项目说明文档

数据结构课程设计

——银行业务

作 者 姓 名： 邓泉

学 号： 1953871

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](#_Toc25184029)

[1.1 项目内容](#_Toc25184030)

1.2 背景分析................................................................................................................

[1.3 功能要求](#_Toc25184031)

[2 设计与实现](#_Toc25184032)

[2.1 数据结构设计](#_Toc18205)

2.2 类结构设计与实现................................................................................................

2.[2.1 LinkNode类设计与实现](#_Toc25184033)

2.[2.2 LinkedQueue类设计与实现.....................................................................](#_Toc25184034)..

[2.3 变量与函数设计](#_Toc15880)

[2.3.1全局变量](#_Toc19984)

[2.3.2 函数](#_Toc13564)

[2.4 主函数设计与实现](#_Toc25184035)

[4 测试](#_Toc25184036)

[4.1 功能测试](#_Toc25184037)

[4.1.1 正常测试，A窗口人多](#_Toc25184038)

[4.1.2 正常测试，B窗口人多](#_Toc25184039)

[4.1.3 最小N（N = 1）](#_Toc25184040)

[4.1.4 一个窗口没有元素的情况](#_Toc25184041)

4.2 边界测试.................................................................................................................

# 

# 1 分析

## 1.1 项目内容

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。

1.2 背景分析

在现实生活中，工作效率的差异广泛存在于各行各业。在工作效率差异已经存在的情况下，管理者如何更有效地安排工作，客户如何更有效地选择办事流程，都是十分重要的事情。因此，编写一个能够根据生产效率差异，自动安排工作的程序，具有深刻的现实意义和广泛的实用价值。

## 1.3 功能要求

本程序根据给定的工作效率，在接收到所有客户的编号后，根据奇偶性将客户分为两组，再按照要求的顺序，输出客户编号。

1. 输入说明：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。
2. 输出说明：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。
3. 测试用例：

|  |  |  |  |
| --- | --- | --- | --- |
| 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 |

# 2 设计

## **2.1 数据结构设计**

根据题目要求，我们选择的用来存储客户编号的数据结构应该具有先进先出FIFO的特征。因此，我们选择队列存储本题中的客户编号数据。

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，和栈一样，队列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。

## **2.2 类结构设计与实现**

## 2.1.1 LinkNode类设计与实现

LinkNode类为LinkedQueue的基础结点，是一个模板类。它能够存储一个T类型的值以及一个指向下一结点的link成员变量。

//结点类

template <class T>

struct LinkNode

{

//构造函数

LinkNode(const T& value, LinkNode<T>\* next = NULL) : data(value), link(next) {}

LinkNode(LinkNode<T>\* next = NULL) : link(next) {}

T data;

LinkNode<T>\* link;

};

## 

## 2.1.2 LinkedQueue类设计与实现

模板类，有两个LinkNode类型的成员变量first以及last，分别指向链表头和末尾。这个类当作一个队列,其中有EnQueue(),DeQueue(),isEmpty(),makeEmpty()函数。

makeEmpty(): 逐个删除队列中的结点

//置空队列

void makeEmpty()

{

LinkNode<T>\* del;

while (front != NULL)

{

del = front;

front = front->link;

delete del;

}

}

isEmpty()：只要简单的判断front是否为空即可简单的判断队列是否为空。

//判断是否为空

bool isEmpty()

{

return front == NULL;

}

EnQueue()：因为是队列，所以这个函数负责将数据加入链表的末尾。添加的过程是：如果头尾指针都为空的话，就让头尾指针指向这个元素，否则就在尾指针之后添加这个元素，并且让尾指针指向它。（而且如果front->link为空，得让它指向rear，不这样的话front只是一个单独的节点，和后续结点没有产生联系）。

//入队

bool EnQueue(const T& x)

{

if (isEmpty()) //空队列时

{

front = rear = new LinkNode<T>(x);

if (front == NULL) //分配新结点失败

{

return false;

}

}

else //非空队列时，在链尾追加新的结点并更新队尾指针

{

rear->link = new LinkNode<T>(x);

if (rear->link == NULL) //分配新结点失败

{

return false;

}

rear = rear->link;

}

return true;

}

DeQueue()：如果队列为空，返回。否则返回队列头元素（first的元素），并且让first指向它的下一个元素，如果first为空则也让last为空。

//出队

bool DeQueue(T& x)

{

if (isEmpty())

{

return false;

}

LinkNode<T>\* p = front;

x = p->data;

front = front->link;

delete p;

return true;

}

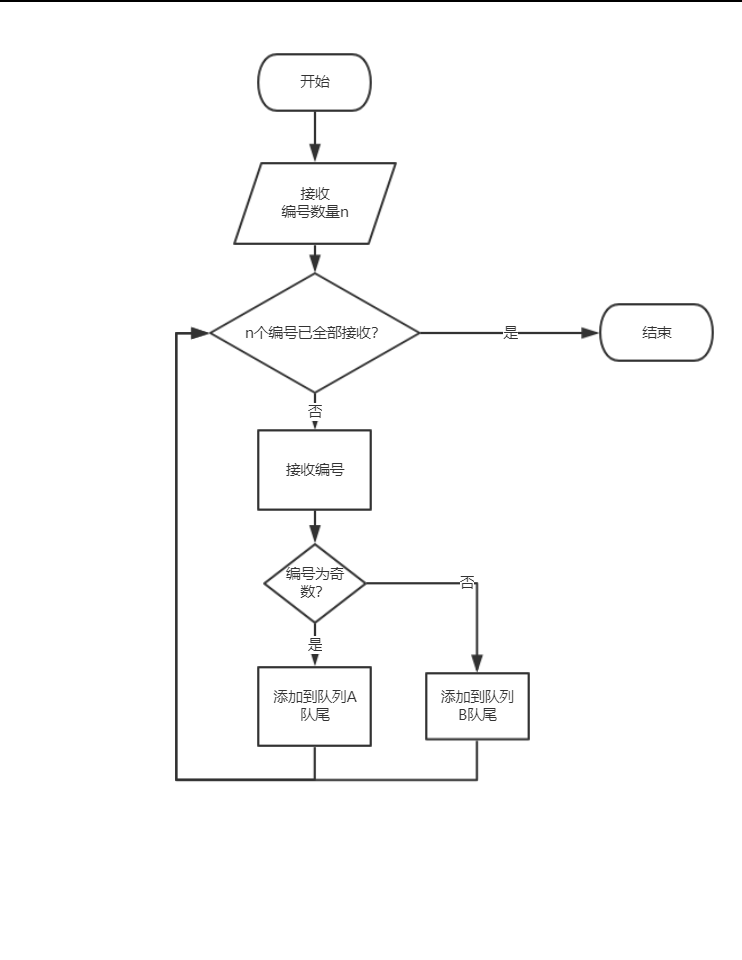
## **2.3 变量与函数设计与实现**

2.3.1 全局变量

LinkedQueue<int> A, B;

2.3.2 函数

输入函数Input()：



代码实现：

//输入

void Input()

{

cout << "请输入顾客总数：";

int N;

cin >> N;

while (N <= 0)

{

cerr << "总数输入错误！请重新输入：";

cin >> N;

}

cout << "请输入N位顾客的编号：";

int customer;

int cnt = 0;

while (true)

{

while (N--)

{

cin >> customer;

if (customer <= 0)

{

cnt++;

}

else if (customer % 2 == 1) //如果是奇数

{

A.EnQueue(customer);

}

else //如果是偶数

{

B.EnQueue(customer);

}

}

if (cnt > 0)

{

cout << "有" << cnt << "位顾客编号输入有误！请补充输入：";

N = cnt;

}

else

{

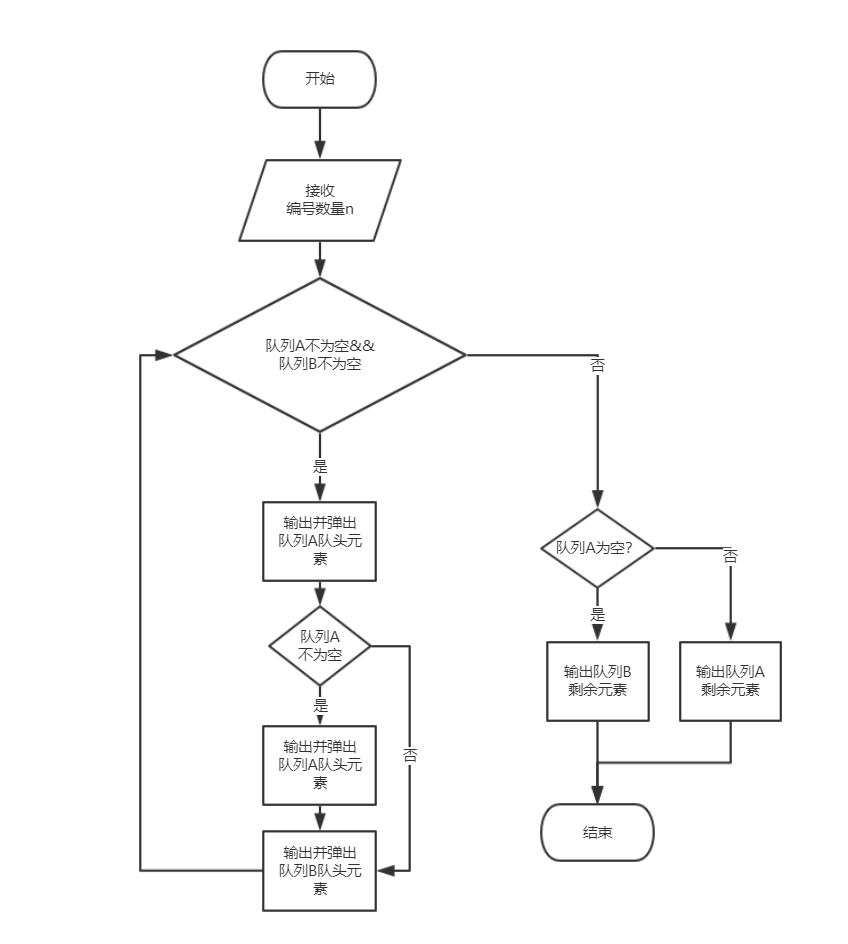
break;

}

}

}

输出函数Output()：



进入循环，循环中，如果队列A不为空，则打印此队列的头节点，并且让头节点指向下一个节点，重复一次，判断A队列是否为空，不为空则打印当前的子节点。接着判断B队列是否为空，不为空打印它的头节点，并且让他指向他的下一个节点。（其实这就是在模拟题目中的情形：当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。）然后继续进入循环判断条件，如果两个队列都为空则退出循环，否则继续循环。值得注意的是,如果其中一个队列为空的话,那么就会把另一个队列的所有元素都输出了

（正常速度：当A窗口处理完两个顾客时，B窗口恰好处理完一个顾客；当某一窗口空时，另一窗口如不空则继续处理。）

代码实现：

void Output()

{

//输出（A出两个，B出一个）

cout << "处理完成的顺序为：";

int customer;

while (!A.isEmpty() || !B.isEmpty())

{

for (int i = 0; !A.isEmpty() && i < 2; i++)

{

A.DeQueue(customer);

cout << customer << " ";

}

if (!B.isEmpty())

{

B.DeQueue(customer);

cout << customer << " ";

}

}

cout << endl;}

## **2.4 主函数设计与实现**

## 

代码实现：

int main()

{

Input();

Output();

system("pause");

return 0;

}

# 3 测试

## 3.1 功能测试

### 3.1.1 正常测试，A窗口人多

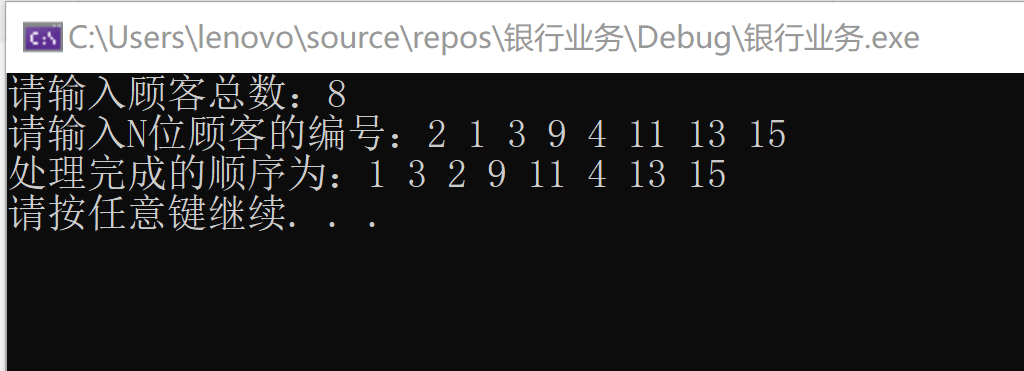
测试用例：

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：**



### 3.1.2 正常测试，B窗口人多

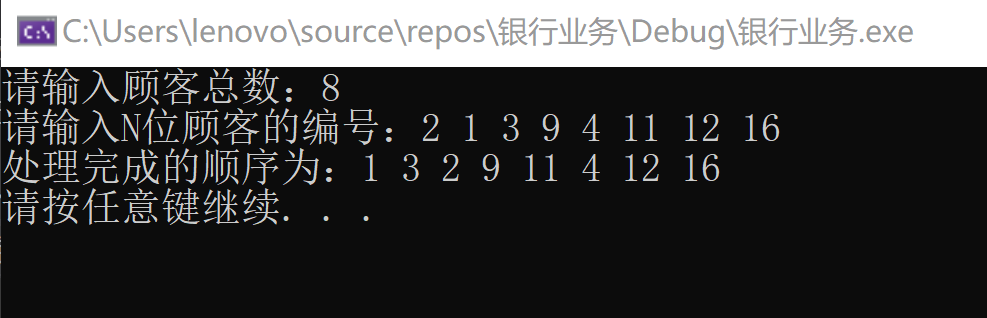
**测试用例：**

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：**



### 3.1.3 最小N

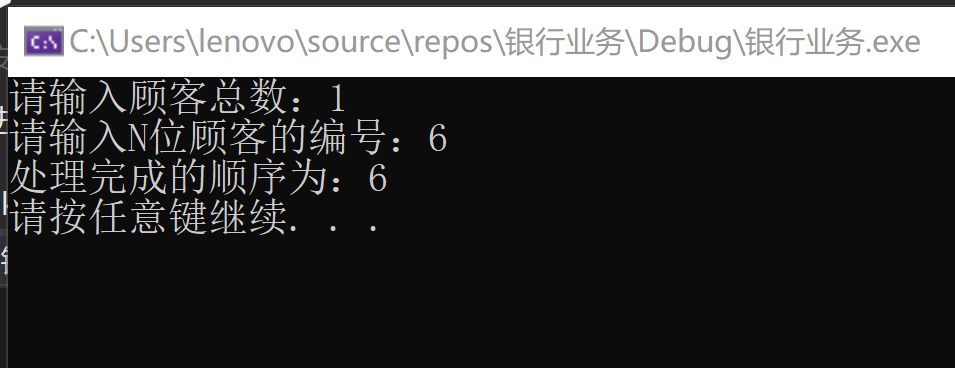
**测试用例：**

1 6

**预期结果：**

6

**实验结果：**



### 3.1.4 一个窗口没有元素的情况

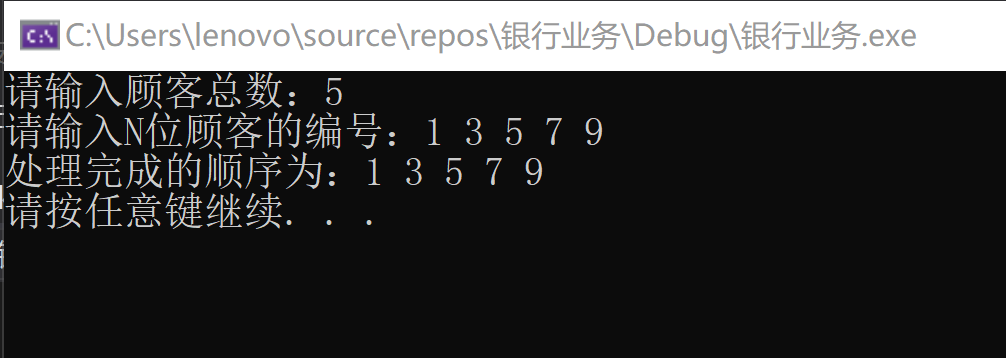
**测试用例：**

5 1 3 5 7 9

**预期结果：**

1 3 5 7 9

**实验结果：**



3.2 边界测试

输入样例：

0

**预期结果：**

正常运行不崩溃，并提示用户重新输入

**实验结果：**

