# 顺序栈运算表达式

## 源码

**SeqStack.h**

|  |
| --- |
| #pragma once  #include <iostream>  using namespace std;  const int StackSize = 10;  template <typename DataType = int>  class SeqStack {  public:  SeqStack(); // 构造函数  ~SeqStack(); // 析构函数  void Push(DataType x); // 入栈操作  DataType Pop(); // 出栈操作  DataType GetTop(); // 去栈顶元素  int Empty(); // 判断栈是否为空  private:  DataType data[StackSize]; // 存放栈元素的数组  int top; // 栈顶元素在数组中的下标  };  // 构造函数  template <typename DataType>  SeqStack<DataType>::SeqStack() {  this->top = -1; // 初始化栈顶元素下标  }  // 析构函数  template <typename DataType>  SeqStack<DataType>::~SeqStack() {  }  // 入栈  template <typename DataType>  void SeqStack<DataType>::Push(DataType x) {  if (this->top == StackSize - 1) throw "栈溢出";  this->data[++top] = x;  }  // 出栈  template <typename DataType>  DataType SeqStack<DataType>::Pop() {  return this->data[this->top--];  }  // 取栈顶  template <typename DataType>  DataType SeqStack<DataType>::GetTop() {  return this->data[this->top];  }  // 判断栈是否为空  template <typename DataType>  int SeqStack<DataType>::Empty() {  return (this->top == -1) ? 1 : 0;  } |

**MyMath.h**

|  |
| --- |
| #pragma once  // 判断运算符优先级  int Precedence(char op);  // 根据二元运算符运算  float Operate(float left, float right, char op); |

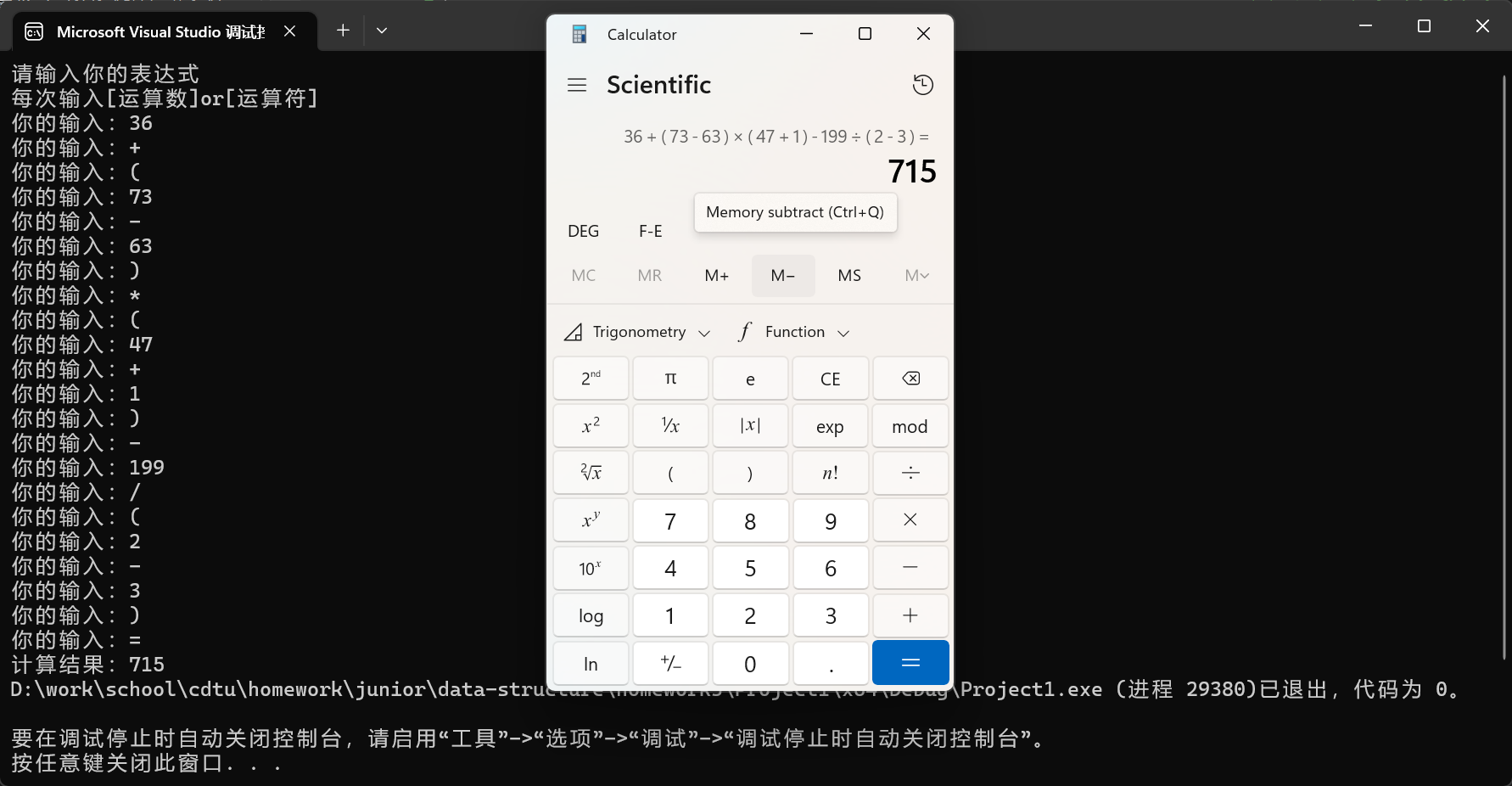
**MyMath.cpp**

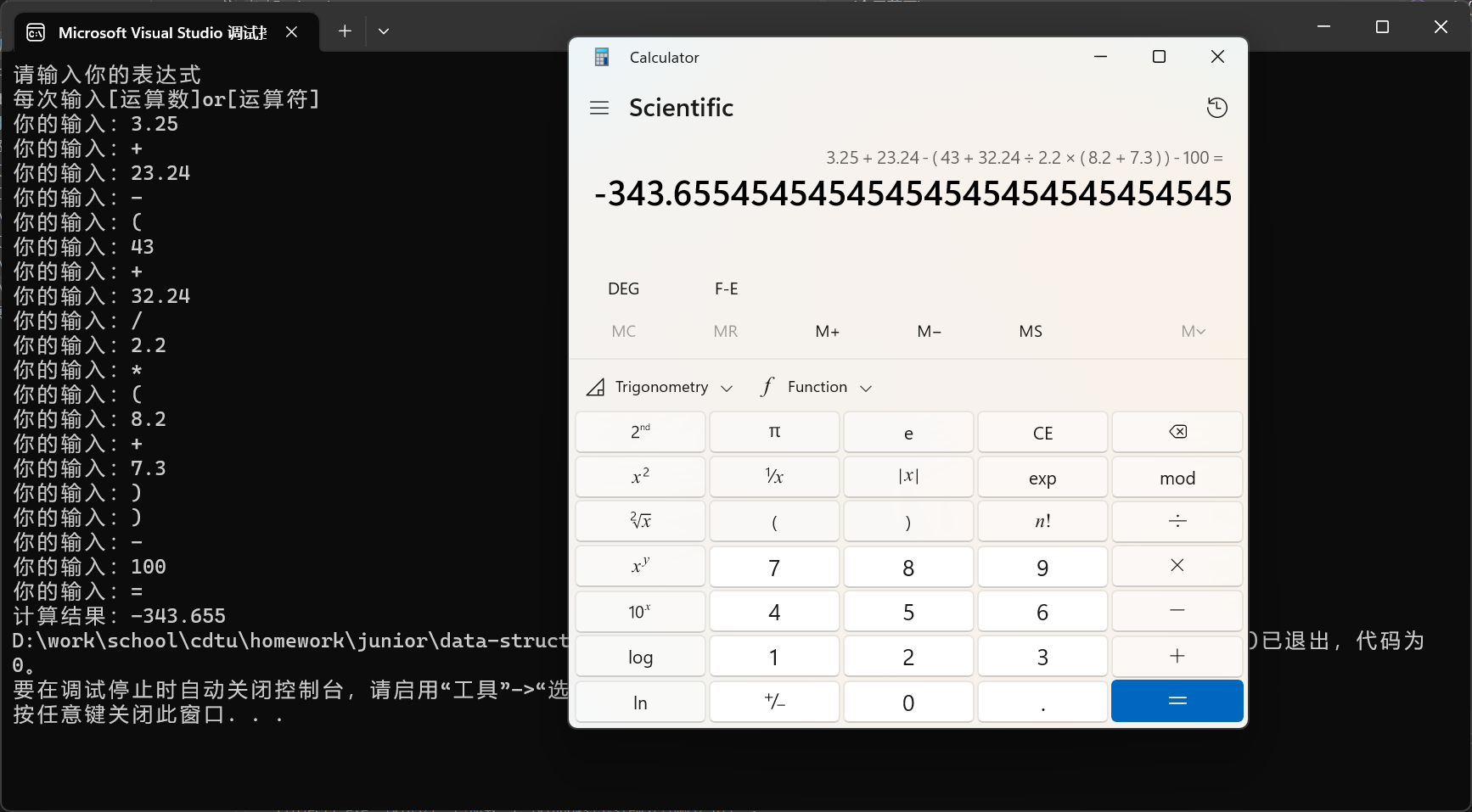
|  |
| --- |
| #include "MyMath.h"  // 判断运算符优先级  int Precedence(char op) {  switch (op) {  case '(':  return 0;  case '#':  return 1;  case '=':  return 1;  case '+':  return 2;  case '-':  return 2;  case '\*':  return 3;  case '/':  return 3;  case ')':  return 4;  default:  throw "不支持的运算符";  }  }  // 根据二元运算符运算  float Operate(float left, float right, char op) {  switch (op) {  case '+':  return left + right;  case '-':  return left - right;  case '\*':  return left \* right;  case '/':  return left / right;  default:  throw "不支持的运算符";  }  } |

**Main.cpp**

|  |
| --- |
| #include "SeqStack.h"  #include "MyMath.h"  #include <string>  using namespace std;  int main() {  string allOps = "#+-\*/=()", userIn = ""; // 定义所有支持的运算符与用户输入  SeqStack<float> nums; // 用于存储运算数  SeqStack<char> ops; // 用于存储运算符  char op = '\0';  float num = 0, left = 0, right = 0; // 用于存储程序会产生的数字  ops.Push('#'); // 将结束符号压入栈中，方便优先级比较    cout << "请输入你的表达式\n每次输入[运算数]or[运算符]" << endl;  do {  if (userIn != "=") {  cout << "你的输入：";  cin >> userIn;  }  // 判断左括号  if (userIn == "(") {  ops.Push('(');  continue;  }  // 判断是否为运算符  if (allOps.find(userIn) == string::npos) {  // 不是运算符则为数字，转为数字并压入数字栈  num = stof(userIn); // 转为数字  nums.Push(num); // 将数字压入数字栈中  continue;  }  // 判断右括号  if (userIn == ")") {  while ((op = ops.Pop()) != '(') {  right = nums.Pop();  left = nums.Pop();  nums.Push(Operate(left, right, op));  }  continue;  }  // 判断运算符优先级  if (Precedence(ops.GetTop()) >= Precedence(userIn[0])) {  // 如果新运算的优先级比栈中低则先弹栈进行运算  right = nums.Pop();  left = nums.Pop();  nums.Push(Operate(left, right, ops.Pop())); // 将运算结果压入数字栈中  // 将用户输入的运算符压入运算符栈中  if (userIn != "=") {  ops.Push(userIn[0]);  }  continue;  }  // 如果新运算的优先级比栈中高则直接压入运算符栈  if (userIn != "=") {  ops.Push(userIn[0]);  }  } while (userIn != "=" || ops.GetTop() != '#');  cout << "计算结果：" << nums.GetTop();  return 0;  } |

## 运行结果





# 门诊看病问题

## 源码

**CirQueue.h**

|  |
| --- |
| #pragma once  const int QueueSize = 50; // 设置循环队列的长度  template <typename DataType>  class CirQueue {  public:  CirQueue(); // 构造函数  ~CirQueue(); // 析构函数  void EnQueue(DataType x); // 入队操作，将元素x入队  DataType DeQueue(); // 出队操作  DataType GetHead(); // 取队头元素  int Empty(); // 判空  int Size(); // 长度  private:  DataType data[QueueSize]; // 数据  int start, end, length; // 队头和队尾以及长度  };  // 长度  template <typename DataType>  int CirQueue<DataType>::Size() {  return length;  }  // 构造函数  template <typename DataType>  CirQueue<DataType>::CirQueue() {  // 将头尾都指向最后一个位置  this->start = QueueSize - 1;  this->end = QueueSize - 1;  this->length = 0; // 长度为0  }  // 析构函数  template <typename DataType>  CirQueue<DataType>::~CirQueue() {  }  // 入队  template <typename DataType>  void CirQueue<DataType>::EnQueue(DataType x) {  if ((this->end + 1) % QueueSize == this->start) throw "满";  this->data[this->end = ((this->end + 1) % QueueSize)] = x;  this->length++;  }  // 出队  template <typename DataType>  DataType CirQueue<DataType>::DeQueue() {  this->length--;  if (this->start == this->end) throw "空队";    return this->data[this->start = (this->start + 1) % QueueSize];  }  // 取队头  template <typename DataType>  DataType CirQueue<DataType>::GetHead() {  if (this->start == this->end) throw "空队";  return this->data[(this->start + 1) % QueueSize];  }  // 判空  template <typename DataType>  int CirQueue<DataType>::Empty() {  return length ? 0 : 1;  } |

**Main.cpp**

|  |
| --- |
| #include "CirQueue.h"  #include <iostream>  #include <string>  using namespace std;  // 用于存储病人信息的结构体  struct Sick {  string id; // 编号  string name;// 姓名  };  /\*  处理菜单选项，并返回进入门诊的人  choice：选项  sicks：排队的病人  current：当前正在门诊的病人  return：返回处理后正在门诊的病人  \*/  Sick\* choiceHandle(int choice, CirQueue<Sick\*>\* sicks, Sick\* current);  int main() {  int userIn = 0;  CirQueue<Sick\*>\* sicks = new CirQueue<Sick\*>;  struct Sick\* sick = nullptr;    do {  try {  cout << "----------输入数字选择对应功能----------" << endl;  if (sick == nullptr) {  cout << "----------------诊室空闲----------------" << endl;  }  else {  cout << "---编号[" + sick->id + "]|姓名[" + sick->name + "]正在门诊-----" << endl;  }  if (sicks->Size()) {  cout << "---请编号[" + sicks->GetHead()->id + "]|姓名[" + sicks->GetHead()->name + "]做好准备-----" << endl;  }  cout << "---------当前还有[" << sicks->Size() << "]人正在排队----------" << endl;  cout << "--------------1、录入病人---------------" << endl;  cout << "--------------2、呼叫病人---------------" << endl;  cout << "--------------0、退出系统---------------" << endl;  cout << "请选择：";  cin >> userIn;  sick = choiceHandle(userIn, sicks, sick);  cout << "----------------------------------------" << endl;  }  catch (const char\* ex) {  cout << "错误的操作" << endl;  exit(-1);  }  } while (userIn);  return 0;  }  /\*  处理菜单选项，并返回进入门诊的人  choice：选项  sicks：排队的病人  current：当前正在门诊的病人  return：返回处理后正在门诊的病人  \*/  Sick\* choiceHandle(int choice, CirQueue<Sick\*>\* sicks, Sick\* current) {  struct Sick\* sick;  switch (choice) {  case 0:  cout << "再见" << endl;  return nullptr;  case 1:  cout << "请输入编号及姓名" << endl;  sick = new Sick;  cin >> sick->id >> sick->name;  sicks->EnQueue(sick);  return current;  case 2:  if (sicks->Size()) {  sick = sicks->DeQueue();  }  else {  cout << "无人等待" << endl;  sick = nullptr;  }  return sick;  default:  cout << "不支持的选项" << endl;  return current;  }  } |

## 运行结果

