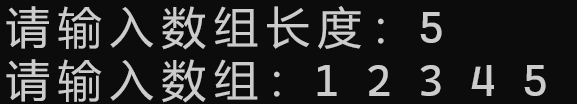
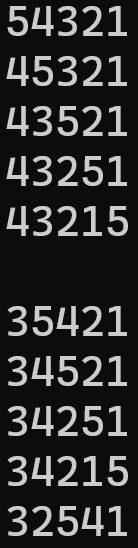
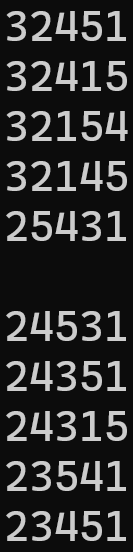
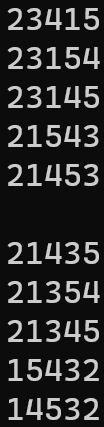
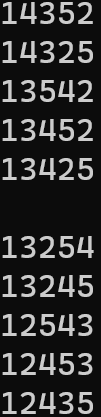
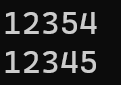
2022212153 陈祥烨 第二章

**第一题**

1 3 4 5 2是合法的输出序列



文件目录:

Queue.hpp

Stack.hpp

源.cpp

源.cpp

#include<iostream>

#include"Stack.hpp"

#include"Queue.hpp"

using namespace std;

template<class T>

void in(Queue<T> input, Stack<T> temp, Queue<T> output);

template<class T>

void out(Queue<T> input, Stack<T> temp, Queue<T> output);

template<class T>

void func(Queue<T> input, Stack<T> temp, Queue<T> output)

{

static int judge = 0;

//判断temp栈空且input队空时，输出output队列的数据

if (input.empty() && temp.empty())

{

T element;

while (!output.empty())

{

output.get\_front(element);

cout << element;

output.serve();

}

cout << " ";

++judge;

if (judge == 5)

{

cout << endl;

judge = 0;

}

cout << endl;

return;

}

//pattern == in, 取出input元素，放入到temp栈中

//pattern ==out, 取出temp元素，放入output队列中

in(input, temp, output);

out(input, temp, output);

}

template<class T>

void in(Queue<T> input, Stack<T> temp, Queue<T> output)

{

if (input.empty())

return;

T element;

input.get\_front(element); //取队列首元素

input.serve(); //删除头元素

temp.push(element); //压栈

func(input, temp, output); //进入功能选择

}

template<class T>

void out(Queue<T> input, Stack<T> temp, Queue<T> output)

{

if (temp.empty())

return;

T element;

temp.get\_top(element);

temp.pop();

output.append(element);

func(input, temp, output);

}

int main(){

cout << "请输入数组长度：";

int n;

cin >> n;

Queue<int> input;

Stack<int> temp(n);

Queue<int> output;

cout << "请输入数组：";

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

{

int num;

cin >> num;

input.append(num);

}

func(input, temp, output);

return 0;

}

Stack.hpp

#pragma once

#include<iostream>

using namespace std;

template<class T>

class Stack

{

private:

//数据

int maxlen; //最大

int count; //总数

T\* data; //数据

public:

Stack(int m); //构造函数

Stack(Stack& st); //复制构造

~Stack(); //析构

bool empty() const; //栈空

bool full() const; //栈满

bool get\_top(T& x)const; //得到栈顶元素

bool push(const T x); //入栈

bool pop(); //出栈

};

template<class T>

Stack<T>::Stack(int m)

{

maxlen = m;

count = 0;

data = new T[maxlen];

}

template<class T>

Stack<T>::Stack(Stack& st)

{

this->count = st.count;

this->maxlen = st.maxlen;

this->data = new T[this->maxlen];

for (int i = 0; i < this->maxlen; i++)

{

this->data[i] = st.data[i];

}

}

template<class T>

Stack<T>::~Stack()

{

if (this->data != NULL)

{

delete[] this->data;

this->data = NULL;

}

}

template<class T>

bool Stack<T>::empty() const

{

return count == 0;

}

template<class T>

bool Stack<T>::full() const

{

return count == maxlen;

}

template<class T>

bool Stack<T>::get\_top(T& x)const

{

if (empty())

return false;

else

{

x = \*(data + count - 1);

return true;

}

}

template<class T>

bool Stack<T>::push(const T x)

{

if (full())

return false;

else

{

\*(data + count) = x;

count++;

return true;

}

}

template<class T>

bool Stack<T>::pop()

{

if (empty() == 1)

return false;

this->count--;

return true;

}

Queue.hpp

#pragma once

#include<iostream>

using namespace std;

template<class T>

struct node

{

T data; //数据域

node<T>\* next; //指针域

};

template<class T>

class Queue

{

public:

Queue(); //默认构造

Queue(Queue<T>& queue); //拷贝构造

~Queue(); //析构

bool empty() const; //队空

bool get\_front(T& x)const; //获取首元素

bool append(const T x); //入队

bool serve(); //出队

private:

int count; //总量

node<T>\* front, \* rear; //头指针，尾指针

};

template<class T>

Queue<T>::Queue()

{

front = new node<T>; //头指针指向头结点，头结点无数据域

rear = front; //尾指针接到头指针上

front->next = NULL; //头指针质置空

count = 0;

}

template<class T>

Queue<T>::Queue(Queue<T>& queue)

{

front = new node<T>; //头指针指向头结点，头结点无数据域

rear = front; //尾指针接到头指针上

front->next = NULL; //头指针质置空

count = queue.count;

//深拷贝

node<T>\* ptr = queue.front;

while (ptr->next != NULL)

{

node<T>\* s = new node<T>; //创建新结点

s->data = ptr->next->data; //存入数据

s->next = NULL; //next置空

rear->next = s; //将新结点连接到尾结点的next

rear = s; //更新尾结点

ptr = ptr->next; //指针指向下一个

}

}

template<class T>

bool Queue<T>::empty()const

{

return rear == front;

}

template<class T>

bool Queue<T>::get\_front(T& x)const

{

if (empty())

return false;

x = front->next->data;

return true;

}

template<class T>

bool Queue<T>::append(const T x)

{

node<T>\* s = new node<T>; //创建新结点

s->data = x; //存入数据

s->next = NULL; //next置空

rear->next = s; //将新结点连接到尾结点的next

rear = s; //更新尾结点

count++; //更新总数量

return true;

}

template<class T>

bool Queue<T>::serve() //出队

{

node<T>\* temp;

if (empty())

return false;

temp = front->next; //暂存首结点

front->next = temp->next; //头指针连接第二个结点

delete temp; //删除头结点

count--; //更新总数量

if (front->next == NULL) //如果队空，则尾结点连接头结点

rear = front;

return true;

}

template<class T>

Queue<T>::~Queue()

{

while (!empty())serve();

delete front;

}

对于连续的三个数来说（a,b,c），b>c||(b<c&&b>a)。

**第二题**

test.cpp

#include"Stack.hpp"

int main()

{

Stack<int> S1(10);

if (S1.empty())

cout << "S1 is empty" << endl;

S1.push(1);

int x;

S1.get\_top(x);

cout << "the top of S1 element is " << x << endl;

Stack<int> S2(S1);

if (!S2.full())

cout << "S2 is not full" << endl;

S2.push(2);

S2.get\_top(x);

cout << "the top of S2 element is " << x << endl;

S1 = S2;

cout << "the top of S1 element is " << x << endl;

return 0;

}

Stack.hpp

#pragma once

#include<iostream>

using namespace std;

template<class T>

class Stack

{

private:

//数据

int maxlen; //最大

T\* top; //栈顶指针

int count; //总量

T\* data; //数据

public:

Stack(int m); //构造函数

Stack(Stack& st); //拷贝构造

Stack& operator=(const Stack& st); //=号 深拷贝

~Stack(); //析构

bool empty() const; //栈空

bool full() const; //栈满

bool get\_top(T& x)const; //得到栈顶元素

bool push(const T x); //入栈

bool pop(); //出栈

};

template<class T>

Stack<T>::Stack(int m)

{

maxlen = m;

data = new T[maxlen];

count = 0;

top = data-1;

}

template<class T>

Stack<T>::Stack(Stack& st)

{

maxlen = st.maxlen;

count = st.count;

this->data = new T[maxlen];

top = data - 1;

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

{

this->data[i] = st.data[i];

top++; //得到栈顶指针的相对位置

}

cout << "Stack 的拷贝构造调用" << endl;

}

template<class T>

Stack<T>& Stack<T>::operator=(const Stack& st)

{

//删

if (this->data != NULL)

{

delete[] this->data;

this->data = NULL;

this->top = NULL;

this->maxlen = 0;

this->count = 0;

}

//深拷贝

this->count = st.count;

this->maxlen = st.maxlen;

this->data = new T[this->maxlen];

this->top = data - 1;

for (int i = 0; i < this->maxlen; i++)

{

this->data[i] = st.data[i];

top++;

}

cout << "Stack 的 = 运算符调用" << endl;

return \*this;

}

template<class T>

Stack<T>::~Stack()

{

if (this->data != NULL)

{

delete[] this->data;

this->data = NULL;

this->top = NULL;

}

}

template<class T>

bool Stack<T>::empty() const

{

return top == data -1;

}

template<class T>

bool Stack<T>::full() const

{

return top == data + maxlen - 1;

}

template<class T>

bool Stack<T>::get\_top(T& x)const

{

if (empty())

return false;

else

{

x = \*top;

return true;

}

}

template<class T>

bool Stack<T>::push(const T x)

{

if (full())

return false;

else

{

top++;

\*top = x;

count++;

return true;

}

}

template<class T>

bool Stack<T>::pop()

{

if (empty())

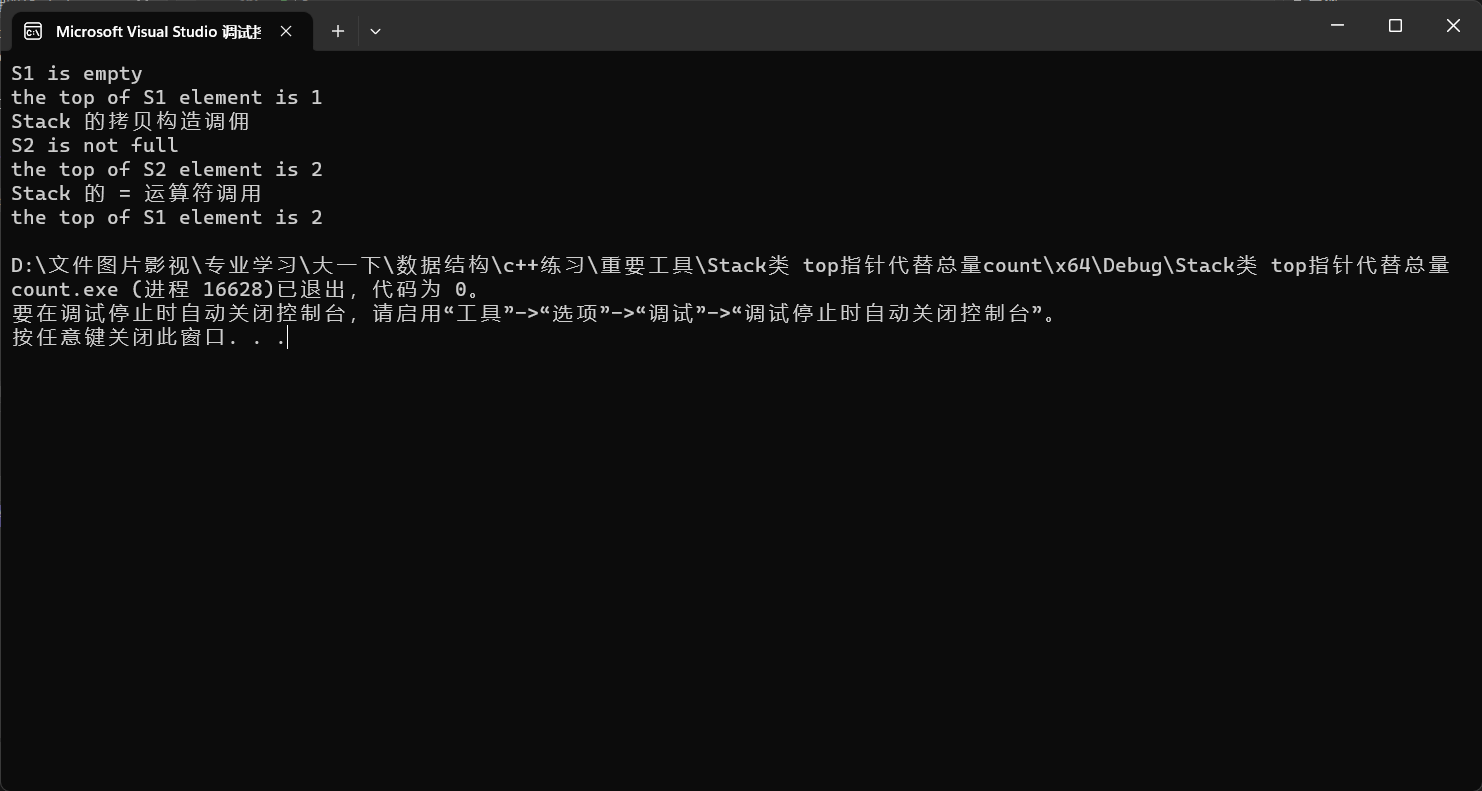
return false;

top--;

count--;

return true;

}



**第三题**

#include<iostream>

using namespace std;

bool match(char a1, char a2)

{

switch (a1)

{

case '(':

if (a2 == ')')

return true;

break;

case '[':

if (a2 == ']')

return true;

break;

case '{':

if (a2 == '}')

return true;

break;

default:

cout << "输入的啥玩意" << endl;

break;

}

return false;

}

int main()

{

char a1;

char a2;

cout << "输入两个0，退出循环" << endl;

while (1)

{

cin >> a1;

cin >> a2;

if (a1 == '0' && a2 == '0')

break;

if (match(a1, a2))

cout << a1 << " 和 " << a2 << " 相匹配" << endl;

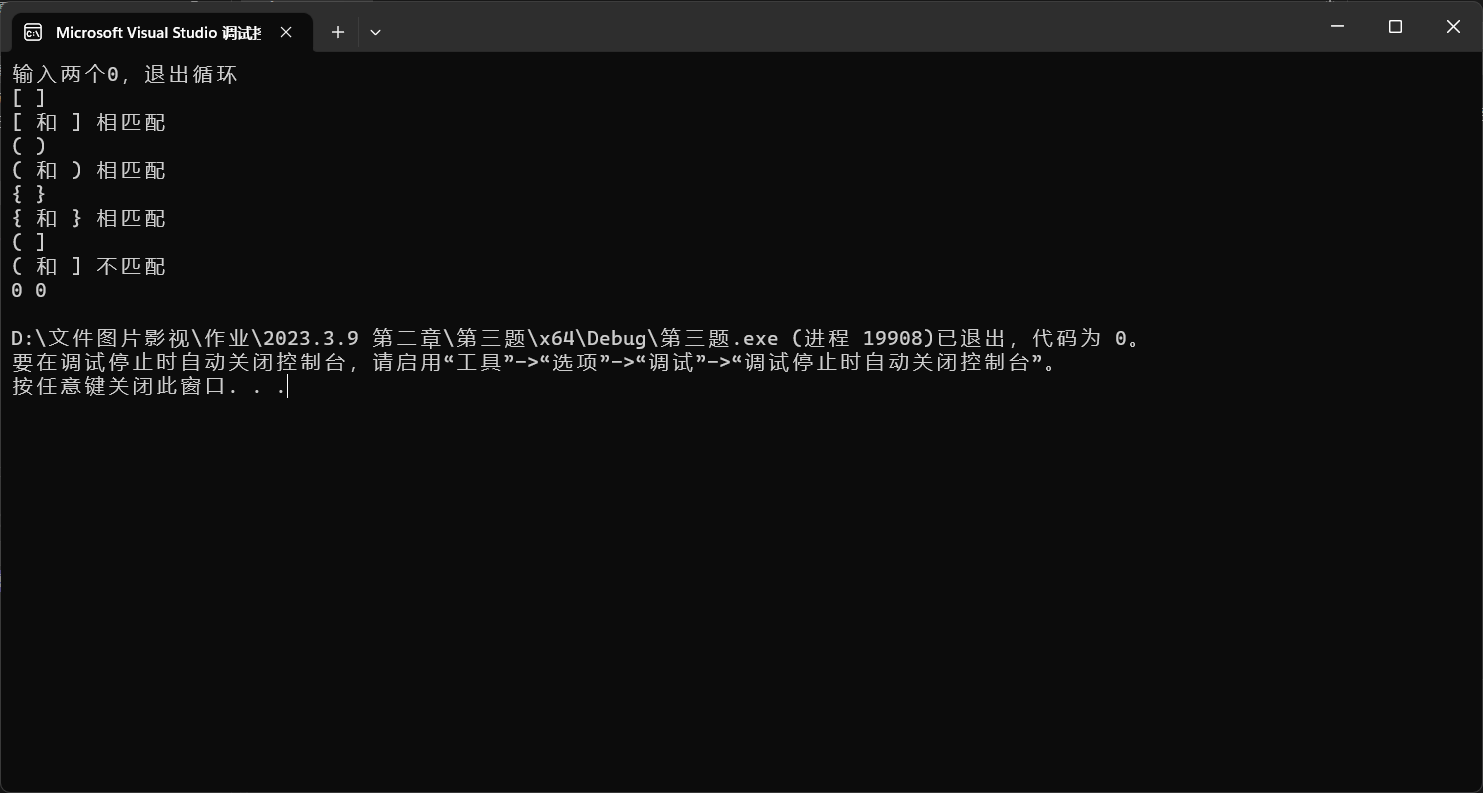
else

cout << a1 << " 和 " << a2 << " 不匹配" << endl;

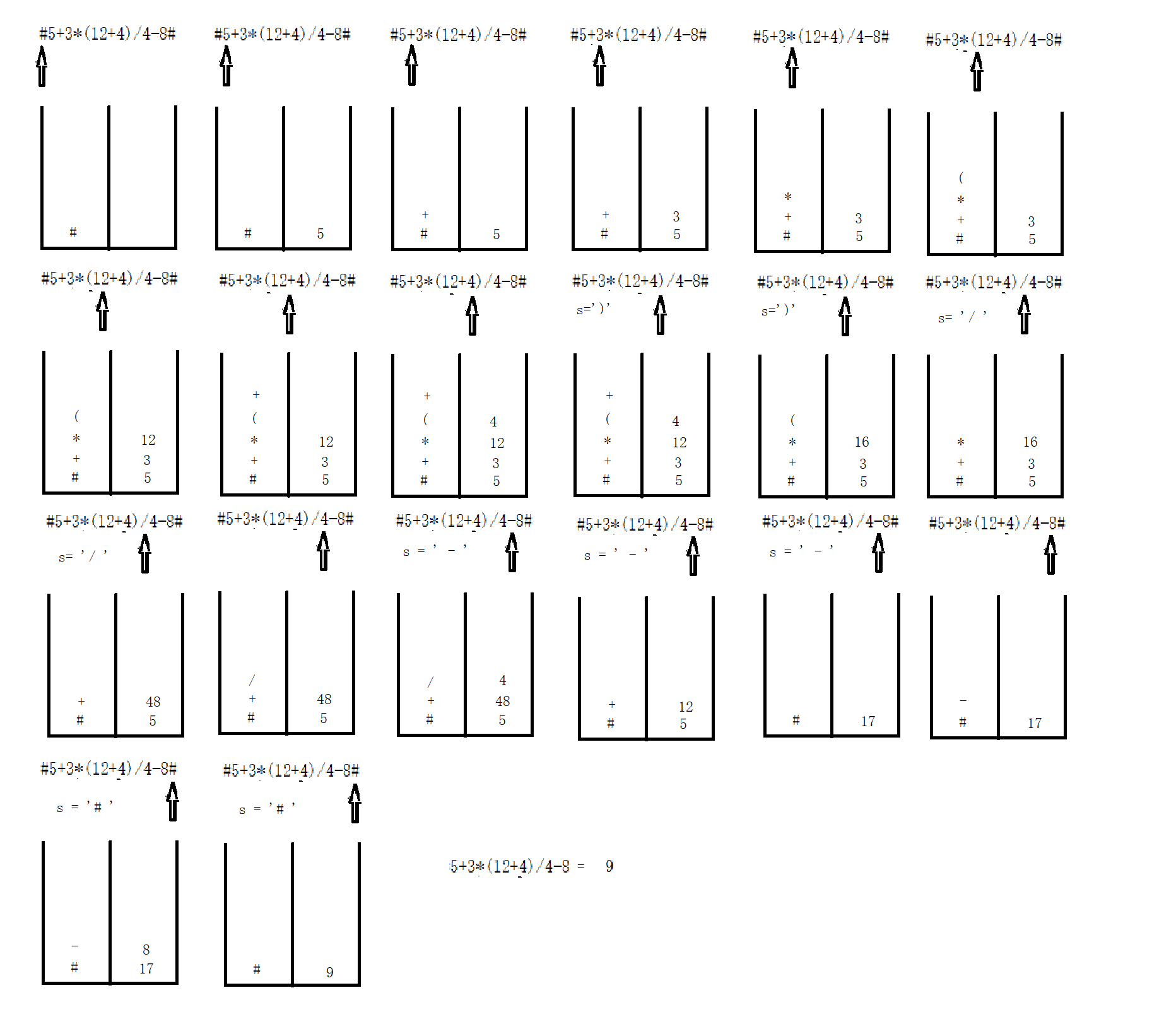
}

return 0;

}



第四题

**第五题**

Stack.hpp

#pragma once

#include<iostream>

using namespace std;

template<class T>

class Stack

{

private:

//数据

int maxlen; //最大

int count; //总数

T\* data; //数据

public:

Stack(int m); //构造函数

Stack(Stack& st); //拷贝构造

Stack& operator=(const Stack& st); //=号 深拷贝

~Stack(); //析构

bool empty() const; //栈空

bool full() const; //栈满

bool get\_top(T& x)const; //得到栈顶元素

bool push(const T x); //入栈

bool pop(); //出栈

};

template<class T>

Stack<T>::Stack(int m)

{

maxlen = m;

count = 0;

data = new T[maxlen];

}

template<class T>

Stack<T>::Stack(Stack& st)

{

this->count = st.count;

this->maxlen = st.maxlen;

this->data = new T[this->maxlen];

for (int i = 0; i < this->maxlen; i++)

{

this->data[i] = st.data[i];

}

}

template<class T>

Stack<T>& Stack<T>::operator=(const Stack& st)

{

//删

if (this->data != NULL)

{

delete[] this->data;

this->data = NULL;

this->maxlen = 0;

this->count = 0;

}

//深拷贝

this->count = st.count;

this->maxlen = st.maxlen;

this->data = new T[this->maxlen];

for (int i = 0; i < this->maxlen; i++)

{

this->data[i] = st.data[i];

}

return \*this;

}

template<class T>

Stack<T>::~Stack()

{

if (this->data != NULL)

{

delete[] this->data;

this->data = NULL;

}

}

template<class T>

bool Stack<T>::empty() const

{

return this->count == 0;

}

template<class T>

bool Stack<T>::full() const

{

return this->count == maxlen;

}

template<class T>

bool Stack<T>::get\_top(T& x)const

{

if (empty())

return false;

else

{

x = \*(data + count - 1);

return true;

}

}

template<class T>

bool Stack<T>::push(const T x)

{

if (full())

return false;

else

{

\*(data + count) = x;

count++;

return true;

}

}

template<class T>

bool Stack<T>::pop()

{

if (empty())

return false;

this->count--;

return true;

}

源.cpp

#include"Stack.hpp"

#include<string>

#include<sstream>

const int char\_cal\_num = 16; //计算机符号的数量

char char\_cal[char\_cal\_num] = //计算机中所有需要的符号

{'0','1','2','3','4','5','6','7','8','9',

'+','-','\*','/',

'(',')' };

bool calculator\_have(char a) //判断a在系统中有没有

{

for (int i = 0; i < char\_cal\_num; i++)

{

if (a == char\_cal[i])

return true;

}

return false;

}

bool isrightexpression(string expression) //判断表达式是否合法

{

//先判断是不是每个符号是本计算机系统所包含的

for(int i = 0;i<expression.size();i++)

{

char c = expression[i];

if (!calculator\_have(c))

return false;

}

//判断所需要的参数

char first\_last;

char secend\_last;

int right\_bracket\_number = 0; //右括号的数量

int num = expression.length(); //循环次数

//最后一个字符不能是左括号和运算符

first\_last = expression.back();

if (first\_last == '+' || first\_last == '-' ||

first\_last == '\*' || first\_last == '/' || first\_last == '(')

return false;

//判断，如果有误则返回

for (int i = 0; i < num - 1; i++)

{

first\_last = expression.back();

expression.pop\_back();

secend\_last = expression.back();

switch (first\_last)

{

//运算符前面必须是数字

case '+':

case '-':

case '\*':

case '/':

{

if (secend\_last!=')' && (secend\_last < '0' || secend\_last > '9'))

return false;

break;

}

//右括号前必须是数字

case ')':

{

if (secend\_last < '0' || secend\_last > '9')

return false;

right\_bracket\_number += 1;

break;

}

//左括号前面必须是运算符，已经有右括号（除首位的左括号）

case '(':

{

if (secend\_last != '+' && secend\_last != '-' &&

secend\_last != '\*' && secend\_last != '/')

return false;

if (right\_bracket\_number == 0)

return false;

right\_bracket\_number--;

break;

}

}

}

//最前面的字符只能是左括号或数字

first\_last = expression.back();

if (first\_last == '+' || first\_last == '-' ||

first\_last == '\*' || first\_last == '/' ||

first\_last == ')')

return false;

return true;

}

string get\_expression(string expression) //将输入的表达式转化为本系统所需要的结构

{

string cal\_expression("#");

cal\_expression += expression;

cal\_expression += "#";

return cal\_expression;

}

//判断优先级

bool ispreferential(char top,char CurrentC)

{

int preferential\_val1 = 0; //优先级用值的大小表示

int preferential\_val2 = 0;

switch (top)

{

case '+':

case '-':

preferential\_val1 = 1;

break;

case '\*':

case '/':

preferential\_val1 = 2;

break;

}

switch (CurrentC)

{

case '+':

case '-':

preferential\_val2 = 1;

break;

case '\*':

case '/':

preferential\_val2 = 2;

break;

}

if (preferential\_val1 >= preferential\_val2)

return true;

else return false;

}

//字符串转换为整数

double StringIntInt(string CurrentS)

{

istringstream is(CurrentS);

int number;

is >> number;

return number;

}

//实现两个数之间的计算

double get\_tempresult(char ope, double number1, double number2)

{

switch (ope)

{

case '+': return number1 + number2; break;

case '-': return number1 - number2; break;

case '\*': return number1 \* number2; break;

case '/': return number1 / number2; break;

}

}

bool getresult(string expression, double& result) //计算结果

{

//先将字符串倒置

reverse(expression.begin(), expression.end());

//需要用到的参数

char CurrentC = expression.back();

string CurrentS;

Stack<char> ope\_st(10);

Stack<double> num\_st(10);

ope\_st.push(CurrentC);

char ope; //运算符

double number1, number2, tempresult;

//读取字符串,为#,开始运行，否则报错

if (CurrentC == '#')

{

expression.pop\_back();

CurrentC = expression.back();

while (CurrentC != '#')

{

//读取尾字符保存到临时字符串中，判断是数还是符号

if ('0' <= CurrentC && CurrentC <= '9')

{

CurrentS += CurrentC;

expression.pop\_back();

CurrentC = expression.back();

//如果下一个是运算符，字符串转换为整数类型，入栈

if (CurrentC < '0' || CurrentC > '9')

{

double number = StringIntInt(CurrentS);

num\_st.push(number);

CurrentS = "";

}

continue;

}

else

{

if (CurrentC == '(')

{

ope\_st.push(CurrentC);

expression.pop\_back();

CurrentC = expression.back();

continue;

}

else if (CurrentC == ')')

{

//取栈顶运算符，计算

ope\_st.get\_top(ope);

ope\_st.pop();

if (ope == '(')

{

expression.pop\_back();

CurrentC = expression.back();

continue;

}

else

{

num\_st.get\_top(number2);

num\_st.pop();

num\_st.get\_top(number1);

num\_st.pop();

tempresult = get\_tempresult(ope, number1, number2);

num\_st.push(tempresult);

}

}

else

{

//如果是运算符，判断优先级

ope\_st.get\_top(ope);

if (ispreferential(ope, CurrentC))

{

//如果栈顶运算符优先级大于currentS，则取出topS进行运算（运算函数）

ope\_st.pop();

num\_st.get\_top(number2);

num\_st.pop();

num\_st.get\_top(number1);

num\_st.pop();

tempresult = get\_tempresult(ope, number1, number2);

num\_st.push(tempresult);

}

else

{

//如果栈顶运算符优先级小于currentS,则入栈

ope\_st.push(CurrentC);

expression.pop\_back();

CurrentC = expression.back();

}

}

}

}

ope\_st.get\_top(ope);

ope\_st.pop();

num\_st.get\_top(number2);

num\_st.pop();

num\_st.get\_top(number1);

num\_st.pop();

result = get\_tempresult(ope, number1, number2);

return true;

}

else

return false;

}

void calculator() //主系统

{

cout << "请输入您所需要计算的表达式: " << endl;

string expression;

string cal\_expression;

while (true) //表达式形式转换

{

getline(cin, expression);

if (!isrightexpression(expression)) //检验表达式合法性

{

cout << "您输入的表达式有误，请重新输入: " << endl;

continue;

}

else

{

cal\_expression = get\_expression(expression);

break;

}

}

double result;

if (getresult(cal\_expression, result))

cout << "结果为: " << result << endl;

else

cout << "计算错误" << endl;

}

int main()

{

cout << "欢迎使用计算器系统！" << endl;

//正整数四则运算

calculator();

return 0;

}

