## 实验三 逆波兰式的产生及计算

**一、实验目的：**

**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。**

**二、实验预习提示**

**1、逆波兰式定义**

**将运算对象写在前面，而把运算符号写在后面。用这种表示法表示的表达式也称做后缀式。逆波兰式的特点在于运算对象顺序不变，运算符号位置反映运算顺序。采用逆波兰式可以很好的表示简单算术表达式，其优点在于易于计算机处理表达式。**

**2、产生逆波兰式的前提**

**中缀算术表达式**

**3、逆波兰式生成的实验设计思想及算法**

否

出错处理

否

否

是

是

是

是

否

是

否

输入一个中缀式表示的简单运算表达式

‘#’入栈

sym=当前输入符号

sym是数字吗?

对数字进行处理，形成一个数字串

栈顶运算符优先级低于sym吗?

将向前看符号入栈

栈顶运算符与sym优先级相等吗?

将栈顶运算符弹出，且输出

栈顶运算符优先级高于sym吗

栈顶是’(‘且sym为’）吗

栈顶运算符出栈

程序结束

**(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。**

**做法如下：将该字符与运算符栈顶的运算符的优先关系相比较。如果，该字符优先关系高于此运算符栈顶的运算符，则将该运算符入栈。倘若不是的话，则将此运算符栈顶的运算符从栈中弹出，将该字符入栈。**

**(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。**

**3、逆波兰式计算的实验设计思想及算法**

读入一个逆波兰算术表达式

sym=当前输入符号

sym是运算符吗?

将该字符入栈

根据运算符的特点从栈顶部取出若干个运算对象进行该运算将运算结果入栈

程序结束

否

是

Sym=‘#’

否

是

**(1)构造一个栈，存放运算对象。**

**(2)读入一个用逆波兰式表示的简单算术表达式。**

**(3)自左至右扫描该简单算术表达式并判断该字符，如果该字符是运算对象，则将该字符入栈。若是运算符，如果此运算符是二目运算符，则将对栈顶部的两个运算对象进行该运算，将运算结果入栈，并且将执行该运算的两个运算对象从栈顶弹出。如果该字符是一目运算符，则对栈顶部的元素实施该运算，将该栈顶部的元素弹出，将运算结果入栈。**

**(4)重复上述操作直至扫描完整个简单算术表达式的逆波兰式，确定所有字符都得到正确处理，我们便可以求出该简单算术表达式的值。**

**三、实验过程和指导：**

**（一）准备：**

**1.阅读课本有关章节，**

**2.考虑好设计方案；**

**3.设计出模块结构、测试数据，初步编制好程序。**

**（二）上课上机：**

**将源代码拷贝到机上调试，发现错误，再修改完善。第二次上机调试通过。**

**（三）程序要求：**

**程序输入/输出示例：**

**输出的格式如下：**

**(1)逆波兰式的生成及计算程序，编制人：姓名，学号，班级**

**(2)输入一以#结束的中缀表达式(包括+—\*/（）数字#)：在此位置输入符号串如(28+68)\*2#**

**(3)逆波兰式为：28&68+2\***

**(4)逆波兰式28&68+2\*计算结果为192**

**备注：(1)在生成的逆波兰式中如果两个数相连则用&分隔，如28和68，中间用&分隔；**

**(2)在此位置输入符号串为用户自行输入的符号串。**

**注意：1.表达式中允许使用运算符（+-\*/）、分割符（括号）、数字，结束符#；**

**2.如果遇到错误的表达式，应输出错误提示信息（该信息越详细越好）；**

**3.对学有余力的同学，测试用的表达式事先放在文本文件中，一行存放一个表达式，同时以分号分割。同时将预期的输出结果写在另一个文本文件中，以便和输出进行对照；**

**（四）程序思路（仅供参考）：**

**模块结构：**

**（1）定义部分：定义常量、变量、数据结构。**

**（2）初始化：设立算符优先分析表、初始化变量空间（包括堆栈、结构体、数组、临时变量等）；**

**（3）控制部分：从键盘输入一个表达式符号串；**

**（4）利用算符优先分析算法进行表达式处理：根据算符优先分析表对表达式符号串进行堆栈（或其他）操作，输出分析结果，如果遇到错误则显示错误信息。**

**（5）对生成的逆波兰式进行计算。**

**（五）练习该实验的目的和思路：**

**程序较复杂，需要利用到程序设计语言的知识和大量编程技巧，逆波兰式的生成是算符优先分析法的应用，是一种较实用的分析法，通过这个练习可大大提高软件开发能力。**

**（六）为了能设计好程序，注意以下事情：**

**1.模块设计：将程序分成合理的多个模块（函数），每个模块做具体的同一事情。**

**2.写出（画出）设计方案：模块关系简图、流程图、全局变量、函数接口等。**

**3.编程时注意编程风格：空行的使用、注释的使用、缩进的使用等。**

**四、上交：**

**1.程序源代码（上交软盘）；**

**2.已经测试通过的测试数据3组；**

**3.实验报告：**

**实验名称**

**实验目的和要求**

**（一）实验内容**

**（1）功能描述：该程序具有什么功能？**

**（2）程序结构描述：函数调用格式、参数含义、返回值描述、函数功能；函数之间的调用关系图。**

**（3）程序总体执行流程图**

**（二）实验过程记录：出错次数、出错严重程度、解决办法摘要。**

**（三）实验总结：你在编程过程中花时多少？多少时间在纸上设计？多少时间上机输入和调试？多少时间在思考问题？遇到了哪些难题？你是怎么克服的？你对你的程序的评价？你的收获有哪些？**

**五、参考源代码**

**#include<stdio.h>**

**#include<math.h>**

**#define max 100**

**char ex[max]; /\*存储后缀表达式\*/**

**void trans(){ /\*将算术表达式转化为后缀表达式\*/**

**char str[max]; /\*存储原算术表达式\*/**

**char stack[max]; /\*作为栈使用\*/**

**char ch;**

**int sum,i,j,t,top=0;**

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

**printf("\*输入一个求值的表达式，以#结束。\*\n");**

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

**printf("算数表达式：");**

**i=0; /\*获取用户输入的表达式\*/**

**do{**

**i++;**

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

**}while(str[i]!='#' && i!=max);**

**sum=i;**

**t=1;i=1;**

**ch=str[i];i++;**

**while(ch!='#'){**

**switch(ch){**

**case '(': /\*判定为左括号\*/**

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

**break;**

**case ')': /\*判定为右括号\*/**

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

**ex[t]=stack[top];top--;t++;**

**}**

**top--;**

**break;**

**case '+': /\*判定为加减号\*/**

**case '-':**

**while(top!=0&&stack[top]!='('){**

**ex[t]=stack[top];top--;t++;**

**}**

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

**break;**

**case '\*': /\*判定为乘除号\*/**

**case '/':**

**while(stack[top]=='\*'||stack[top]=='/'){**

**ex[t]=stack[top];top--;t++;**

**}**

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

**break;**

**case ' ':break;**

**default:while(ch>='0'&&ch<='9'){ /\*判定为数字\*/**

**ex[t]=ch;t++;**

**ch=str[i];i++;**

**}**

**i--;**

**ex[t]='#';t++;**

**}**

**ch=str[i];i++;**

**}**

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

**ex[t]=stack[top];t++;top--;**

**}**

**ex[t]='#';**

**printf("\n\t原来表达式：");**

**for(j=1;j<sum;j++)**

**printf("%c",str[j]);**

**printf("\n\t后缀表达式：",ex);**

**for(j=1;j<t;j++)**

**printf("%c",ex[j]);**

**}**

**void compvalue(){ /\*计算后缀表达式的值\*/**

**float stack[max],d; /\*作为栈使用\*/**

**char ch;**

**int t=1,top=0; /\*t为ex下标，top为stack下标\*/**

**ch=ex[t];t++;**

**while(ch!='#'){**

**switch(ch){**

**case '+':**

**stack[top-1]=stack[top-1]+stack[top];**

**top--;**

**break;**

**case '-':**

**stack[top-1]=stack[top-1]-stack[top];**

**top--;**

**break;**

**case '\*':**

**stack[top-1]=stack[top-1]\*stack[top];**

**top--;**

**break;**

**case '/':**

**if(stack[top]!=0)**

**stack[top-1]=stack[top-1]/stack[top];**

**else{**

**printf("\n\t除零错误!\n");**

**exit(0); /\*异常退出\*/**

**}**

**top--;**

**break;**

**default:**

**d=0;**

**while(ch>='0'&&ch<='9'){**

**d=10\*d+ch-'0'; /\*将数字字符转化为对应的数值\*/**

**ch=ex[t];t++;**

**}**

**top++;**

**stack[top]=d;**

**}**

**ch=ex[t];t++;**

**}**

**printf("\n\t计算结果:%g\n",stack[top]);**

**}**

**main(){**

**trans();**

**compvalue();**

**}**