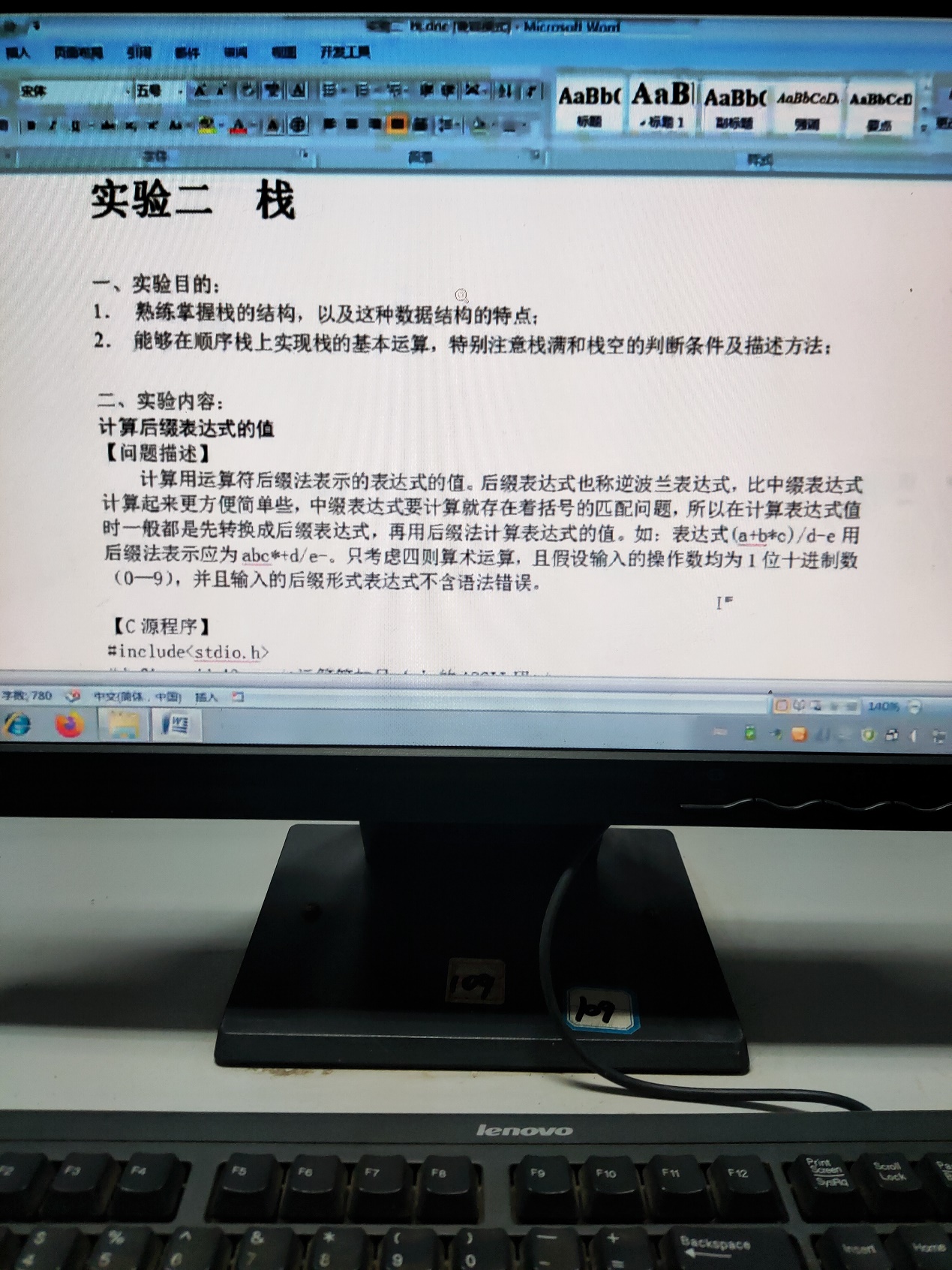
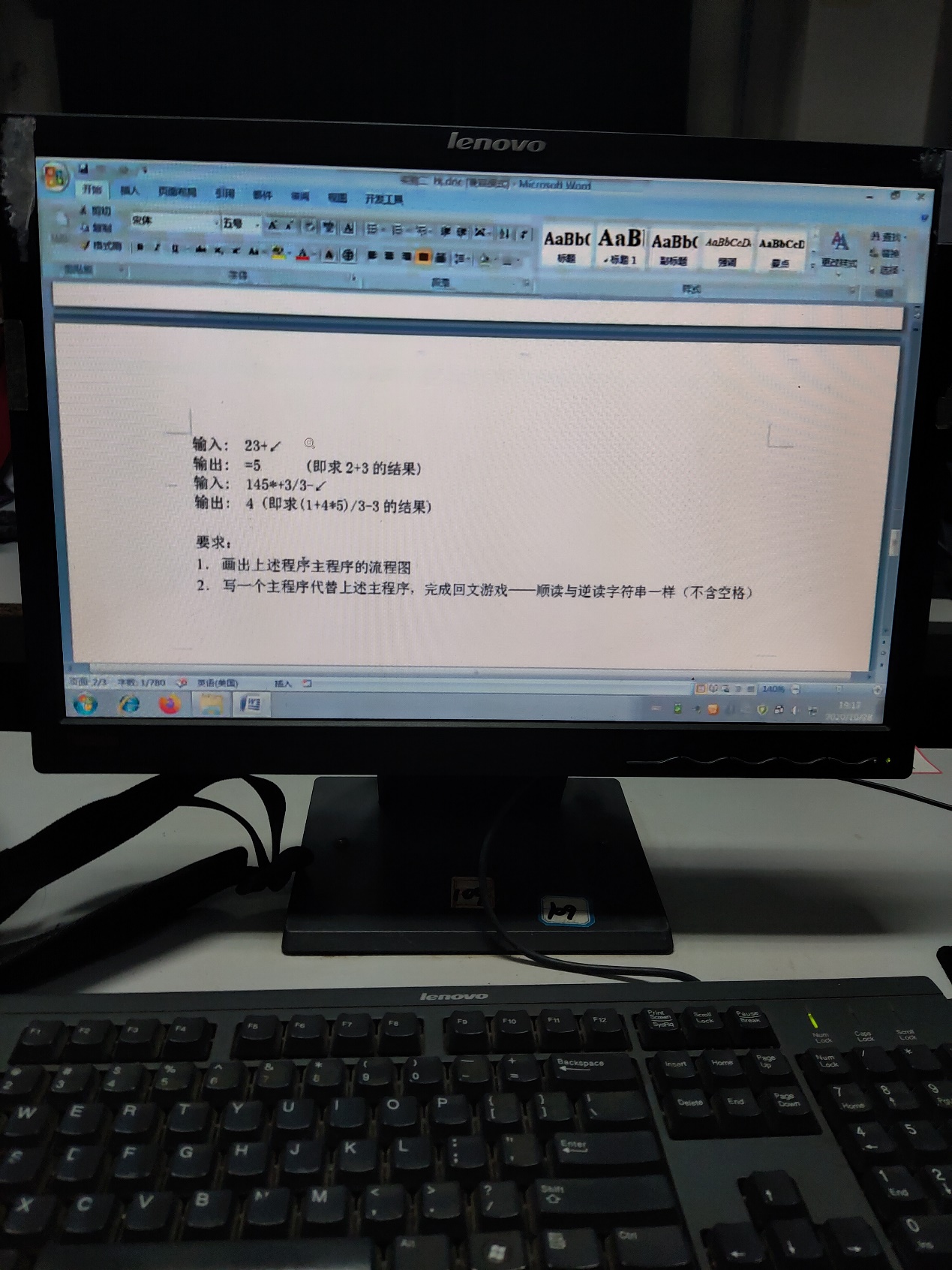
本次实验主要进行了栈的应用，以及一些相关函数的调用：

具体题目信息如下：

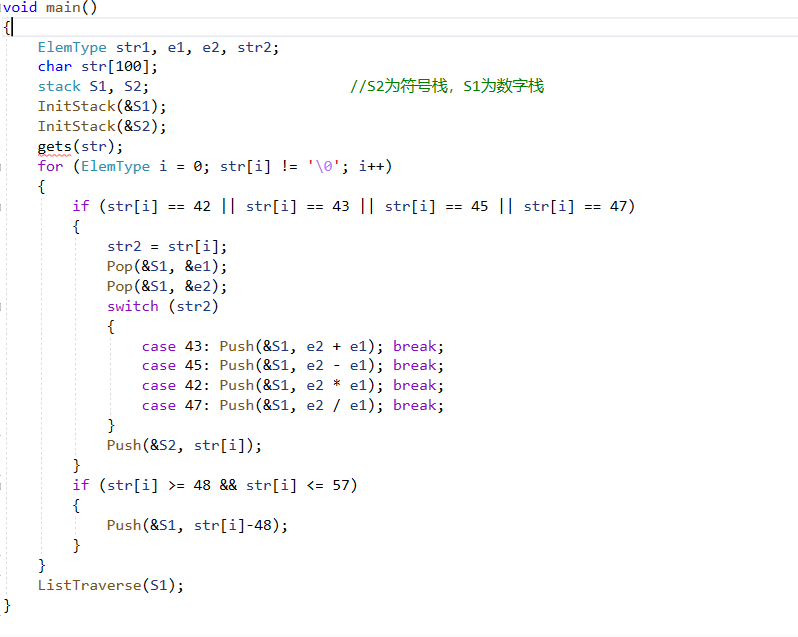




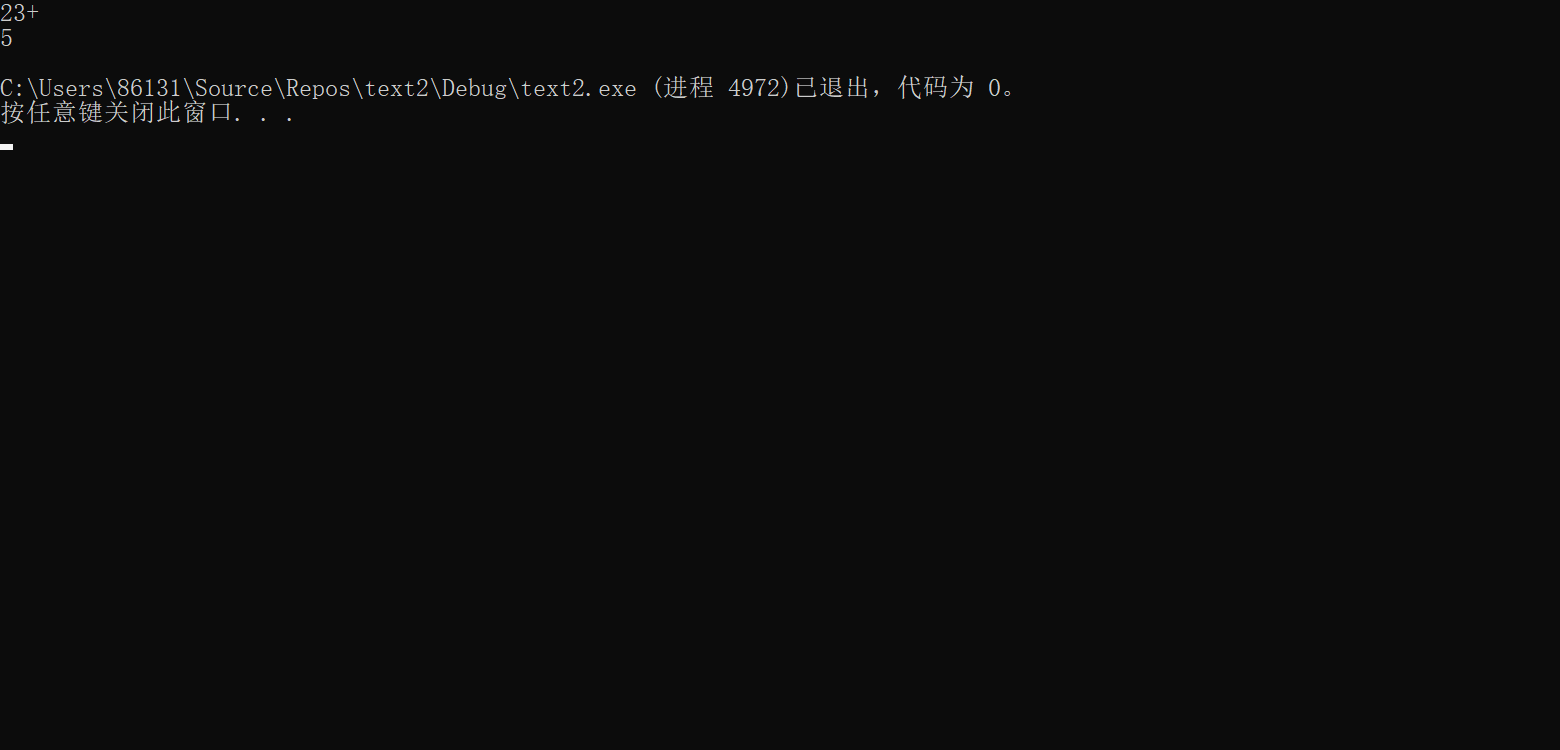
在思考该问题时应考虑到：

1. 要求输入一连串的数字进行四则运算（考虑优先级），而有多少个式子输入我们不得而知，可以使用gets（）；函数（输入类型为char，为ASCII码值的形式）。
2. 我们知道‘栈’的特点是先进后出，也是需要考虑顺序的问题。
3. 现需编写主程序，其余函数已由老师给出。
4. 第一反应是需要两个‘栈’来分别存放数字和符号，在编写过程中发现只要顺序一定，一个栈同样可以实现。

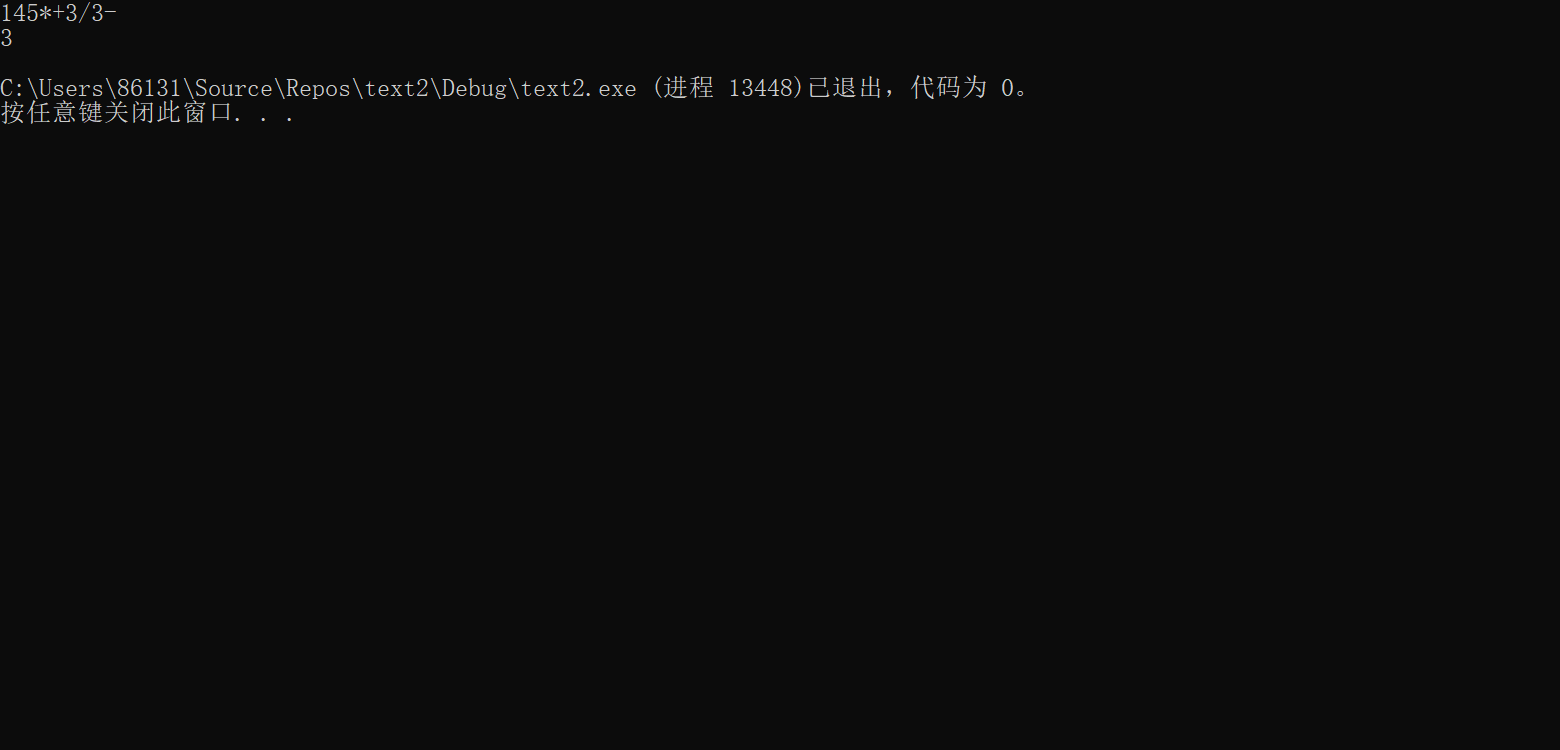
**编写程序**



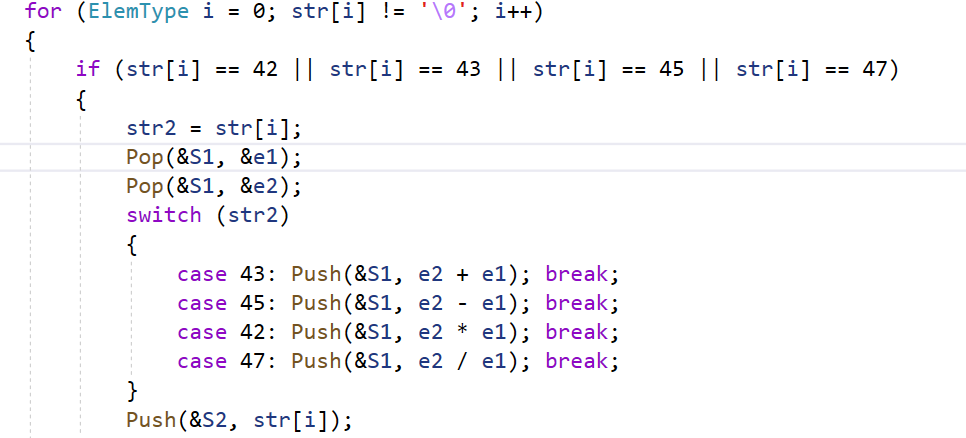
在最后输出是采用了Listtraverse（S1）函数中自带的输出来输出结果，其原理是每次运算总会有两个因子和一个符号参加计算，机制为一旦遇到符号便会从S1中弹出两个数字与遇到的符号进行运算，然后再将运算结果压入栈S1中。

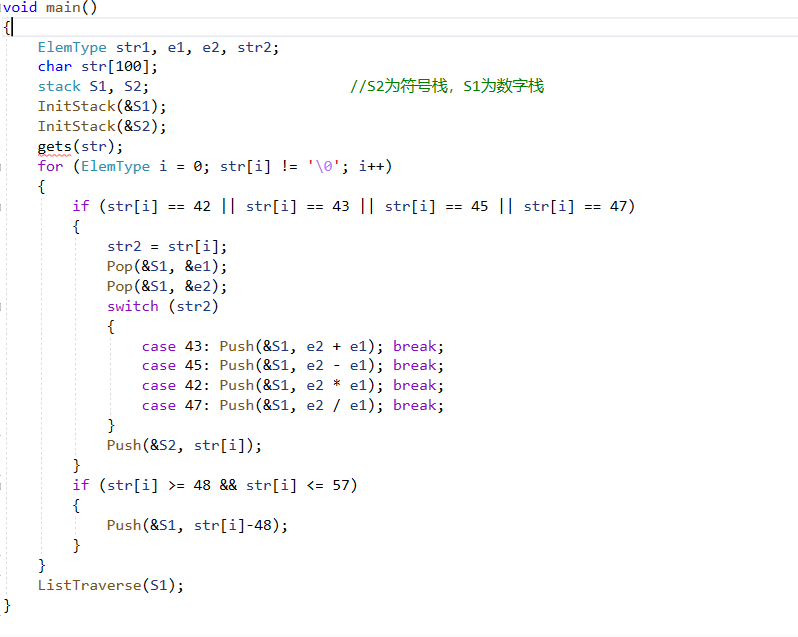


先测试23+（2+3）的运算结果，正确！

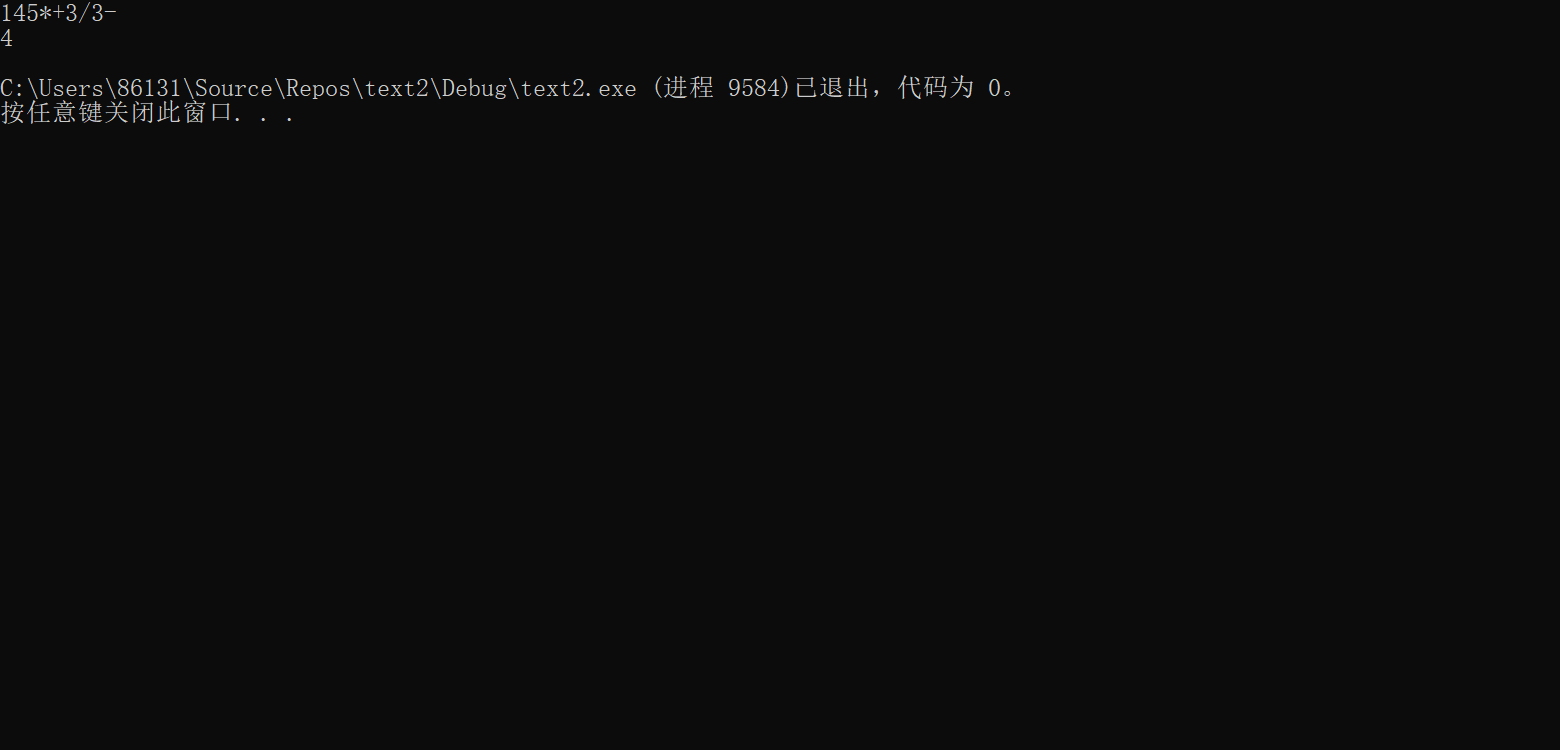
再测试145\*+3/3- ((1+4\*5)/3-3)结果，出错！

其原因是弹出的顺序有问题，而在做加法时可以交换符号两边的加数而不改变结果。





修改，运行。



结果正确！

源代码：

#include<string.h>

#include<ctype.h>

#include<malloc.h> /\* malloc()等 \*/

#include<limits.h> /\* INT\_MAX等 \*/

#include<stdio.h> /\* EOF(=^Z或F6),NULL \*/

#include<stdlib.h> /\* atoi() \*/

#include<io.h> /\* eof() \*/

#include<math.h> /\* floor(),ceil(),abs() \*/

#include<process.h> /\* exit() \*/

#define add 43 //'+'

#define subs 45 //'-'

#define mult 42 //'\*'

#define div 47 //'/'

//0-48 1-49 2-50

#define TRUE 1

#define FALSE 0

#define OK 1

#define ERROR 0

#define INFEASIBLE -1

typedef int Status;

#define LIST\_INIT\_SIZE 100

#define LISTINCREMRNT 10

typedef int ElemType;

typedef struct

{

ElemType\* base; //储存空间基地址

ElemType\* top; // 记录当前链表长度

int stacksize; //链表规模

} stack;

Status InitStack(stack\* S)

{

(\*S).base = (ElemType\*)malloc(LIST\_INIT\_SIZE \* sizeof(ElemType));

if (!(\*S).base)

{

exit(OVERFLOW);

}

(\*S).top = (\*S).base;

(\*S).stacksize = LIST\_INIT\_SIZE;

return OK;

}

Status DestroyStack(stack\* S)

{

free((\*S).base);

(\*S).base = NULL;

(\*S).top = NULL;

(\*S).stacksize = 0;

return OK;

}

Status ClearStack(stack\* S)

{ /\* 把S置为空栈 \*/

(\*S).top = (\*S).base;

return OK;

}

Status StackEmpty(stack S)

{ /\* 若栈S为空栈，则返回TRUE，否则返回FALSE \*/

if (S.top == S.base)

return TRUE;

else

return FALSE;

}

int StackLength(stack S)

{

return S.top - S.base;

}

Status GetTop(stack S, ElemType\* e)

{ /\* 若栈不空，则用e返回S的栈顶元素，并返回OK；否则返回ERROR \*/

if (S.top > S.base)

{

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

return OK;

}

else

return ERROR;

}

Status Push(stack\* S, ElemType e)

{

if ((\*S).top - (\*S).base >= (\*S).stacksize)

{

(\*S).base = (ElemType\*)realloc((\*S).base, ((\*S).stacksize + LISTINCREMRNT) \* sizeof(ElemType));

if (!(\*S).base)

exit(OVERFLOW);

(\*S).top = (\*S).base + (\*S).stacksize;

(\*S).stacksize += LISTINCREMRNT;

}

\*(S->top) = e;

(S->top)++;

}

Status Pop(stack\* S, ElemType\* e)

{

if ((\*S).top == (\*S).base)

return ERROR;

(\*S).top--;

\*e = \*((\*S).top);

}

Status ListTraverse(stack S)

{

ElemType\* e = S.base;

while (e < S.top)

{

printf("%d \n", \*e++);

}

return OK;

}

void convention()

{

stack s;

unsigned n; /\* 非负整数 \*/

ElemType e;

InitStack(&s); /\* 初始化栈 \*/

printf("n(>=0)=");

scanf("%u", &n);

while (n)

{

Push(&s, n % 8);

n = n / 8;

}

while (!StackEmpty(s))

{

Pop(&s, &e);

printf("%d", e);

}

}

void main()

{

ElemType str1, e1, e2, str2;

char str[100];

stack S1, S2; //S2为符号栈，S1为数字栈

InitStack(&S1);

InitStack(&S2);

gets\_s(str);

for (ElemType i = 0; str[i] != '\0'; i++)

{

if (str[i] == 42 || str[i] == 43 || str[i] == 45 || str[i] == 47)

{

str2 = str[i];

Pop(&S1, &e1);

Pop(&S1, &e2);

switch (str2)

{

case 43: Push(&S1, e2 + e1); break;

case 45: Push(&S1, e2 - e1); break;

case 42: Push(&S1, e2 \* e1); break;

case 47: Push(&S1, e2 / e1); break;

}

Push(&S2, str[i]);

}

if (str[i] >= 48 && str[i] <= 57)

{

Push(&S1, str[i]-48);

}

}

ListTraverse(S1);

}