|  |
| --- |
| 中南大学  《数据结构》课程实验  实验报告  实验题目 利用栈进行表达式求值  专业班级 软件工程2005班  学 号 8209200504  姓 名 李均浩  **实验成绩：**  **批阅教师：**  2021年4月9日 |

一、需求分析

1.程序任务

本程序主要利用栈的基本操作，实现用算符优先法对算术表达式求值。对本设计系统实现+、-、\*、/、%和乘方（^）运算，支持多位数字以及小数的运算。

2.输入以及输出的形式



图1 程序输入输出形式

3.程序功能

实现对多位数字、带有小数的中缀表达式的求值。

4.测试数据

(1)正确输入

(a)((6+7\*(8/6)/6)+3)/100#

预期输出：0.243333

(b) ((2^3/6)^2+6/8)\*5#

预期输出：12.638888

(2)非法输入

(a) (4$6)+9\*4#

预期：输出用户提示，清除键盘缓存区，销毁栈，并跳回程序开头重新输入。

(b) sqwr+666#

预期：输出用户提示，清除键盘缓存区，销毁栈，并跳回程序开头重新输入。

二、概要设计

1.抽象数据类型定义：

ADT Stack {

数据对象：D＝{ai| ai∈ElemSet, i=1,2,...,n, n≥0 }

数据关系：R1＝{ <ai-1,ai>| ,ai-1,ai∈D, i=2,...,n }

约定an端为栈顶，a1端为栈底。

基本操作：

InitStack(&S)

操作结果：构造一个空栈 S。

DestroyStack(&S)

初始条件：栈 S 已存在。

操作结果：栈 S 被销毁。

ClearStack(&S)

初始条件：栈 S 已存在。

操作结果：将 S 清为空栈。

StackEmpty(S)

初始条件：栈 S 已存在。

操作结果：若栈 S 为空栈，则返回TRUE，否则返回FALSE。

StackLength(S)

初始条件：栈 S 已存在。

操作结果：返回栈 S 中元素个数，即栈的长度。

GetTop(S, &e)

初始条件：栈 S 已存在且非空。

操作结果：用 e 返回S的栈顶元素。

这是取栈顶元素的操作，只以 e 返回栈顶元素，并不将它从栈中删除。

Push(&S, e)

初始条件：栈 S 已存在。

操作结果：插入元素 e 为新的栈顶元素。

Pop(&S, &e)

初始条件：栈 S 已存在且非空。

操作结果：删除 S 的栈顶元素，并用 e 返回其值。

StackTraverse(S, visit( ))

初始条件：栈 S 已存在且非空，visit( )为元素的访问函数。

操作结果：从栈底到栈顶依次对S的每个元素调用函数visit( )，

　　　　　　　一旦visit( )失败，则操作失败。

}

2.主程序的流程



图2 主程序的流程

三、详细设计

1.模块伪码

(1) InitStack Begin （传入一个NumStack类型的引用变量）

s.base = (Elemtype\*)申请内存空间，大小为(STACK\_INIT\_SIZE \* sizeof(Elemtype));

如果 (s.base == NULL)

{

弹出错误信息("Unable to allocate to memory space");

退出程序，错误代码：(OVERFLOW);

}

其他则{

s.top = s.base;

s.stack\_size = STACK\_INIT\_SIZE;

返回SUCCESS;

}

End

(2) Push Begin(传入 NumStack& s, Elemtype e)

如果((s.top - s.base) >= s.stack\_size) {//检查是否栈存满

//重新追加空间，大小为STACK\_INCREMENT

s.base = (Elemtype\*)realloc(s.base, s.stack\_size + STACK\_INCREMENT);

//检查时是否成功分配到了内存空间

如果(s.base == NULL)

{

perror("Unable to allocate to memory space");

exit(OVERFLOW);

}

//更新栈顶位置和栈大小(stack\_size)记录

s.top = s.base + s.stack\_size;

s.stack\_size = s.stack\_size + STACK\_INCREMENT;

}

\*s.top = e;

s.top++;

返回 SUCCESS;

End

(3) Pop Begin(传入 NumStack& s, Elemtype& e)

如果(s.top == s.base)

{

返回 ERROR;

}

其他则

{

s.top--;

e = \*s.top;

返回 SUCCESS;

}

End

(4) StackEmpty Begin(传入 NumStack\* s)

如果((\*s).base == (\*s).top)

返回 TRUE;

其他则

返回 FALSE;

End

(5) GetTop Begin(传入 OperatorStack\* s)

如果(!StackEmpty(s))

{

char\* temp = s->top;

temp--;

返回\*(temp);

}

其他则返回 '!';

End

(6) DisplayStack Begin(传入 OperatorStack\* s)

如果 (StackEmpty(s))返回;

for 初始化 i = 0 进行（s->top - s->base）次 步长为 1

{

printf("%c ", s->base[i]);

}

printf(" ");

End

(7) isOperator Begin(传入 char c)

如果(c == '+' || c == '-' || c == '\*' || c == '/' || c == '(' || c == ')' || c == '%' || c == '^' || c == '#')

返回TRUE;

其他则

返回FALSE;

End

(8) Calculate Begin(传入 double temp\_1, double temp\_2, char op)

判断(op)的值

{

如果是'+':

返回 temp\_1 + temp\_2;

如果是'-':

返回 1.0 \* temp\_1 - temp\_2;

如果是'\*':

返回 temp\_1 \* temp\_2;

如果是'/':

返回 1.0 \* temp\_1 / temp\_2;

如果是'^':

返回 1.0\*pow(temp\_1, temp\_2);

如果是'%':

返回 转换为double类型((int)temp\_1 % (int)temp\_2);

}

End

(8) DestroyStack Begin(传入 NumStack& s)

如果 (s.base != NULL)

{

释放(s.base)的内存;

s.base = NULL;

s.top = NULL;

s.stack\_size = 0;

返回 SUCCESS;

}

其他则

返回 ERROR;

End

(9) isValidInput Begin(传入 char c)

如果 (c是数字或者c是+、-、\*、/、^、%、#)

{

返回 true;

}

返回 false;

End

2.函数调用关系图



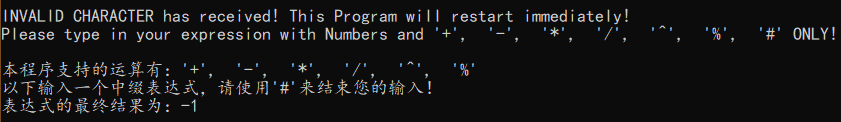
图3 函数调用关系图

四、调试分析

1.问题复现

(1)在处理非法输入的时候只能应对未键入’#’的情况

(a)错误信息



(b)错误源码

//处理非法输入

if (!isValidInput(temp\_char))

{

cout << endl << "INVALID CHARACTER has received! This Program will restart immediately!" << endl \

<< "Please type in your expression with Numbers and \'+\', \'-\', \'\*\', \'/\', \'^\', \'%\', \'#\' ONLY!" << endl;

DestroyStack(num\_stack);

DestroyStack(temp\_num\_stack);

DestroyStack(operator\_stack);

temp\_char = ' ';

//rewind(stdin);

goto start;

}

(c)错误解释

虽然将所有栈以及临时存储字符的temp\_char变量全部重置了，但是程序从头开始之后会继续读入之前的字符，导致程序在最后一次跳入开头时直接读入’#’，以-1的结果结束程序，无法实现重新输入程序。

(d)解决方案

在遇到非法字符的时候加入清除键盘缓存区的语句，在程序跳回开头之前清空缓存。

2. 算法的时空分析

(1)改进设想

暂未有缩短运行时间的方法。

程序编写中有部分变量可以通过一定方式省去，能节省运行占用的空间。

3. 经验与体会

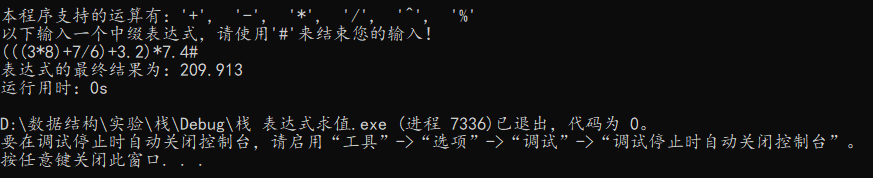
本次实验使用了顺序栈，利用栈先进后出，后进先出的特性实现了表达式的求值，栈的这一特性能帮助我们实现很多使用的功能，包括字符的匹配，操作的回滚等。以后可以多多尝试使用。在使用栈的时候要时刻注意是否有溢出的可能，若进行压栈操作的时候检查存入数据之后是否会触及栈顶，如果将要溢出，则要重新分配空间。

五、用户使用说明

1.按照提示输入一个中缀表达式，用’#’来结束输入（示例：(((3\*8)+7/6)+3.2)\*7.4#）

****

2.获得表达式的值

****

*※3.通过源代码首部的 #define DEBUG\_MODE\_ON 标记可以打开/关闭显示运算过程中出入栈的情况*

六、测试结果

(1)输入：9+4/7-9+6/7^2#

输出：

(2)输入：((6+7\*(8/6)/6)+3)/100#

输出：

(3)输入：((2^3/6)^2+6/8)\*5#

输出：

(4)输入：(5^2%3)+2#

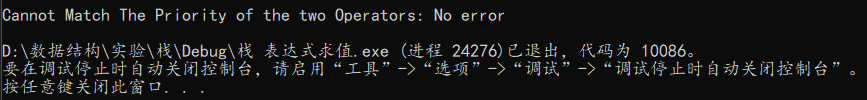
输出：

(5)输入：(2+1/3)^(3+1.4)\*(6-5%3)#

输出：

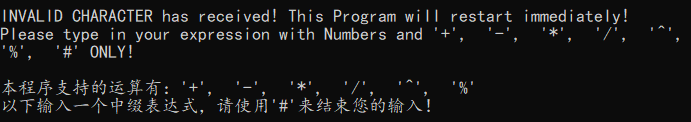
以下为错误输入的样例：

(6)输入：(3+4))/4+11#

处理：

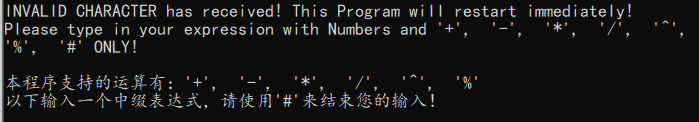
结束程序，返回预先设定的无法匹配引发的错误代码10086

(7)输入：2\*(23+4)=

处理：

输出提示信息，返回至程序开头

(8)输入：9+p\*(3+4)#

处理：

输出提示信息，返回至程序开头

七、附录

#pragma once

#include <iostream>

#include <cstdio>

#include <malloc.h>

#include <string>

#define ERROR 0

#define SUCCESS 1

#define TRUE 1

#define FALSE 0

#define STACK\_INIT\_SIZE 300

#define STACK\_INCREMENT 10

typedef int Status;

typedef char Elemtype;

typedef struct SqStack

{

Elemtype\* base;

Elemtype\* top;

int stack\_size;

}Stack;

//初始化一个栈

Status InitStack(Stack& s)

{

s.base = (Elemtype\*)malloc(STACK\_INIT\_SIZE \* sizeof(Elemtype));

if (s.base == NULL)

{

perror("Unable to allocate to memory space");

exit(OVERFLOW);

}

else {

s.top = s.base;

s.stack\_size = STACK\_INIT\_SIZE;

return SUCCESS;

}

}

//获取栈顶的数据元素

Elemtype GetTop(Stack s)

{

if (s.top != s.base)

return \*(s.top - 1);

}

//将新的元素推入栈中

Status Push(Stack& s, Elemtype e)

{

if ((s.top - s.base) >= s.stack\_size) {//检查是否栈存满

//重新追加空间，大小为STACK\_INCREMENT

s.base = (Elemtype\*)realloc(s.base, s.stack\_size + STACK\_INCREMENT);

//检查时是否成功分配到了内存空间

if (s.base == NULL)

{

perror("Unable to allocate to memory space");

exit(OVERFLOW);

}

//更新栈顶位置和栈大小(stack\_size)记录

s.top = s.base + s.stack\_size;

s.stack\_size = s.stack\_size + STACK\_INCREMENT;

}

//\*s.top++ = e;

\*s.top = e;

s.top++;

return SUCCESS;

}

Status Pop(Stack& s, Elemtype& e)

{

if (s.top == s.base)

{

return ERROR;

}

else

{

s.top--;

e = \*s.top;

return SUCCESS;

}

}

//销毁一个栈

Status DestroyStack(Stack& s)

{

if (s.base != NULL)

{

free(s.base);

s.base = NULL;

s.top = NULL;

s.stack\_size = 0;

return SUCCESS;

}

else

return ERROR;

}

Status ClearStack(Stack& s)

{

if (s.base != NULL)

{

s.top = s.base;

return SUCCESS;

}

else

return ERROR;

}

Status StackEmpty(Stack s)

{

if (s.base == s.top)

return TRUE;

else

return FALSE;

}

int GetLength(Stack s)

{

if (s.base == s.top)

return 0;

else

{

return s.top - s.base;

}

}

Status StackTraverse(Stack s, Status(\*visit)(Elemtype))

{

Elemtype\* traverser = s.base;

while (traverser < s.top)

{

if (!visit(\*traverser))

break;

traverser++;

}

return SUCCESS;

}

源代码 1 StackBasicOperation.h

#include <iostream>

#include <cstdio>

#include <malloc.h>

#define ERROR 0

#define SUCCESS 1

#define TRUE 1

#define FALSE 0

#define STACK\_INIT\_SIZE 10

#define STACK\_INCREMENT 5

#define LITTLE\_NUM\_CAPACITY 100

//ERROR\_EXIT\_CODE

#define OPERATOR\_CANNOT\_MATCH 10086

#define PRIORITY\_CANNOT\_GET 12580

#define INVALID\_INPUT 888

//开启栈储存信息显示

#define DEBUG\_MODE\_ON

typedef int Status;

typedef double Elemtype;

using namespace std;

typedef struct {

char\* base;

char\* top;

int stack\_size;

}OperatorStack;

typedef struct {

Elemtype\* base;

Elemtype\* top;

int stack\_size;

}NumStack;

//初始化一个栈

Status InitStack(NumStack& s)

{

s.base = (Elemtype\*)malloc(STACK\_INIT\_SIZE \* sizeof(Elemtype));

if (s.base == NULL)

{

perror("Unable to allocate to memory space");

exit(OVERFLOW);

}

else {

s.top = s.base;

s.stack\_size = STACK\_INIT\_SIZE;

return SUCCESS;

}

}

Status InitStack(OperatorStack& s)

{

s.base = (char\*)malloc(STACK\_INIT\_SIZE \* sizeof(char));

if (s.base == NULL)

{

perror("Unable to allocate to memory space");

exit(OVERFLOW);

}

else {

s.top = s.base;

s.stack\_size = STACK\_INIT\_SIZE;

return SUCCESS;

}

}

//将新的元素推入栈中

Status Push(NumStack& s, Elemtype e)

{

if ((s.top - s.base) >= s.stack\_size) {//检查是否栈存满

//重新追加空间，大小为STACK\_INCREMENT

s.base = (Elemtype\*)realloc(s.base, s.stack\_size + STACK\_INCREMENT);

//检查时是否成功分配到了内存空间

if (s.base == NULL)

{

perror("Unable to allocate to memory space");

exit(OVERFLOW);

}

//更新栈顶位置和栈大小(stack\_size)记录

s.top = s.base + s.stack\_size;

s.stack\_size = s.stack\_size + STACK\_INCREMENT;

}

\*s.top = e;

s.top++;

return SUCCESS;

}

Status Push(OperatorStack& s, char e)

{

if ((s.top - s.base) >= s.stack\_size) {//检查是否栈存满

//重新追加空间，增量大小为STACK\_INCREMENT

s.base = (char\*)realloc(s.base, s.stack\_size + STACK\_INCREMENT);

//检查时是否成功分配到了内存空间

if (s.base == NULL)

{

perror("Unable to allocate to memory space");

exit(OVERFLOW);

}

//更新栈顶位置和栈大小(stack\_size)记录

s.top = s.base + s.stack\_size;

s.stack\_size = s.stack\_size + STACK\_INCREMENT;

}

\*s.top = e;

s.top++;

return SUCCESS;

}

Status Pop(NumStack& s, Elemtype& e)

{

if (s.top == s.base)

{

return ERROR;

}

else

{

s.top--;

e = \*s.top;

return SUCCESS;

}

}

Status Pop(OperatorStack& s, char& e)

{

if (s.top == s.base)

{

return ERROR;

}

else

{

s.top--;

e = \*s.top;

return SUCCESS;

}

}

Status StackEmpty(NumStack\* s)

{

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

return TRUE;

else

return FALSE;

}

Status StackEmpty(OperatorStack\* s)

{

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

return TRUE;

else

return FALSE;

}

char GetTop(OperatorStack\* s)

{

if (!StackEmpty(s))

{

char\* temp = s->top;

temp--;

return \*(temp);

}

else return '!';

}

double GetTop(NumStack\* s)

{

if (!StackEmpty(s))

{

double\* temp = s->top;

temp--;

return \*(temp);

}

else return -1;

}

void DisplayStack(OperatorStack\* s)

{

if (StackEmpty(s))return;

for (int i = 0; i < s->top - s->base; i++)

{

printf("%c ", s->base[i]);

}

printf(" ");

}

void DisplayStack(NumStack\* s)

{

if (StackEmpty(s))return;

for (int i = 0; i < s->top - s->base; i++)

{

printf("%f ", s->base[i]);

}

printf(" ");

}

Status isOperator(char c)

{

if (c == '+' || c == '-' || c == '\*' || c == '/' || c == '(' || c == ')' || c == '%' || c == '^' || c == '.' || c == '#')

return TRUE;

else

return FALSE;

}

char ComparePriority(char a, char b)

{

if (a == '+')

{

if (b == '\*' || b == '/' || b == '(' || b == '^' || b == '%')

return '<';

else

return '>';

}

else if (a == '-')

{

if (b == '\*' || b == '/' || b == '(' || b == '^' || b == '%')

return '<';

else

return '>';

}

else if (a == '\*')

{

if (b == '(' || b == '^')

return '<';

else

return '>';

}

else if (a == '/')

{

if (b == '(' || b == '^')

return '<';

else

return '>';

}

else if (a == '%')

{

if (b == '(' || b == '^')

return '<';

else

return '>';

}

else if (a == '^')

{

if (b == '(')

return '<';

else

return '>';

}

else if (a == '(')

{

if (b == ')')

return '=';

else if (b == '#')

return '!';

else

return '<';

}

else if (a == ')')

{

if (b == '(')

return '!';

else

return '>';

}

else if (a == '#')

{

if (b == ')')

return '!';

if (b == '#')

return '=';

else

return '<';

}

}

double Calculate(double temp\_1, double temp\_2, char op)

{

switch (op)

{

case '+':

return temp\_1 + temp\_2;

case '-':

return 1.0 \* temp\_1 - temp\_2;

case '\*':

return temp\_1 \* temp\_2;

case '/':

return 1.0 \* temp\_1 / temp\_2;

case '^':

return 1.0 \* pow(temp\_1, temp\_2);

case '%':

return static\_cast<double>((int)temp\_1 % (int)temp\_2);

}

}

Status DestroyStack(NumStack& s)

{

if (s.base != NULL)

{

free(s.base);

s.base = NULL;

s.top = NULL;

s.stack\_size = 0;

return SUCCESS;

}

else

return ERROR;

}

Status DestroyStack(OperatorStack& s)

{

if (s.base != NULL)

{

free(s.base);

s.base = NULL;

s.top = NULL;

s.stack\_size = 0;

return SUCCESS;

}

else

return ERROR;

}

#ifdef DEBUG\_MODE\_ON

Status StackDebug(OperatorStack operator\_stack, NumStack num\_stack)

{

printf("DEBUG INFORMATION:\n");

printf("目前的OperatorStack栈：");

DisplayStack(&operator\_stack);

printf("\n目前的NumStack栈：");

DisplayStack(&num\_stack);

printf("\n\n");

return SUCCESS;

}

#endif

bool isValidInput(char c)

{

if ((c >= '0' && c <= '9') || isOperator(c))

{

return true;

}

return false;

}

int main()

{

//建立三个栈并将其初始化

OperatorStack operator\_stack;

NumStack num\_stack;

NumStack temp\_num\_stack;

start:

InitStack(operator\_stack);

InitStack(num\_stack);

InitStack(temp\_num\_stack);

double sum = 0;

double digit;

int exponent = 0;

//储存从符号栈(OperatorStack)中弹出的符号字符

char operator\_for\_cal;

//临时储存从数字栈(NumStack)中弹出的数字字符

double left\_num, right\_num;

cout << "\n本程序支持的运算有：\'+\', \'-\', \'\*\', \'/\', \'^\', \'%\'" << endl;

cout << "以下输入一个中缀表达式，请使用\'#\'来结束您的输入！" << endl;

Push(operator\_stack, '#');

char temp\_char = getchar();

time\_t time\_start = time(0);

//处理非法输入

if (!isValidInput(temp\_char))

{

cout << endl << "INVALID CHARACTER has received! This Program will restart immediately!" << endl \

<< "Please type in your expression with Numbers and \'+\', \'-\', \'\*\', \'/\', \'^\', \'%\', \'#\' ONLY!" << endl;

DestroyStack(num\_stack);

DestroyStack(temp\_num\_stack);

DestroyStack(operator\_stack);

temp\_char = ' ';

rewind(stdin);

goto start;

}

Status error\_indicator = 0;

while (temp\_char != '#' || GetTop(&operator\_stack) != '#')

{

while (!isOperator(temp\_char))

{

Push(temp\_num\_stack, temp\_char - '0');

temp\_char = getchar();

if (!isValidInput(temp\_char))

{

cout << endl << "INVALID CHARACTER has received! This Program will restart immediately!" << endl \

<< "Please type in your expression with Numbers and \'+\', \'-\', \'\*\', \'/\', \'^\', \'%\', \'#\' ONLY!" << endl;

DestroyStack(num\_stack);

DestroyStack(temp\_num\_stack);

DestroyStack(operator\_stack);

temp\_char = ' ';

rewind(stdin);

goto start;

}

if (temp\_char == '.')

{

temp\_char = getchar();

if (!isValidInput(temp\_char))

{

cout << endl << "INVALID CHARACTER has received! This Program will restart immediately!" << endl \

<< "Please type in your expression with Numbers and \'+\', \'-\', \'\*\', \'/\', \'^\', \'%\', \'#\' ONLY!" << endl;

DestroyStack(num\_stack);

DestroyStack(temp\_num\_stack);

DestroyStack(operator\_stack);

temp\_char = ' ';

rewind(stdin);

goto start;

}

int little\_exp = -1;;

while (!isOperator(temp\_char))

{

sum += (temp\_char - '0') \* pow(10, little\_exp);

little\_exp--;

temp\_char = getchar();

if (!isValidInput(temp\_char))

{

cout << endl << "INVALID CHARACTER has received! This Program will restart immediately!" << endl \

<< "Please type in your expression with Numbers and \'+\', \'-\', \'\*\', \'/\', \'^\', \'%\', \'#\' ONLY!" << endl;

DestroyStack(num\_stack);

DestroyStack(temp\_num\_stack);

DestroyStack(operator\_stack);

temp\_char = ' ';

rewind(stdin);

goto start;

}

}

}

}

while (StackEmpty(&temp\_num\_stack) == FALSE)

{

Pop(temp\_num\_stack, digit);

sum = sum + digit \* pow(10, exponent);

exponent++;

}

exponent = 0;

if (sum != 0)

{

Push(num\_stack, (double)sum);

sum = 0;

}

#ifdef DEBUG\_MODE\_ON

StackDebug(operator\_stack, num\_stack);//Debug

#endif

if (isOperator(temp\_char))

{

switch (ComparePriority(GetTop(&operator\_stack), temp\_char))

{

case '<':

Push(operator\_stack, temp\_char);

temp\_char = getchar();

if (!isValidInput(temp\_char))

{

cout << endl << "INVALID CHARACTER has received! This Program will restart immediately!" << endl \

<< "Please type in your expression with Numbers and \'+\', \'-\', \'\*\', \'/\', \'^\', \'%\', \'#\' ONLY!" << endl;

DestroyStack(num\_stack);

DestroyStack(temp\_num\_stack);

DestroyStack(operator\_stack);

temp\_char = ' ';

rewind(stdin);

goto start;

}

#ifdef DEBUG\_MODE\_ON

StackDebug(operator\_stack, num\_stack);//Debug

#endif

break;

case '>':

Pop(operator\_stack, operator\_for\_cal);

Pop(num\_stack, right\_num);

Pop(num\_stack, left\_num);

#ifdef DEBUG\_MODE\_ON

StackDebug(operator\_stack, num\_stack);//Debug

#endif

Push(num\_stack, Calculate(left\_num, right\_num, operator\_for\_cal));

#ifdef DEBUG\_MODE\_ON

cout << "The Calculate Function is called!" << endl << endl;

#endif

#ifdef DEBUG\_MODE\_ON

StackDebug(operator\_stack, num\_stack);//Debug

#endif

break;

case '=':

Pop(operator\_stack, operator\_for\_cal);

temp\_char = getchar();

if (!isValidInput(temp\_char))

{

cout << endl << "INVALID CHARACTER has received! This Program will restart immediately!" << endl \

<< "Please type in your expression with Numbers and \'+\', \'-\', \'\*\', \'/\', \'^\', \'%\', \'#\' ONLY!"\

<< endl;

DestroyStack(num\_stack);

DestroyStack(temp\_num\_stack);

DestroyStack(operator\_stack);

temp\_char = ' ';

rewind(stdin);

goto start;

}

#ifdef DEBUG\_MODE\_ON

StackDebug(operator\_stack, num\_stack);//Debug

#endif

break;

case '!':

#ifdef DEBUG\_MODE\_ON

StackDebug(operator\_stack, num\_stack);//Debug

#endif

perror("Cannot Match The Priority of the two Operators");

exit(OPERATOR\_CANNOT\_MATCH);

}

}

#ifdef DEBUG\_MODE\_ON

cout << "temp\_char为：" << temp\_char << endl << endl;

#endif

}

time\_t time\_shutdown = time(0);

cout << "表达式的最终结果为：" << GetTop(&num\_stack) << endl << "运行用时：" << time\_shutdown - time\_start << 's' << endl;

cout << endl << endl;

DestroyStack(num\_stack);

DestroyStack(temp\_num\_stack);

DestroyStack(operator\_stack);

temp\_char = ' ';

rewind(stdin);

goto start;

return 0;

}