

# 项目说明文档

## 数据结构课程设计

### ——银行业务

作者姓名: 翟晨昊

学 号: 1952216

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.....	- 4 -
1.1	背景分析 .....	- 4 -
1.2	功能分析 .....	- 4 -
2	设计.....	- 5 -
2.1	数据结构设计.....	- 5 -
2.2	类结构设计 .....	- 5 -
2.3	成员与操作设计 .....	- 5 -
2.4	系统设计 .....	- 9 -
3	实现.....	- 10 -
3.1	顾客分类功能的实现 .....	- 10 -
3.1.1	顾客分类功能流程图 .....	- 10 -
3.1.2	顾客分类功能核心代码.....	- 11 -
3.1.3	顾客分类功能描述 .....	- 13 -
3.2	办理业务流程功能的实现 .....	- 14 -
3.2.1	办理业务流程功能流程图.....	- 14 -
3.2.2	办理业务流程功能核心代码.....	- 15 -
3.2.3	办理业务流程功能描述.....	- 17 -
3.3	总体功能的实现.....	- 18 -
3.3.1	总体功能流程图 .....	- 18 -
3.3.2	总体功能核心代码 .....	- 19 -
3.3.3	总体功能截屏示例 .....	- 20 -
4	测试.....	- 21 -
4.1	功能测试 .....	- 21 -
4.1.1	正常功能测试, A 窗口人多 .....	- 21 -
4.1.2	正常功能测试, B 窗口人多 .....	- 21 -
4.1.3	正常功能测试, AB 窗口人数一样多 .....	- 21 -
4.1.4	顾客人数最小 .....	- 22 -
4.1.5	顾客全在 A 窗口 .....	- 22 -
4.1.6	顾客全在 B 窗口 .....	- 22 -
4.2	边界测试 .....	- 24 -
4.2.1	顾客数量为零 .....	- 24 -
4.3	出错测试 .....	- 25 -
4.3.1	顾客编号个数比顾客总数多 .....	- 25 -
4.3.2	输入顾客个数错误 .....	- 25 -

4.3.3 顾客编号输入不合法 .....	- 25 -
4.3.4 序列中存在两个编号相同的客户 .....	- 26 -

# 1 分析

## 1.1 背景分析

在日常生活中，我们经常会遇到排队的情况：取票，吃饭，结账等等。在银行处理业务时，也经常会遇到排队的情况。一般来讲，对于不同的窗口，处理速度也不一样。本次项目假设某银行有 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 个顾客时，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。假定不考虑顾客前后到达的时间间隔，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。项目需要设计一个程序，来将顾客按照处理的先后顺序进行排列。

## 1.2 功能分析

作为一个银行业务办理程序，首先我们要能够储存顾客的序列，并将他们按照编号的奇数和偶数进行分类。随后要模拟银行办理业务的流程，当 A 窗口每处理完 2 个顾客时，B 窗口处理完 1 个顾客，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。最后，要把输出的顾客序列进行展示，并等待用户决定是否要进行新一轮的银行业务办理流程。

综上所述，本程序中需要有输入，排列顾客，输出的功能。

## 2 设计

### 2.1 数据结构设计

如上功能分析所述，该银行业务办理程序会有大量插入序列，从序列中提出的操作，且顾客的办理顺序满足先进先出，因此考虑使用链式队列的数据结构。将顾客插入序列就插入在队列的末尾，每次提出顾客就从队首提出。

### 2.2 类结构设计

首先，本程序需要进行顾客入队与出队等操作，采用的是链式队列的数据结构。链式队列一般包括两个抽象数据类型——链式队列结点类（LinkedList 类）与链式队列类（LinkedList 类），LinkedList 类来储存每位顾客的编号，LinkedList 类则将这些顾客按输入顺序排列，并提供入队，出队等操作。随后，程序还需要保存最后生成的顾客办理顺序，并将他们输出。由于每次的顾客数量未知，因此采用 Vector 向量类来保存最后的顾客办理顺序。为了使向量类与链式队列类更具有泛用性，本程序将 Vector 类与 LinkedList 类都设计为了模板类。

### 2.3 成员与操作设计

向量类（Vector）：

```
template <typename ElementType> class Vector {
public:
    ~Vector();
    Vector();
    ElementType& operator[](const int x);
    const ElementType& operator[](const int x)const;
    Vector<ElementType>(const Vector<ElementType>& rhs);
    Vector<ElementType>& operator=(const Vector<ElementType>& rhs);
    bool isFull();
    bool isEmpty();
    void pushBack(const ElementType& temp);
    void popBack();
    void clear();
    int getSize();
    void reSize(int newSize);
private:
    void extendSize();
    int size;
    int maxSize;
    ElementType* myVector;
};
```

**私有成员:**

```
int size;//Vector 中已经存储的元素数量
int maxSize;//Vector 中已经申请的空间大小, 元素数量如果超过要再次申请
ElementType* myVector;//存储的元素序列的起始地址
```

**私有操作:**

```
void extendSize();
//在 Vector 申请的空间已经被占满时再次申请空间, 每次扩充至 maxsize*2+1
是因为刚开始的 maxsize 为 0
```

**公有操作:**

```
Vector();
//构造函数, 初始化指针并将 size 与 maxSize 置为 0
```

```
~Vector();
//析构函数, 调用 clear() 函数来删除元素, 释放内存
```

```
ElementType& operator[](const int x);
//重载[]运算符, 返回 ElementType&类型
```

```
const ElementType& operator[](const int x)const;
//重载[]运算符, 返回 const ElementType&类型
```

```
Vector<ElementType>(const Vector<ElementType>& rhs);
//复制构造函数, 将一个 Vector 复制给另一个 Vector
```

```
Vector<ElementType>& operator=(const Vector<ElementType>& rhs);
//重载=运算符, 可以将一个 Vector 赋给另一个 Vector
```

```
bool isFull();
//判断是否 Vector 中申请的内存已经被占满
```

```
bool isEmpty();
//判断 Vector 是否为空
```

```
void pushBack(const ElementType& temp);
//在 Vector 末尾添加一个元素
```

```
void popBack();
//删除 Vector 最末尾的元素
```

```
void clear();
//删除 Vector 中的所有元素并释放内存
```

```
int getSize();  
//返回 Vector 已经存储的元素数量  
  
void reSize(int newSize);  
//重设 Vector 的大小，若比之前小，则会抛弃多余的元素
```

**链式队列结点类 (LinkedListNode):**

```
template<typename ElementType>  
class LinkedListNode {  
public:  
    friend class LinkedListQueue<ElementType>;  
    LinkedListNode() = default;  
    LinkedListNode(const ElementType& inputNum) : number(inputNum) {};  
private:  
    LinkedListNode<ElementType>* next = nullptr;  
    ElementType number = 0;  
};
```

**私有成员:**

LinkedListNode<ElementType>\* next;//指针域，指向该结点的后一个结点  
ElementType number;//结点中顾客的编号

**公有操作:**

```
friend class LinkedListQueue<ElementType>;  
//将 LinkedListQueue 类声明为友元
```

```
LinkedListNode() = default;  
//默认构造函数
```

```
LinkedListNode(const ElementType& inputNum);  
//含参构造函数
```

**链式队列类 (LinkedListQueue):**

```
template<typename ElementType>  
class LinkedListQueue {  
public:  
    LinkedListQueue();  
    ~LinkedListQueue();  
    void makeEmpty();  
    int getSize();  
    bool isEmpty();  
    void enqueue(const ElementType& inputData);
```

```

    bool deQueue();
    ElementType getFront();
    bool checkNumber(ElementType curNum);
private:
    int size;
    ListNode<ElementType>* front;
    ListNode<ElementType>* rear;
};

```

**私有成员:**

```

int size;//队列中元素个数
ListNode<ElementType>* front;//指针域，指向队首结点
ListNode<ElementType>* rear;//指针域，指向队尾结点

```

**公有操作:**

```

LinkedQueue();
//默认构造函数，开辟一个附加头结点并将队列结点个数设为 0

~LinkedQueue();
//析构函数，通过调用 makeEmpty() 将队列中的结点删除，实现对内存的回收

void makeEmpty();
//删除队列中的所有元素并释放内存

int getSize();
//返回队列中已经存储的元素数量

bool isEmpty();
//判断队列是否为空

void enqueue(const ElementType& inputData);
//入队

bool deQueue();
//出队

ElementType getFront();
//获取队首结点的顾客编号

bool checkNumber(ElementType curNum);
//检查队列中是否有顾客编号相同

```



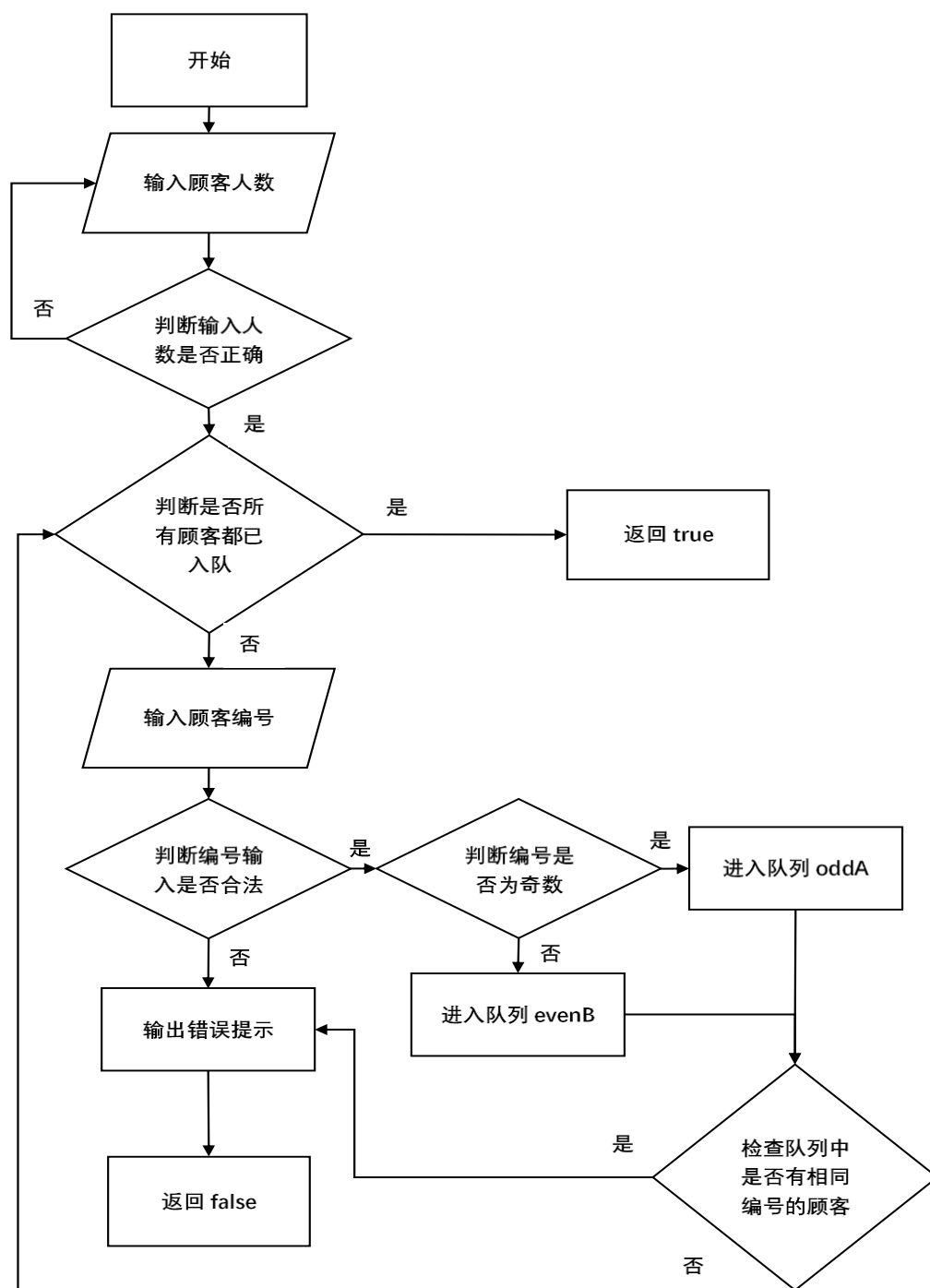
## 2.4 系统设计

程序在使用时，首先输入顾客总数与每一位顾客的编号，按照编号的奇偶性将顾客分到两个队列中，按照项目要求进行办理业务处理后，展示生成的顾客序列。在模拟一次银行办理业务后，用户可以选择重新进行一轮新的银行业务办理流程，或者退出系统。

### 3 实现

#### 3.1 顾客分类功能的实现

##### 3.1.1 顾客分类功能流程图



## 3.1.2 顾客分类功能核心代码

```

void init(int& customerNum)
{
    cout << "请在一行内输入顾客的总数和每位顾客的编号(录入等于总数后将不再录入):" << endl;
    cin >> customerNum;
    while (customerNum <= 0 || customerNum > 1000)
    {
        cout << "输入人数数量不正确或超出上限!" << endl;
        cout << "请重新输入: ";
        cin.clear();
        cin.ignore(numeric_limits<streamsize>::max(), '\n');
        cin >> customerNum;
    }
}

bool customerClassification(LinkedQueue<int>& oddA, LinkedQueue<int>& evenB, int customerNum)
{
    int curNum = 0;
    for (int i = 0; i < customerNum; i++)
    {
        cin >> curNum;
        if (curNum <= 0 || cin.fail())
        {
            cin.clear();
            cin.ignore(numeric_limits<streamsize>::max(), '\n');
            cout << "输入不合法!!!";
            return false;
        }
        if (curNum % 2 == 1)
        {
            if (!oddA.checkNumber(curNum))
            {
                cin.clear();
                cin.ignore(numeric_limits<streamsize>::max(), '\n');
                cout << "序列中存在两个编号相同的客户!";
                return false;
            }
            oddA.enqueue(curNum);
        }
        else if (curNum % 2 == 0)
        {

```

```

        if (!evenB.checkNumber(curNum))
        {
            cin.clear();
            cin.ignore(numeric_limits<streamsize>::max(), '\n');
            cout << "序列中存在两个编号相同的客户!";
            return false;
        }
        evenB.enqueue(curNum);
    }
}
cin.clear();
cin.ignore(numeric_limits<streamsize>::max(), '\n');
return true;
}

```

LinkedList 类中:

```

template<typename ElementType>
bool LinkedList<ElementType>::checkNumber(ElementType curNum)
{
    ListNode<int>* temp = front;
    for (int i = 0; i < size; i++)
    {
        temp = temp->next;
        if (temp->number == curNum)
        {
            return false;
        }
    }
    return true;
}

template<typename ElementType>
void LinkedList<ElementType>::enqueue(const ElementType& inputData)
{
    ListNode<ElementType>* temp = new ListNode<ElementType>(inputData);
    rear->next = temp;
    rear = rear->next;
    size++;
}

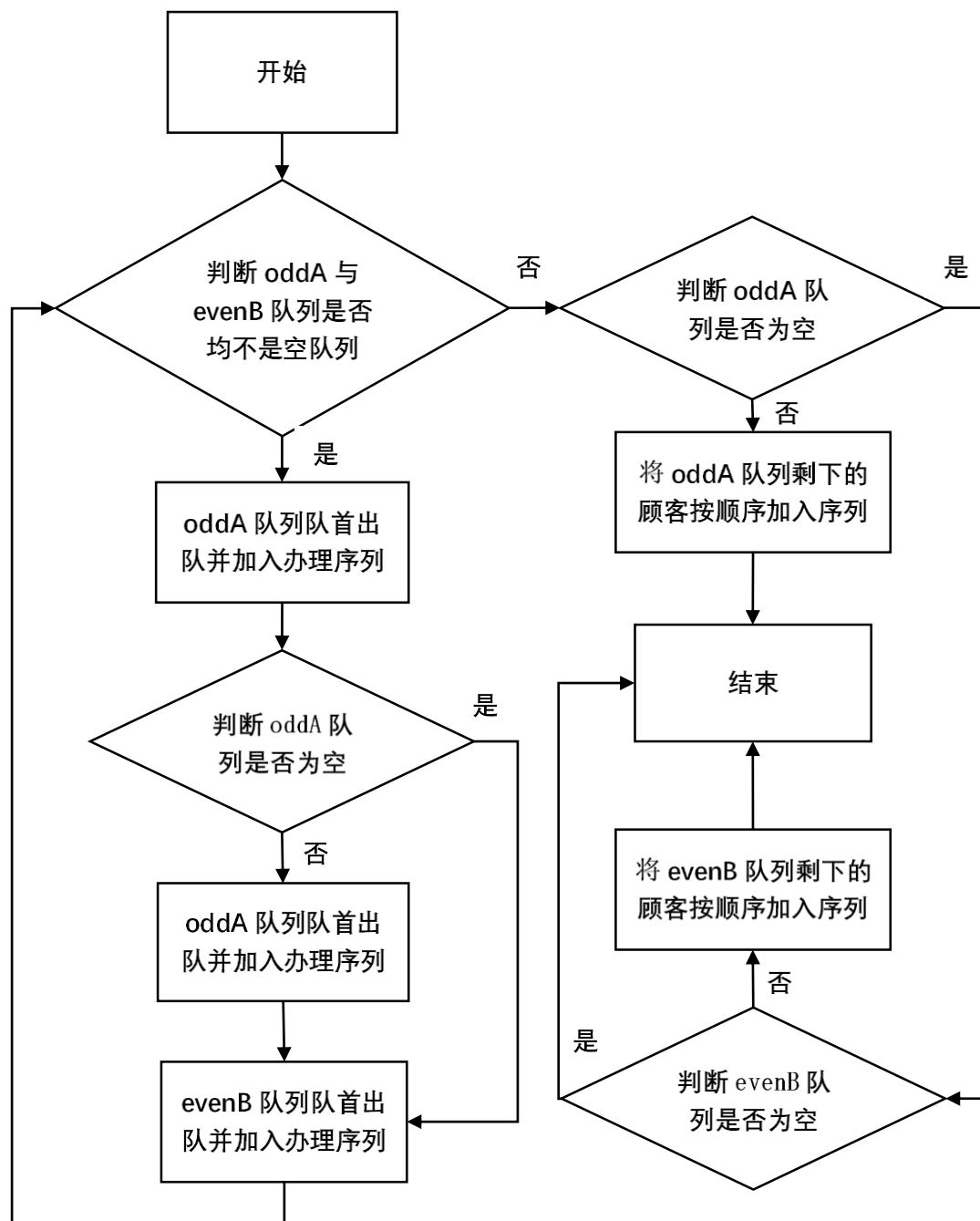
```

### 3.1.3 顾客分类功能描述

刚开始要在一行内输入顾客的总数和每位顾客的编号，如果顾客编号个数比输入的顾客总数要多，那么录入等于总数后将不再录入。随后按顺序每次判断一位顾客的编号，如果是奇数就进入 oddA 队列，如果是偶数就进入 evenB 队列。同时在进入队列前还要检查队列中是否已经有该编号的顾客存在。

## 3.2 办理业务流程功能的实现

### 3.2.1 办理业务流程功能流程图



## 3.2.2 办理业务流程功能核心代码

```
void businessProcessing(LinkedQueue<int>& oddA, LinkedQueue<int>& evenB
, Vector<int>&output)
{
    while (!oddA.isEmpty() && !evenB.isEmpty())
    {
        output.pushBack(oddA.getFront());
        if (!oddA.dequeue())
        {
            break;
        }
        if (!oddA.isEmpty())
        {
            output.pushBack(oddA.getFront());
            if (!oddA.dequeue())
            {
                break;
            }
        }
        output.pushBack(evenB.getFront());
        if (!evenB.dequeue())
        {
            break;
        }
    }
    while (!oddA.isEmpty())
    {
        output.pushBack(oddA.getFront());
        if (!oddA.dequeue())
        {
            break;
        }
    }
    while (!evenB.isEmpty())
    {
        output.pushBack(evenB.getFront());
        if (!evenB.dequeue())
        {
            break;
        }
    }
}
```

LinkedList 类中:

```
template<typename ElementType>
bool LinkedList<ElementType>::deQueue()
{
    if (isEmpty())
    {
        return false;
    }
    ListNode<ElementType>* temp = front->next;
    front->next = temp->next;
    delete temp;
    temp = nullptr;
    size--;
    if (size == 0)
    {
        rear = front;
    }
    return true;
}
```

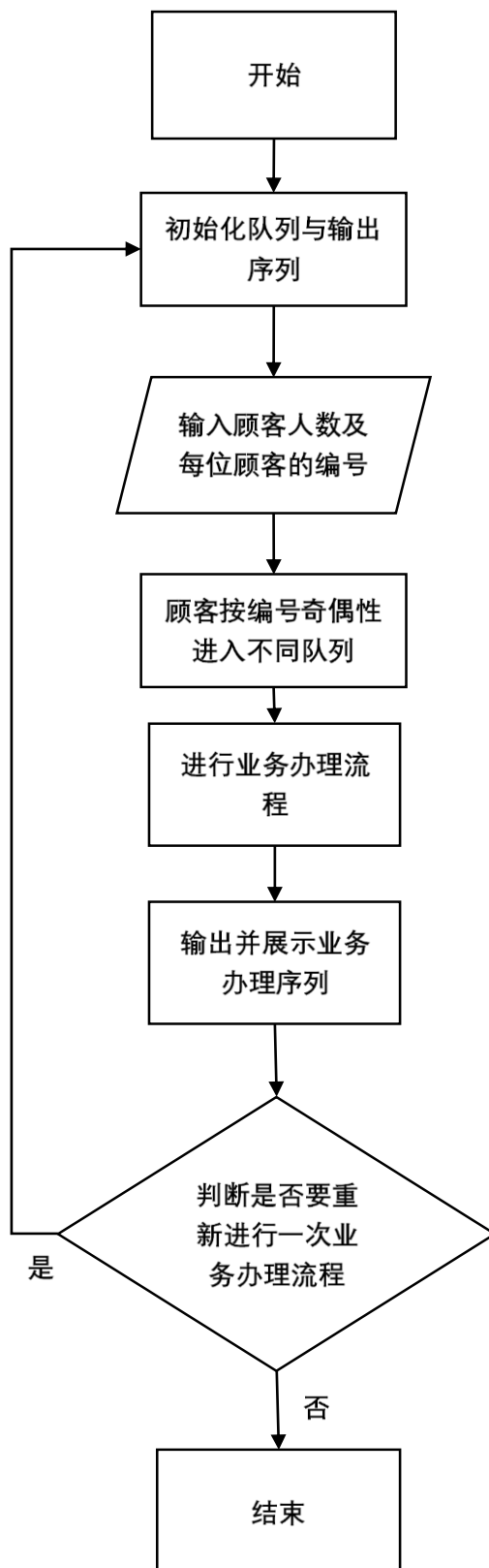


### 3.2.3 办理业务流程功能描述

项目要求 A 窗口每处理完 2 个顾客是，B 窗口处理完 1 个顾客。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。因此一次循环中先让 oddA 队首出队，此时如果 oddA 队列不为空，则要再次让 oddA 队首出队，随后再让 evenB 队首出队。当 oddA 与 evenB 队列中至少有一个为空队列时跳出循环。检查是哪个队列不为空，将剩下的顾客直接加入到输出序列中。

### 3.3 总体功能的实现

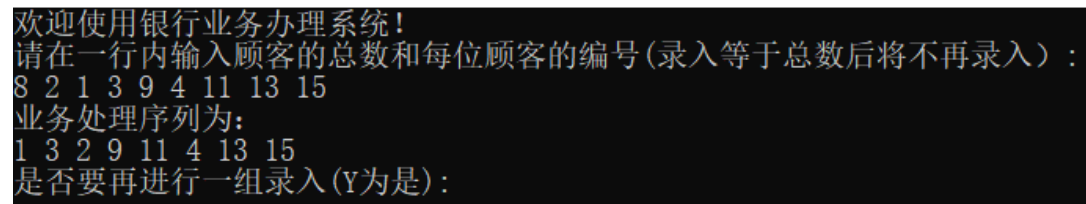
#### 3.3.1 总体功能流程图



### 3.3.2 总体功能核心代码

```
int main()
{
    string judge = "Y";
    int customerNum = 0;
    cout << "欢迎使用银行业务办理系统! " << endl;
    while (judge == "Y")
    {
        LinkedQueue<int> oddA;
        LinkedQueue<int> evenB;
        Vector<int> output;
        init(customerNum);
        if (customerClassification(oddA, evenB, customerNum))
        {
            businessProcessing(oddA, evenB, output);
            cout << "业务处理序列为: " << endl;
            for (int i = 0; i < output.getSize(); i++)
            {
                cout << output[i];
                if (i != output.getSize() - 1)
                {
                    cout << ' ';
                }
            }
            cout << endl;
            cout << "是否要再进行一组录入(Y 为是):";
            cin >> judge;
        }
        cout << "欢迎再次使用! " << endl;
        return 0;
    }
}
```

### 3.3.3 总体功能截屏示例



```
欢迎使用银行业务办理系统！  
请在 一行内输入 顾客的总数和每位顾客的编号(录入等于总数后将不再录入)：  
8 2 1 3 9 4 11 13 15  
业务处理序列为：  
1 3 2 9 11 4 13 15  
是否要再进行一组录入(Y为是)：
```

## 4 测试

### 4.1 功能测试

#### 4.1.1 正常功能测试，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：

```
欢迎使用银行业务办理系统！
请在一行内输入顾客的总数和每位顾客的编号(录入等于总数后将不再录入)：
8 2 1 3 9 4 11 13 15
业务处理序列为：
1 3 2 9 11 4 13 15
是否要再进行一组录入(Y为是)：
```

#### 4.1.2 正常功能测试，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：

```
欢迎使用银行业务办理系统！
请在一行内输入顾客的总数和每位顾客的编号(录入等于总数后将不再录入)：
8 2 1 3 9 4 11 12 16
业务处理序列为：
1 3 2 9 11 4 12 16
是否要再进行一组录入(Y为是)：
```

#### 4.1.3 正常功能测试，AB 窗口人数一样多

测试用例：9 7 5 3 2 9 4 15 13 12

预期结果：7 5 2 3 9 4 15 13 12

实验结果:

```

欢迎使用银行业务办理系统!
请在 一行内输入 顾客的总数和每位顾客的编号(录入等于总数后将不再录入):
9 7 5 3 2 9 4 15 13 12
业务处理序列为:
7 5 2 3 9 4 15 13 12
是否要再进行一组录入(Y为是):

```

#### 4.1.4 顾客人数最小

测试用例: 1 6

预期结果: 6

实验结果:

```

欢迎使用银行业务办理系统!
请在 一行内输入 顾客的总数和每位顾客的编号(录入等于总数后将不再录入):
1 6
业务处理序列为:
6
是否要再进行一组录入(Y为是):

```

#### 4.1.5 顾客全在 A 窗口

测试用例: 8 1 3 5 7 9 11 13 15

预期结果: 1 3 5 7 9 11 13 15

实验结果:

```

欢迎使用银行业务办理系统!
请在 一行内输入 顾客的总数和每位顾客的编号(录入等于总数后将不再录入):
8 1 3 5 7 9 11 13 15
业务处理序列为:
1 3 5 7 9 11 13 15
是否要再进行一组录入(Y为是):

```

#### 4.1.6 顾客全在 B 窗口

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：**

```
欢迎使用银行业务办理系统！
请在 一行内输入 顾客的总数和每位顾客的编号(录入等于总数后将不再录入)：
8 2 4 6 8 10 12 14 16
业务处理序列为：
2 4 6 8 10 12 14 16
是否要再进行一组录入(Y为是)：
```

## 4.2 边界测试

### 4.2.1 顾客数量为零

测试用例：0

预期结果：程序给出提示信息，程序正常运行不崩溃。

实验结果：

```
欢迎使用银行业务办理系统！
请在 一行内输入 顾客的总数和每位顾客的编号(录入等于总数后将不再录入)：
0
输入人数数量不正确或超出上限！
请重新输入：
```



### 4.3 出错测试

#### 4.3.1 顾客编号个数比顾客总数多

测试用例：8 2 5 9 7 11 4 16 15 12 19

预期结果：程序忽略多出的顾客编号(12 19)，输出 5 9 2 7 11 4 15 16

实验结果：

```

欢迎使用银行业务办理系统！
请在同一行内输入顾客的总数和每位顾客的编号(录入等于总数后将不再录入)：
8 2 5 9 7 11 4 16 15 12 19
业务处理序列为：
5 9 2 7 11 4 15 16
是否要再进行一组录入(Y为是)：

```

#### 4.3.2 输入顾客个数错误

测试用例：

-2 5 6 7 8

wasd 4 7 8 9

测试 1 9 5 6

预期结果：程序给出提示信息，程序正常运行不崩溃。

实验结果：

```

欢迎使用银行业务办理系统！
请在同一行内输入顾客的总数和每位顾客的编号(录入等于总数后将不再录入)：
-2 5 6 7 8
输入人数数量不正确或超出上限！
请重新输入：wasd 4 7 8 9
输入人数数量不正确或超出上限！
请重新输入：测试 1 9 5 6
输入人数数量不正确或超出上限！
请重新输入：

```

#### 4.3.3 顾客编号输入不合法

测试用例：

8 5 6 2 1 -7 11 4 9

8 4 s 5 sd 1 12 16 7

8 7 测试 4 5 2 3 15 17

预期结果：程序给出提示信息，程序正常运行不崩溃。

**实验结果：**

```

欢迎使用银行业务办理系统！
请在一行内输入顾客的总数和每位顾客的编号(录入等于总数后将不再录入)：
8 5 6 2 1 -7 11 4 9
输入不合法！！！
是否要再进行一组录入(Y为是):Y
请在一行内输入顾客的总数和每位顾客的编号(录入等于总数后将不再录入)：
8 4 s 5 sd 1 12 16 7
输入不合法！！！
是否要再进行一组录入(Y为是):Y
请在一行内输入顾客的总数和每位顾客的编号(录入等于总数后将不再录入)：
8 7 测试 4 5 2 3 15 17
输入不合法！！！
是否要再进行一组录入(Y为是):

```

**4.3.4 序列中存在两个编号相同的客户**

**测试用例：**8 1 5 7 11 6 5 9 15

**预期结果：**程序给出提示信息，程序正常运行不崩溃。

**实验结果：**

```

欢迎使用银行业务办理系统！
请在一行内输入顾客的总数和每位顾客的编号(录入等于总数后将不再录入)：
8 1 5 7 11 6 5 9 15
序列中存在两个编号相同的客户！
是否要再进行一组录入(Y为是):

```