项目说明文档

数据结构课程设计

——**银行业务**

作 者 姓 名： 陈翔飞

学 号： 1851756

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 3](#_Toc27523408)

[1.1 背景分析 3](#_Toc27523409)

[1.2 功能分析 3](#_Toc27523410)

[2 设计 3](#_Toc27523411)

[2.1 数据结构设计 3](#_Toc27523412)

[2.2 类结构设计 3](#_Toc27523413)

[2.3 成员与操作设计 3](#_Toc27523414)

[2.4 系统设计 6](#_Toc27523415)

[3 实现 7](#_Toc27523416)

[3.1 顾客分离功能的实现 7](#_Toc27523417)

[3.1.1 顾客分离流程图 7](#_Toc27523418)

[3.1.2 顾客分离核心代码 8](#_Toc27523419)

[3.2业务处理功能的实现 9](#_Toc27523420)

[3.2.1 业务处理核心代码 9](#_Toc27523421)

[3.2.2 业务处理截屏示例 10](#_Toc27523422)

[3.3 总体系统的实现 11](#_Toc27523423)

[3.3.1 总体系统流程图 11](#_Toc27523424)

[3.3.2 总体系统核心代码 11](#_Toc27523425)

[3.3.3 总体系统截屏示例 12](#_Toc27523426)

[4 测试 12](#_Toc27523427)

[4.1 功能测试 12](#_Toc27523428)

[4.1.1 A窗口人多 12](#_Toc27523429)

[4.1.2 B窗口人多 13](#_Toc27523430)

[4.1.3 只有B窗口有人 13](#_Toc27523431)

[4.1.4 只有A窗口有人 14](#_Toc27523432)

[4.1.5 最小N 14](#_Toc27523433)

# 1 分析

## 背景分析

项目的实际场景是银行处理业务。现在有A窗口和B窗口，A窗口和B窗口处理业务的速度不一 样，每当A处理完2个顾客时，B处理完1个顾客。

## 1.2 功能分析

在项目的背景下，需要实现的功能是给定到达银行的顾客序列，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.按照业务完成的顺序输出顾客序列。并且不考虑顾客先后到达的时间间隔，当A，B同时处理完顾客时，A窗口的顾客优先输出。

# 2 设计

## 2.1 数据结构设计

如上功能分析所述，项目需要存储顾客的处理顺序，排在前面的顾客先进行业务服务，满足先进先出原则，所以需要数据结构队列。 这里队列采用链式结构存储，所以还需要一个队列节点类。

## 2.2 类结构设计

为了保证设计的数据结构的泛用性，本项目选择将Node类和Queue类都设计为模板类， Node类储存实际的数据，Queue类储存队首和队尾，并提供入队、出队、返回队首等接口。为了存放最后的处理顺序序列，还需要一个动态数组，这里选用Vector。

## 2.3 成员与操作设计

**队列结点类（Node）**

**类定义：**

1. **template**<**typename** ElementType> **class** Node
2. {
3. **private**:
4. **friend** **class** Queue<ElementType>;
5. Node<ElementType>() : Pre(nullptr), Next(nullptr) {};
6. Node<ElementType>(**const** ElementType& e) :Element(e) {};
7. Node<ElementType>\* Next=nullptr, \*Pre=nullptr;
8. ElementType Element;
9. };

**私有成员：**

Node<ElementType>\* Next=nullptr; //队列中的前一个节点

Node<ElementType>\* Pre=nullptr;//队列中的后一个节点

ElementType Element;//队列元素

**私有操作：**

Node<ElementType>() : Pre(nullptr), Next(nullptr) {};

//默认构造函数

Node<ElementType>(const ElementType& e) :Element(e) {};

//拷贝构造函数

**队列类（Queue）**

**类定义：**

1. **template**<**typename** ElementType> **class** Queue
2. {
3. **public**:
4. Queue<ElementType>();
5. ~Queue<ElementType>();
6. **int** GetSize();
7. **bool** Empty();
8. **void** EnQueue(ElementType E);
9. **void** DeQueue();
10. ElementType Front();
11. **void** Clear();
12. **private**:
13. **int** Size=0;
14. Node<ElementType>\* Head, \*Tail;
15. };

**私有成员：**

int Size = 0;//队列的大小;

Node<ElementType>\* Head;//队首节点

Node<ElementType>\* Tail;//队尾节点

**公有操作：**

Queue<ElementType>();

//默认构造函数

~Queue<ElementType>();

//析构函数，通过调用Clear（）来回收内存

int GetSize();

//返回队列大小

bool Empty();

//判断队列是否为空

void EnQueue(ElementType E);

//入队操作

void DeQueue();

//出队操作

ElementType Front();

//返回队首

void Clear();

//清空队列，回收内存

**向量类（Vector）**

**类定义：**

1. **template**<**typename** ElementType> **class** Vector
2. {
3. **public**:
4. ~Vector<ElementType>();
5. Vector<ElementType>() = **default**;
6. Vector<ElementType>(**const** Vector<ElementType> & v);
7. Vector<ElementType>& operator = (**const** Vector<ElementType>&v);
8. **void** PushBack(**const** ElementType& t);
9. **void** PopBack();
10. **void** Clear();
11. **int** GetSize() **const**;
12. **void** ReSize(**int** NewSize);
13. ElementType& operator[](**int** Index) **const**;
14. **bool** Empty()**const**;
15. **const** ElementType& Back() **const**;
16. **private**:
17. **void** Extend();
18. **int** Size = 0;
19. **int** Capacity = 0;
20. ElementType\* Array = nullptr;
21. };

**私有成员：**

int Size;//Vector中实际储存的元素数量

int Capacity;//Vector已经申请的空间

ElementType\* Array;//储存的数据的起始地址

**私有操作：**

void Extend();

//扩容函数，当容量不足时调用

**公有操作：**

~Vector<ElementType>();

//析构函数，通过调用Clear()来释放内存

Vector<ElementType>() = default;

//默认构造函数

Vector<ElementType>(const Vector<ElementType> & v);

//拷贝构造函数

Vector<ElementType>& operator = (const Vector<ElementType>&v);

//重载=运算符，使该类支持赋值运算

void PushBack(const ElementType& t);

//向Vector末尾添加一个元素

void PopBack();

//删除末尾的元素

void Clear();

//清空Vector，释放内存

int GetSize() const;

//返回储存元素的数量

void ReSize(int NewSize);

//重设Vector的大小

ElementType& operator[](int Index) const;

//重载[]运算符，使Vector可以像数组一样使用

bool Empty()const;

//判断Vector是否为空

const ElementType& Back() const;

//返回末尾的元素

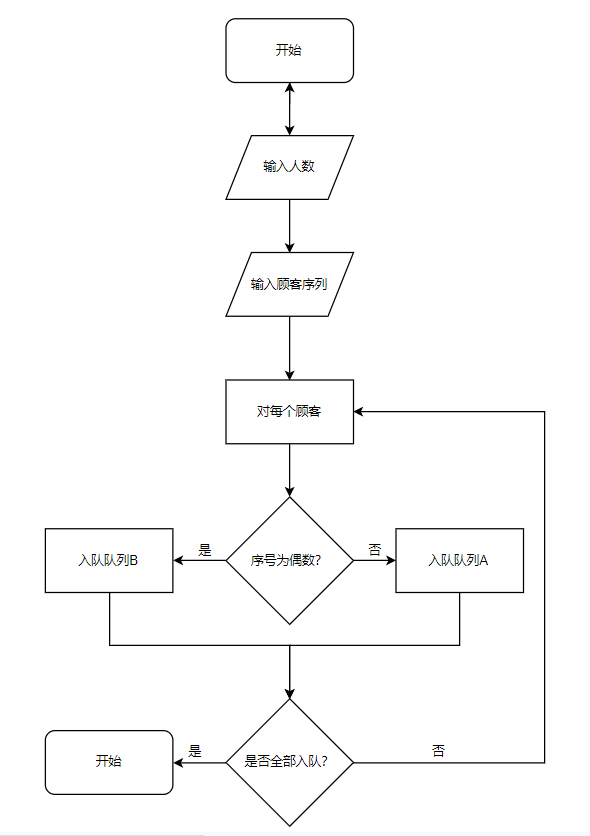
## 2.4 系统设计

系统首先询问有多少顾客，然后让用户输入顾客到达序列，在求出顾客处理顺序之后将其输出。

# 3 实现

## 3.1 顾客分离功能的实现

### 3.1.1 顾客分离流程图



### 3.1.2 顾客分离核心代码

**用户界面接口：**

1. **void** CustomerSeparation(Queue<**int**>& A, Queue<**int**>& B, **int** &CustomerNum)
2. {
3. cout << "请输入顾客的总数:" << endl;
4. **int** CustomerNum = 0;
5. cin >> CustomerNum;
6. **while** (CustomerNum <= 0)
7. {
8. cout << "请输入一个正整数！" << endl;
9. cin >> CustomerNum;
10. }
11. **int** i;
13. **for** (i = 0; i < CustomerNum; ++i)
14. {
15. **int** CurNum;
16. cin >> CurNum;
17. **if** (CurNum % 2 == 0)
18. B.EnQueue(CurNum);
19. **else**
20. A.EnQueue(CurNum);
21. }
22. }

**队列入队内部实现：**

1. **template**<**typename** ElementType>
2. **inline** **void** Queue<ElementType>::EnQueue(ElementType E)
3. {
4. Node<ElementType>\* Temp = **new** Node<ElementType>(E);
5. Temp->Pre = Tail->Pre;
6. Temp->Next = Tail;
7. Tail->Pre = Temp;
8. Temp->Pre->Next = Temp;
9. Size++;
10. }

## 3.2业务处理功能的实现

### 3.2.1 业务处理核心代码

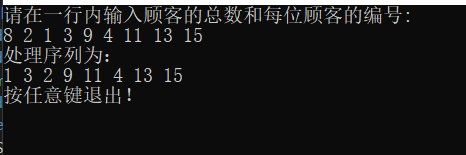
**用户界面接口：**

1. **void** ProcessBusiness(Queue<**int**>& A, Queue<**int**>& B, Vector<**int**> &OutSequence)
2. {
3. **int** B\_BusyTime = 1; //窗口B处理当前业务剩余时间
4. **while** (!A.Empty() && !B.Empty())//当两个队列都不为空时
5. {
6. OutSequence.PushBack(A.Front());//A优先进入输出序列
7. A.DeQueue();
8. **if** (B\_BusyTime == 0)//如果B已经处理完当前业务
9. {
10. OutSequence.PushBack(B.Front());//队首进入输出序列
11. B.DeQueue();
12. B\_BusyTime = 1;//业务剩余时间重设为1
13. }
14. **else** --B\_BusyTime;//如果未处理完，则将剩余时间-1
15. }
16. **while** (!A.Empty())//如果A不为空，则把A剩余元素全放入输出队列
17. {
18. OutSequence.PushBack(A.Front());
19. A.DeQueue();
20. }
21. **while** (!B.Empty())//如果B不为空，则把B剩余元素全放入输出队列
22. {
23. OutSequence.PushBack(B.Front());
24. B.DeQueue();
25. }
26. }

**队列出队内部实现：**

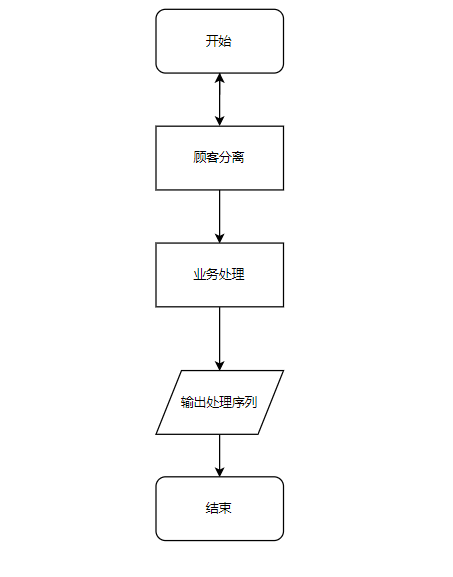
1. **template**<**typename** ElementType>
2. **inline** **void** Queue<ElementType>::DeQueue()
3. {
4. **if** (Empty()) **return**;
5. Node<ElementType>\* Temp = Head->Next;
6. Head->Next = Temp->Next;
7. Head->Next->Pre = Head;
8. **delete** Temp;
9. Size--;
10. }

### 3.2.2 业务处理截屏示例



## 3.3 总体系统的实现

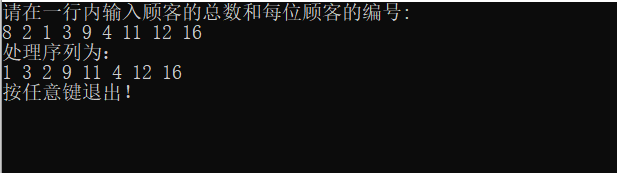
### 3.3.1 总体系统流程图



### 3.3.2 总体系统核心代码

1. **int** main(**void**)
2. {
3. **int** i;
4. **int** CustomerNum = 0;
5. Vector<**int**> OutSequence;
6. Queue<**int**> A, B;
7. CustomerSeparation(A, B,CustomerNum);
8. ProcessBusiness(A, B, OutSequence);
9. cout << "处理序列为：" << endl;
10. **for** (i = 0; i < CustomerNum-1; ++i)
11. {
12. cout << OutSequence[i] << " ";
13. }
14. cout << OutSequence[i];
15. **while** (getchar() != '\n') **continue**;//清除行末换行符
16. cout << endl << "按任意键退出！" << endl;
17. getchar();//等待退出
19. **return** 0;
20. }

### 3.3.3 总体系统截屏示例



# 4 测试

## 4.1 功能测试

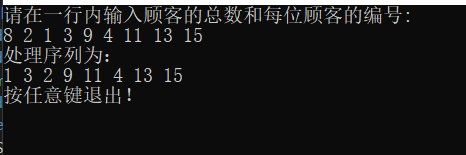
### 4.1.1 A窗口人多

**测试用例**：8 2 1 3 9 4 11 13 15

**预期结果**：

1 3 2 9 11 4 13 15

**实验结果：**



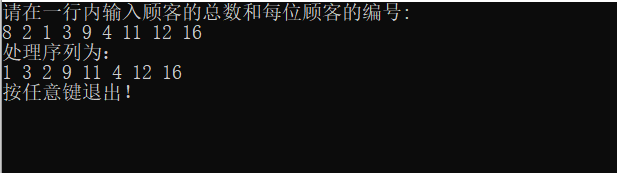
### 4.1.2 B窗口人多

**测试用例：**8 2 1 3 9 4 11 12 16

**预期结果：**

1 3 2 9 11 4 12 16

**实验结果：**



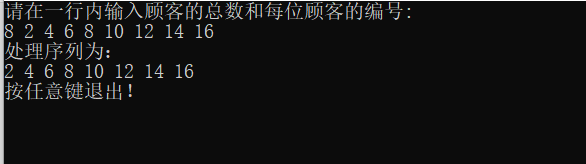
### 4.1.3 只有B窗口有人

**测试用例：**8 2 4 6 8 10 12 14 16

**预期结果：**

2 4 6 8 10 12 14 16

**实验结果：**



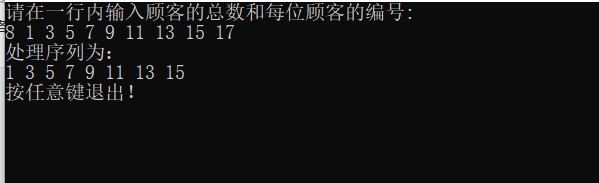
### 4.1.4 只有A窗口有人

**测试用例：**8 1 3 5 7 9 11 13 15 17

**预期结果：**

1 3 5 7 9 11 13 15

**实验结果：**



### 4.1.5 最小N

**测试用例：**1 6

**预期结果：**

6

**实验结果：**

