

课程设计

|  |  |  |  |
| --- | --- | --- | --- |
| 学院名称： | 信息产业学院 | | |
| 课程名称： | 数据结构 | | |
| 设计名称： | 银行排队系统 | | |
| 专业班级： | 211060104 | | |
| 成员1： | 21106010413 | 成员2： | 21106010401 |
| 成员3： | 21106010403 | 成员4： | 21106010428 |
| 学生姓名： | 刘钟泽、师玉娜、任嘉钰、赵昊峰 | | |
| 指导教师： | 高俊 | | |

2023年6月7日

目录

[一、 项目分析 4](#_Toc137222582)

[1.1 项目背景 4](#_Toc137222583)

[1.2 项目需求分析 5](#_Toc137222584)

[1.3 项目要求 5](#_Toc137222585)

[**1.3.1** 功能要求 5](#_Toc137222586)

[**1.3.2** 输入格式 6](#_Toc137222587)

[**1.3.3** 输出格式 6](#_Toc137222588)

[**1.3.4** 项目示例 6](#_Toc137222589)

[二、 项目设计 6](#_Toc137222590)

[2.1 数据结构设计 6](#_Toc137222591)

[2.2 类设计 6](#_Toc137222592)

[**2.2.1** 结点类（ListNode） 8](#_Toc137222593)

[**2.2.2** 双向链表类（List） 8](#_Toc137222594)

[**2.2.3** 队列类（Queue） 10](#_Toc137222595)

[三、 项目实现 12](#_Toc137222596)

[3.1 项目主体功能流程图 12](#_Toc137222597)

[3.2 项目主体功能代码 12](#_Toc137222598)

[**3.2.1** 数据结构设计 12](#_Toc137222599)

[**3.2.2** 类结构设计 12](#_Toc137222600)

[**3.2.3** 模板类以及链式队列类（List\_Queue） 14](#_Toc137222601)

[**3.2.4** 队列比较类（Compare\_Queue） 15](#_Toc137222602)

[**3.2.5** 成员与操作设计，具体链式队列函数实现： 16](#_Toc137222603)

[四、 用户使用说明 22](#_Toc137222604)

[4.1 银行业务功能流程简介 22](#_Toc137222605)

[4.2 用户使用方式 23](#_Toc137222606)

[五、 实现 24](#_Toc137222607)

[5.1 银行业务功能的实现 24](#_Toc137222608)

[**5.1.1** 银行业务功能核心代码 24](#_Toc137222609)

[**5.1.2** 银行业务功能截屏示例 27](#_Toc137222610)

[**5.1.3** 银行业务功能边界测试 29](#_Toc137222611)

[六、 设计小结 30](#_Toc137222612)

[6.1 可提升的点 30](#_Toc137222613)

[6.2 团队总结 31](#_Toc137222614)

[**6.2.1** 课设方面 31](#_Toc137222615)

[**6.2.2** 社会方面 31](#_Toc137222616)

[**6.2.3** 个人方面 31](#_Toc137222617)

[七、 主要参考文献和附录 31](#_Toc137222618)

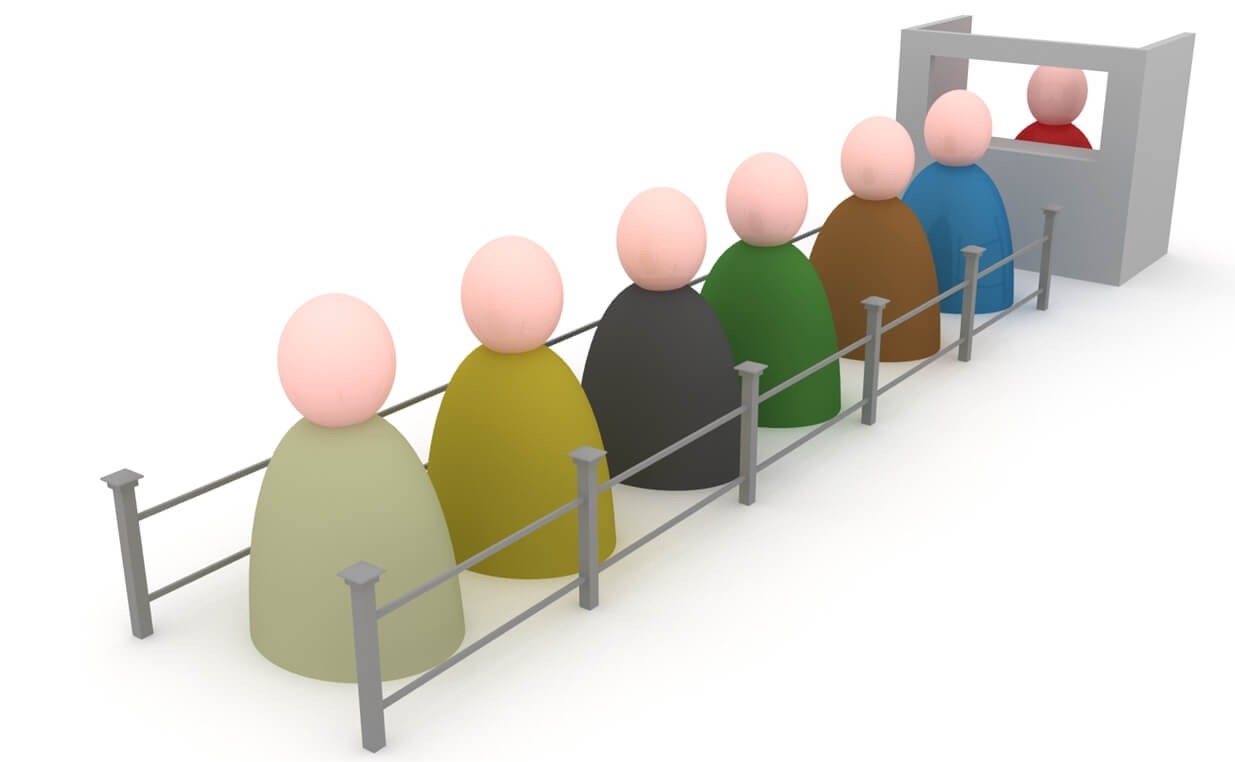
[7.1 参考文献 31](#_Toc137222619)

[7.2 附录：源代码 33](#_Toc137222620)

# 项目分析

## 项目背景

我们经常会遇到各种各样需要排队等待的场景。无论是在校园里，在食堂打饭、在图书馆借书、在教室签到；还是在社会上，在银行办理业务、在医院挂号、在商场结账，都需要我们按照一定的顺序排队。这些场景中，我们可以把每个等待者看作一个节点，并用指针连接起来形成一个链表。这样就可以方便地对链表进行增加、删除和遍历的操作，实现排队的功能。链表是一种动态的数据结构，它不需要预先分配固定大小的空间，而是根据需要动态地分配和释放节点。这样就可以避免数组那样需要一片连续的空间，而且可能会造成空间浪费或者空间不足的问题。链表也有不同的类型，如单链表、双链表、循环链表等，它们各有各的优缺点和适用景。

图 1.1‑1排队图

排队不仅是一种生活常态，也是一种社会规范。在很多领域，排队还是一种必要的方式，如公共交通、景点门票等。例如，为了保证公交车的运行效率和乘客的安全，公交车站通常会设置专门的上车区域，让乘客按照线性排队或者两端开口排队的方式上车。在一些热门的景点，为了缓解排队拥堵的问题，采用了线上预订和取票的方式，让游客可以提前安排好行程，避免浪费时间。 排队对于个人和社会都有着重要的意义。首先，排队可以维护公共秩序，防止发生拥挤和混乱的情况，保障人们的生命财产安全。如果没有人自觉排队，可能会导致各种事故和纠纷，影响社会稳定。其次，排队可以体现文明素养，展现人与人之间的相互尊重和关怀，营造出良好的社会氛围。

因此，国家制定了排队日，旨在希望人们在排队的一点一滴中形成一种习惯和意识，让更多的人不自觉地履行这一公民的基本义务。 在实际排队过程中，除了人的自身素质因素外，还涉及到许多复杂的数学模型。比如，M/M/1排队模型是最基本和最常用的排队模型 ，其 中M表示顾客到达服从泊松分布 ，M也表示服务时间服从指数分布 ，1则表示只有一个服务器 。这个模型可以用来描述很多简单的排队场景，如银行柜台、餐厅收银台等。当然，在实际场景中，各种因素都会对排队 产生影响，如顾客的到达间隔不稳定 ，需要考虑开放多个服务窗口 等等。因此，研究排队问题具有极高的现实应用价值。 无论是理论研究还是应用探讨，排队都是一个值得深入探究的领域。实际上，有许多专业的排队研究机构和排队管理软件在帮助我们进一步提高排队的效率和质量 ，为我们的日常生活提供更加便捷的服务。毫无疑问，人们应该认识到排队的重要性，自觉地排队，从而构建更加和谐、安全的社会环境。

因此我们的研究很有意义。

## 项目需求分析

针对于银行业务这一项目，本项目在实现的过程中，应当能够满足以下的需求：

* **功能完善**

系统应当能够满足基本需求，即能够正确的按照完成顺序输出顾客的编号。

* **执行效率高**

针对数据量比较大的情况，本系统也应该具有在较短时间内求解出正确答案的能力。

* **健壮性**

当用户输入的数据非法时，系统应当能够识别并处理错误，而非直接崩溃退出。

## 需求预分析

* **功能完善**

为了实现这一功能，我们可以使用一个队列来存储顾客的编号，并按照先进先出的原则进行出队操作。队列是一种适合用于排队场景的数据结构，它可以用链表来实现，也可以用数组来实现。链表实现的队列可以动态地分配和释放空间，而数组实现的队列需要预先确定一个固定大小的空间。

* **执行效率高**

为了提高执行效率，我们可以使用一些优化算法来减少不必要的计算和存储。比如，我们可以使用一个变量来记录当前队列中最大的编号，并在每次出队或入队时更新这个变量。这样就可以避免每次输出时都要遍历整个队列来寻找最大值。

* **健壮性**

为了增强健壮性，我们可以使用一些异常处理机制来捕获和处理可能发生的错误。比如，我们可以使用try-catch语句来捕获输入数据格式不正确、队列为空、队列溢出等异常，并给出相应的提示信息或者恢复措施。

## 项目要求

### 功能要求

某银行有A，B两个业务窗口，且处理业务的速度不一样，其中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。给定到达银行的顾客序列，请按照业务完成的顺序输出顾客序列。假定不考虑顾客信后到达的时间间隔，并且当不同窗口同时处理完2个顾客时，A窗口的顾客优先输出。

### 输入格式

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。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。数字间以空格分隔。

### 输出格式

按照业务处理完成的顺序输出顾客的编号。数字键以空格分隔，但是最后一个编号不能有多余的空格。

### 项目示例

表 1.3‑1输入输出示例

|  |  |  |  |
| --- | --- | --- | --- |
| 序号 | 输入 | 输出 | 说明 |
| 1 | 8 2 1 3 9 4 11 13 15 | 1 3 2 9 11 4 13 15 | 正常测试，A窗口人多 |
| 2 | 8 2 1 3 9 4 11 12 16 | 1 3 2 9 11 4 12 16 | 正常测试，B窗口人多 |
| 3 | 1 6 | 6 | 最小N |

# 项目设计

## 数据结构设计

如上述功能分析所示，本项目需要实现一种先进先出的数据结构，也即队列。因此，本项目需要使用队列这一数据结构，从而满足所需要项目需求。

## 类设计

### 类设计概述

本项目中使用到队列（Queue）这一数据结构，而队列的底层数据类型则采用了链表进行实实现。经典的链表一般包括两个抽象数据类型（ADT）——链表结点类（ListNode）与链表类（List），而两个类之间的耦合关系可以采用嵌套、继承等多种关系。

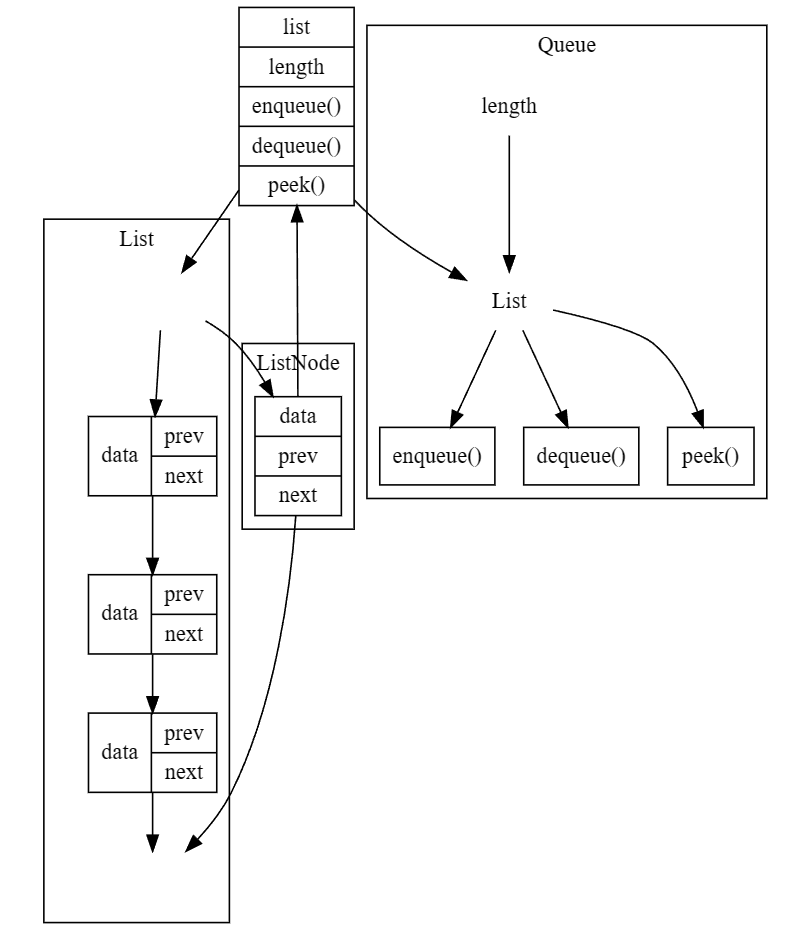


图 2.2‑1类的结构设计

为了实现代码的复用性，本系统实现了一个**链表**。采用struct描述链表结点类（ListNode），这样使得链表结点类（List）可以直接访问链表结点而不需要定义友元关系。本系统实现的链表结构各种操作的时间复杂度如下：

* 插入操作：O(n)
* 删除操作：O(n)
* 查询操作：O(n)
* 遍历操作：O(n)

### 结点类（ListNode）

链表的结点中存储的数据有链表数据、后继结点和前驱结点。其UML图如下所示：

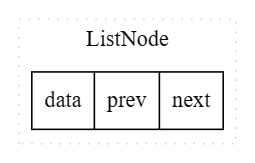


图 2.2‑2结点类的结构设计

### 双向链表类（List）

链表的实现原理大同小异，不同之处在于：是否带头结点、是否带尾结点、每一个结点是否带前驱结点等。为了使得链表中各种操作的时间复杂度都尽可能低，因此这里选择了带头结点和尾结点的双向链表来实现链表中的各种操作。也即：选择了牺牲空间来达到降低时间复杂度的效果。

链表的UML图如下所示：

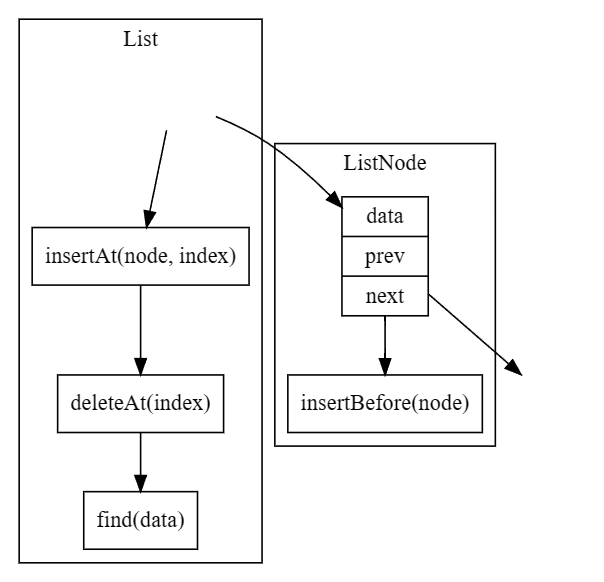


图 2.2‑3双链表类的结构设计

为了便于对链表进行遍历、插入、删除、查找等操作，增加了一个iterator类。iterator类内部存储一个链表节点指针，同时，通过运算符重载相应的自增、自减、判等等操作。

iterator类、ListNode类和List类的关系如下图所示：

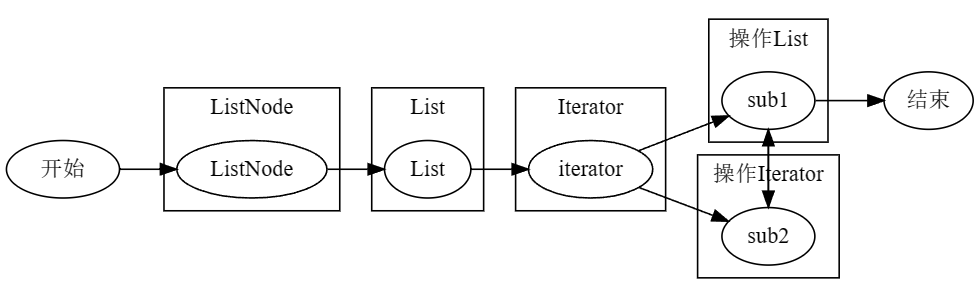


图 2.2‑4iterator类、ListNode类和List类的关系

其中，List类中的主要函数如下所示：

* inline int **size**()const

返回链表中结点的个数，不包括头结点。

* inline bool **empty**()const

判断链表是否为空，也即链表中结点的个数是否为0。

* void **push\_back**(**Type** data)

在链表尾部插入新的数据，也即新增一个结点并且加入到链表末端。

* void **push\_front**(**Type** data)

在链表头部插入新的数据，也即新增一个结点并且加入到链表的头部。

* **iterator** **find**(const **Type**& data)const

在链表中查找值为data的元素是否存在，返回该位置的迭代器，若查找失败返回空指针对应的迭代器。

* **Type** **remove**(**iterator** index)

移除迭代器所处位置的元素，返回移除位置元素的值。

* void **erase\_last**()

移除链表末端的元素，即最后的结点。

* void **erase\_first**()

移除链表首端的元素，即第一个结点。

* void **clear**()

清空链表，即删除链表中所有的结点。

* **iterator** **begin**()

返回链表第一个元素所在位置的迭代器。

* **iterator** **end**()

返回链表尾结点后的空结点的迭代器。

### 队列类（Queue）

队列是计算机程序中常用的数据结构，常常用于计算机模拟现实事务，如：排队等。队列是一种特殊的线性表，特殊之处在于它只允许在表的前端（front）进行删除操作，而在表的后端（rear）进行插入操作，因此是一种操作受限制的线性表。进行插入操作的端称为队尾，进行删除操作的端称为队头。队列中没有元素时，被称为空队列。

队列的数据元素又称为队列元素。在队列中插入一个队列元素称为入队，从队列中删除一个队列元素称为出队。因为队列只允许在一端插入，在另一端删除，所以只有最早进入队列的元素才能最先从队列中删除，故队列又称为先进先出（FIFO—first in first out）线性表。

队列的包括了顺序队列和循环队列，实现存储的底层数据可以通过向量或者链表完成。本项目中的队列以链表作为底层数据结构，实现了顺序队列。其UNL图如下所示：

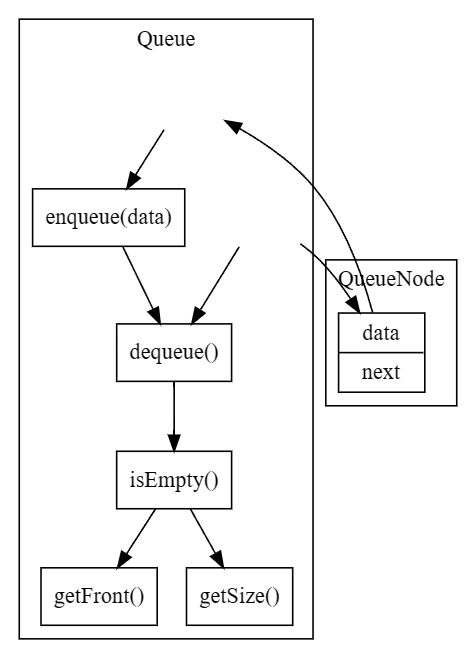


图 2.2‑5队列类的设计

队列中主要函数如下所示：

* inline bool **empty**()const

判断队列是否为空，也即队列内部链表是否为空。

* inline int **size**()const

返回队列中链表节点的个数。

* void **push**(const **Type**& i)

在队列尾部加入一个元素，也即入队。

* void **pop**()

删除队列头部的元素，也即出队。

* const **Type**& **front**()const

获取队首的元素值。

通过上述函数操作，即完成了一个队列所需要的最基本操作。其中，队的各个操作的时间复杂度如下所示：

* 入队操作：O(1)
* 出队操作：O(1)
* 读取操作：O(1)

# 项目实现

## 项目主体功能流程图

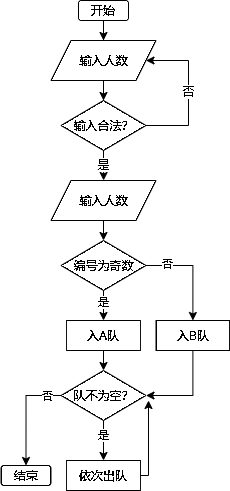


图 3.1‑1项目主体功能流程图

## 项目主体功能代码

### 数据结构设计

该项目明显是对队列的一个应用实现典例，此题采用了链式队列进行操作，构建三个链式队列A，B，C用以解决问题。

### 类结构设计

模板类以及具体链式队列的实现类，以链表结点，头尾指针作为数据成员

为实现封装性：私有数据成员

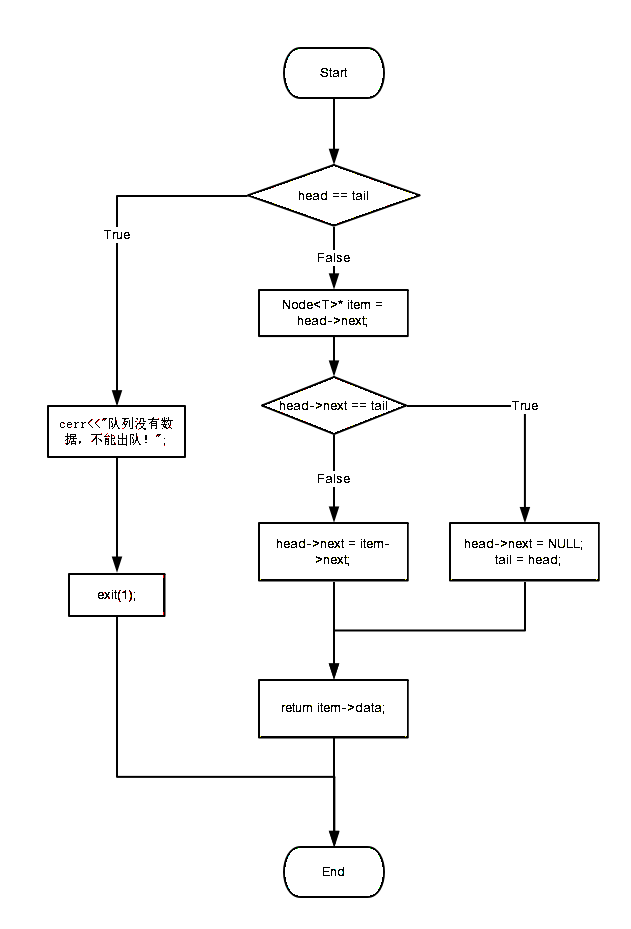


图 3.1‑2出列图

struct Link\_node //队列结点，包含数据以及指针

{

T data;

Link\_node\* next;

};

Link\_node\* rear, \* front; //队尾，队头指针

### 模板类以及链式队列类（List\_Queue）

#pragma once

#include<iostream>

using namespace std;

template<class T>

class Queue //队列模板类创建

{

public:

virtual void clear() = 0; //将队列清空

virtual int size()const = 0; //求队列长度

virtual void In\_Queue(T& x) = 0; //入队列

virtual T Out\_Queue() = 0; //出队列

virtual T Read\_Queue()const = 0; //读取队头元素

virtual bool Is\_empty()const = 0; //判断队列是否为空

};

template<class T>

class Link\_Queue :public Queue<T>

{

public:

Link\_Queue() { front = NULL; rear = NULL; } //构造函数将队头，队尾指针置空

~Link\_Queue() { clear(); } //析构函数清除队列

void clear(); //将队列清空

int size()const; //求队列长度

void In\_Queue(T& x); //入队列

T Out\_Queue(); //出队列

T Read\_Queue()const; //读取队头元素

bool Is\_empty()const; //判断队列是否为空

private:

struct Link\_node //队列结点，包含数据以及指针

{

T data;

Link\_node\* next;

};

Link\_node\* rear, \* front; //队尾，队头指针

};

### 队列比较类（Compare\_Queue）

class Compare\_Queue //比较队列类

{

private:

Link\_Queue<int>Queue\_A; //定义三个int型队列

Link\_Queue<int>Queue\_B;

Link\_Queue<int>Queue\_C;

public:

Compare\_Queue() {}; //构造函数

~Compare\_Queue() {}; //析构函数

void Handling(); //办理业务实现函数

};

### 成员与操作设计，具体链式队列函数实现：

template<class T>

void Link\_Queue<T>::clear() //清空队列

{

Link\_node\* p;

while (front != NULL) //当队列头指针不为空（即队列不为空时）进行头指针后移逐个删除队列元素

{

p = front;

front = front->next;

delete p;

}

rear = NULL; //将尾指针置空

}

template<class T>

int Link\_Queue<T>::size()const

{

Link\_node\* p = front;

int count = 0;

while (p != NULL) //当头指针不为空时依次后移。count++，进行计数

{

count++;

p = p->next;

}

return count;

}

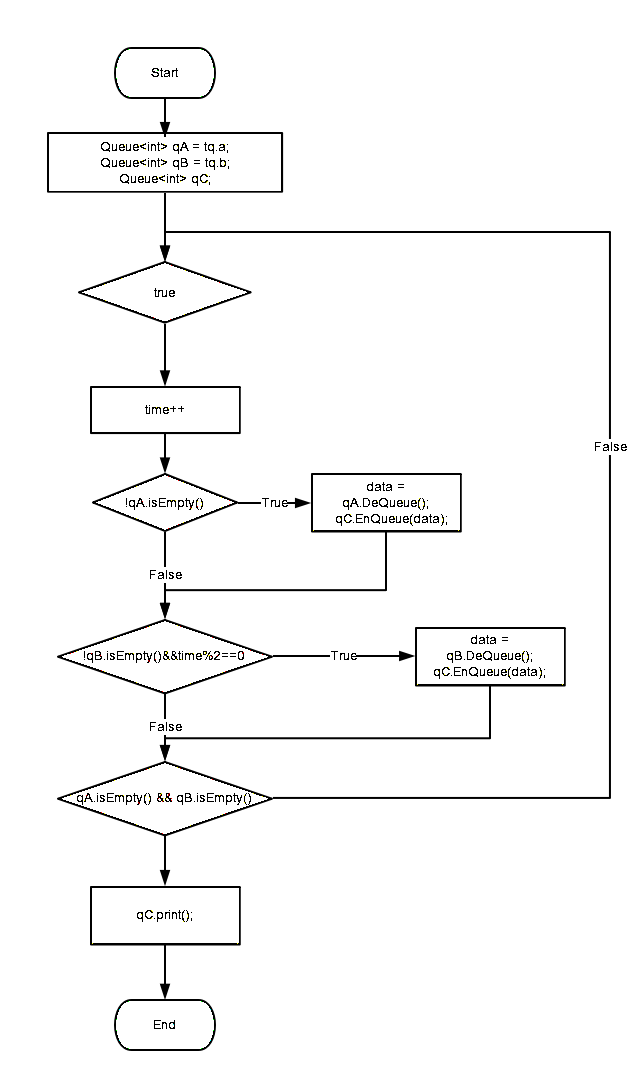


图 3.1‑3ostreamfun类

template<class T>

void Link\_Queue<T>::In\_Queue(T& x)

{

Link\_node\* p = new Link\_node;

if (p == NULL)

{

cout << "存储分配错误！" << endl; //检查开辟新结点是否成功

exit(1);

}

if (rear == NULL) //如果此队列中无元素则进行插入第一个元素操作

{

p->data = x;

rear = p;

front = p;

rear->next = NULL;

}

else //如果队列中有元素则进行在队尾插入操作

{

p->data = x;

rear->next = p;

rear = p;

rear->next = NULL;

}

}

template<class T>

T Link\_Queue<T>::Out\_Queue()

{

Link\_node\* p = front;

T x;

while (front != NULL) //判断队列中是否有元素

{

if (p == NULL)

{

cout << "存储分配错误！" << endl; //判断建立结点是否成功

exit(1);

}

else //如果成功返回头指针数据值，并将头指针后移

{

p = front;

x = p->data;

front = front->next;

if (front == NULL) //原来只有一个元素出队后队列为空

{

rear == NULL; //修改队尾指针

}

delete p;

return x;

}

}

}

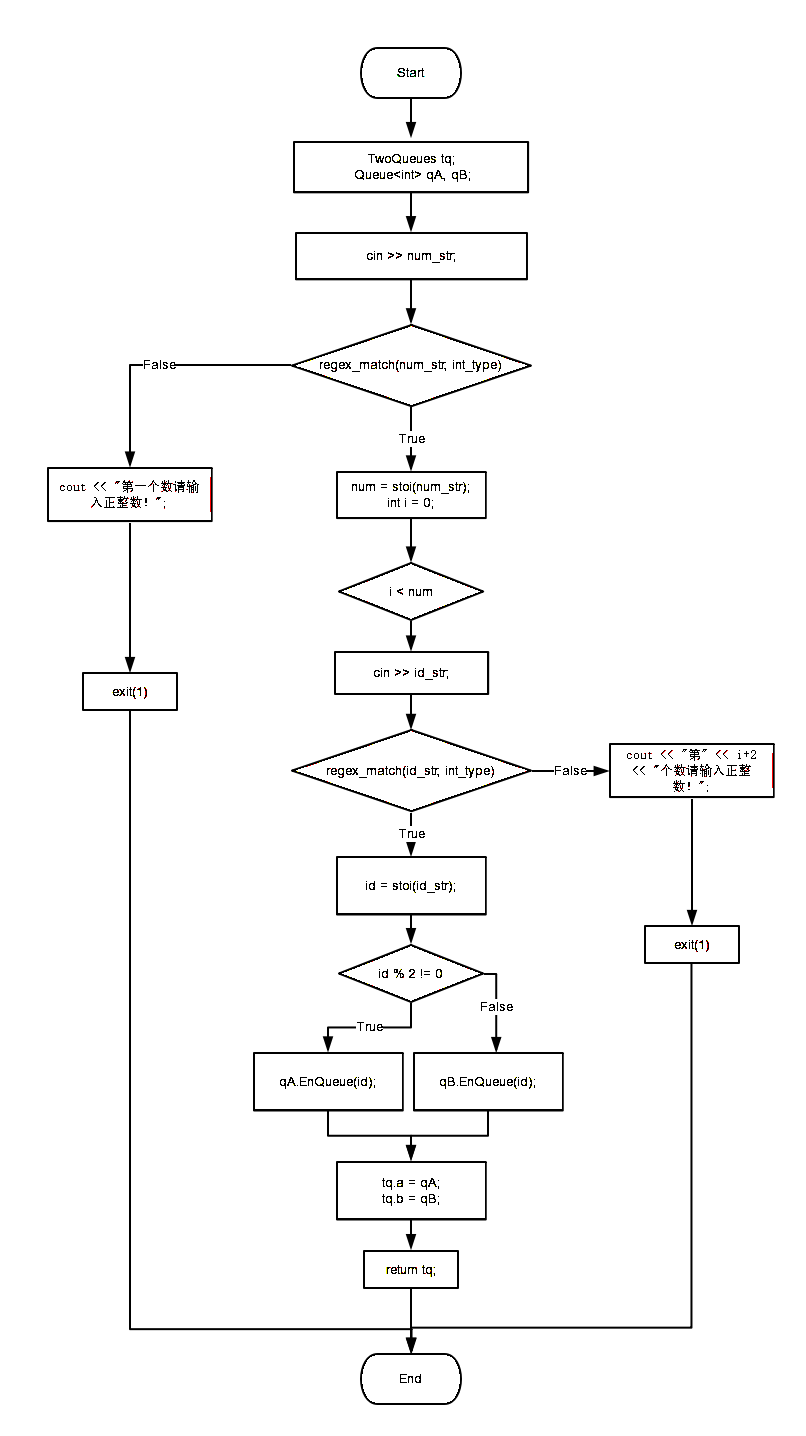


图 3.1‑4 istreamfun类

template<class T>

T Link\_Queue<T>::Read\_Queue()const

{

if (front != NULL) //如果队列中有元素，返回队头元素的值

{

return front->data;

}

else

{

cout << "队列中无元素，不可读取队列首元" << endl;

}

}

template<class T>

bool Link\_Queue<T>::Is\_empty()const

{

if (rear == NULL || front == NULL) //头指针或者尾指针为空则队列为空

{

return true;

}

else

{

return false;

}

}

# 用户使用说明

## 银行业务功能流程简介

银行业务功能是将输入的编号判断奇、偶以后分别输入到A、B中，根据题目所给规则输出队列到C中再进行输出C中元素

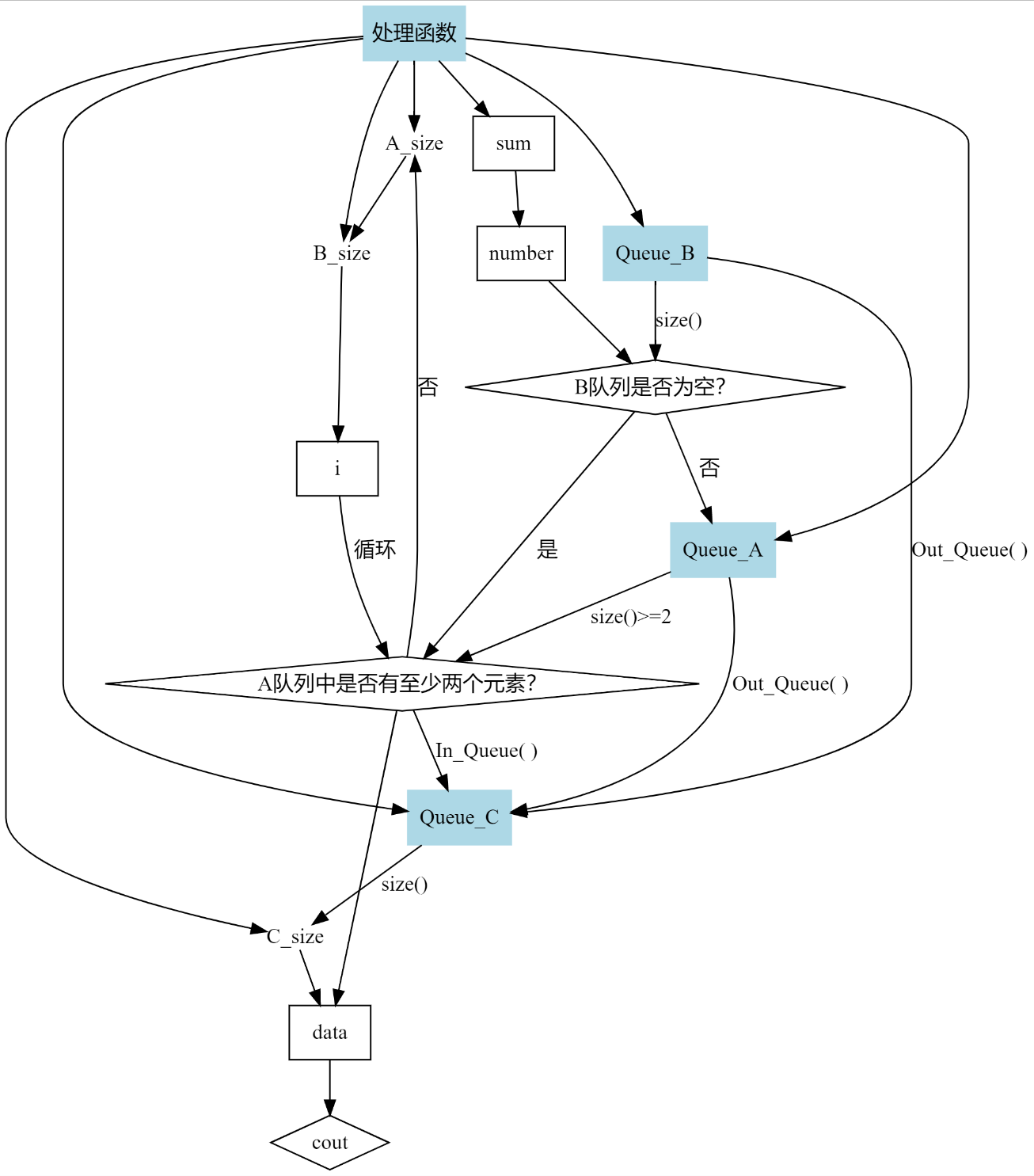


图 4.1‑1 银行业务功能流程

## 用户使用方式

银行排队业务是指在银行等候区，客户按照自己的需要排队等待服务的业务。银行排队业务的基本流程如下：

* 确认排队目的并在等待区等待；
* 当前一位顾客办理完毕后，前往柜台进行业务服务；
* 完成业务服务后，顾客离开柜台。

在代码中，使用了链队列来实现银行排队业务。其中，队列头指针为front，队列尾指针为rear，使用了模板类来定义不同类型的数据元素，包含了队列的入队、出队和判断队列是否为空等操作。具体实现中，分别将奇数和偶数编号的客户存储在不同的队列中，业务处理时，从A队列中取出两个客户，从B队列中取出一个客户，将三个客户依次存放在C队列中，然后根据条件进行不同的处理。如果A队列中的元素小于2个且B队列不为空，则先输出A中的元素，再输出B中的元素。如果B队列已经为空，则将A队列中的元素全部输出。最后，将C队列中的元素依次输出即可。

针对此代码，编写一个用户手册，详细说明银行排队业务的操作流程，包括以下两部分内容：

排队信息处理

* **顾客进入等待区：**当顾客进入银行后，前往等待区等待服务。在等待区，客户需要排队等候。
* **顾客办理业务：**当柜台业务人员处理完前一位顾客的业务后，会根据排队顺序呼叫下一位顾客前往柜台进行业务办理。
* **顾客离开柜台：**当顾客完成业务服务后，即可离开柜台，结束本次排队服务。

退出程序

用户可能存在排队等候的情况，但还未进行任何业务时需要离开，此时可以直接选择退出队列功能。

* **特殊情况处理：**当A队列中元素少于2，B队列不为空时，先将A队列元素全部输出，再将B队列元素全部输出；如果B队列已经为空，则将A队列中的元素全部输出；如果C队列中没有元素，则输出“无人在排队”。
* **操作顺序：**银行排队业务需要按照先进先出、先来先服务的原则进行操作，即先处理排在前面的客户，再处理排在后面的客户。

# 实现

## 银行业务功能的实现

### 银行业务功能核心代码

void Compare\_Queue::Handling(){

int sum, number = 0, data; //总的排队人数sum， 不同人的编号number，用于暂时存储数据

int A\_size, B\_size, C\_size; //记录A,B,C队列的长度

cin >> sum;

for (int i = 0; i < sum; i++){ //循环输入总人数以及人员编号

cin >> number;

if ((number % 2) == 0) //人员编号为偶数进入B队列

{

Queue\_B.In\_Queue(number);

}

else //人员编号为奇数进入A队列

{

Queue\_A.In\_Queue(number);

}

}

while ((Queue\_A.size() >= 2) && (Queue\_B.Is\_empty() == false)) //当A队列还有至少两个元素，且B队列不为空时

{

data = Queue\_A.Out\_Queue(); //输出A队列中的两个元素，B队列中的一个元素

Queue\_C.In\_Queue(data);

data = Queue\_A.Out\_Queue();

Queue\_C.In\_Queue(data);

data = Queue\_B.Out\_Queue();

Queue\_C.In\_Queue(data);

}

if ((Queue\_A.size() < 2) && (Queue\_B.Is\_empty() == false)) //当A队列中元素少于2，B队列不为空时

{

B\_size = Queue\_B.size();

A\_size = Queue\_A.size();

for (int i = 0; i < A\_size; i++) //先将A队列元素全部输出，再将B队列元素全部输出

{

data = Queue\_A.Out\_Queue();

Queue\_C.In\_Queue(data);

}

for (int i = 0; i < B\_size; i++)

{

data = Queue\_B.Out\_Queue();

Queue\_C.In\_Queue(data);

}

}

if (Queue\_B.Is\_empty() == true) //如果B队列中元素为空

{

A\_size = Queue\_A.size();

for (int i = 0; i < A\_size; i++) //将A中元素全部放入C中

{

data = Queue\_A.Out\_Queue();

Queue\_C.In\_Queue(data);

}

}

if (Queue\_C.Is\_empty() == false) //如果C中有元素

{

C\_size = Queue\_C.size();

for (int i = 0; i < C\_size; i++) //将C中元素全部输出

{

data = Queue\_C.Out\_Queue();

cout << data;

if (C\_size - 1 != i) //保证C输出元素最后一个后无括号

{

cout << " ";

}

}

}

else //C中无元素

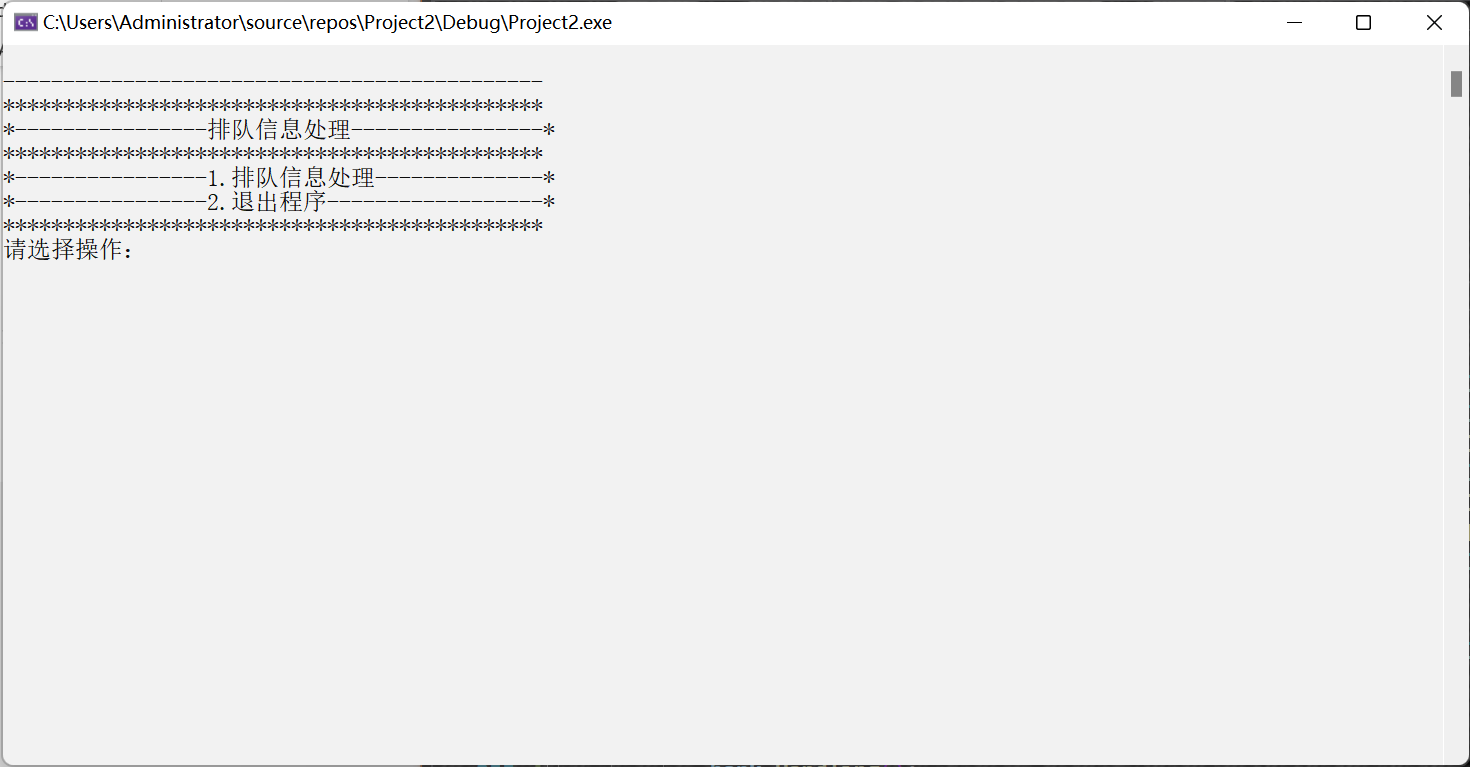
{

cout << "无人在排队" << endl;

}

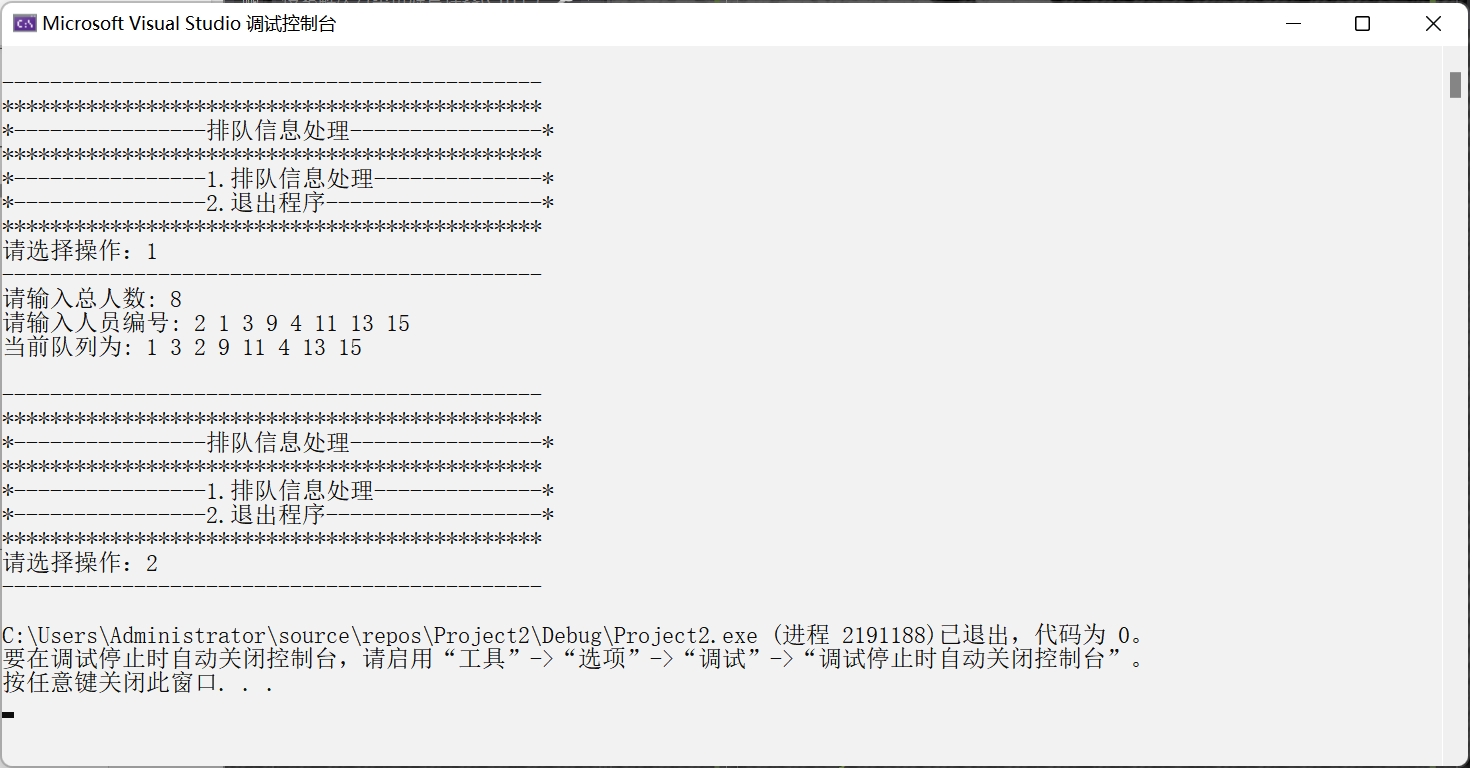
}

### 银行业务功能截屏示例

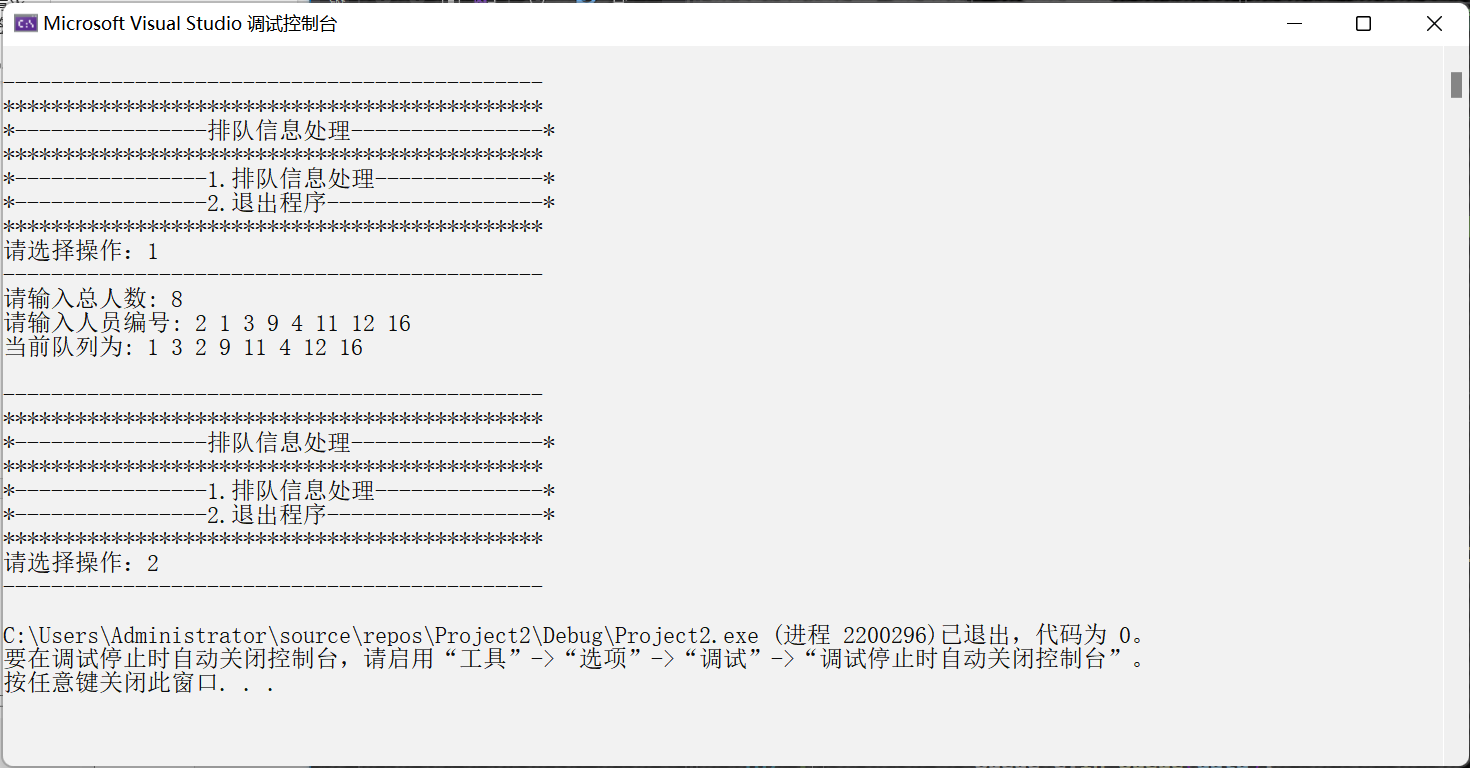


正常的银行业务功能展示，未涉及边界情况。

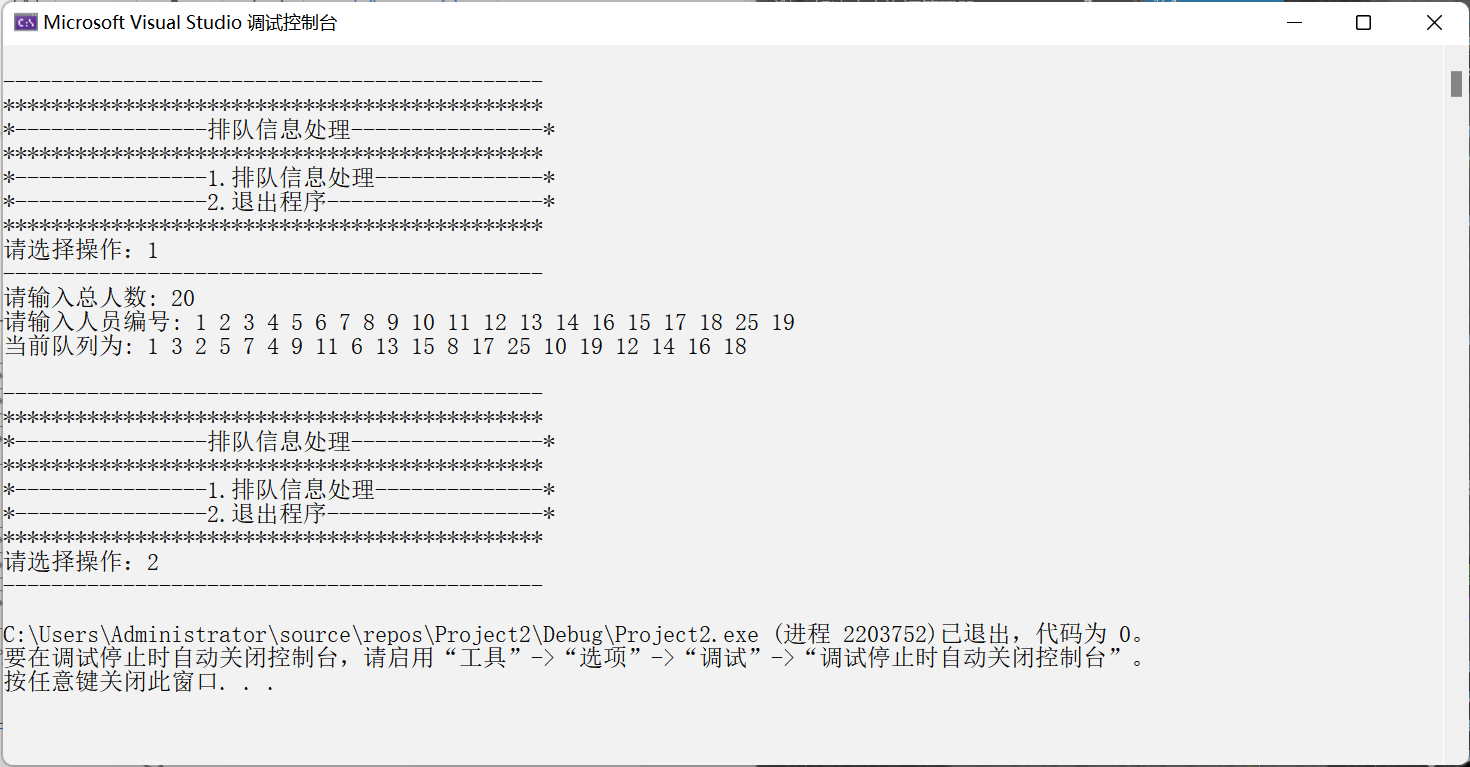
测试一：



测试二：

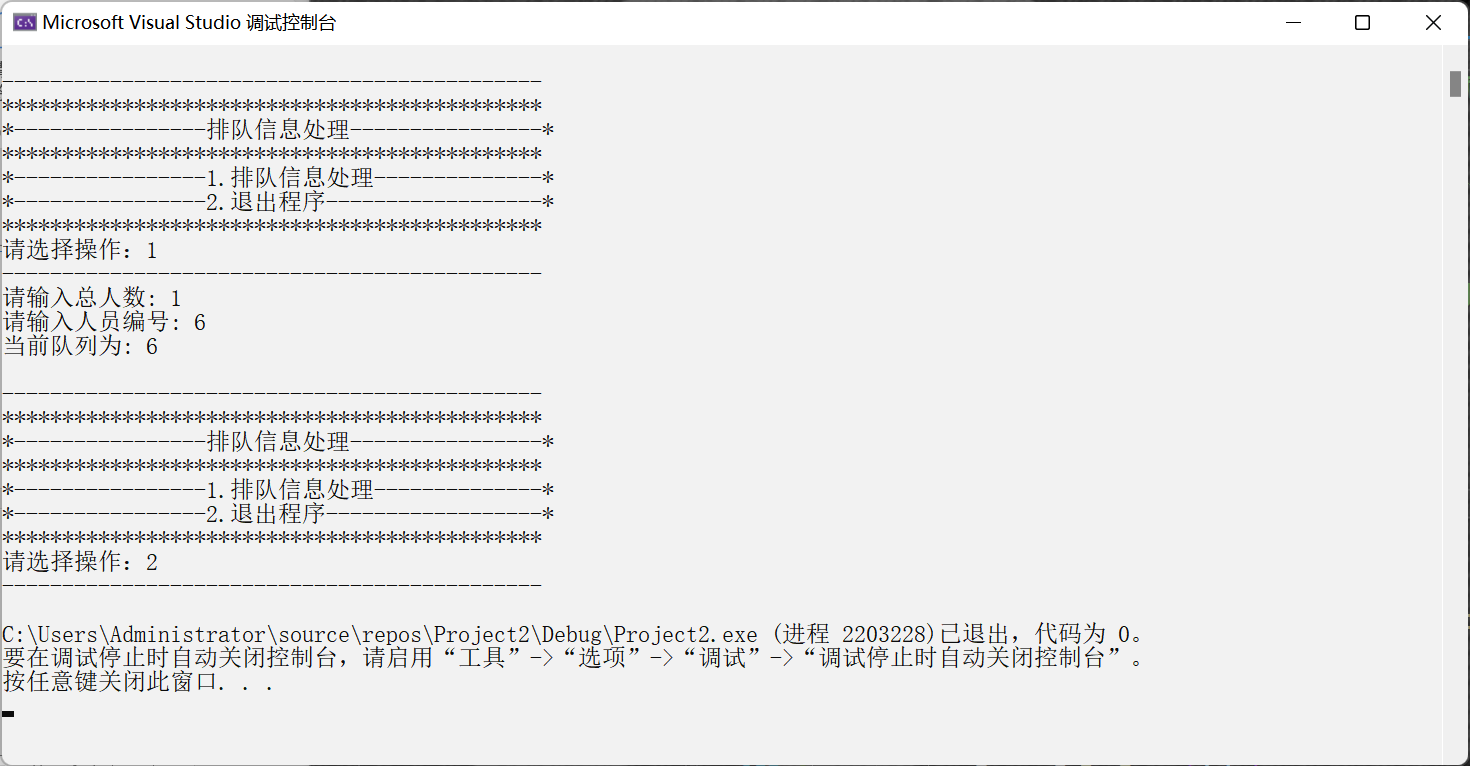


测试三：

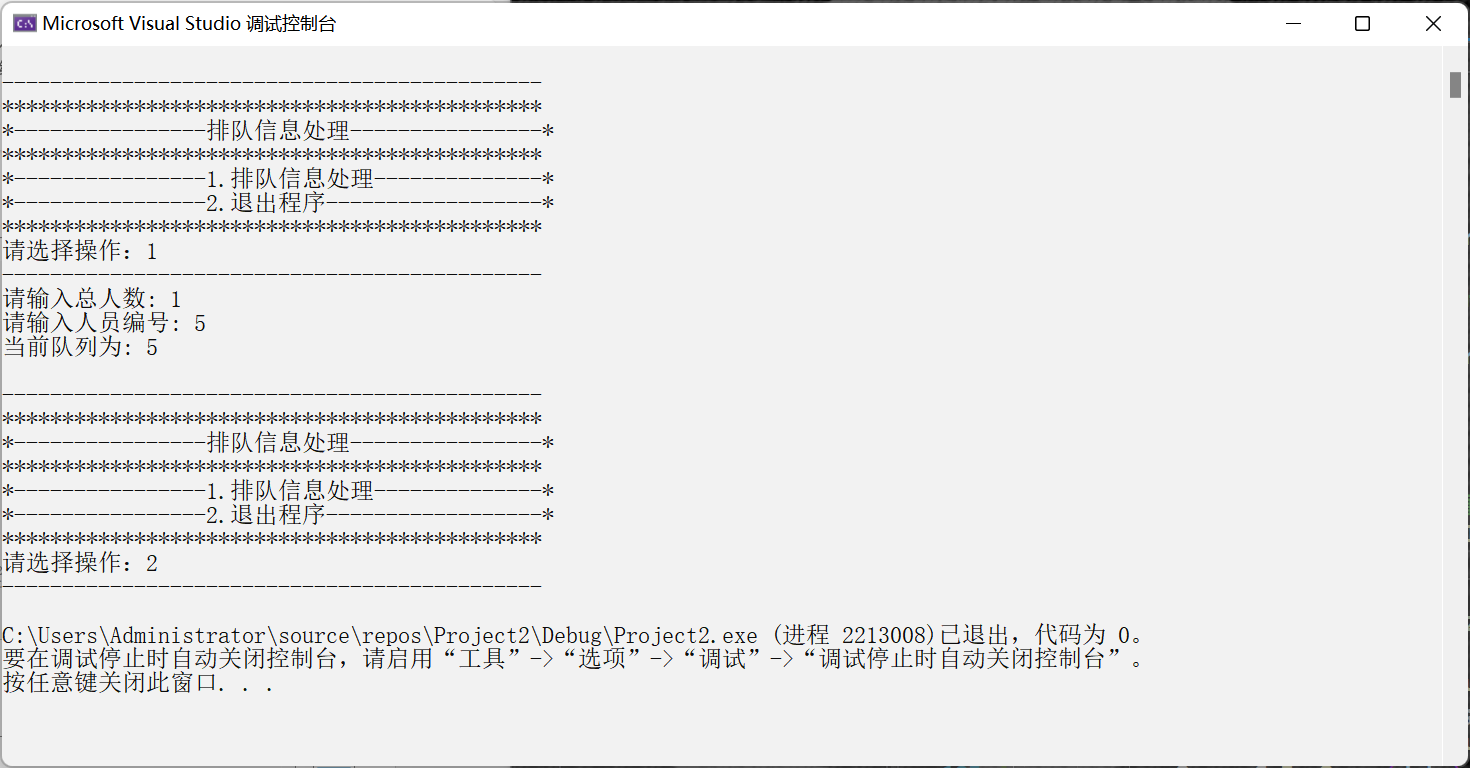


### 银行业务功能边界测试

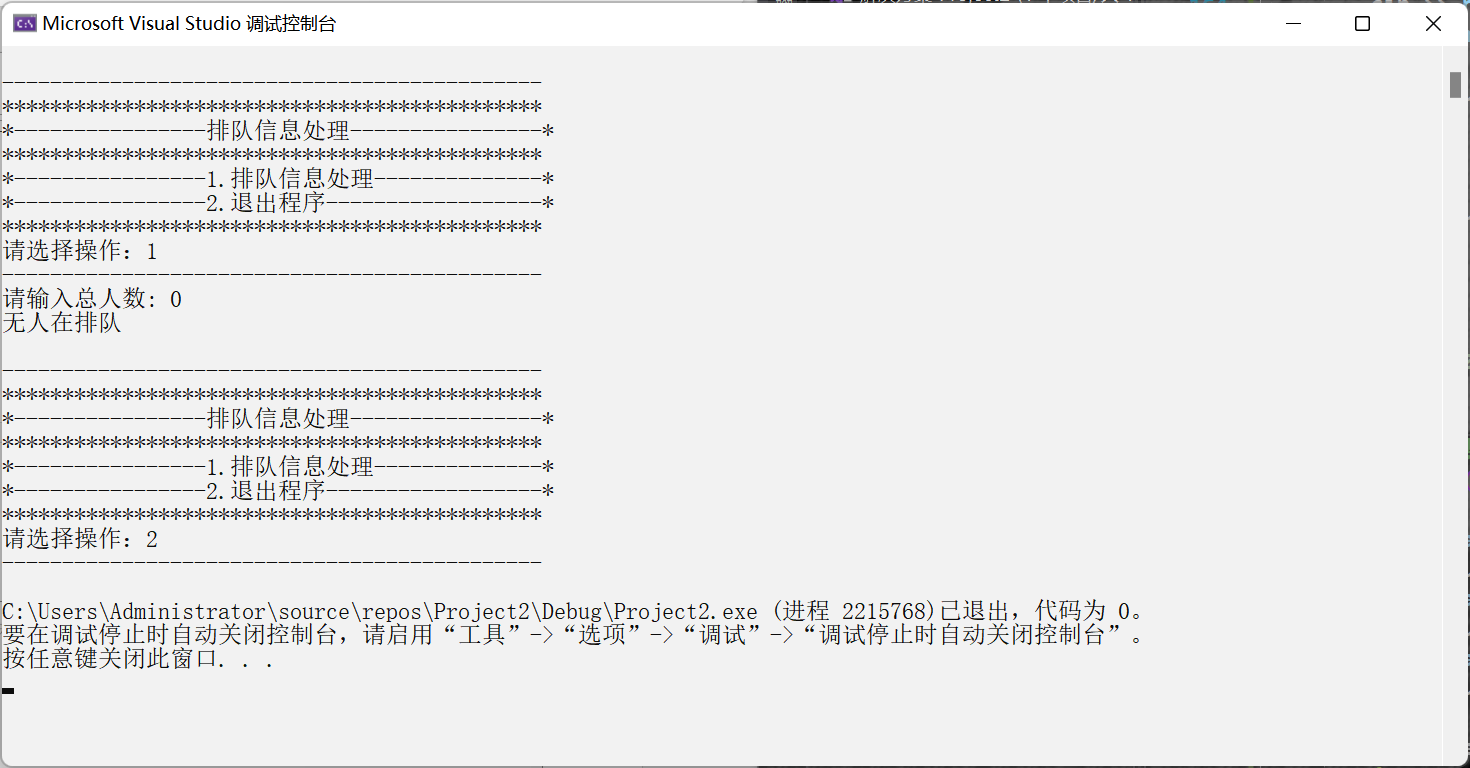
测试一（只有一个队列有元素）：



测试二：



测试三（两队列均无元素）：

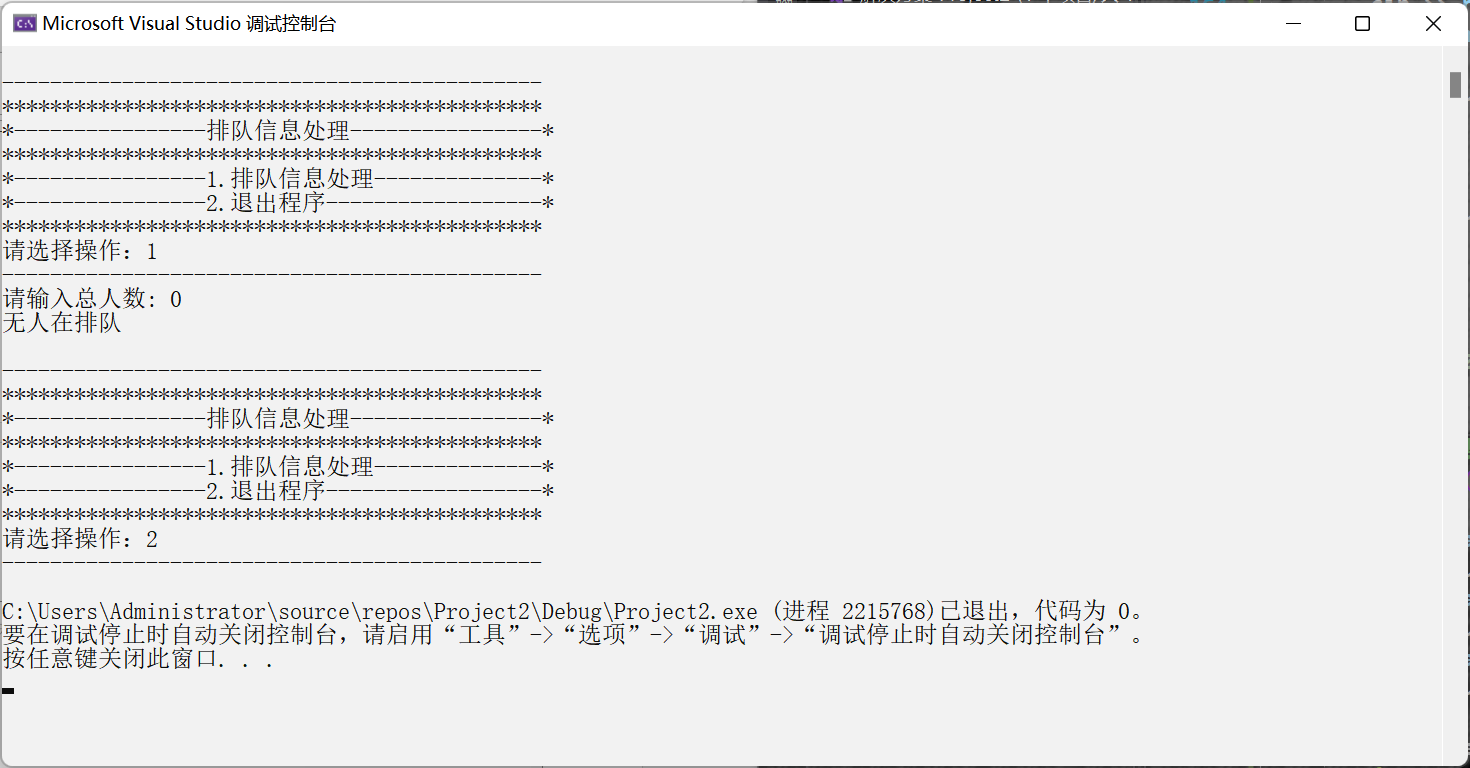


# 设计小结

## 可提升的点

在这次银行业务系统的编程中我掌握了队列的基本类实现，用模板类先进行声明再在子类中进行具体实现，在这次的银行业务设计中，我完成了基本建要求，也在一些地方做出了改进。

改进一、我进行了输入0时的情况的边界考虑



改进二、进行了无限制排队人数，采用链式栈，具有通用性。

## 团队总结

### 课设方面

在链队列中，节点之间通过指针相互连接，连接顺序容易出现问题导致出现指针错误。这是一个常见的编程错误，往往是由于对链队列的数据结构不够理解而导致的。为了避免这种错误，我们需要深入理解数据结构，尤其是涉及到指针连接的部分。我们需要掌握链队列的基本操作，如创建、插入、删除、遍历等，并且注意在每次操作后保持链队列的完整性和正确性。我们还需要注意链队列的边界条件，如空队列、满队列、只有一个元素的队列等，并且在编程时进行合理的判断和处理。只有这样，我们才能避免大部分指针错误的产生；在编写银行排队系统时，需要考虑系统的复杂度，以选择相对合适的算法。系统的复杂度包括时间复杂度和空间复杂度，分别反映了算法执行所需的时间和空间资源。一般来说，我们希望选择时间复杂度和空间复杂度都较低的算法，以提高系统的性能和响应速度。但是，在实际情况中，时间复杂度和空间复杂度往往存在着一定的权衡和折中关系。我们需要根据不同场景中的需求和限制，进行合理的选择和优化。使用合适的算法和数据结构，可以更好地提高系统的性能和响应速度。链队列是一种非常实用的数据结构，在银行排队系统中可以发挥其优势。链队列可以快速插入和删除队列中的元素，可以灵活应对不同场景中的需求；同时，链队列可以在不限定队列长度的情况下使用，具有动态增加队列长度的能力，能够很好地满足现实场景中用户数量不确定、业务时间不固定等特点。

### 社会方面

我们进行编写了一个银行排队系统，使用了链队列的数据结构，实现了先来先服务的排队规则。用户通过操作菜单可以添加自己的信息到队列中，等待柜员取出进行业务办理。该银行排队系统是一个简单的应用场景，也可以应用到其他需要排队等候的场景中，比如医院挂号、机场候机、餐厅取号等。链队列可以快速插入和删除队列中的元素，可以灵活应对不同场景中的需求，是一个非常实用的数据结构。同时，链队列可以在不限定队列长度的情况下使用，具有动态增加队列长度的能力，能够很好地满足现实场景的需求。通过这个银行排队系统，我们可以体会到链队列的优点和特性，也可以学习到如何设计和实现一个简单而有效的排队系统。这个银行排队系统不仅可以提高银行业务的效率和质量，也可以提高用户的满意度和信任度。这是一个有意义的课设项目，也是一个有趣的编程挑战。

### 个人方面

通过本次课设共同完成任务和合作达成目标，小组成员不仅可以加深对团队协作和集体利益的认识，增强合作意识，而且可以有效分工合作，避免单打独斗的情况发生，从而提升整个小组的工作效率和工作质量。同时每个小组成员还可以学习新技能、拓展视野、提高专业能力，从而提高自身的竞争力和可持续发展能力。这样既有利于小组的整体发展，也有利于个人的成长和进步。通过这样的课设活动，小组成员可以更好地适应社会的需求和变化，培养出积极主动、团结协作、创新求进的精神和品质。在这个过程中，小组成员之间也可以建立深厚的友谊和信任，相互支持和鼓励，共同面对困难和挑战，共同分享成功和喜悦。这是一次难忘的经历，也是一次宝贵的学习。

# 主要参考文献和附录

## 参考文献

1. 叶晴晴,陈钰.带有工作故障的M/M/1重试排队系统的性能分析[J].河南师范大学学报(自然科学版),2023,51(03):82-89.DOI:10.16366/j.cnki.1000-2367.2023.03.010.
2. 周楠,李福川,宣萱.基于双排队系统的软件可靠性增长模型[J].计算机工程与设计,2023,44(02):447-456.DOI:10.16208/j.issn1000-7024.2023.02.018.
3. 柯洪昌,王慧,孔德刚,张雪峰.基于Android的在线手机银行排队系统设计[J].现代信息科技,2022,6(24):27-30.DOI:10.19850/j.cnki.2096-4706.2022.24.007.
4. 林宜锋.基于ENSP的中学网络规划研究与设计[J].中国新通信,2022,24(18):85-89.
5. 石英男,樊瑶,张泞君.基于红帽Linux的银行排队叫号系统[J].软件导刊,2020,19(11):131-135.
6. 王鹏.高校基础网络规划与设计[J].信息与电脑(理论版),2020,32(21):210-212.
7. 赵海宇.医院网络的规划与设计[J].福建电脑,2019,35(06):46-47.DOI:10.16707/j.cnki.fjpc.2019.06.012.
8. 安华萍.校园网学生宿舍无线网络规划与设计[J].电脑编程技巧与维护,2019(05):174-176.DOI:10.16184/j.cnki.comprg.2019.05.067.
9. 刘航.高职院校校园网络规划与设计的分析与探讨[J].信息与电脑(理论版),2018(18):157-158.
10. 何天聪. 银行排队系统分析及窗口服务改进研究[D].深圳大学,2018.
11. 王学芹.校园网络建设的规划与设计[J].高考,2017(15):7.
12. 卢敏颢. TS银行网点客户排队系统优化研究[D].河北工业大学,2016.
13. 金烈淼. 银行排队叫号管理系统的分析与设计[D].云南大学,2016.
14. 谢统薇. 基于多平台银行预约排队系统的研究与应用[D].郑州大学,2015.
15. 何迎盛,余建华.银行智能排队系统设计与改进[J].科技广场,2015(01):66-69.DOI:10.13838/j.cnki.kjgc.2015.01.013.
16. 卢子卿. 银行客户排队系统的研究[D].安徽农业大学,2014.

## 附录：源代码

#include<iostream>

using namespace std;

template<class T>

class Queue //队列模板类创建

{

public:

virtual void clear() = 0; //将队列清空

virtual int size()const = 0; //求队列长度

virtual void In\_Queue(T& x) = 0; //入队列

virtual T Out\_Queue() = 0; //出队列

virtual T Read\_Queue()const = 0; //读取队头元素

virtual bool Is\_empty()const = 0; //判断队列是否为空

};

template<class T>

class Link\_Queue :public Queue<T>

{

public:

Link\_Queue() { front = NULL; rear = NULL; } //构造函数将队头，队尾指针置空

~Link\_Queue() { clear(); } //析构函数清除队列

void clear(); //将队列清空

int size()const; //求队列长度

void In\_Queue(T& x); //入队列

T Out\_Queue(); //出队列

T Read\_Queue()const; //读取队头元素

bool Is\_empty()const; //判断队列是否为空

private:

struct Link\_node //队列结点，包含数据以及指针

{

T data;

Link\_node\* next;

};

Link\_node\* rear, \* front; //队尾，队头指针

};

template<class T>

void Link\_Queue<T>::clear() //清空队列

{

Link\_node\* p;

while (front != NULL) //当队列头指针不为空（即队列不为空时）进行头指针后移逐个删除队列元素

{

p = front;

front = front->next;

delete p;

}

rear = NULL; //将尾指针置空

}

template<class T>

int Link\_Queue<T>::size()const

{

Link\_node\* p = front;

int count = 0;

while (p != NULL) //当头指针不为空时依次后移。count++，进行计数

{

count++;

p = p->next;

}

return count;

}

template<class T>

void Link\_Queue<T>::In\_Queue(T& x)

{

Link\_node\* p = new Link\_node;

if (p == NULL)

{

cout << "存储分配错误！" << endl; //检查开辟新结点是否成功

exit(1);

}

if (rear == NULL) //如果此队列中无元素则进行插入第一个元素操作

{

p->data = x;

rear = p;

front = p;

rear->next = NULL;

}

else //如果队列中有元素则进行在队尾插入操作

{

p->data = x;

rear->next = p;

rear = p;

rear->next = NULL;

}

}

template<class T>

T Link\_Queue<T>::Out\_Queue()

{

Link\_node\* p = front;

T x;

while (front != NULL) //判断队列中是否有元素

{

if (p == NULL)

{

cout << "存储分配错误！" << endl; //判断建立结点是否成功

exit(1);

}

else //如果成功返回头指针数据值，并将头指针后移

{

p = front;

x = p->data;

front = front->next;

if (front == NULL) //原来只有一个元素出队后队列为空

{

rear == NULL; //修改队尾指针

}

delete p;

return x;

}

}

}

template<class T>

T Link\_Queue<T>::Read\_Queue()const

{

if (front != NULL) //如果队列中有元素，返回队头元素的值

{

return front->data;

}

else

{

cout << "队列中无元素，不可读取队列首元" << endl;

}

}

template<class T>

bool Link\_Queue<T>::Is\_empty()const

{

if (rear == NULL || front == NULL) //头指针或者尾指针为空则队列为空

{

return true;

}

else

{

return false;

}

}

class Compare\_Queue //比较队列类

{

private:

Link\_Queue<int>Queue\_A; //定义三个int型队列

Link\_Queue<int>Queue\_B;

Link\_Queue<int>Queue\_C;

public:

Compare\_Queue() {}; //构造函数

~Compare\_Queue() {}; //析构函数

void Handling(); //办理业务实现函数

};

void Compare\_Queue::Handling()

{

int sum, number = 0, data; //总的排队人数sum， 不同人的编号number，用于暂时存储数据

int A\_size, B\_size, C\_size; //记录A,B,C队列的长度

cout << "请输入总人数: " ;

cin >> sum;

if(sum != 0){

cout << "请输入人员编号: ";

cout << endl;

}

for (int i = 0; i < sum; i++) //循环输入总人数以及人员编号

{

cin >> number;

if ((number % 2) == 0) //人员编号为偶数进入B队列

{

Queue\_B.In\_Queue(number);

}

else //人员编号为奇数进入A队列

{

Queue\_A.In\_Queue(number);

}

}

while ((Queue\_A.size() >= 2) && (Queue\_B.Is\_empty() == false)) //当A队列还有至少两个元素，且B队列不为空时

{

data = Queue\_A.Out\_Queue(); //输出A队列中的两个元素，B队列中的一个元素

Queue\_C.In\_Queue(data);

data = Queue\_A.Out\_Queue();

Queue\_C.In\_Queue(data);

data = Queue\_B.Out\_Queue();

Queue\_C.In\_Queue(data);

}

if ((Queue\_A.size() < 2) && (Queue\_B.Is\_empty() == false)) //当A队列中元素少于2，B队列不为空时

{

B\_size = Queue\_B.size();

A\_size = Queue\_A.size();

for (int i = 0; i < A\_size; i++) //先将A队列元素全部输出，再将B队列元素全部输出

{

data = Queue\_A.Out\_Queue();

Queue\_C.In\_Queue(data);

}

for (int i = 0; i < B\_size; i++)

{

data = Queue\_B.Out\_Queue();

Queue\_C.In\_Queue(data);

}

}

if (Queue\_B.Is\_empty() == true) //如果B队列中元素为空

{

A\_size = Queue\_A.size();

for (int i = 0; i < A\_size; i++) //将A中元素全部放入C中

{

data = Queue\_A.Out\_Queue();

Queue\_C.In\_Queue(data);

}

}

if (Queue\_C.Is\_empty() == false) //如果C中有元素

{

C\_size = Queue\_C.size();

cout << "当前队列为: ";

for (int i = 0; i < C\_size; i++) //将C中元素全部输出

{

data = Queue\_C.Out\_Queue();

cout << data;

if (C\_size - 1 != i) //保证C输出元素最后一个后无括号

{

cout << " ";

}

}

}

else //C中无元素

{

cout << "无人在排队" << endl;

}

}

void MainMenu()

{

Compare\_Queue bank;

int menu;

do {

cout << endl;

cout << "---------------------------------------------" << endl;

cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*" << endl;

cout << "\*----------------排队信息处理----------------\*" << endl;

cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*" << endl;

cout << "\*----------------1.排队信息处理--------------\*" << endl;

cout << "\*----------------2.退出程序------------------\*" << endl;

cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*" << endl;

cout << "请选择操作：";

cin >> menu;

cout << "---------------------------------------------" << endl;

switch (menu)

{

case 1:

bank.Handling();

break;

case 2:

break;

default:

cout << "输入有误，请重新输入" << endl;

break;

}

} while (menu != 2);

}

int main()

{

MainMenu();

return 0;

}