第一题代码如下

#include <iostream>

#include <cstring>

#include <string.h>

using namespace std;

const int StackSize = 10; //10是示例性的数据，根据实际问题具体定义

template<typename DataType> //定义模板类SeqStack

class SeqStack {

public:

SeqStack(); //构造函数，初始化一个空栈

~SeqStack(); //析构函数

void Push(DataType x); //入栈操作，将元素x入栈

DataType Pop(); //出栈操作，将栈顶元素弹出

DataType GetTop(); //取栈顶元素（并不删除）

int Empty(); //判断栈是否为空

int Precedence(char ch);

int Operate(int x, char ch, int y);

private:

DataType data[StackSize]; //存放栈元素的数组

int top; //游标，栈顶指针，为栈顶元素在数组中的下标

};

template<typename DataType>

SeqStack<DataType>::SeqStack() {

top = -1;

}

template<typename DataType>

SeqStack<DataType>::~SeqStack() {

}

//入栈操作，将元素x入栈

template<typename DataType>

void SeqStack<DataType>::Push(DataType x) {

if (top == StackSize - 1) throw "上溢";

top++;

data[top] = x;

}

//出栈操作，将栈顶元素弹出

template<typename DataType>

DataType SeqStack<DataType>::Pop() {

DataType datapop;

if (top == -1) throw "下溢";

datapop = data[top--];

return datapop;

}

//取栈顶元素（并不删除）

template<typename DataType>

DataType SeqStack<DataType>::GetTop() {

if (top != -1)

return data[top];

}

template<typename DataType>

int SeqStack<DataType>::Empty() {

if (top == -1)

return 1;

else

return 0;

}

//判断优先级

template<typename DataType>

int SeqStack<DataType>::Precedence(char ch) {

int z = 0;

switch (ch) {

case '+':

z = 1;

break;

case '-':

z = 1;

break;

case '\*':

z = 2;

break;

case '/':

z = 2;

break;

case '#':

z = 0;

break;

case '=':

z = 0;

break;

case '(':

z = 3;

break;

default:

printf("error!\n");

}

return z;

}

//进行相加

template<typename DataType>

int SeqStack<DataType>::Operate(int x, char ch, int y) {

int z = 0;

switch (ch) {

case '+':

z = x + y;

break;

case '-':

z = x - y;

break;

case '\*':

z = x \* y;

break;

case '/':

z = x / y;

break;

default:

printf("error!\n");

}

return z;

}

int main() {

SeqStack<int> OPTR{}, OPND{}; //定义顺序栈变量S //OPTR存储运算符，OPND存储操作数和运算结果

char w;

int x, y, z;

char op[9];

strcpy(op,"+-\*/()=#"); //op为运算符集合

//init(OPTR);

OPTR.Push('#');

//init(OPND);

//栈初始化，并在运算符栈的栈底压入表达式左边虚设的字符“#”

w = getchar(); //从终端读入一个字符

while (!((w == '#') && (OPTR.GetTop() == '#'))) {

if (!strchr(op, w)) {

OPND.Push(atoi(&w));

w = getchar(); //操作数进操作数栈

} else if (OPTR.Precedence(w) > OPTR.Precedence(OPTR.GetTop())) { //比较优先数

OPTR.Push(w);

w = getchar();

} else {

char theta;

theta = OPTR.Pop(); //弹出栈顶运算符

y = OPND.Pop();

x = OPND.Pop(); //连续弹出两个操作数

z = OPND.Operate(x, theta, y); //进行运算xθy

OPND.Push(z); //将运算结果压入操作数栈

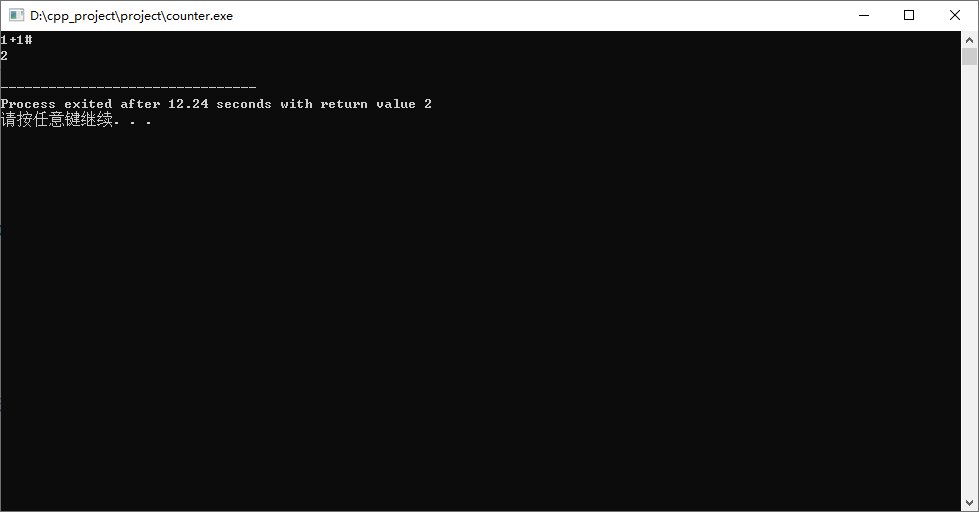
}

}

cout << z << endl;

return (OPND.GetTop()); //从操作数栈顶取出表达式运算结果返回

}



第二题代码如下

#include <iostream>

#include <queue>

#include <chrono>

#include <thread>

using namespace std;

int main() {

queue<int> patientQueue;

// 获取病人编号并排队

while (true) {

cout << "请输入病人编号（输入0结束）：" << endl;

int patientId;

cin >> patientId;

if (patientId == 0) {

break;

}

patientQueue.push(patientId);

}

// 模拟看病过程

while (!patientQueue.empty()) {

int currentPatient = patientQueue.front();

patientQueue.pop();

// 呼叫当前病人看病

cout << "请病人 " << currentPatient << " 就诊" << endl;

if (!patientQueue.empty()) {

// 提示下一个准备看病的病人编号

cout << "下一个病人准备就诊，编号为：" << patientQueue.front() << endl;

}

// 暂停2秒钟

this\_thread::sleep\_for(chrono::seconds(2));

}

cout << "所有病人已看完。" << endl;

return 0;

}

