#include <stdio.h>

#include <stdlib.h>

#define TRUE 1

#define FALSE 0

#define OK 1

#define ERROR 0

#define INFEASIBLE -1

#define OVERFLOW -2

#define STACK\_INIT\_SIZE 100 //存储空间初始分配量

#define STACKINCREMENT 10 //存储空间分配增量

typedef char SElemType;

typedef char OperandType; //表达式求值的运算类型

typedef int Status;

typedef struct

{

SElemType \*base;

SElemType \*top;

int stacksize;

}SqStack;

//构造一个空栈

Status InitStack(SqStack \*S)

{

S->base = (SElemType \*)malloc(STACK\_INIT\_SIZE \* sizeof(SElemType));

if(!S->base)

{

printf("内存分配失败!\n");

exit(OVERFLOW);

}

S->top = S->base;

S->stacksize = STACKINCREMENT;

return OK;

}

//若栈不为空，则用e返回S的栈顶元素，并返回OK；否则返回ERROR

Status GetTop(SqStack \*S, SElemType \*e)

{

if(S->top == S->base)

return ERROR;

\*e = \*(S->top - 1);

return OK;

}

//插入元素e为新的栈顶元素

Status Push(SqStack \*S, SElemType e)

{

if(S->top - S->base >= STACK\_INIT\_SIZE) //栈满， 追加存储空间

{

S->base = (SElemType \*)realloc(S->base, (S->stacksize + STACKINCREMENT) \* sizeof(SElemType));

if(!S->base)

{

printf("内存分配失败!\n");

exit(OVERFLOW);

}

S->top = S->base + S->stacksize;

S->stacksize += STACKINCREMENT;

}

\*S->top++ = e;

return OK;

}

//若栈不为空，则删除S的栈顶元素，用e返回其值，并返回Ok；否则返回ERROR

Status Pop(SqStack \*S, SElemType \*e)

{

if(S->top == S->base)

return ERROR;

\*e = \*--S->top;

return OK;

}

//销毁栈S，使其不复存在

Status StackDestroy(SqStack \*S)

{

free(S->base);

S->base = NULL;

S->top = NULL;

S->stacksize = 0;

return OK;

}

//清空栈S，保留栈底指针

void ClearStack(SqStack \*S)

{

S->top = S->base;

}

//根据教科书表3.1，判断两符号的优先关系

char Precede(char t1, char t2){

int i,j;

char pre[][7]={

//运算符之间的优先级制作成一张表格

{'>','>','<','<','<','>','>'},

{'>','>','<','<','<','>','>'},

{'>','>','>','>','<','>','>'},

{'>','>','>','>','<','>','>'},

{'<','<','<','<','<','=','0'},

{'>','>','>','>','0','>','>'},

{'<','<','<','<','<','0','='}};

switch(t1){

case '+': i=0; break;

case '-': i=1; break;

case '\*': i=2; break;

case '/': i=3; break;

case '(': i=4; break;

case ')': i=5; break;

case '=': i=6; break;

}

switch(t2){

case '+': j=0; break;

case '-': j=1; break;

case '\*': j=2; break;

case '/': j=3; break;

case '(': j=4; break;

case ')': j=5; break;

case '=': j=6; break;

}

return pre[i][j];

}

//判断c是否为运算符

Status In(OperandType c)

{

switch(c)

{

case '+':

case '-':

case '\*':

case '/':

case '(':

case ')':

case '=':

return TRUE;

default:

return FALSE;

}

}

//二元运算(a theta b)

OperandType Operate(OperandType a, OperandType theta, OperandType b)

{

OperandType c;

switch(theta)

{

case '+':

c = a + b;

break;

case '-':

c = a - b;

break;

case '\*':

c = a \* b;

break;

case '/':

c = a / b;

break;

}

return c;

}

//算术表达式求值的算符优先算法，设OPTR和OPND分别为运算符栈和运算数栈，OP为运算符集合

OperandType EvaluateExpression()

{

SqStack OPTR, OPND;

OperandType a, b, d, x, theta;

char c; //存放有键盘输入的字符串

char z[6]; //存放整数字符串

int i;

;

InitStack(&OPTR); //初始化运算符栈

Push(&OPTR, '='); //=是表达式结束符

InitStack(&OPND); //初始化运算数栈

c = getchar();

GetTop(&OPTR, &x);

while(c != '=' || x != '=')

{

if(In(c)) //是7种运算符之一

{

switch(Precede(x, c))

{

case '<': //当前已经压栈一个运算符（x）比后一个运算符（c）低时，就将c压栈

Push(&OPTR, c);

c = getchar();

break;

case '=':

Pop(&OPTR, &x); //脱括号并接收下一字符

c = getchar();

break;

case '>':

Pop(&OPTR, &theta); //退栈并将运算结果压入OPND中

Pop(&OPND, &b);

Pop(&OPND, &a);

Push(&OPND, Operate(a, theta, b));

break;

}

}

else if(c >= '0' && c <= '9') //c是操作数

{

i = 0;

do

{

z[i] = c;

i ++;

c = getchar();

}while(c >= '0' && c <= '9');

z[i] = 0;

d = atoi(z); //将字符数组转为整型存于d

Push(&OPND, d);

}

else //c为非法字符

{

printf("ERROR3\n");

exit(1);

}

GetTop(&OPTR, &x);

}

GetTop(&OPND, &x);

StackDestroy(&OPTR);

StackDestroy(&OPND);

return x;

}

int main()

{

printf("请输入算术表达式:\n");

printf("%d\n", EvaluateExpression());

return 0;

}