**项目说明文档**

**数据结构课程设计**

**——银行业务**

作 者 姓 名： 张梓瀚

学 号： 2051943

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 项目分析 - 3 -](#_Toc121308686)

[1.1项目背景 - 3 -](#_Toc121308687)

[1.2 项目要求 - 3 -](#_Toc121308688)

[1.2.1 项目要求 - 3 -](#_Toc121308689)

[1.2.2 输入格式 - 3 -](#_Toc121308690)

[1.2.3 输出格式 - 3 -](#_Toc121308691)

[1.2.4 项目示例 - 3 -](#_Toc121308692)

[2 项目设计 - 3 -](#_Toc121308693)

[2.1 数据结构设计 - 3 -](#_Toc121308694)

[2.2 类设计 - 4 -](#_Toc121308695)

[2.2.1 链表结点类（ListNode） - 4 -](#_Toc121308696)

[2.2.2 链表类（List） - 4 -](#_Toc121308697)

[2.2.3 队列类（Queue） - 5 -](#_Toc121308698)

[2.3 银行业务处理 - 6 -](#_Toc121308699)

[2.3.1 主要思想 - 6 -](#_Toc121308700)

[2.3.2代码 - 7 -](#_Toc121308701)

[3 项目测试 - 7 -](#_Toc121308702)

[3.1 A窗口人多的情况 - 7 -](#_Toc121308703)

[3.2 B窗口人多的情况 - 7 -](#_Toc121308704)

[3.3 人数最少的情况 - 8 -](#_Toc121308705)

[3.4 有窗口没有顾客的情况 - 8 -](#_Toc121308706)

# 1 项目分析

## 1.1项目背景

队列是计算机中非常常用的一种数据结构，在任务调度、消息处理等场景中有着丰富的应用。在日常生活中，我们在商店、医院等场所也经常遇到排队的情形，因此队列的应用场景非常广泛。

## 1.2 项目要求

### 1.2.1 项目要求

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。

### 1.2.2 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。

### 1.2.3 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。

### 1.2.4 项目示例

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

# 2 项目设计

## 2.1 数据结构设计

本项目采用链表存储的队列实现。

## 2.2 类设计

本题需要对银行排队人员的行为进行模拟，使用队列实现这一功能，为了实现代码复用，使用之前题目中编写好的链表来实现链式队列。

### 2.2.1 链表结点类（ListNode）

template <typename T>

class ListNode

{

T\* data;

ListNode<T>\* next;

public:

//默认构造函数

ListNode<T>();

//赋值构造函数

ListNode<T>(const T& src);

//获取数据

const T& get\_data() const;

//获取后继结点

const ListNode<T>\* get\_next() const;

//修改数据

void set\_data(const T& src);

friend class List<T>;

};

### 2.2.2 链表类（List）

链表的结点中存储的数据有链表数据data和后继结点next。为提高代码的泛用性，本题中ListNode类采用模板类实现。

template <typename T>

class List

{

private:

ListNode<T>\* start;

int size = 0;

inline ListNode<T>\* findLastByData(const T& tar);

inline ListNode<T>\* findLastByPos(int pos);

public:

//默认构造函数

List();

//析构函数

~List();

//获取头结点

const ListNode<T>\* get\_head();

//查找指定位置的结点

ListNode<T>\* Locate(int pos);

//向链表尾部插入结点

inline void insert(const T& s);

//向特定位置插入结点

inline bool insert(const T& s, int pos);

//根据数据内容查找结点

inline ListNode<T>\* findByData(const T& s);

//根据在链表中的位置查找结点

inline ListNode<T>\* findByPos(int tarID);

//修改结点数据

inline void modify(T tar, const T& new\_s);

//删除指定位置的结点

inline void remove(int tarID);

//按序打印链表所有结点

inline void print\_all(ostream& os = cout);

//获取链表长度

int get\_list\_size();

//判断链表是否为空

bool is\_empty();

};

### 2.2.3 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。队列类代码如下：

template<typename T>

class Queue : List<T>

{

T head = 0;

public:

//元素入队（插入链表末尾）

void enqueue(const T& data);

//元素出队（删去链表第一个结点）

const T& dequeue();

//判断队列是否为空

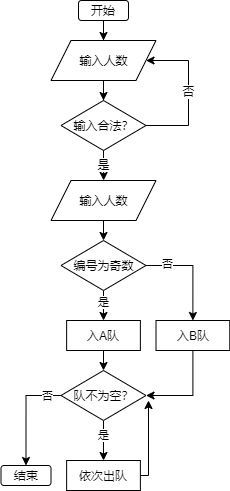
bool isEmpty();

};

## 2.3 银行业务处理

### 2.3.1 主要思想

先按序号的奇偶性将顾客分别加入两个窗口对应的队列，再不断按照A窗口两人、B窗口一人的顺序交替出队，若队列为空说明当前该窗口无人，跳过出队操作，最终出队序列即为顾客办完业务的顺序。



### 2.3.2代码

Queue<int> q1, q2;

int num, id;

std::cin >> num;

for (int i = 1; i <= num; i++)

{

cin >> id;

//先将顾客分别入队

if (id & 1)

q1.enqueue(id);

else

q2.enqueue(id);

}

while (!(q1.isEmpty() && q2.isEmpty()))

{

//若所在队列不为空，按题目描述依次让完成业务的顾客出队并输出

if (!q1.isEmpty())

cout << q1.dequeue() << ' ';

if (!q1.isEmpty())

cout << q1.dequeue() << ' ';

if (!q2.isEmpty())

cout << q2.dequeue() << ' ';

}

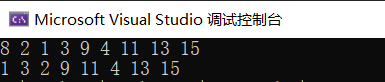
# 3 项目测试

## 3.1 A窗口人多的情况

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：

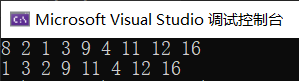


## 3.2 B窗口人多的情况

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：

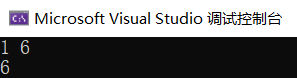


## 3.3 人数最少的情况

测试用例：1 6

预期结果：6

实验结果：



## 3.4 有窗口没有顾客的情况

测试用例：4 1 3 5 7

预期结果：1 3 5 7

实验结果：

