**实验7 中缀表达式转后缀表达式，并计算表达式的值**

【问题描述】

计算用运算符后缀法表示的表达式的值。后缀表达式也称逆波兰表达式，比中缀表达式计算起来更方便简单些，中缀表达式要计算就存在着括号的匹配问题，所以在计算表达式值时一般都是先转换成后缀表达式，再用后缀法计算表达式的值。如：表达式(a+b\*c)/d-e用后缀法表示应为abc\*+d/e-。只考虑四则算术运算，且假设输入的操作数均为1位十进制数（0~9），并且输入的后缀形式表达式不含语法错误。

【数据描述】

#define add 43 //运算符加号'+'的ASCII码

#define subs 45 //运算符减号'-'的ASCII码

#define mult 42 //运算符乘号'\*'的ASCII码

#define div 47 //运算符除号'/'的ASCII码

#define MAXSIZE 100

typedef struct{

int stkdata[MAXSIZE]; //用数组来表示栈空间，定义长度为MAXSIZE的栈

int top; //栈顶指针

}STKzone;

typedef STKzone \*STK;

typedef enum{ true=1,false=0 } bool;

typedef enum{ ok,error } status;

【算法描述】

用流程图表示见图3-3。

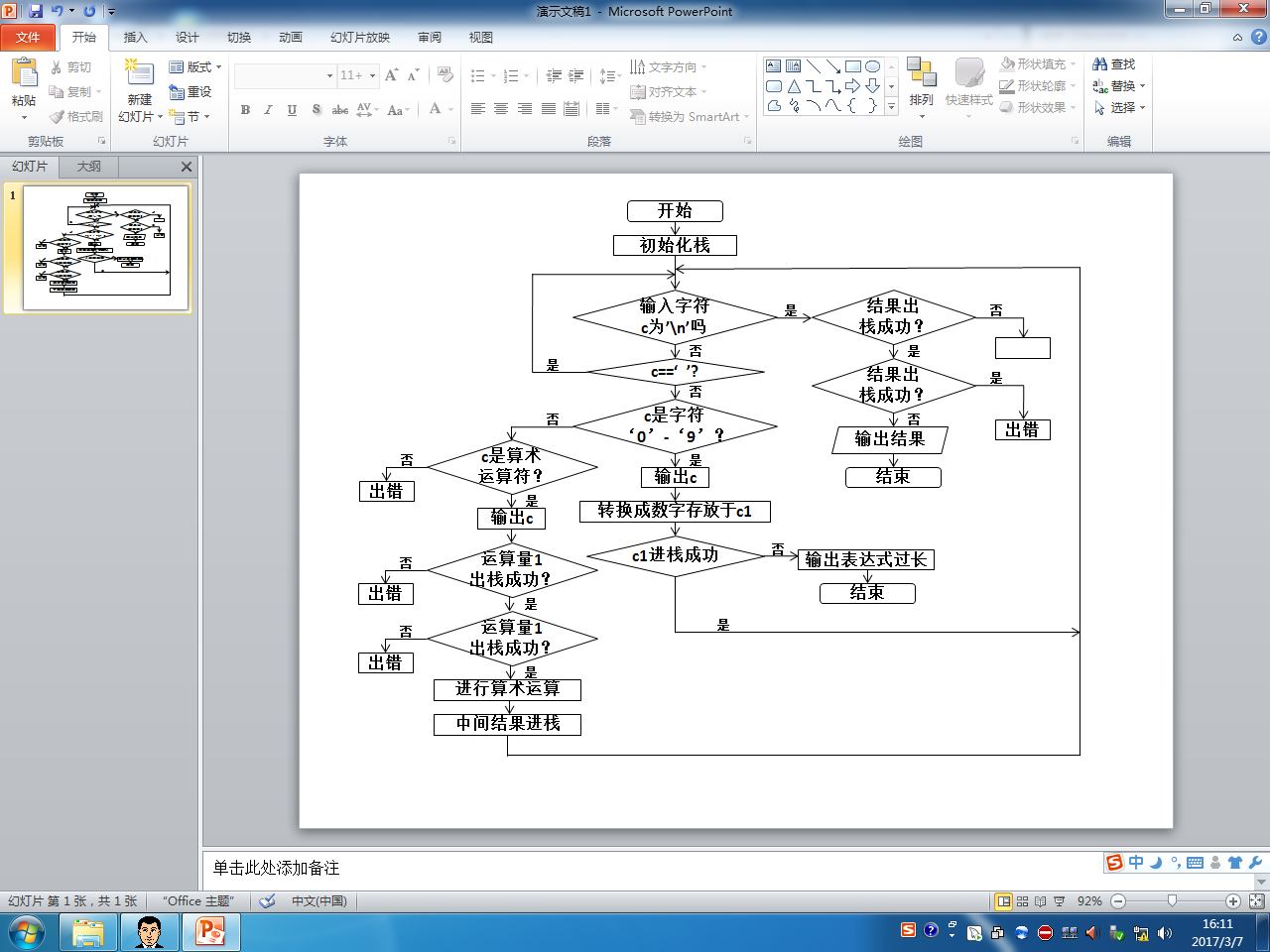


图3-3 计算表达式值的流程图

【C源程序】

#include "stdio.h"

#include "stdlib.h"

#define add 43 //运算符加号'+'的ASCII码

#define subs 45 //运算符减号'-'的ASCII码

#define mult 42 //运算符乘号'\*'的ASCII码

#define div 47 //运算符除号'/'的ASCII码

#define MAXSIZE 100

typedef struct{

int stkdata[MAXSIZE]; //用数组来表示栈空间，定义长度为MAXSIZE的堆栈

int top; //栈顶

}STKzone;

typedef STKzone \*STK;

typedef enum{ok,error}status;

STKzone expSTKzone;

STK expSTK;

STK initSTK(STKzone \*stack\_zone){ //执行栈初始化，建栈指针

STK p;

p=stack\_zone;

p->top=0;

return p;

}

status push(int \*term,STK pstk){ //将一结构型数据送入栈中

if(pstk->top==MAXSIZE)

return error; //栈满，进栈失败

pstk->stkdata[pstk->top]=\*term;

(pstk->top)++; //栈顶指针移动

return ok;

}//push

int emptySTK(STK pstk){ //判断栈是否为空栈

return(pstk->top==0);

}

status pop(int \*pdata, STK pstk){ //从栈中取出一结构型数据

if(emptySTK(pstk))

return error;

(pstk->top)--; //退栈

\*pdata=pstk->stkdata[pstk->top];

return ok;

}

void synerror(){

printf("\n表达式语法错!\n");

exit(0);

}

int eval(char tag,int a1,int a2){

switch(tag){

case add: return(a1+a2);

case subs: return(a1-a2);

case mult: return(a1\*a2);

case div: return(a1/a2);

}

return ok;

}

void main(){

char c;

int opd1,opd2,temp;

int c1;

expSTK=initSTK(&expSTKzone);

printf("\n后置表达式( 操作数用逗号间隔 ): ");

while((c=getchar())!='\n'){

if((c>47)&&(c<58)){ //判断是否是0-9的字符

c1=0;

while(c!=','){ //把输入的字符型数字转换成数字

c1=10\*c1+(c-48);

c=getchar();

}

printf("%d ",c1);

if(push(&c1,expSTK)==error){ //运算分量进栈

printf("\n表达式太长\n");

exit(0);

}

}

else

if((c==add)||(c==subs)||(c==mult)||(c==div)){

printf("%c ",c);

if(pop(&opd1,expSTK)==error) //将运算量1出栈

synerror();

if(pop(&opd2,expSTK)==error) //将运算量2出栈

synerror();

temp=eval(c,opd2,opd1); //计算得到结果

push(&temp,expSTK); //将运算结果进栈

}

else

synerror(); //出现非法字符

}//while

if(pop(&opd1,expSTK)==error)

synerror();

if(!(emptySTK(expSTK)))

synerror();

printf("= %-7d\n",opd1);

}//main\_end

【测试数据】

输入：20,30,+

输出：20 30 + = 50  （即求2+3的结果）

输入：100,4,50,\*+3,/120,-

输出：1 4 5 \* + 3 / 3 - = -20 （即求(100+4\*50)/3-120的结果）

【说明】

本算法中对后置法表示的表达式求值按如下规则进行：自左向右扫描，每遇到一个n+1元组( opd1,opd2,…,opdn,opr )（其中opd为操作数，opr为n元运算符），就计算一次opr( opd1,opd2,…,opdn )的值，其结果取代原来表达式中n+1元组的位置，再从表达式开头重复上述过程，直到表达式中不含运算符为止。

【实验题】

用栈实现一般表达式（即中缀表达式）到后缀表达式的转换

【要求】

将该程序与上述源码合并，完成：输入中缀表达式转换成后缀表达式，并对后缀表达式求值。