****

**Tongji University**

**数据结构课程设计实验报告**

**005-银行业务**

专 业： 软 件 工 程

指导教师： 张 颖

学 号： 2 1 5 3 0 6 1

姓 名： 谢 嘉 麒

1. **项目概览**
   1. **项目背景**

本次项目以银行业务处理作为事实依据，实际反映了一类生活实际问题的求解方法。在银行业务，酒店排队乃至政府公务处理方面，都需要考虑怎样排队以及处理问题能使效率达到最大化。因此对于排队问题的研究，在日常生活中起着重要意义。

* 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。

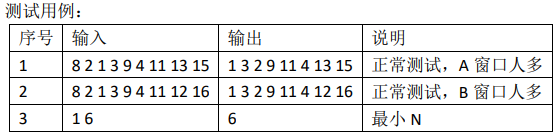
* 1. **项目要求**
     1. **项目输入要求**

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。

* + 1. **项目输出要求**

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。

* + 1. **项目测试用例**

****

* 1. **需求分析**

考虑到银行业务处理是民生根本问题，对于时效性，处理效率，流量大小，复杂程度都有较高的要求，因此本次项目设计也需要从这几个问题为基点出发，优化程序。

* + 1. **功能的完备性**

要求项目功能设计完整，输入输出处理严谨规范，对于错误输入需要提示和重新输入环节，程序结束后需要询问客户是否继续等。

* + 1. **程序的正确性**

要求程序输入无误，计算正确，输出合理合法的顾客编号。

* + 1. **较高的执行效率**

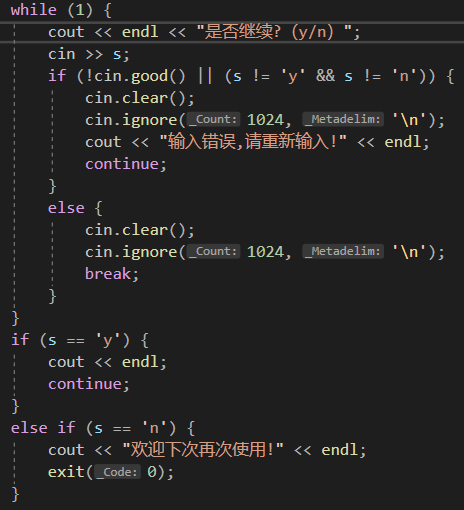
需要对于程序主要算法和数据结构进行优化，降低时间复杂度和空间复杂度，保证代码的运行效率，提高业务处理速度。

* + 1. **具备良好的健壮性**

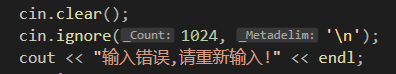
要求程序在用户使用时适时提示，程序中设置多个非法操作警戒断点，进行出错提示，避免程序崩溃。

1. **项目设计思路**
   1. **项目主要功能设计**
      1. **用户交互界面的设计**

考虑到银行业务往往需要连续不断地进行持续操作，因此若每次程序运行完成后都结束程序，会导致下一次业务需要重启程序重新进行，这一点可能成为业务办理效率下降的隐患，因此进行设计使程序能够重复运行，解决此类问题。



代码主要通过while循环进行嵌套操作，若用户希望重复运行，则其输入’y’后程序会跳转回程序首部。同时此处也使用了输入错误处理的相关内容，代码如下：



当用户输入错误时，则需要调用cin函数集中的cin.clear()和cin.ignore()函数截断输入流并清空缓冲区，保证用户输入和程序读入合理合法。

* + 1. **AB窗口业务办理速率的算法处理**

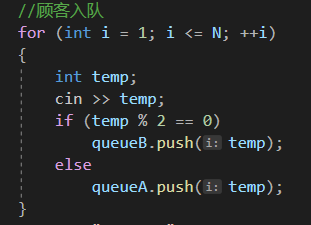
当客户编号数据全部键入程序存储单元后（本次程序使用队列进行存储，详见本报告“数据结构设计”模块），通过输出流进行逐个打印。而考虑到A/B两窗口的业务处理速率存在差异，A处理两个顾客的时间，B处理一个顾客，则需要设计算法使这种差异在输出中体现。

设计采用for循环作为差分计数器，将A窗口的for循环次数设定为2，B窗口不设循环，则使得A窗口每输出两个客户编号后，B窗口才在其后输出一个客户编号，不断循环直至客户编号全部输出，则实现了办理效率的差异处理。

同样的，顾客入队时只需要用取余法判断客户编号是否为偶数即可。

具体代码实现如下：

1. 顾客入队代码：



1. 客户输出代码：



* 1. **项目数据结构设计**

根据银行业务的处理办法以及排队规则常识可以推断出，本次项目设计需要使用以先进先出为特点的数据结构来存储客户编号，保证先来办理的客户先被受理业务。因此，选择队列作为本次项目的数据结构设计，依托链表作为底层数据结构。

* + 1. **单链表数据结构的设计**

考虑到本次项目对于时间复杂度和空间复杂度具有较高要求，因此选择使用单链表来作为队列数据结构的底层基础。而单链表的设计又由链表结点结构体和链表类组成，类中设定多个成员函数作为功能依靠。

1. 单链表结点结构体的设计：
2. T data; //表内数据
3. LinkNode<T>\* link; //结点指针
4. LinkNode(LinkNode<T>\* ptr = NULL) { link = ptr; }
5. LinkNode()
6. 单链表类的设计：
7. List() { first = new LinkNode<T>; }

List(const T& x) { first = new LinkNode<T>(x); }

1. //复制构造函数

List(List<T>& L);

1. //置空链表

void makeEmpty();

1. //链表类析构函数

~List();

1. //计算链表的长度

int Length()const;

1. //返回附加头节点的地址

LinkNode<T>\* getHead()const { return first; }

1. //在链表中查找含有数据x的结点

LinkNode<T>\* Search(T x);

1. //搜索第i个元素的地址

LinkNode<T>\* locate(int i);

1. //取出第i个元素的值

bool getData(int i, T& x)const;

1. //用x修改第i个元素的值

void setData(int i, T& x);

1. //在第i个元素后插入x

bool Insert(int i, T& x);

1. //删去第i个元素，x返回该元素的值

bool Remove(int i, T& x);

1. //判断表是否空，空则返回true

bool Isempty()const

1. //输出链表内所有结点

void outputAll();

1. //重载函数：赋值

List<T>& operator = (List<T>& L);

1. //后插法建立单链表

void inputRear(T endTag);

1. //前插法建立单链表

void inputFront(T endTag);

1. //在尾部插入新数据

void insert\_rear(const T x);

1. //删除第一个数据结点

void delete\_first();

单链表存储结点的头尾指针：

LinkNode<T>\* first; //单链表的表头指针

LinkNode<T>\* tail; //单链表的表尾指针

* + 1. **队列类的设计与使用**

队列类主要以先进先出为特点，实现元素的入队，出队，取队首元素和判空操作。

队（Queue）同栈一样，也是一种运算受限的线性表，规定只能在队首进行元素的出队，在队尾进行元素的入队；

若想向队中添加新元素，则需要（也仅能）将新元素加入队尾，使其成为新的队尾元素，这一过程被称为入队（Push）；

而若想从队中删除元素，则需要找到队首元素并将其删除，使其下一个元素成为新的队首，这一过程被称为出队（Pop）；

而这些过程又以链表作为底层存储来进行，设计如下：

1. 定义链式队列存储：

//内部定义一个链表

List<Type> data;

1. 队列成员函数的实现：
2. //构造函数

//默认构造函数

Queue();

1. //判断队列是否为空

inline bool empty()const;

1. //返回队列大小

inline int size()const;

1. //入队

void push(const Type& i);

1. //出队

void pop();

1. //获取队首元素

const Type& front()const;

Type& front();

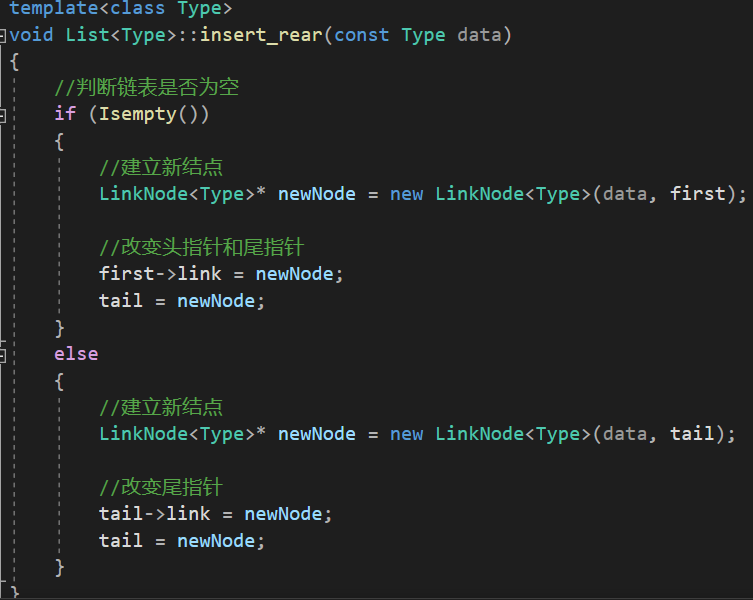
1. **算法设计与程序流程框架**
   1. **入队出队操作的设计**

本次项目功能重点依托于队列数据结构的入队与出队设计，由于这两个函数在程序中调用频率较多，因此优化这两个函数成为优化程序，提高程序运行效率的突破口。

* + 1. **入队操作的设计**

本次队列类的实现将链表尾作为队列尾，因此入队操作即是在链表尾部添加一个新元素。故链表类中的push函数只需要将参数传入链表类insert\_rear函数中即可。

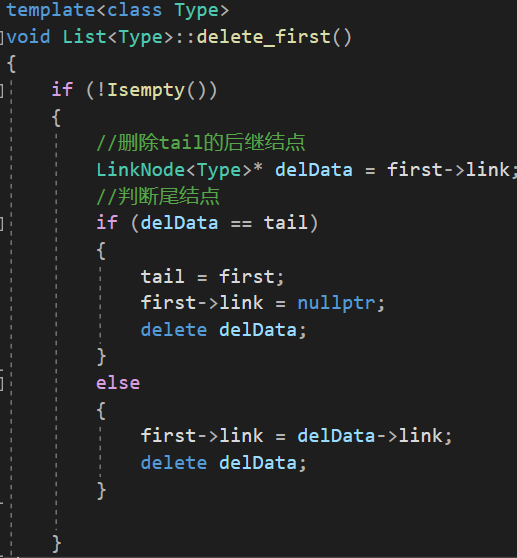
为了避免遍历链表的操作，降低时间复杂度，链表类的实现设计了尾指针指向链表尾部，通过牺牲空间的方法来提高运行效率。在新元素入队时，只需要将新元素插入链表尾指针指向的位置，并更新尾指针即可。



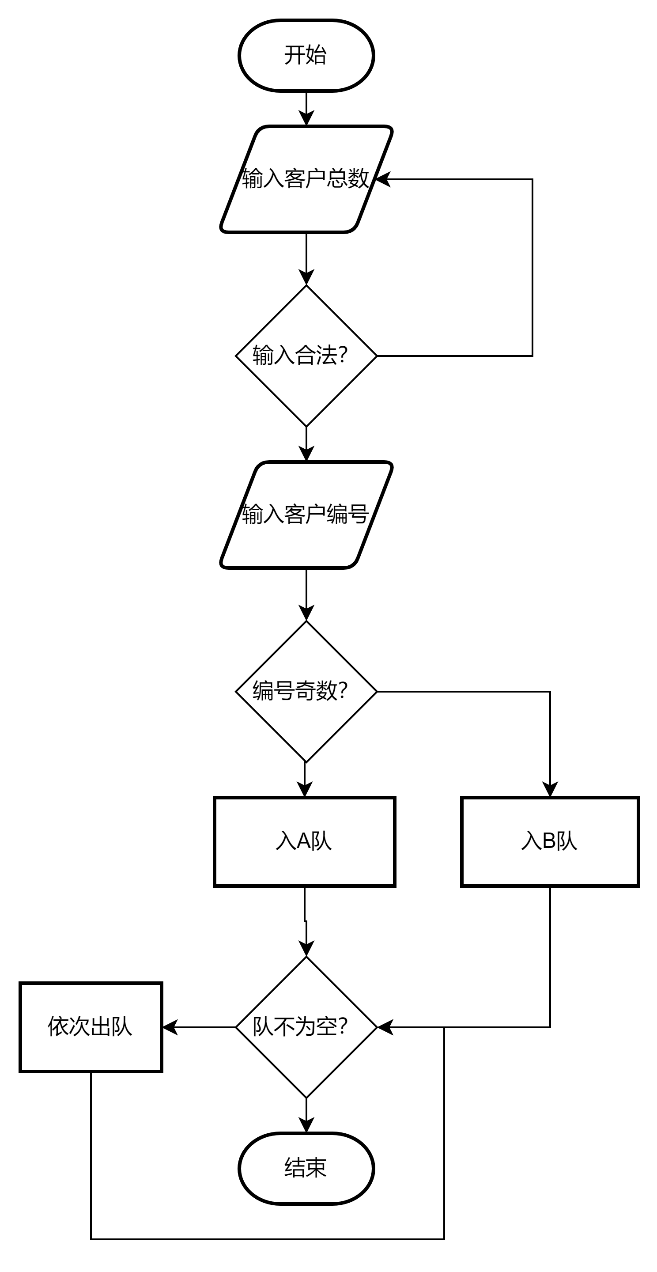
* + 1. **出队操作的设计**

与入队类似的，本次队列类的实现将链表头作为队列头，因此出队操作即是在链表头部删除一个元素。故链表类中的pop函数只需要调用链表类delete\_first函数即可。

链表类的实现设计了尾指针指向链表尾部，通过牺牲空间的方法来提高运行效率。删除队首元素时，只需要将头指针指向的元素删除，并更新头指针即可。

****

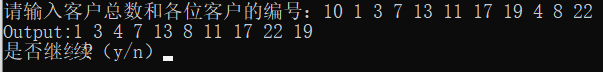
* 1. **程序整体流程图**

****

1. **程序测试**
   1. **A窗口人数较多的情况测试**

测试数据：10 1 3 7 13 11 17 19 4 8 22

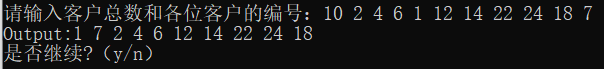
预期结果：1 3 4 7 13 8 11 17 22 19



* 1. **B窗口人数较多的情况测试**

测试数据：10 2 4 6 1 12 14 22 24 18 7

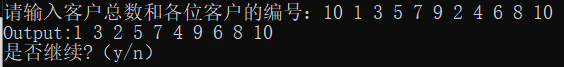
预期结果：1 7 2 4 6 12 14 22 24 18



* 1. **AB窗口人数相同的情况测试**

测试数据：10 1 3 5 7 9 2 4 6 8 10

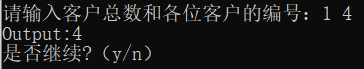
预期结果：1 3 2 5 7 4 9 6 8 10



* 1. **人数最小的情况**

测试数据：1 4

预期结果：4



* 1. **输入非法时的错误处理**

测试数据：-2

预期结果：输入错误，请重新输入。

