项目说明文档

数据结构课程设计

——银行业务

作 者 姓 名： 杨滕超

学 号： 2151298

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 项目概述](#_Toc495668153) 3

[1.1 项目背景](#_Toc495668154) 3

[1.2 项目功能](#_Toc495668155) 3

[1.3 项目分析](#_Toc495668154) 3

[2 项目设计](#_Toc495668156) 4

[2.1 总体设计](#_Toc495668157) 4

[2.2 类设计](#_Toc495668158) 4

[2.2.1 List类](#_Toc495668163) 4

[2.2.2 Queue类](#_Toc495668163) 4

[2.2.3 Vector类](#_Toc495668163) 5

[2.2.4 Vector游标类](#_Toc495668163) 6

[3 项目实现](#_Toc495668161) 8

[3.1 总体实现](#_Toc495668162) 8

[3.1.1 总体实现流程图 8](#_Toc495668167)

[3.1.2 总体实现思路](#_Toc495668167) 9

[3.1.3 总体实现主要代码](#_Toc495668167) 9

[3.2 排队功能实现 1](#_Toc495668166)0

[3.2.1 排队功能实现流程图 1](#_Toc495668167)0

[3.2.2 排队功能实现思路](#_Toc495668167) 10

[3.2.3 排队功能实现主要代码](#_Toc495668168) 10

[3.3 出队入队功能具体实现 1](#_Toc495668170)2

[4 项目测试](#_Toc495668161) 13

[4.1 正确输入](#_Toc495668174) 13

[4.1.1 A窗口人数多](#_Toc495668176) 13

[4.1.2 B窗口人数多](#_Toc495668176) 13

[4.1.3 A、B窗口人数相同](#_Toc495668176) 13

[4.1.4 最小人数](#_Toc495668176) 13

[4.2 输入非法](#_Toc495668174) 14

[4.2.1 输入顾客总数为负数](#_Toc495668176) 14

[4.2.2 输入顾客编号为非法字符](#_Toc495668176) 14

1 项目概述

* 1. 项目背景

银行排队作为日常生活中一种常见的现象，广泛地发生于每个人的点点滴滴。可以发现排队这件事同样渗透于我们的日常生活，并且发挥着及其重要的作用。排队不仅仅是个人文明素养的表现，更是提高办事效率的有效方法。但对于银行排队，相信在日常生活中每个人都或多或少的感受过等待良久的苦楚，如何排队如何减少排队时间仍然是一个值得研究的问题。

基于上述考虑，本项目将其抽象为模型，研究排队在某种特定情况的输出序列，对现实的排队具有指导性的意义。

* 1. 项目功能

用户输入第一个正整数N，表示顾客总数，接着输入N个数字代表顾客的编号，数字均以空格分隔。本项目将银行排队分为两个窗口A和B，A窗口每处理完2个顾客，B窗口处理完一个顾客。按照要求，我们将顾客编号为奇数的顾客分到A窗口办理业务，为偶数的顾客则到B窗口办理业务。基于上述过程，按照业务处理完成的顺序输出顾客的编号若两个窗口同时完成业务，则优先输出A窗口的顾客编号。

输出以空格分隔，但最后一个是输出不能有多余的空格。

* 1. 项目分析

1. 考虑到在用户输入错误时，程序不会崩溃，而是会输出输入错误的提示，保证程序的正确运行。
2. 考虑到程序效率问题，选取适合的数据结构进行对题目过程的模拟，保证程序在较低的时间复杂度下完成，本项目的事件复杂度为O(n)。
3. 满足题目的基本要求，在其基础上添加输入提示，提高了对用户的友好性。

2 项目设计

2.1 总体设计

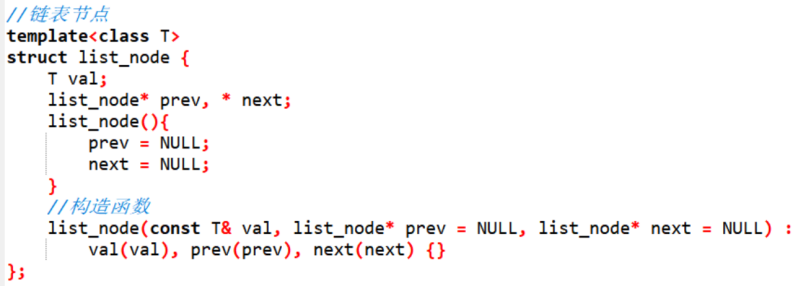
当我们看到到“排队”问题，优先考虑到队列这种数据结构，这是一种实现了“先进先出”的结构，满足了本项目的要求。

2.2 类设计

本项目需要用到List类作为底层实现Queue类，利用Vector类记录输入的顾客编号。

2.2.1 List类

·链表结点类设计：其中包括需要储存的数据，前驱节点和后继节点。如下图所示：



·链表类设计：数据成员包括，头结点，尾结点以及记录节点个数数据。

主要功能(与本题有关)包括：

·//返回链表大小inline int size() const;

·//判断链表是否为空inline bool empty() const;

·//清空链表void clear();

**·//最后添加元素void emplace\_back(const T& x)；**

**·//最后删除元素void pop\_back();**

**·//开头添加元素void emplace\_front(const T& x);**

**·//开头删除元素void pop\_front();**

2.2.2 Queue类

队列和栈一样，同样是一种运算受限的线性表，它允许的操作是在表的前端(front)进行删除，表的后端(rear)进行插入，遵循先进先出的原则。其中的前端称为队头，后端称为队尾。队列的实现同样分为两种方式，顺序队列和链式队列，对于顺序队列，为了实现空间的循环利用，采用循环队列的方式，其中的要点在于队头指针和队尾指针往后移动增加1后对最大空间大小取模。而对于链式队列，其优点在于没有空间的限制，可以灵活地申请空间。

基于上述考虑，我们采用以链表为基础的队列。考虑到链表无论是在头或尾的插入删除结点的时间复杂度均为O(1)，因此我们将链表头作为队列头，链表为作为队列尾。

该类主要成员函数如下：

//判断队列是否为空

inline bool empty() const {

return data.empty();

}

//获得队列中元素个数

inline int size() const {

return data.Size;

}

//清空队列

void clear() {

data.clear();

}

//在队列为插入元素

void emplace(const T& x) {

data.emplace\_back(x);

}

//在队列头删除元素

void pop() {

data.pop\_front();

}

//查询队列头的元素值

T& front() {

return data.getHead()->next->val;

}

2.2.3 Vector类

向量作为存储同一种类型数据的一维数组，相当于是数组的扩展，在其基础上增加方便程序员工作的操作。其中的优点在于：对于查找某个位置的元素的时间复杂度为O(1)，但是它的缺点也十分明显：向量的删除与插入的时间代价是巨大的，时间复杂度基本是O(n)级别；但特殊的是，在最后插入元素时，时间复杂度为O(1)。

我们设计Vector类的同时，同样设计了它的迭代器。

主要成员函数如下：

·//构造函数

Vector();

Vector(int size);

Vector(const Vector<T>& V);

Vector(int size, const T& val);

·//清空Vector

void clear();

·//最后添加元素

void emplace\_back(const T& x);

·//最后删除元素

void pop\_back();

·//返回最后元素

const T back() const ;

T& back() ;

·//返回开头元素

const T front() const ;

T& front() ;

·//插入

void insert(const Vector<T>::vector\_iterator& it, const T& x);

·//删除

void remove(const Vector<T>::vector\_iterator& it);

·//迭代器begin

inline Vector<T>::vector\_iterator begin() ;

·//迭代器end

inline Vector<T>::vector\_iterator end() ;

·//返回迭代器的查找

typename Vector<T>::vector\_iterator find(const T& x);

·//重载[]

T& operator[](const int index) ;

·//重载=

Vector<T>& operator=(const Vector<T>& V);

·//重新设置大小

void resize(int size);

void resize(int size, const T& val);

2.2.4 Vector游标类

其中数据成员包含了指向Vector中数组元素的指针，并实现的重载\*、->、++、--、<、>等操作符。

主要函数如下：

·//重载\*

T& operator\*();

·//重载->

T\* operator->();

·//重载==

inline bool operator==(const vector\_iterator& it) const ;

·//重载!=

inline bool operator!=(const vector\_iterator& it) const;

·//重载不等号

inline bool operator<(const vector\_iterator& it) const;

inline bool operator>(const vector\_iterator& it) const ;

inline bool operator<=(const vector\_iterator& it) const;

inline bool operator>=(const vector\_iterator& it) const ;

·//重载自加

inline Vector<T>::vector\_iterator& operator++();

·//重载自减

inline Vector<T>::vector\_iterator& operator--();

·//后置自加

Vector<T>::vector\_iterator operator++(int);

·//后置自减

Vector<T>::vector\_iterator operator--(int);

·//与数字相加

friend Vector<T>::vector\_iterator operator+(const vector\_iterator& it,int num);

·//与数字相减

friend Vector<T>::vector\_iterator operator-(const vector\_iterator& it,int num);

·//自加

Vector<T>::vector\_iterator operator+=(int num);

·//自减

Vector<T>::vector\_iterator operator-=(int num);

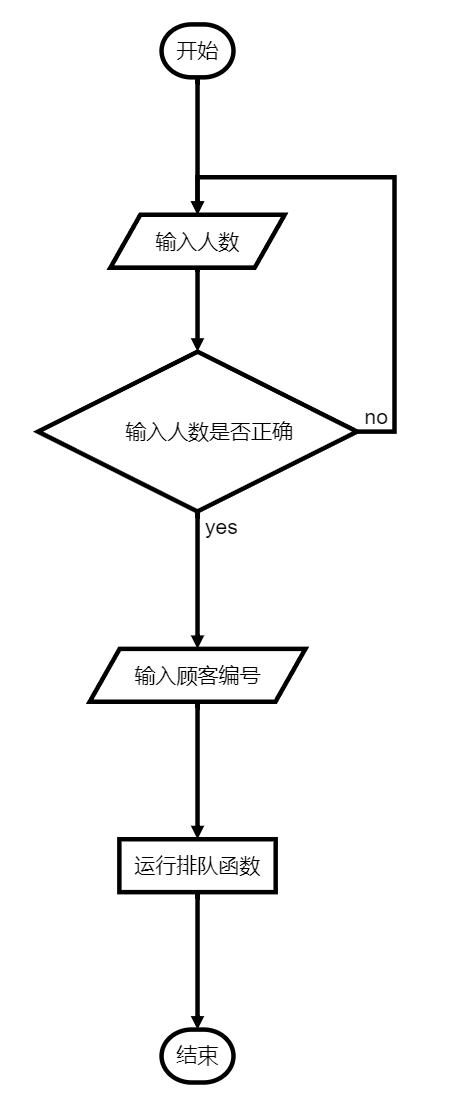
·//两个相减

friend int operator-(const vector\_iterator& it1, const vector\_iterator& it2);

3 项目实现

3.1 总体实现

3.1.1 总体实现流程图



3.1.2 总体实现思路

首先要求用户输入顾客总数N，接着输入N个顾客编号，最后进入排队函数进行输出。对于数字的输入，我们需要考虑输入错误的处理。

3.1.3 总体实现主要代码

cout << "input number" << endl;

while (1){

cin >> size;

if (cin.fail() || size < 1){

cout << "输入人数有误，请重新输入" << endl;

cin.clear();

cin.ignore(65536, '\n');

}

else

break;

}

Vector<int> nums(size, 0);

cout << endl << "请输入顾客编号" << endl;

for (int i = 0; i < size; ++i){

cin >> nums[i];

if (cin.fail()){

cout << "输入有误，请重新输入!" << endl;

cin.clear();

cin.ignore(65536, '\n');

--i;//重新输入该项

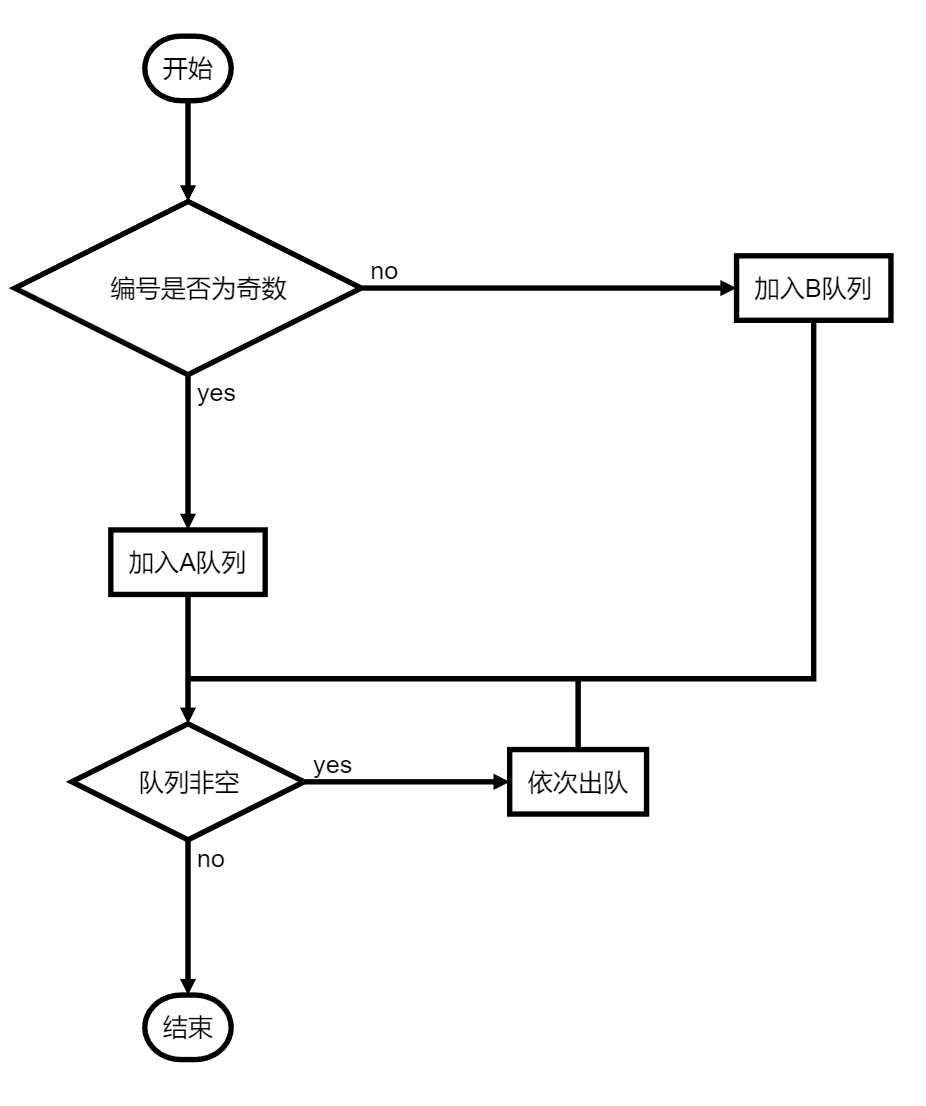
}

}

bank(nums);//银行排队函数

3.2 排队功能实现

3.2.1 排队功能实现流程图



3.2.2 排队功能实现思路

实例化两个队列A和B，将Vector的对象nums以引用的方式传入函数，这样可以提高程序的效率，避免没有必要的复制。

接着将奇数编号的顾客加入A队列，偶数编号的顾客加入B队列。

并且，以两个队列至少有一个非空为循环条件，优先输出A队列中队头的两个元素，再输出B队列中队头的一个元素。在这里需要注意的是：输出之前需要判断队列是否非空，否则将会引起程序的错误。

关于保证最后一个没有空格，我们对第一个输出的顾客编号特殊判断，使其输出不带空格，其余的顾客输出格式均以“空格 + 编号”为其形式。

3.2.3 排队功能实现主要代码

主要代码如下：

Queue<int> A, B;

for (int i = 0; i < Size; ++i)

{

if (nums[i] % 2)

A.emplace(nums[i]);

else

B.emplace(nums[i]);

}

cout << endl << "业务完成顺序为" << endl;

while (Size)

{

//每一轮A出两个B出一个

if (!A.empty())

{

if (Size == nums.size())

cout << A.front();

else

cout << ' ' << A.front();

A.pop();

--Size;

}

if (!A.empty())

{

cout << ' ' << A.front() ;

A.pop();

--Size;

}

if (!B.empty())

{

if (Size == nums.size())

cout << B.front();

else

cout <<' ' << B.front();

B.pop();

--Size;

}

}

cout << endl;

3.3 出队入队功能的具体实现

对于本项目的重点，即为队列的出队入队操作，关于这个操作如何基于链表的操作实现，本人将在此进行分析。

Queue类中的出对入队函数如下：

//在队列为插入元素

void emplace(const T& x) {

data.emplace\_back(x);

}

//在队列头删除元素

void pop() {

data.pop\_front();

}

深入List类中的emplace\_back函数和pop\_front函数，代码如下：

//最后添加元素

template<class T>

void List<T>::emplace\_back(const T& x){

//申请新的空间

list\_node<T>\* node = new list\_node<T>(x);

++Size;//元素个数更新

//连接最后一个

tail->next = node;

node->prev = tail;

//尾结点更新

tail = node;

}

//删除开头元素

template<class T>

void List<T>::pop\_front(){

if (empty())//空了就不能删除

return;

--Size;//元素个数减少

//要删除的节点的下一个

list\_node<T>\* temp = head->next->next;

list\_node<T>\* del = head->next;

delete del;

head->next = temp;

if (temp)//如果下一个不为空

temp->prev = head;

else//下一个为空，更新尾结点指向头结点

tail = head;

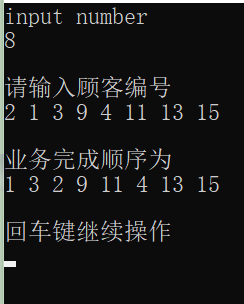
}

4 项目测试

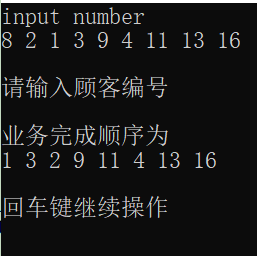
以下测试均符合预期。

4.1 正确输入

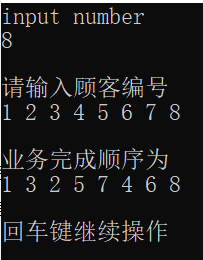
4.1.1 A窗口人数多



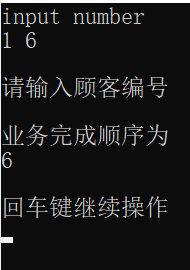
4.1.2 B窗口人数多



4.1.3 A、B窗口人数相同

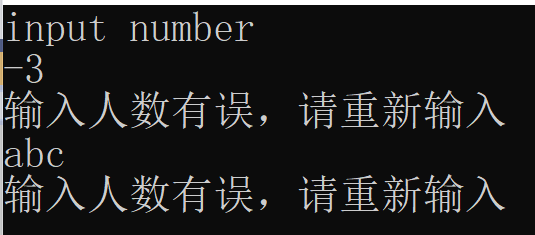


4.1.4 最小人数



4.2 输入非法

4.2.1 输入顾客总数为负数



4.2.2 输入顾客编号为非法字符

