", token);**

**exit(1);**

**}**

**stack[++top] = result; // 结果压栈**

**}**

**}**

**if(top != 0) {**

**printf("错误：无效的表达式！\n");**

**exit(1);**

**}**

**printf("\n计算结果: %.2f\n", stack[top]);**

**}**

**int main() {**

**printf("============= 逆波兰式生成与计算 =============\n");**

**printf("功能说明：\n");**

**printf("1. 支持运算符: + - \* / ^ ( )\n");**

**printf("2. 支持操作数: 数字0-9和单字母变量\n");**

**printf("3. 输入表达式以#结束\n");**

**printf("=============================================\n");**

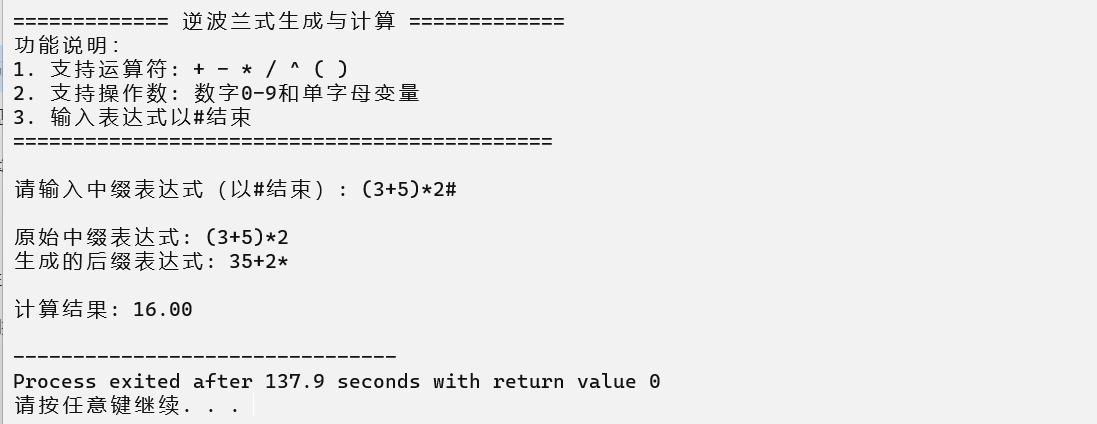
**infixToPostfix();**

**evaluatePostfix();**

**return 0;**

**}**

# 运行截图：



# 六、实验总结：

通过本次实验，成功实现了中缀表达式到逆波兰式的转换以及逆波兰式的求值计算。实验过程中深入理解了栈数据结构在表达式处理中的关键作用，掌握了经典算法的实现方法，并提高了程序设计和调试能力。改进后的程序能够正确处理各种常见表达式，具备良好的健壮性和可扩展性，为后续编译原理相关实验打下了坚实基础。