项目说明文档

数据结构课程设计

——银行业务

同济大学

Tongji University

目 录

[1 分析 1](#_Toc60058446)

[1.1 项目要求 1](#_Toc60058447)

[1.2 项目分析 1](#_Toc60058448)

[2 设计 1](#_Toc60058449)

[2.1 数据结构设计 1](#_Toc60058450)

[2.2 成员与操作设计 2](#_Toc60058451)

[2.3 系统设计 3](#_Toc60058452)

[3 实现 4](#_Toc60058453)

[3.1 在顺序队列中插入一个元素： 4](#_Toc60058454)

[3.1.1 插入功能图文描述 4](#_Toc60058455)

[3.1.2 插入功能核心代码 5](#_Toc60058456)

[3.1.3 插入功能截屏示例 5](#_Toc60058457)

[3.2 删除功能的实现 6](#_Toc60058458)

[3.2.1 删除功能流程图 6](#_Toc60058459)

[3.2.2 删除功能核心代码 7](#_Toc60058460)

[3.2.3 删除功能截屏示例 8](#_Toc60058461)

[3.3 其他功能的实现 8](#_Toc60058462)

[3.3.1 判断队列是否为空： 8](#_Toc60058463)

[3.3.2 返回队列大小 9](#_Toc60058464)

[3.3.3 从队头到队尾将队列中的元素依次输出： 9](#_Toc60058465)

[4 测试 10](#_Toc60058466)

[4.1 队列测试 10](#_Toc60058467)

[4.1.4 测试集1： 10](#_Toc60058468)

[4.1.5 测试集2： 11](#_Toc60058469)

[4.2 项目测试 11](#_Toc60058470)

[4.2.1 正常测试，A窗口人多 11](#_Toc60058471)

[4.2.2 正常测试，B窗口人多 12](#_Toc60058472)

[4.2.3 最小N 13](#_Toc60058473)

[4.3 出错测试 13](#_Toc60058474)

[4.3.1 乱码与错误输入测试 13](#_Toc60058475)

[4.3.2 数据中某一项输入错误 14](#_Toc60058476)

# 1 分析

## 1.1 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 项目分析

在银行办理业务时，处理原则是先到的先处理，这正好和队列FIFO（先进先出）的原则不谋而合。为完成该项目，应该实现队列类。

# 2 设计

## 2.1 数据结构设计

如上功能分析所述，该项目需要实现一个队列类。该类大致需要以下功能：

定义顺序队列类型，使其具有如下功能：

(1) 建立一个空队列；

(2) 释放队列空间，将队列销毁；

(3) 将队列清空，变成空队列；

(4) 判断队列是否为空；

(5) 返回队列内的元素个数；

(6) 将队头元素弹出队列（出队）；

(7) 在队列中加入一个元素（入队）；

(8) 从队头到队尾将队列中的元素依次输出。

## 2.2 成员与操作设计

为了使程序能够泛用，使用类模板类

**Queue类：**

**私有成员：**

T\* e; //存储元素信息

int S; //当前元素个数

int MAXS; //最大能存储多少个元素

int \_front; //队头

int \_rear; //队尾

const int INIT\_SIZE = 100;

**ADT接口：**

Queue();

//构造函数，对私有成员进行赋值以及内存申请

Queue(Queue<T>& )

//赋值构造函数，由于涉及到变量的动态申请，所以不能使用默认构造函数

Queue<T>& operator=(Queue<T>& )

//复制构造函数，由于涉及到变量的动态申请，所以不能使用默认复制函数

bool empty()

//返回队列是否为空

int size()

//返回队列的大小

T front()

//返回队列的首元素，若队列为空，会报错

Status popQueue(T& );

//从队列中弹出元素，若队列为空，不会报错

T pop();

//从队列中弹出元素，若队列为空，会报错

void push(T);

//将元素插入到队列中

void printQueue();

//从队首到队尾打印队列

~Queue();

//析构函数

## 2.3 系统设计

按照顺序将所有顾客的编号读入，若为奇数插入A队列，若为偶数插入B队列。而后当A和B都不为空时，A每输出两个顾客，B输出一个顾客；最后，把有剩余的队列中的元素输出即可。

# 3 实现

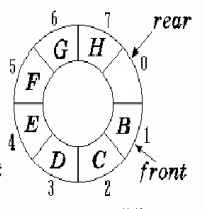
## 3.1 在顺序队列中插入一个元素：

### 3.1.1 插入功能图文描述

首先可以通过队列的大小来判断队列是否已经“真满”了（申请的空间是否全部被占用），如果未满，将新加入的元素赋值给e[rear]，即赋值给队尾元素，再将rear加1，同样，如若rear和MAXS相等，则将rear置为零，这样，就构成了一个循环队列，在这之后，更新队列的大小。

如果已经“真满”了，按照课本上的做法，将无法再插入新元素。我对其进行了改进：我在新的空间上再次申请了一段连续的内存空间，大小为原先大小的两倍。之后将就空间上，从front到rear的元素转移到新空间的0-S（队列的大小）上，相当于做了一次一一映射。

形象地看，可以将原来的顺序数组视作下图：



而将front到rear的元素一一映射到新申请的连续空间最靠前的S（队列的大小）个位置上。而后更新front，rear以及MAXS的值，删除原先申请的空间。

此时，“真满”的队列就被转化成了“未满”的队列，可以按照向“未满”队列中加入元素的方法（第一段）对其进行操作。

### 3.1.2 插入功能核心代码

template<class T>

void Queue<T>::push(T ele)

{

//如果队列已满，更新申请的空间

if (S == MAXS)

{

//记录旧位置

T\* oldE = e;

//申请新空间

e = new(nothrow) T[MAXS \*= 2];

if (!e)exit(OVERFLOW);

//将旧储存的值转移到新位置

for (int i = 0; i < S; ++i)e[i] = oldE[(i + \_front) % S];

//更新\_front和\_rear的值

\_front = 0, \_rear = S;

//删除旧位置

delete[]oldE;

}

//加入新元素

e[\_rear] = ele;

//更新\_rear的值

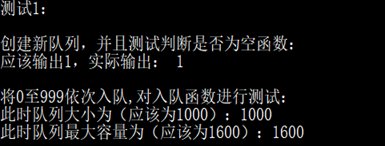
\_rear = (\_rear + 1) % MAXS;

//更新大小

++S;

}

### 3.1.3 插入功能截屏示例



## 3.2 删除功能的实现

### 3.2.1 删除功能流程图

开始

队列为空？

报错，结束程序

记录\_front的值

\_front = (\_front + 1) % MAXS

返回记录的值

结束

### 3.2.2 删除功能核心代码

template<class T>

T Queue<T>:: pop()

{

if (!S) { printf("队列为空！无法弹出！\n"); exit(-1); }

//更新大小

--S;

//记录删除的元素

T ele = e[\_front];

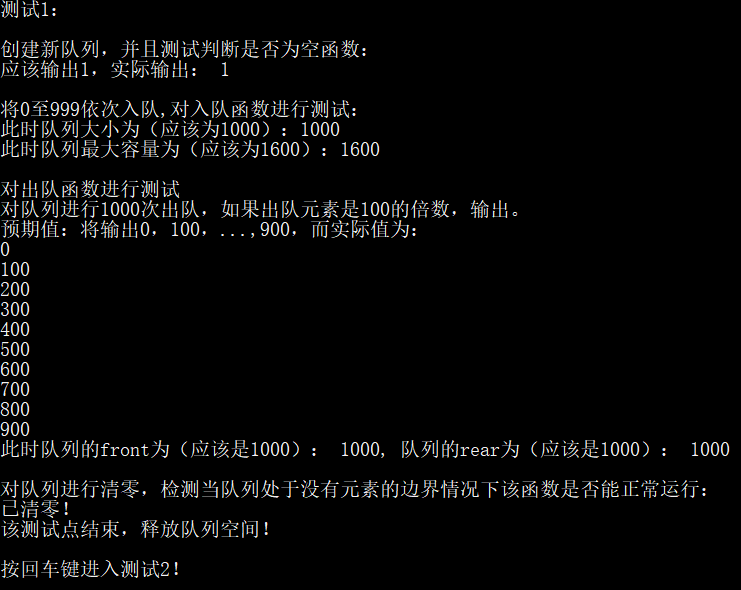
//更新\_front的位置

\_front = (\_front + 1) % MAXS;

return ele;

}

### 3.2.3 删除功能截屏示例



## 3.3 其他功能的实现

### 3.3.1 判断队列是否为空：

通过队列中储存的S来判断队列是否为空。

代码；bool empty(){ return !S;}

### 3.3.2 返回队列大小

返回队列中储存的S值。

代码：int size(){ return S;}

### 3.3.3 从队头到队尾将队列中的元素依次输出：

由于队列的存储结构是顺序表，只是抽象数据类型提供的接口受限，使得无法访问或对队列非首元素进行操作。

但是在该ADT接口的实现上，可以利用顺序表的性质，在不改变队首元素（将其逐个取出）的基础上，提供一种打印队列全体元素的功能。故只需要从队首开始遍历该顺序表，打印所有遍历到的元素；而当遍历到MAXS的时候，返回顺序表物理位置的首地址，继续遍历，直到队尾为止。

代码：

template<class T>

void Queue<T>::printQueue()

{

if (!S)

{

cout << "队列为空！" << endl;

return;

}

cout << "队列从队头到队尾包含：" << endl;

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

{

cout << e[(i + \_front) % MAXS] << ' ';

if (!((i + 1) % 20))cout << endl;

}

cout.put('\n');

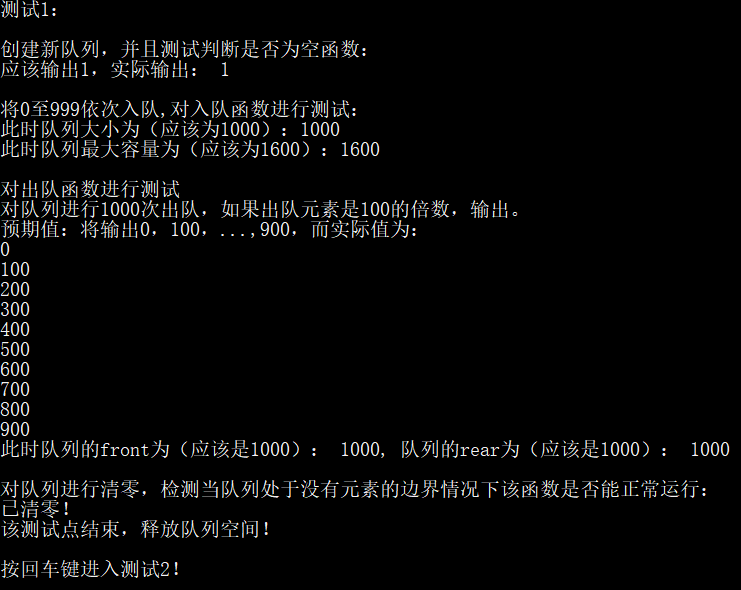
}

# 4 测试

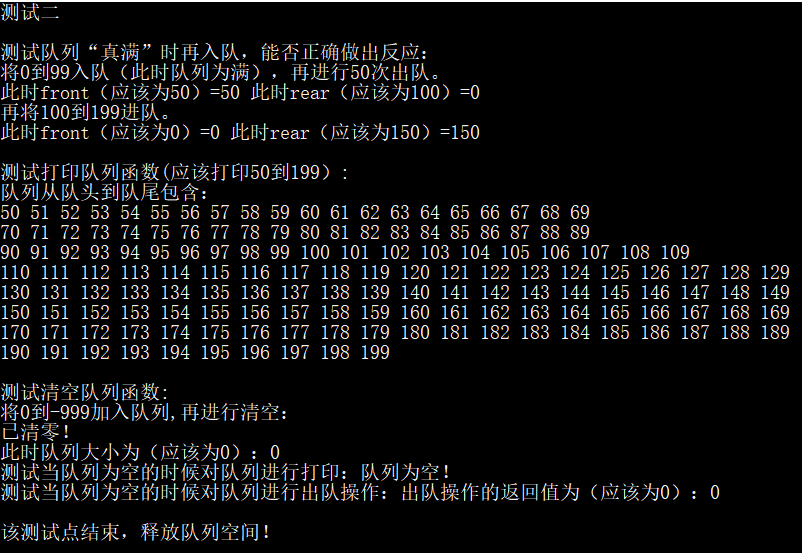
## 4.1 队列测试

（在测试程序中进行测试）

### 4.1.4 测试集1：



### 4.1.5 测试集2：



## 4.2 项目测试

### 4.2.1 正常测试，A窗口人多

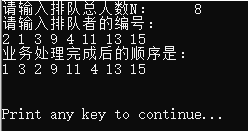
**测试用例：**

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：**



### 4.2.2 正常测试，B窗口人多

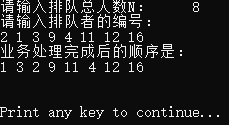
**测试用例：**

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：**



### 4.2.3 最小N

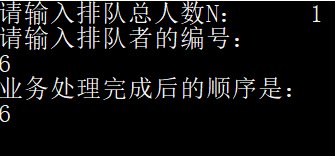
**测试用例：**

1

6

**预期结果：**6

**实验结果：**



## 4.3 出错测试

### 4.3.1 乱码与错误输入测试

**测试用例：**

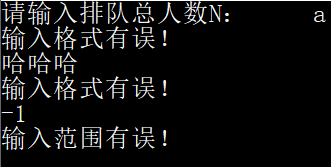
a

哈哈哈

-1

**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：**

****

### 4.3.2 数据中某一项输入错误

**测试用例：**

**7**

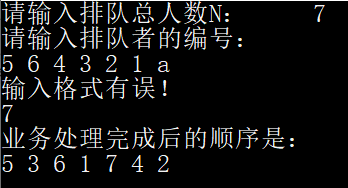
5 6 4 3 2 1 a

7

**预期结果：**对错误数据给出提示信息，要求重新输入

输出结果：5 3 6 1 7 4 2

**实验结果：**

****