项目说明文档

数据结构课程设计

——银行业务

作 者 姓 名： 张喆

学 号： 1754060

指 导 教 师： 张颖

学院、 专业： 软件学院 软件工程

同济大学

Tongji University

目 录

[1 分析 1](#_Toc532235651)

[1.1 背景分析 1](#_Toc532235652)

[1.2 功能分析 1](#_Toc532235653)

[2 设计 2](#_Toc532235654)

[2.1 数据结构设计 2](#_Toc532235655)

[2.2 类结构设计 2](#_Toc532235656)

[2.3 成员与操作设计 2](#_Toc532235657)

[2.4 系统设计 2](#_Toc532235658)

[3 实现 3](#_Toc532235659)

[3.1 读入顾客功能的实现 3](#_Toc532235660)

[3.1.1 读入顾客功能流程图 3](#_Toc532235661)

[3.1.2 读入顾客功能核心代码 3](#_Toc532235662)

[3.1.3 读入顾客功能截屏示例 4](#_Toc532235663)

[3.2 处理业务功能的实现 4](#_Toc532235664)

[3.2.1 处理业务功能流程图 4](#_Toc532235665)

[3.2.2 处理业务功能核心代码 4](#_Toc532235666)

[3.2.3 处理业务功能截屏示例 5](#_Toc532235667)

[3.3 输出功能的实现 6](#_Toc532235668)

[3.3.1 输出功能流程图 6](#_Toc532235669)

[3.3.2 输出功能核心代码 6](#_Toc532235670)

[3.3.3 输出功能截图示例 6](#_Toc532235672)

[3.4 总体系统的实现 7](#_Toc532235673)

[3.4.1 总体系统核心代码 7](#_Toc532235674)

[3.4.2 总体系统截屏示例 7](#_Toc532235675)

[4 测试 8](#_Toc532235676)

[4.1 功能测试 8](#_Toc532235677)

[4.1.1 读入顾客功能测试 8](#_Toc532235678)

[4.1.2 业务处理功能测试 10](#_Toc532235679)

[4.1.3 输出功能测试 11](#_Toc532235680)

[4.2 边界测试 12](#_Toc532235681)

[4.2.1 最小顾客人数 12](#_Toc532235682)

[4.3 出错测试 12](#_Toc532235683)

# 1 分析

## 1.1 背景分析

银行业务系统的设计师银行不可缺少的部分之一。既是银行为顾客提供优良服务的承载着，又是银行能高效处理业务的秘诀，所以一个良好的银行业务系统应能按照每个窗口的业务员的处理能力相应调整业务处理的顺序。

随着银行业主数量和业务员工作量的日益庞大，如何管理如此庞大的数据显得极为复杂，传统的手工管理量大而容易出错。

随着计算机科学技术的不断成熟，使用计算机对银行业务系统进行管理，具有手工管理无法比拟的优势。这些优点能够极大地提高银行和顾客的效率，也是银行走向信息化、科学化、国际化的重要条件。因此，开发一套银行业务系统具有十分重要的意义。

## 1.2 功能分析

作为一个最简易的银行业务系统，首先应该有的功能就是输入待处理的顾客信息。

其次满足，编号为奇数的顾客需要到A窗口办理业务，为偶数的顾客则去B窗口的条件；并且A窗口处理速度是B窗口的2倍----即当A窗口每处理完2个顾客是，B窗口处理完1个顾客。

最后按照业务处理完成的顺序输出顾客的编号。

（同时需要注意，数字间以空格分隔，但是最后一个编号不能有多余的空格。）

# 2 设计

# 2.1 数据结构设计

如上功能分析所述，该系统规定A窗口的处理速度是B窗口的两倍，而且我们日常去银行办理业务也是遵循排队的原则，因此考虑使用队列作为数据结构。定义QueueA和QueueB分别储存A、B两个窗口的业务情况，再定义Qresult作为结果队列，最后只需要按照A窗口的处理速度是B窗口的两倍的设定处理Qresult作为输出即可。

## 2.2 类结构设计

定义银行类Bank来进行银行业务的处理，在Bank类中定义用于表示A、B两个窗口的队列QueueA和QueueB，又定义了储存结果的队列Qresult。

同时把对银行业务的处理全部封装在Bank类中，简洁直观。

## 2.3 成员与操作设计

**银行类（Bank）**

**私有成员：**

queue<int> QueueA, QueueB; //AB窗口

queue<int> Qresult; //结果队列, 用于保存处理顺序

**银行类的构造函数：**

Bank() = default;

Bank(int sum);

Bank::Bank(int sum)

{

Init(sum);

}

**公有操作：**

void Init(int sum); //输入顾客序列

void dealBusiness(); //处理业务

void showOrder(); //输出顾客的处理顺序

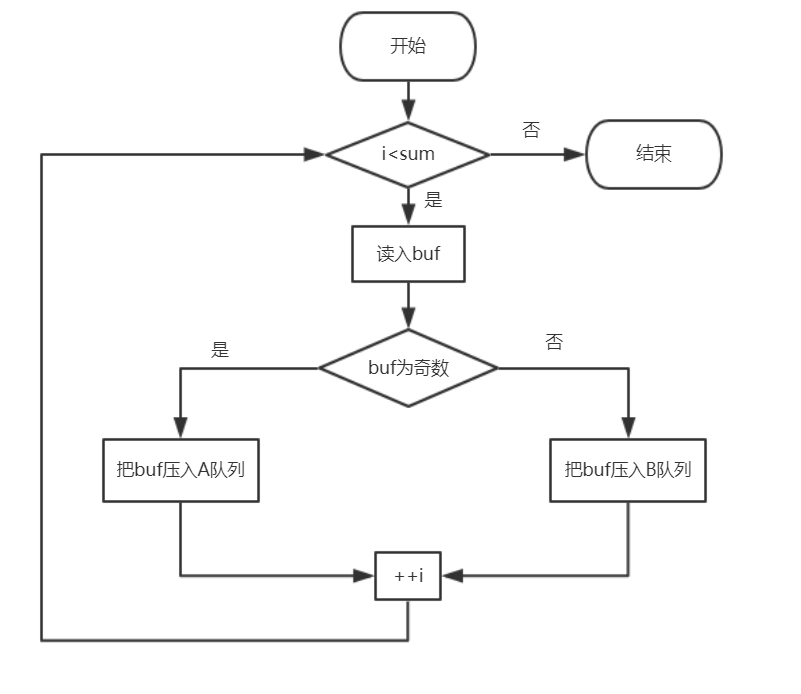
## 2.4 系统设计

系统首先接收由用户指定的顾客人数sum，调用构造函数定义Bank类型的对象，之后调用Bank类的方法dealBusiness()处理银行业务最后调用showOrder()输出处理顺序。

# 3 实现

## 3.1 读入顾客功能的实现

### 3.1.1 读入顾客功能流程图



### 3.1.2 读入顾客功能核心代码

void Bank::Init(int sum)

{

int buf;

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

{

cin >> buf;

if (buf % 2) //如果是奇数则加入A窗口

{

QueueA.push(buf);

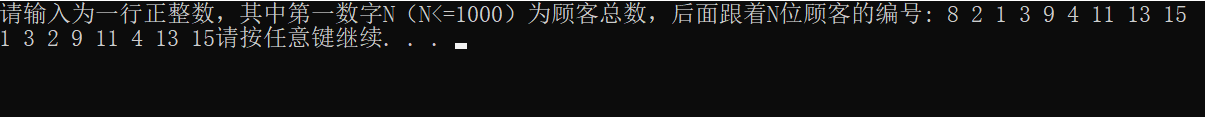
}

else { QueueB.push(buf); } //否则加入B窗口

}

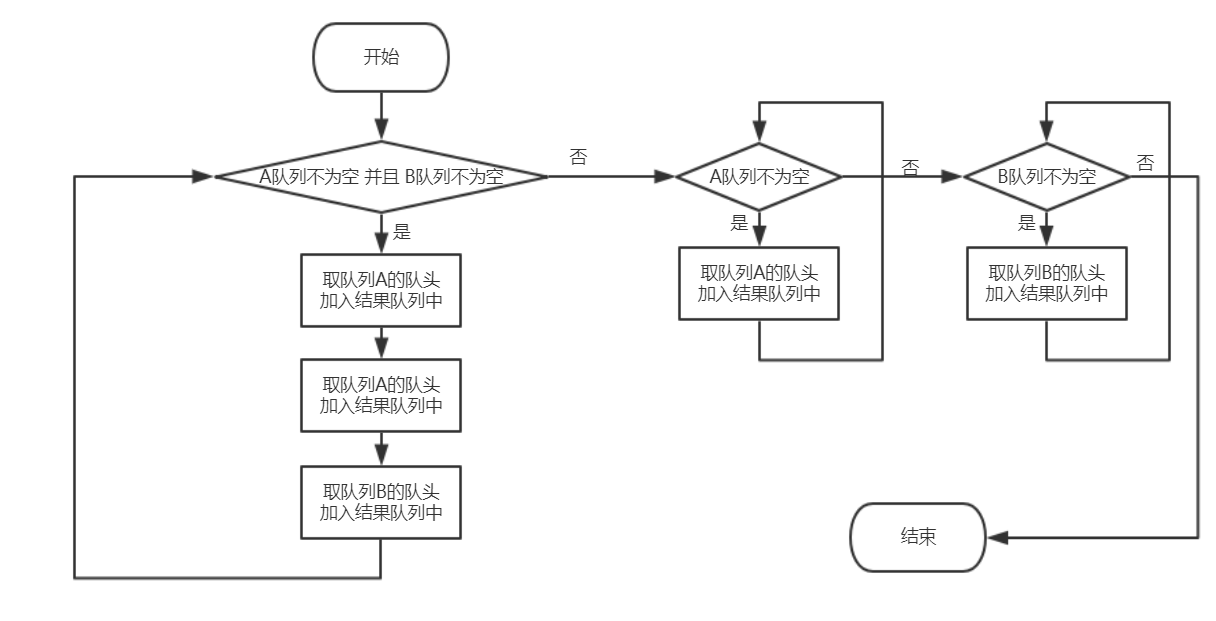
}

### 3.1.3 读入顾客功能截屏示例



## 3.2 处理业务功能的实现

### 3.2.1 处理业务功能流程图



### 3.2.2 处理业务功能核心代码

void Bank::dealBusiness()

{

while (!QueueA.empty() && !QueueB.empty())

//两个窗口对应的队列都不为空时

//按照A窗口处理速度是B窗口的两倍 把结果计入结果队列中

{

Qresult.push(QueueA.front());

QueueA.pop();

Qresult.push(QueueA.front());

QueueA.pop();

Qresult.push(QueueB.front());

QueueB.pop();

}

while (!QueueA.empty())

//如果A有剩余,把A的其余部分直接加入结果中

//此时B一定为空

{

Qresult.push(QueueA.front());

QueueA.pop();

}

while (!QueueB.empty())

//同理,如果B有剩余,则把B的起源于部分直接加入结果中

{

Qresult.push(QueueB.front());

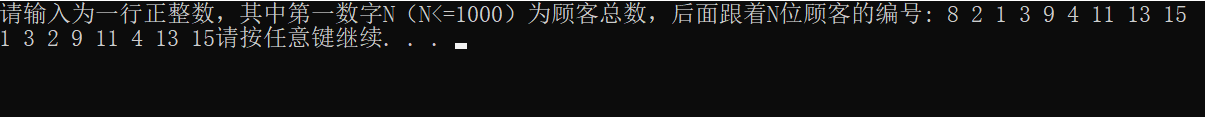
QueueB.pop();

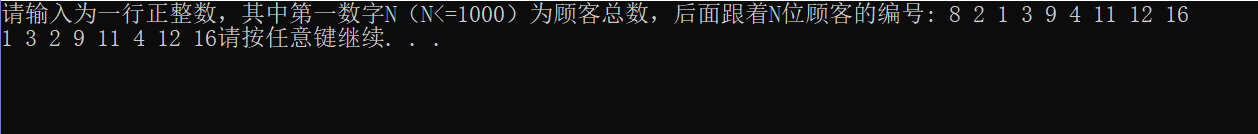
}

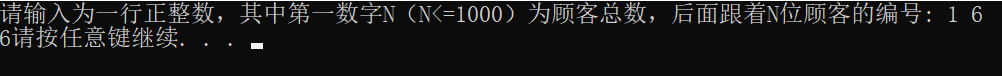
}

}

### 3.2.3 处理业务功能截屏示例

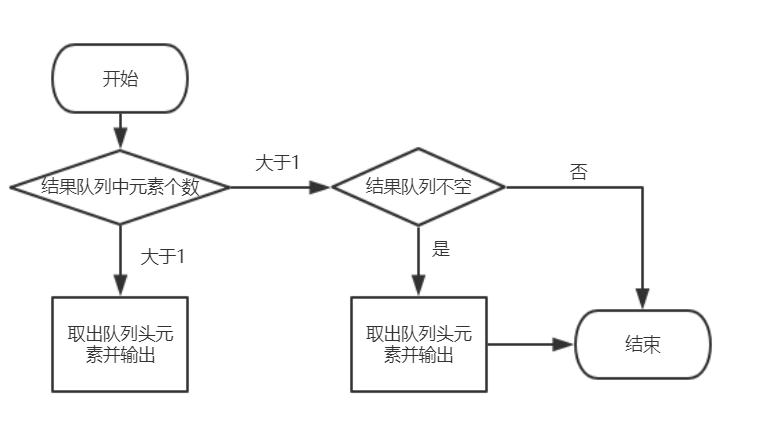






## 3.3 输出功能的实现

### 3.3.1 输出功能流程图



### 3.3.2 输出功能核心代码

void Bank::showOrder()

{

while (Qresult.size() > 1)

//输出结果

{

cout << Qresult.front() << ' ';

Qresult.pop();

}

if (Qresult.size())

//如果结果队列中有最后一个元素, 则输出它并且结尾无空格

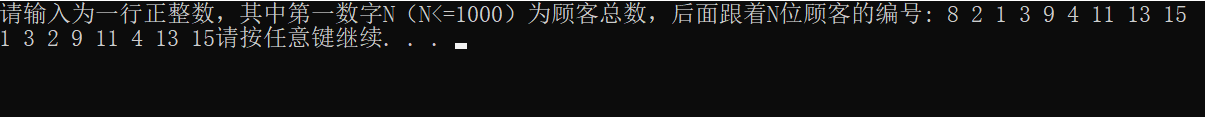
{

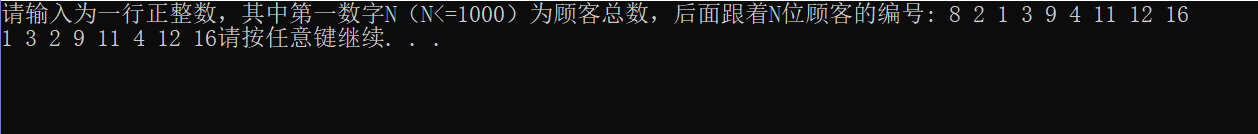
cout << Qresult.front();

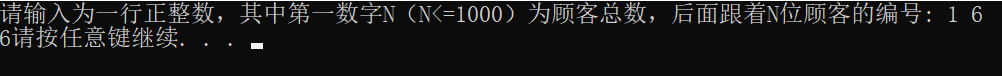
}

### }

### 3.3.3 输出功能截图示例







## 3.4 总体系统的实现

### 3.4.1 总体系统核心代码

int sum;

cout << "请输入为一行正整数，其中第一数字N（N<=1000）为顾客总数，后面跟着N位顾客的编号: ";

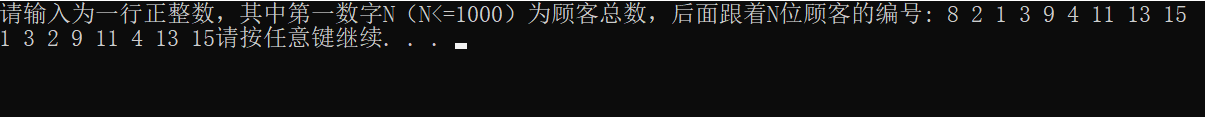
cin >> sum; //由于输入数据为一整行,因此没办法对总人数做检查

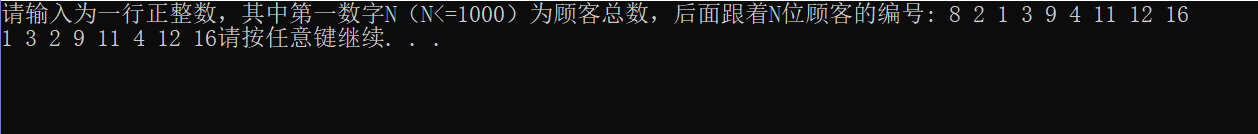
Bank bank(sum);

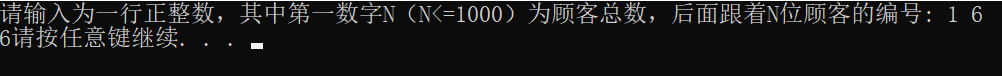
bank.dealBusiness();

bank.showOrder();

### 3.4.2 总体系统截屏示例







# 4 测试

## 4.1 功能测试

### 4.1.1 读入顾客功能测试

**测试用例**：8 2 1 3 9 4 11 13 15

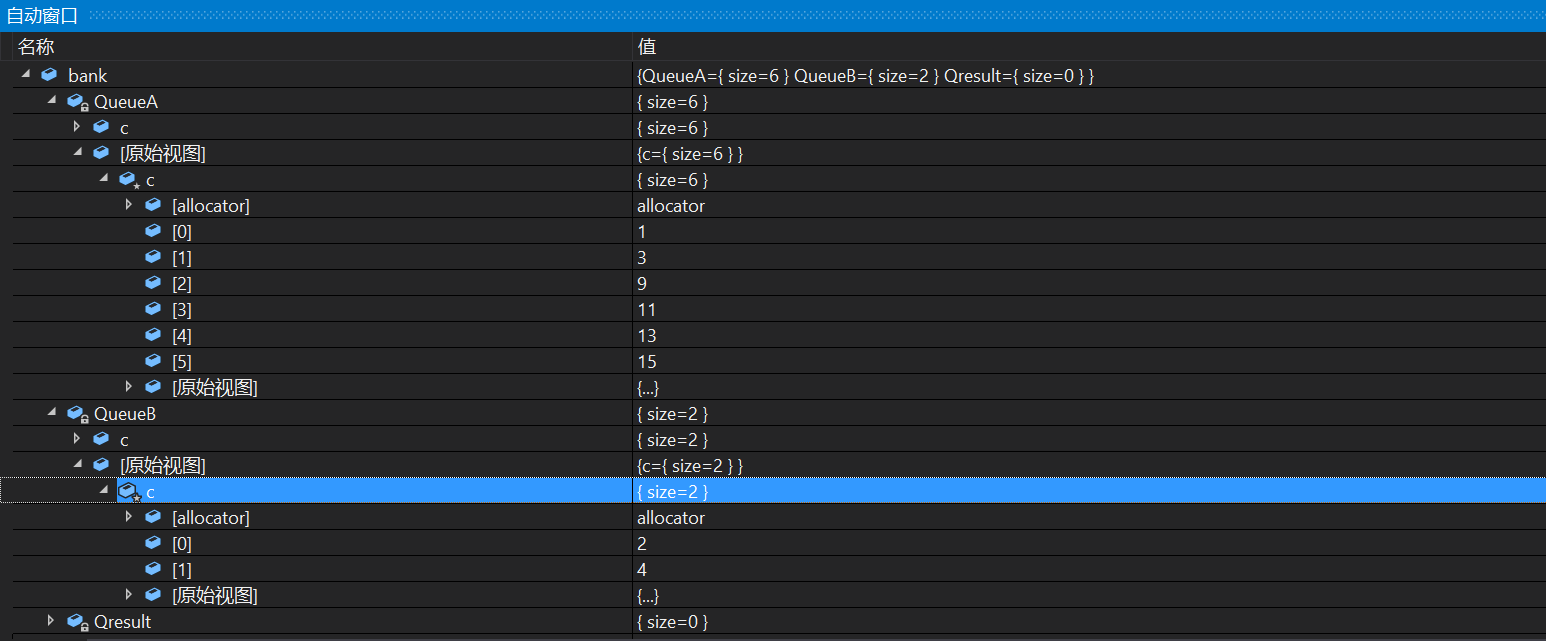
**预期结果**：

正确储存在QueueA和QueueB中

QueueA中储存1 3 9 11 13 15

QueueB中储存2 4

**实验结果**



**测试用例**：8 2 1 3 9 4 11 12 16

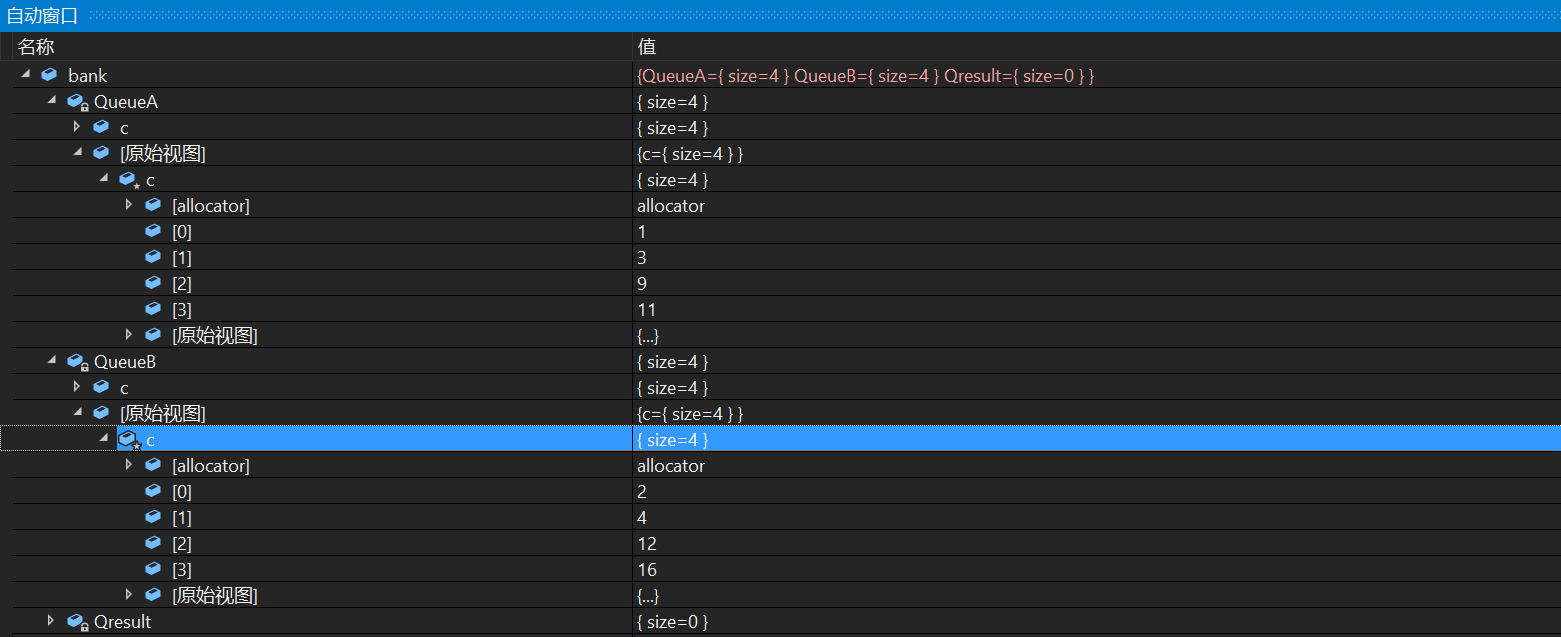
**预期结果**：

正确储存在QueueA和QueueB中

QueueA中储存1 3 9 11

QueueB中储存2 4 12 16

**实验结果：**



**测试用例**：1 6

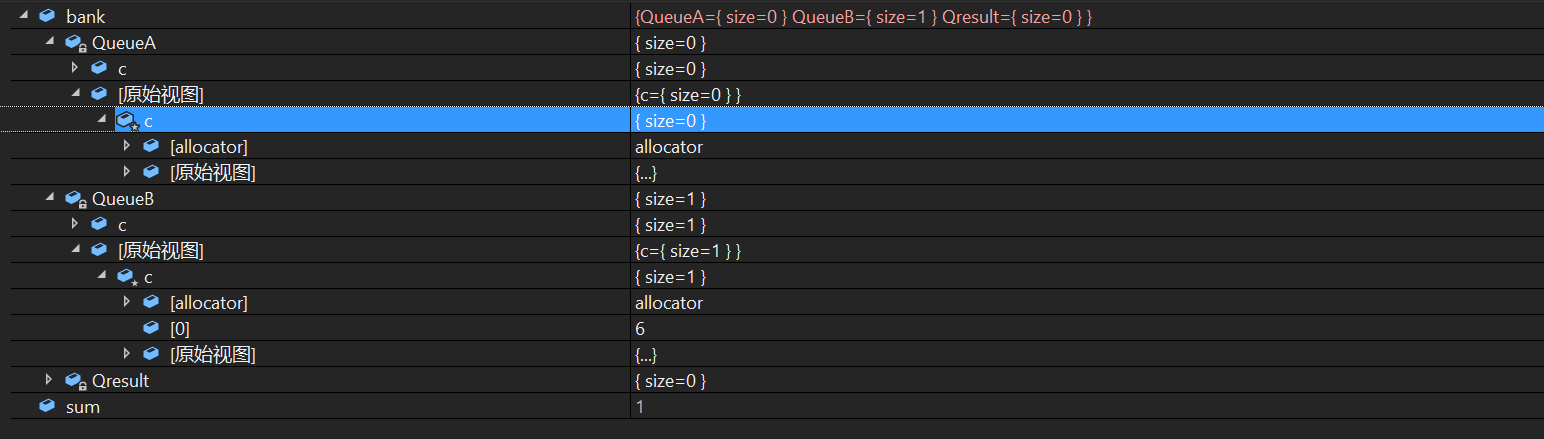
**预期结果**：

正确储存在QueueA和QueueB中

QueueA为空

QueueB中储存6

**实验结果：**



### 4.1.2 业务处理功能测试

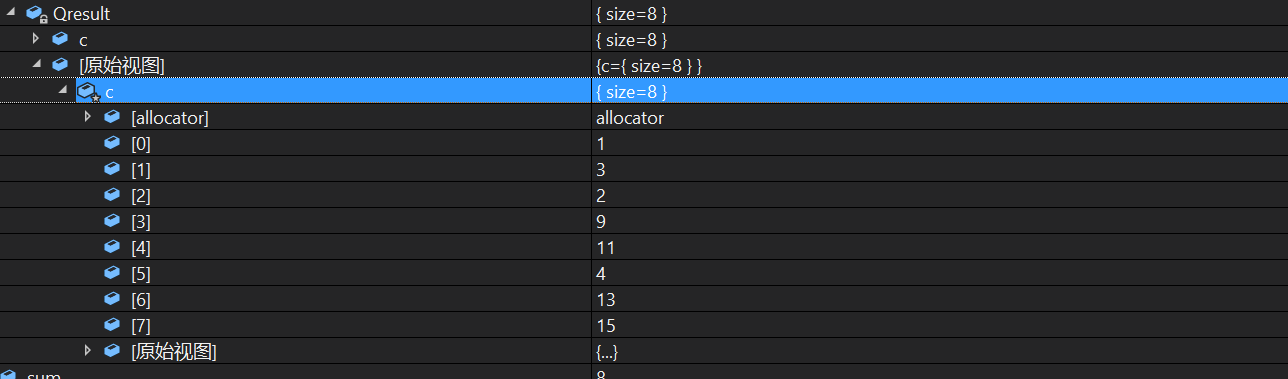
**测试用例**：8 2 1 3 9 4 11 13 15

**预期结果**：

Qresult正确储存带处理序列

Qresult中存储1 3 2 9 11 4 13 15

**实验结果：**



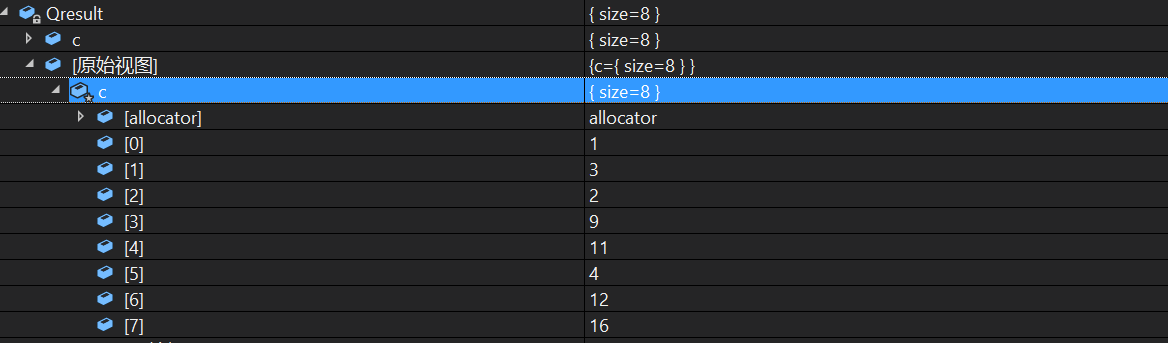
**测试用例**：8 2 1 3 9 4 11 12 16

**预期结果**：

Qresult正确储存带处理序列

Qresult中存储1 3 2 9 11 4 12 16

**实验结果：**



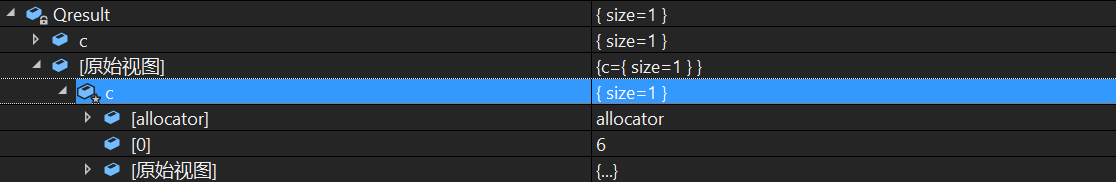
**测试用例**：1 6

**预期结果**：

Qresult正确储存带处理序列

Qresult中存储6

**实验结果：**



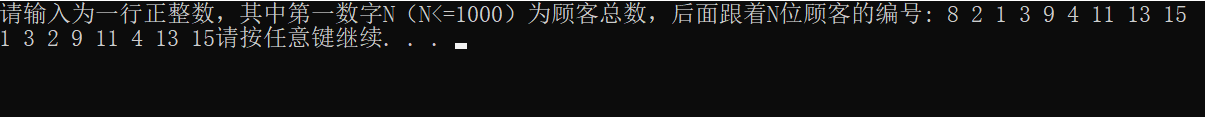
### 4.1.3 输出功能测试

**测试用例**：8 2 1 3 9 4 11 13 15

**预期结果**：

输出1 3 2 9 11 4 13 15 （结尾没有多余的空格）

**实验结果：**

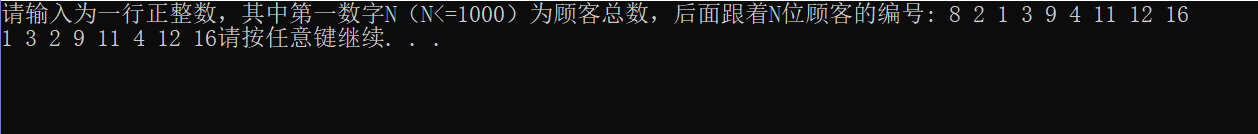


**测试用例**：8 2 1 3 9 4 11 12 16

**预期结果**：

输出1 3 2 9 11 4 12 16 （结尾没有多余的空格）

**实验结果：**

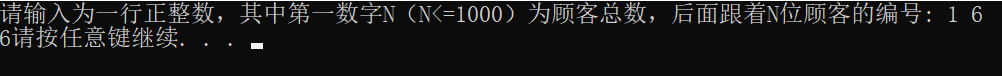


**测试用例**：1 6

**预期结果**：

输出6 （结尾没有多余的空格）

**实验结果：**



## 4.2 边界测试

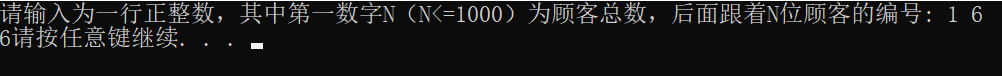
### 4.2.1 最小顾客人数

**测试用例**：1 6

**预期结果**：

输出6 （结尾没有多余的空格）

**实验结果：**



## 4.3 出错测试

由于本题目要求输入为一行正整数，所以没办法对输入人数做检测，以达到如果输入人数不满足题设要求可以让用户自定义重新输入