#include<stdio.h>

#include<malloc.h>

#include "源.h"

#define MaxSize 50

typedef char ElemType;

struct {

char ch;//运算符

int pri;//优先级

}lpri[] = { {'=',0},{'(',1},{'\*',5},{'/',5},{'+',3},{'-',3},{')',8} ,{'^',7}},

rpri[] = { {'=',0},{'(',8},{'\*',4},{'/',4},{'+',2},{'-',2},{')',1},{'^',6}};

typedef struct SqStack {

ElemType data[MaxSize];

int top;

}SqStack;

void InitStack(SqStack \*&S) {

S = (SqStack \*)malloc(sizeof(SqStack));

S->top = -1;

}

bool Push(SqStack\*& S, ElemType e) {//进栈

if (S->top == MaxSize - 1) {

return false;

}

S->top++;

S->data[S->top] = e;

return true;

}

bool Pop(SqStack\*& S, ElemType& e) {//出栈

if (S->top == -1) {

return false;

}

e = S->data[S->top];

S->top--;

return true;

}

int leftpri(char op) {//左运算符优先级

int i;

for (i = 0;i < 8;i++) {

if (lpri[i].ch == op) {

return lpri[i].pri;

}

}

}

int rightpri(char op) {//右运算符优先级

int i;

for (i = 0;i < 8;i++) {

if (rpri[i].ch == op) {

return rpri[i].pri;

}

}

}

bool inop(char ch) {//判断ch是否为运算符

if (ch == '(' || ch == ')' || ch == '+' || ch == '-' || ch == '\*' || ch == '/' || ch == '^') {

return true;

}

else {

return false;

}

}

int Precede(char op1, char op2) {//op1和op2运算符优先级的比较结果

if (leftpri(op1) == rightpri(op2)) {

return 0;

}

else if (leftpri(op1) < rightpri(op2)) {

return -1;

}

else {

return 1;

}

}

void trans(char\* exp, char postexp[]) {//算术转换为后缀表达式

struct {

char data[MaxSize];

int top;

}op;

int i = 0;

op.top = -1;

op.top++;

op.data[op.top] = '=';

while (\*exp != '\0') {//处理每一个字符

if (inop(\*exp)==false) {//判断为数字时

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

postexp[i++] = \*exp;

exp++;

}

postexp[i++] = '#';

}

else {//为运算符的情况

switch (Precede(op.data[op.top], \*exp)) {

case -1://进栈

op.top++;

op.data[op.top] = \*exp;

exp++;

break;

case 0://只有括号满足这种情况

op.top--;

exp++;

break;

case 1://退栈

postexp[i++] = op.data[op.top];

op.top--;

break;

}

}

}

while (op.data[op.top] != '=') {

postexp[i++] = op.data[op.top];

op.top--;

}

postexp[i] = '\0';

}

float compvalue(char exp[]) {//后缀表达式求值

struct {

float data[MaxSize];

int top;

}st;

st.top = -1;

float d;

char ch;

int t = 0;

ch = exp[t];

t++;

while (ch != '\0') {

int h;

switch (ch) {

case'+':

st.data[st.top - 1] = st.data[st.top - 1] + st.data[st.top];

st.top--;

break;

case'-':

st.data[st.top - 1] = st.data[st.top - 1] - st.data[st.top];

st.top--;

break;

case'\*':

st.data[st.top - 1] = st.data[st.top - 1] \* st.data[st.top];

st.top--;

break;

case'/':

st.data[st.top - 1] = st.data[st.top - 1]/st.data[st.top];

st.top--;

break;

case'^':

h = st.data[st.top - 1];

while (st.data[st.top] != 1) {

st.data[st.top-1] = st.data[st.top-1] \*h;

st.data[st.top]--;

}

st.top--;

break;

default:

d = 0;

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

d = 10 \* d + ch - '0';

ch = exp[t];

t++;

}

st.top++;

st.data[st.top] = d;

}

ch = exp[t];

t++;

}

return st.data[st.top];

}

int main() {

float b=0.0;

char exp[] = "(1+2)^3-5\*5";

char postexp[MaxSize] = {'\0'};

trans(exp, postexp);

printf("中缀表达式:%s\n", exp);

printf("后缀表达式:%s\n", postexp);

compvalue(postexp);

printf("表达式的值:%g\n", compvalue(postexp));

return 0;

}



