**北京科技大学实验报告**

学院：计通学院 专业： 物联网工程 班级：物联1602

姓名：周康能 学号： 41624657 实验日期： 2018 年 5 月 21日

**实验名称：《数据结构》实验2栈的应用**

**实验目的：理解栈后进先出的特点，熟悉进栈、出栈等栈的基本操作，深入理解栈的相关算法及其应用场合，并能用于解决实际问题。**

**实验内容：**

算术表达式求值：输入中缀形式的算术表达式，如：5+(4-2)\*3，将其转换成后缀表达式并输出：542-3\*+，然后对后缀表达式求值（本例结果为11）并将结果输出。

要求：

（1）“中缀表达式转换后缀表达式”和“后缀表达式求值”采用独立函数实现；

（2）操作数支持多位数和小数；

（3）运算符仅考虑+、-、\*、/、（、）、#（#可用作结束符）；

（4）中缀表达式可从键盘输入也可以从文件输入。对输入的中缀表达式要进行合法性检查（表达式头尾以及运算符左右可以包含若干空格）；

（5）可根据用户需求，多次计算不同的表达式；

（6）自行定义栈类型以及需要的栈的基本操作。

**问题分析与算法思路：**

数据结构：

1、中缀表达式转后缀表达式，其中用到的数据结构为栈。

2、后缀表达式求值也用到栈。

算法：

1、中缀表达式转后缀表达式

从左到右扫描表达式，设置一个栈s存放操作符;

对于遇到的每个分量x，分各种情况处理。

2、后缀表达式求值

遇一个符号就pop出两个数计算，算好后再入栈，如此反复。

3、表达式检验

用到标志位

本程序的匹配函数（即表达式检验函数）具有创新意识。

本程序时间复杂度和空间复杂度合适。

**算法描述：**

1. 链式栈的结构，pop，push，gettop函数等

2、中缀表达式转后缀表达式

从左到右扫描表达式，设置一个栈s存放操作符;

对于遇到的每个分量x，分以下几种情况处理：

1) x = 操作数：输出x;

2) x = '('：x进栈；

3) x = 操作符（非括号）：

while (1) {

if (EmptyStack(s)) break;

y = GetTop(s);

if (y = '(') break;

if (y 优先级< x) break;

y出栈并输出；

}

x进栈;

4) x = ')' ：//之前进栈的'('与')'之间的操作符应先计算！

反复出栈，输出出栈的操作符，直到遇'('，退掉;

当扫描完毕时，若栈非空，则将栈中内容依次出栈输出;

3、后缀表达式求值

for (int i = 0; i < length; i++)

{

是空格时 continue；

是数字，入栈；

是各种符号，出两个数，计算后入栈

}

4、表达式检验用到标志位

对括号有flag\_l,flag\_r;对运算符号有一个标志位last，每次读到符号和数字都会变化，读完表达式不符合就会报错

**程序实现：**

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Copyright(c) :Ellis

\* All rights reserved.

\*

\* 文件名称:USTB2018\_data\_structure\_test2

\* 简要描述:输入中缀算术表达式，如：5+(4-2)\*3，将其转换成后缀表达式并输出：542-3\*+，再对后缀表达式求值（本例结果为11）并将结果输出

\* 要求：1、操作数支持多位数和小数

\* 2、运算符仅考虑+、-、\*、/、（、）、#（#可用作结束符）

\* 3、中缀表达式从键盘输入或从文件输入。对输入的中缀表达式要进行合法性检查（表达式头尾以及运算符左右可以包含若干空格）

\* 算法描述：

\* 从左到右扫描表达式，设置一个栈s存放操作符;

\* 对于遇到的每个分量x，分以下几种情况处理：

\* 1) x = 操作数：输出x;

\* 2) x = '('：x进栈；

\* 3) x = 操作符（非括号）：

\* while (1) {

\* if (EmptyStack(s)) break;

\* y = GetTop(s);

\* if (y = '(') break;

\* if (y 优先级< x) break;

\* y出栈并输出；

\* }

\* x进栈;

\* 4) x = ')' ：//之前进栈的'('与')'之间的操作符应先计算！

\* 反复出栈，输出出栈的操作符，直到遇'('，退掉;

\* 当扫描完毕时，若栈非空，则将栈中内容依次出栈输出;

\*

\*

\*

\* 创建日期:20180416\_22:31

\* 作者:Ellis

\* 说明:完成表达式和浮点数的表示，还差计算和注释

\*

\* 修改日期:20180417\_8:37

\* 作者:Ellis

\* 说明:完成所有任务

\* 注意：无

\*

\* 修改日期:20180511\_14：01

\* 作者:Ellis

\* 说明:修补了若干bug

\* 注意：无

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* 名称 : 宏定义、头文件、命名空间区

\* 创建日期: 20180416

\* 作者 : Ellis

\* 说明 : 无

\*

\* 修改日期:

\* 作者 :

\* 说明 :

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#define \_CRT\_SECURE\_NO\_DEPRECATE //规避安全检查

#include <stdio.h>

#include <string>

#include <stdlib.h>

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* 名称 : 结构体定义区

\* 创建日期: 20180416

\* 作者 : Ellis

\* 说明 : 链式的堆栈，符号+数+后继指针

\*

\* 修改日期:

\* 作者 :

\* 说明 :

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

typedef struct \_link\_stack

{

char num\_operator;

double num\_number;

\_link\_stack \*next;

}stack,\*Lstack;

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* 名称 : 函数定义区

\* 创建日期: 20180417

\* 作者 : Ellis

\* 说明 : 定义各种函数,集成化，容易看

\*

\* 修改日期:

\* 作者 :

\* 说明 :

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

Lstack CreateStack(); //创建栈

bool IsEmpty(Lstack L); //判断是否为空

void Push\_Operator(Lstack top, char cal\_operator); //入栈符号

void Push\_Num(Lstack top, double cal\_num); //入栈数字

char Pop\_Operator(Lstack top); //出栈符号

double Pop\_Num(Lstack top); //出栈数字

char GetTop(Lstack top); //取栈顶符号

char GetTop\_Num(Lstack top); //取栈顶数字

int Priority\_check(char ch1, char ch2); //优先级判定

void Mid\_Post(char infix[100], char\* suffix); //中缀转后缀操作

double Postcount(char \*suffix); //计算值

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Function Name:void main()

Author :Ellis

Date :20180417

Description :

Inputs :

Outputs :

Notes :完成任务

Revision :

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void main()

{

while(1)

{

getchar();

printf("Enter Y to Start , N to end.\n");

int c;

c = getchar();

getchar();

if (c == 'Y')

{

//------------------------------------------------------------

//输入中序

//------------------------------------------------------------

printf("txt or keyboard?\nEntet 1 to load txt,2 to use keyboard.\n");

char d;

d = getchar();

getchar();

char suffix[100] = "";

char ins[100];

if (d == '1')

{

FILE \*file;

char name\_of\_txt[20];

printf("Please enter the name of the txt...\n");

scanf("%s", name\_of\_txt);

file = fopen(name\_of\_txt, "r");

if (!file)

{

printf("Could not open this txt!\n");

system("pause");

}

else

{

printf("Open the txt successfully!\n");

}

char ch;

int k = 0;

while (

(ch = fgetc(file)) != EOF //逐个字符去读取

)

{

ins[k] = ch;

k++;

ch = NULL;

}

ins[k] = '\0';

}

else if (d == '2')

{

printf(">>>Please enter your string:\t");

gets(ins);

}

else

{

printf("Errot enter!\n");

system("pause");

return;

}

//------------------------------------------------------------

//Infix2Suffix("2 \* ( 2 + 2 - 2 + 3) \* 2 / 3", suffix);

//Infix2Suffix(" 2+3\*(7-4)", suffix);

//------------------------------------------------------------

//中缀转后缀

//------------------------------------------------------------

printf("\n");

Mid\_Post(ins, suffix);

printf("The infix you enter in is: %s\t", ins);

printf("\n");

printf("The suffix is: %s\t", suffix);

printf("\n");

//------------------------------------------------------------

//------------------------------------------------------------

//后缀计算值

//------------------------------------------------------------

printf(">>>------------------------------\n");

double x = Postcount(suffix);

printf("The result is :\t %lf", x);

printf("\n");

//------------------------------------------------------------

}

else if (c == 'N')

{

system("pause");

return;

}

else

{

printf("Error enter!Please try again!\n");

continue;

}

}

system("pause");

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Function Name:Lstack CreateStack()

Author :Ellis

Date :20180417

Description :

Inputs :

Outputs :

Notes :

Revision :

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

Lstack CreateStack()

{

Lstack top = (Lstack)malloc(sizeof(stack));

top->next = NULL;

top->num\_operator = 0;

top->num\_number = 0;

return top;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Function Name:bool IsEmpty(Lstack L)

Author :Ellis

Date :20180417

Description :

Inputs :

Outputs :

Notes :

Revision :

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

bool IsEmpty(Lstack L)

{

return L->next ? false : true;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Function Name:void Push\_Operator(Lstack top, char cal\_operator)

Author :Ellis

Date :20180417

Description :

Inputs :

Outputs :

Notes :

Revision :

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void Push\_Operator(Lstack top, char cal\_operator)

{

Lstack node = (Lstack)malloc(sizeof(stack));

node->num\_operator = cal\_operator;

node->num\_number = 0;

node->next = top->next;

top->next = node;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Function Name:void Push\_Num(Lstack top, double cal\_num)

Author :Ellis

Date :20180417

Description :

Inputs :

Outputs :

Notes :

Revision :

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void Push\_Num(Lstack top, double cal\_num)

{

Lstack node = (Lstack)malloc(sizeof(stack));

node->num\_operator = 0;

node->num\_number = cal\_num;

node->next = top->next;

top->next = node;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Function Name:char Pop\_Operator(Lstack top)

Author :Ellis

Date :20180417

Description :

Inputs :

Outputs :

Notes :

Revision :

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

char Pop\_Operator(Lstack top)

{

if (IsEmpty(top))

{

return 0;

}

else

{

Lstack temp = top->next;

top->next = temp->next;

char x = temp->num\_operator;

free(temp);

return x;

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Function Name:double Pop\_Num(Lstack top)

Author :Ellis

Date :20180417

Description :

Inputs :

Outputs :

Notes :

Revision :

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

double Pop\_Num(Lstack top)

{

if (IsEmpty(top))

{

return 0;

}

else

{

Lstack temp = top->next;

top->next = temp->next;

double x = temp->num\_number;

free(temp);

return x;

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Function Name:char GetTop(Lstack top)

Author :Ellis

Date :20180417

Description :

Inputs :

Outputs :

Notes :

Revision :

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

char GetTop(Lstack top)

{

if (IsEmpty(top))

{

return false;

}

else

{

return top->next->num\_operator;

}

//return IsEmpty(top) ? false : top->next->num\_operator;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Function Name:char GetTop\_Num(Lstack top)

Author :Ellis

Date :20180417

Description :

Inputs :

Outputs :

Notes :

Revision :

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

char GetTop\_Num(Lstack top)

{

if (IsEmpty(top))

{

return false;

}

else

{

return top->next->num\_number;

}

//return IsEmpty(top) ? false : top->next->num\_operator;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Function Name:int Priority\_check(char ch1, char ch2)

Author :Ellis

Date :20180417

Description :

Inputs :

Outputs :

Notes :

Revision :

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

int Priority\_check(char ch1, char ch2)

{

if ((ch1 == '+' || ch1 == '-') && (ch2 == '+' || ch2 == '-'))

{

return 0;

}

else if ((ch1 == '\*' || ch1 == '/') && (ch2 == '\*' || ch2 == '/'))

{

return 0;

}

else if ((ch1 == '+' || ch1 == '-') && (ch2 == '\*' || ch2 == '/'))

{

return 1;

}

else if ((ch1 == '\*' || ch1 == '/') && (ch2 == '+' || ch2 == '-'))

{

return 0;

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Function Name:void Mid\_Post(char infix[100], char\* suffix)

Author :Ellis

Date :20180417

Description :

Inputs :

Outputs :

Notes :

Revision :

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void Mid\_Post(char infix[100], char\* suffix)

{

Lstack L = CreateStack();

int length = strlen(infix);

int k = 0; //suffix赋值的标志移动位

int count\_oper = 0;

int count\_num = 0;

int flag\_l = 0; //对括号的判定标志位（左）

int flag\_r = 0; //对括号的判定标志位（右）

int last = 1;

for (int i = 0; i < length; i++)

{

//suffix[k] = ' ';

//k++;

if (infix[i] == ' ')

{

continue;

}

else if (infix[i] <= '9' && infix[i] >= '0')

{

//if (flag\_num == 1)

//{

// //continue;

// ;

//}

//else

//{

// printf("Error!\n"); //数字输出

// system("pause");

// //exit(1);

//}

//flag\_oper = 0; //符号渴望

count\_num++;

last = 1;

suffix[k] = infix[i];

k++;

//suffix[k] = ' ';

//k++;

while ((infix[i + 1] <= '9' && infix[i + 1] >= '0') || infix[i + 1] == '.')

{

suffix[k] = infix[i + 1];

k++;

i++;

}

suffix[k] = ' ';

k++;

}

else if (infix[i] == '(') //扫到左括号进栈

{

Push\_Operator(L, infix[i]);

flag\_l = 1;

last = 0;

//char x = GetTop(L);

//printf("%c", x);

}

else if (infix[i] == '+' || infix[i] == '-' || infix[i] == '\*' || infix[i] == '/')

{

//if (flag\_oper ==1) //遇到符号

//{ //若栈顶为（跳出

// //continue; //若栈顶优先级低，跳出

// ; //出栈，输出

//} //入栈符号

//else

//{

// printf("Error!\n");

// system("pause");

// //exit(1);

//}

//flag\_oper = 0; //数字准许

count\_oper++;

last = 0;

//if (IsEmpty(L))

//{

// break;

//}

//else if (GetTop(L) == '(')

//{

// continue;

// //i = i - 1;

//}

while (1)

{

if (IsEmpty(L))

{

break;

}

char y = GetTop(L);

//printf("%c\n", y);

if (y == '(')

{

break;

}

if (!Priority\_check(infix[i], y))

{

break;

}

suffix[k] = y;

Pop\_Operator(L);

k++;

suffix[k] = ' ';

k++;

}

Push\_Operator(L, infix[i]);

printf("%c\n", suffix[i]);

}

else if (infix[i] == ')') //除非遇到(，否则一直出栈输出

{

flag\_r = 1;

last = 1;

int flag\_k = 0; //得到（的标志位，匹配的标志位

while (1)

{

if (GetTop(L) == '(')

{

Pop\_Operator(L);

flag\_k = 1;

break;

}

suffix[k] = Pop\_Operator(L);

k++;

suffix[k] = ' ';

k++;

}

if (flag\_k == 0)

{

printf("Error!\n");

//exit(1);

system("pause");

}

if (flag\_r & flag\_l)

{

flag\_l = 0;

flag\_r = 0;

}

}

else if (infix[i] == '#')

{

if (count\_num - count\_oper != 1)

{

printf("Error!\n");

system("pause");

//exit(1);

}

if ((!flag\_l)&&(!flag\_r))

{

;

}

else

{

printf("Error!\n");

system("pause");

}

if (last == 1)

{

;

}

else

{

printf("Error!\n");

system("pause");

}

break;

}

else

{

printf("Error!\n");

system("pause");

//exit(1);

}

}

while (!IsEmpty(L)) //最后若栈不为空，全输出出栈

{

suffix[k] = Pop\_Operator(L);

k++;

suffix[k] = ' ';

k++;

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Function Name:double Postcount(char \*suffix)

Author :Ellis

Date :20180417

Description :简单来说，就是pop一个符号，两个数，计算后入栈

Inputs :

Outputs :

Notes :

Revision :

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

double Postcount(char \*suffix)

{

Lstack L = CreateStack();

int length = strlen(suffix);

double result;

char temp[100];

int k = 0;

for (int i = 0; i < length; i++)

{

if (suffix[i] == ' ')

{

continue;

}

else if (suffix[i] <= '9' && suffix[i] >= '0')

{

temp[k] = suffix[i];

k++;

while ((suffix[i + 1] <= '9' && suffix[i + 1] >= '0') || suffix[i + 1] == '.')

{

temp[k] = suffix[i + 1];

k++;

i++;

}

double x = atof(temp);

Push\_Num(L, x);

k = 0;

memset(temp, 0, sizeof(temp));

}

else if (suffix[i] == '+')

{

double x1 = Pop\_Num(L);

double x2 = Pop\_Num(L);

double x = x1 + x2;

Push\_Num(L, x);

}

else if (suffix[i] == '-')

{

double x1 = Pop\_Num(L);

double x2 = Pop\_Num(L);

double x = x2 - x1;

Push\_Num(L, x);

}

else if (suffix[i] == '\*')

{

double x1 = Pop\_Num(L);

double x2 = Pop\_Num(L);

double x = x2 \* x1;

Push\_Num(L, x);

}

else if (suffix[i] == '/')

{

double x1 = Pop\_Num(L);

double x2 = Pop\_Num(L);

double x = x2 / x1;

Push\_Num(L, x);

}

}

if (!L->next->next)

{

return result = Pop\_Num(L);

}

else

{

return false;

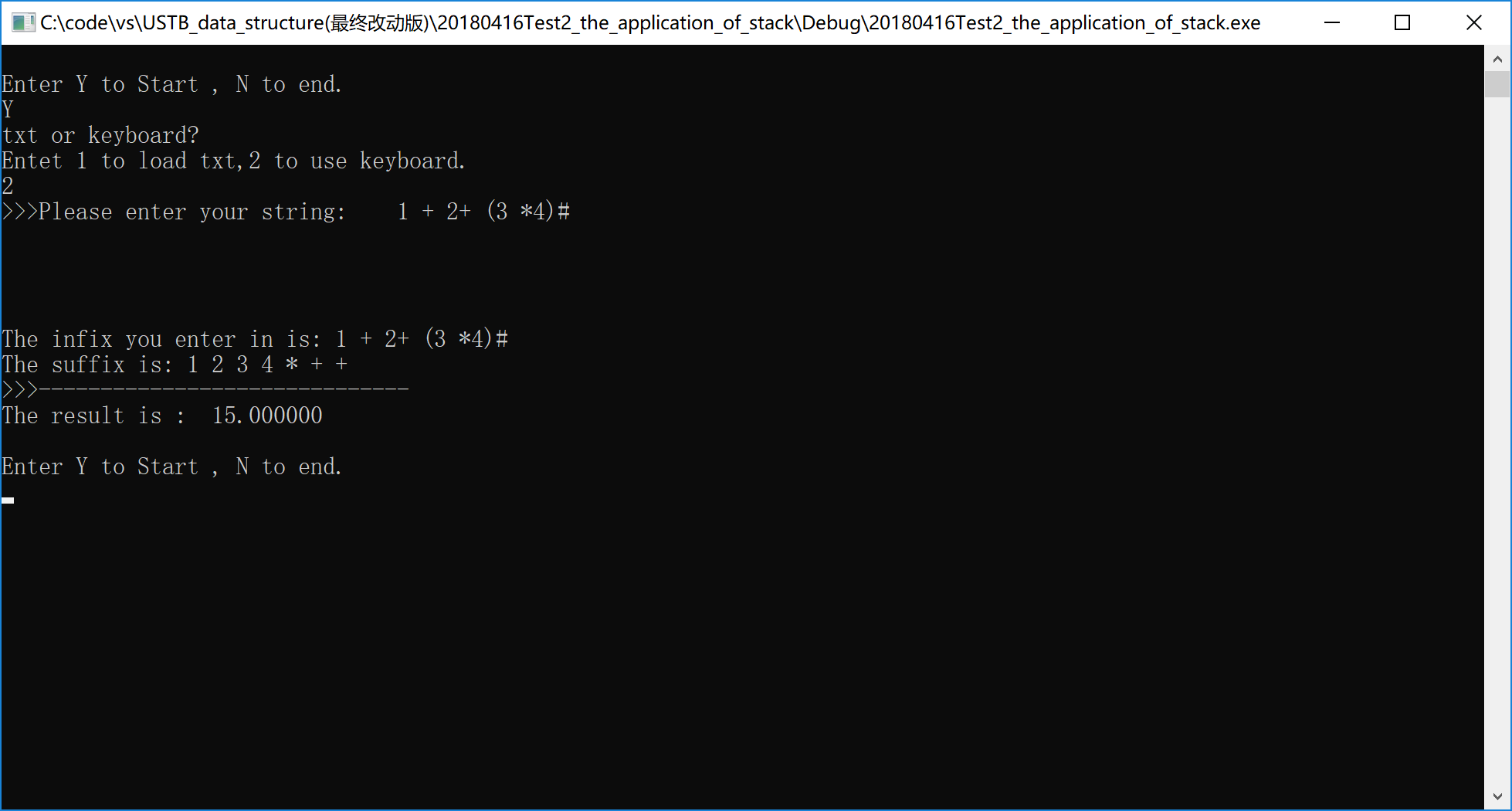
}

}

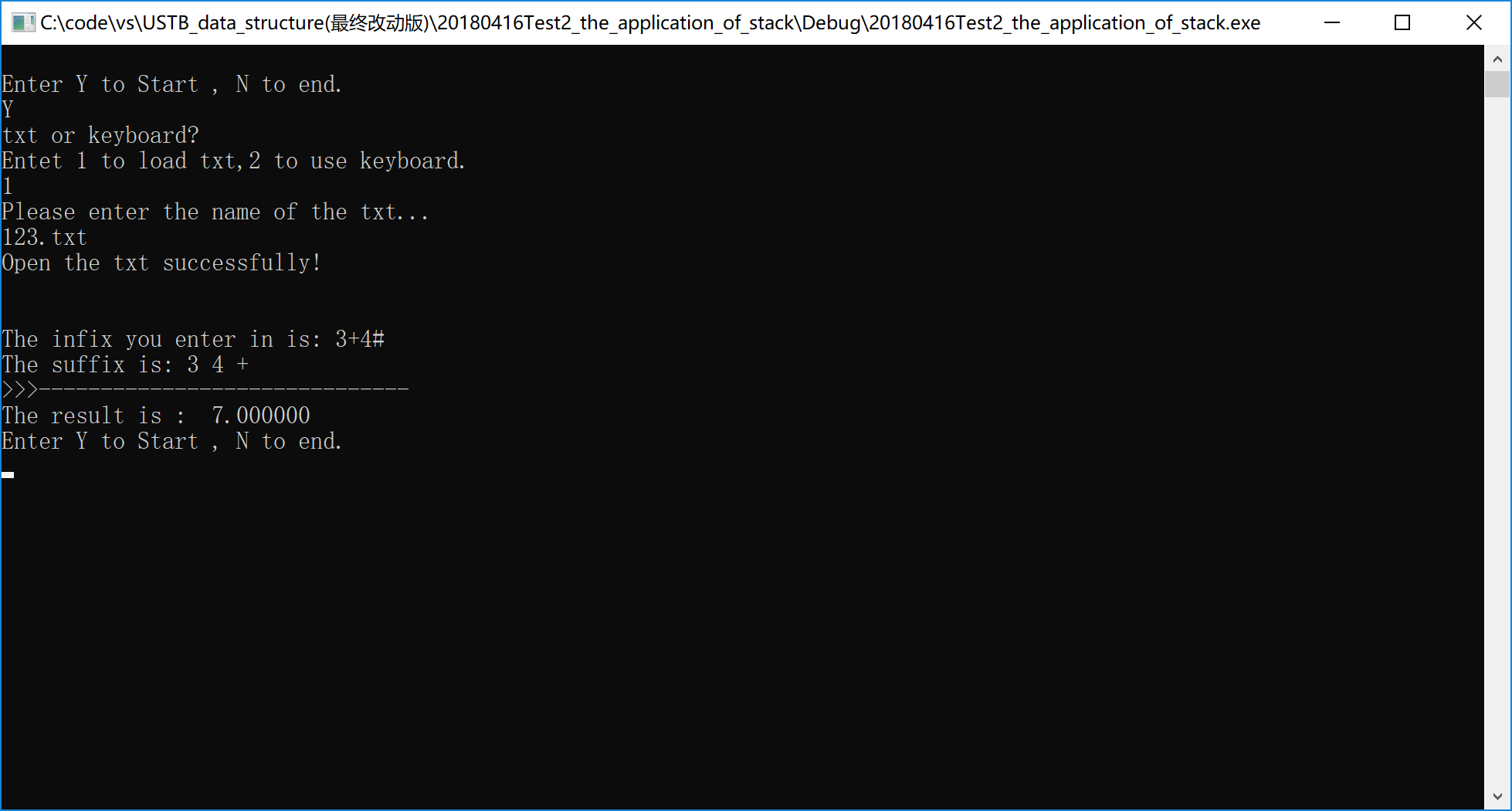
/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

**测试：**

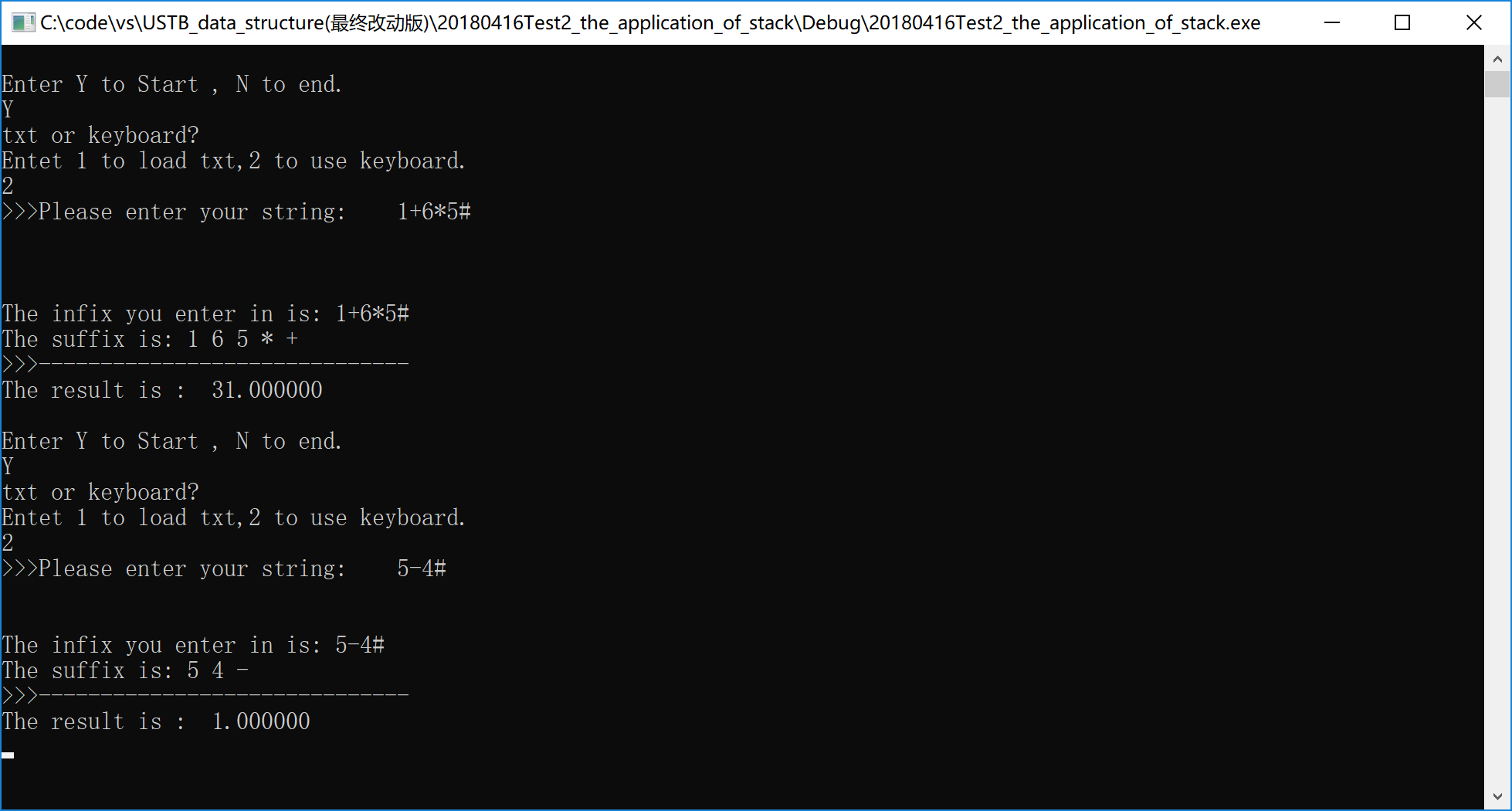
首先测试键盘输入正确性：



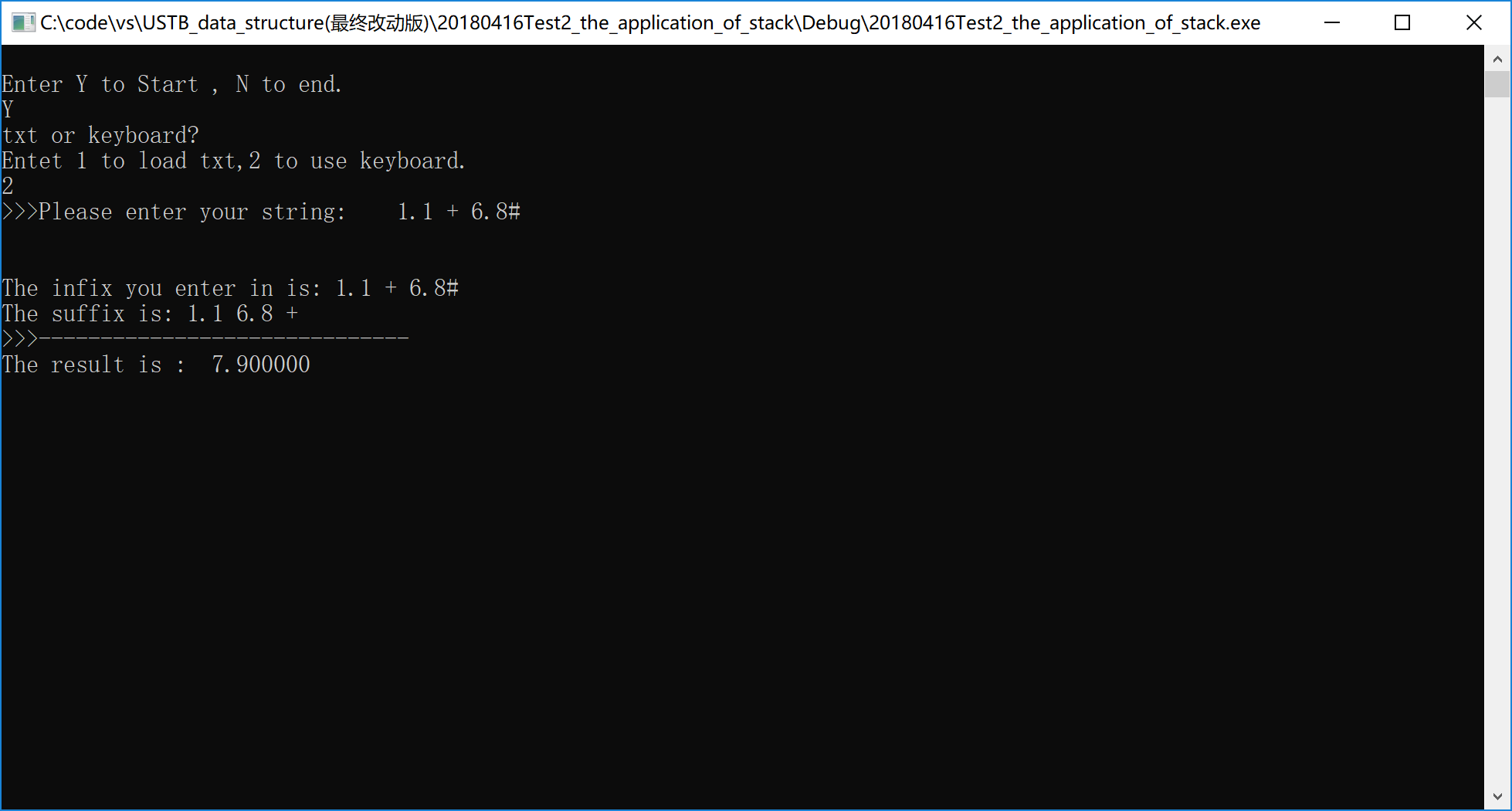
测试txt输入正确性：



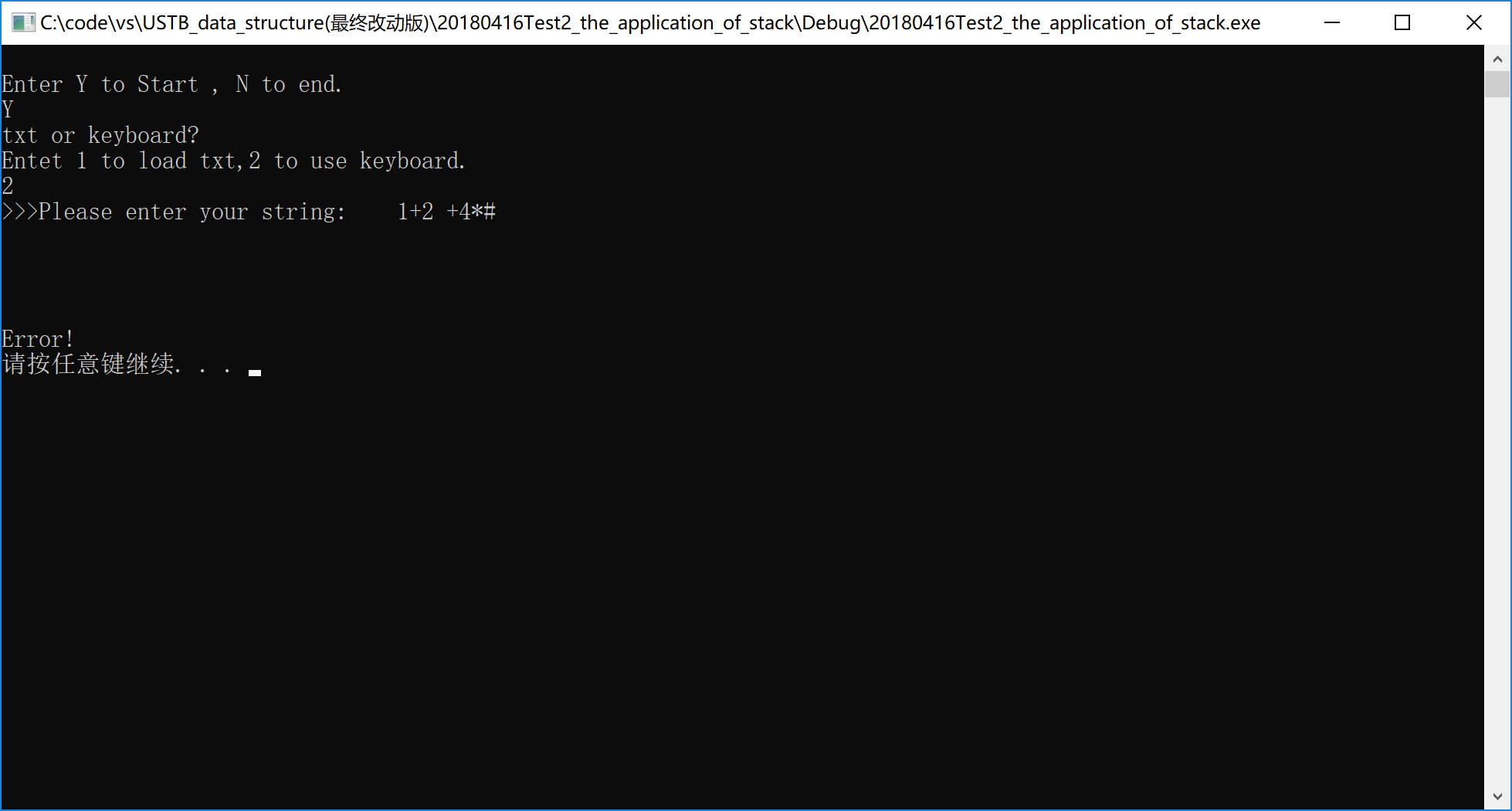
测试重试：



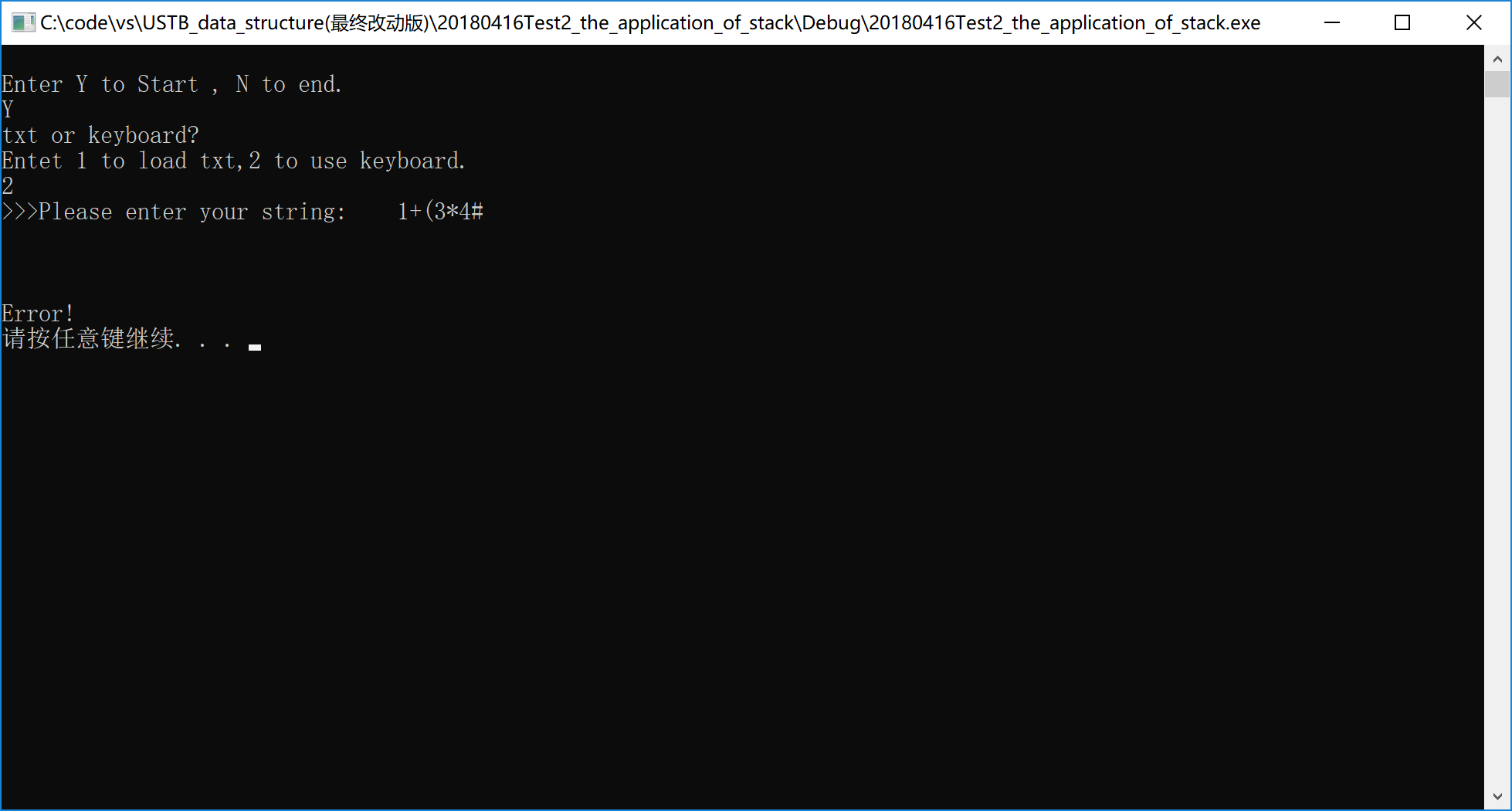
测试小数：



测试表达式不完整：



测试括号不匹配：



**算法的有效性分析：**

时间复杂度及空间复杂度：本程序时间复杂度为O(n)，空间复杂度合适。

有效性：算法正确，模块性强，修改方便，注释完整，可读性强。对符号以及括号的检验有自己独特的方法，具有创新点

不足：程序太长，看着累。命名还需要好好学习一下，命名有点乱。