项目说明文档

数据结构课程设计

——银行业务

作 者 姓 名： 李佳诺

学 号： 1751188

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 3

1.1 背景分析 3

1.2 功能分析 3

2 设计 3

2.1 算法设计 3

2.2 数据结构的设计 4

2.3 类成员与操作设计 4

队列类(Queue) 4

3 实现 5

3.1 链表类的实现 5

3.1.1 构造函数Queue（） 5

3.1.2 判断是否空队函数IsEmpty（） 5

3.1.3 判断是否满队函数IsFull（） 5

3.1.4 入队函数EnQueue（） 6

*入队函数流程图* 6

3.1.5 出队函数DeQueue（） 8

*出队函数流程图* 8

3.2 主函数的实现 9

4 测试 11

4.1 功能性测试 11

4.1.1 正常测试，A窗口人多： 11

4.1.2 正常测试，B窗口人多： 12

4.1.3 最小N: 12

4.2 出错性测试 13

# 1 分析

## 背景分析

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。

## 功能分析

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。

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。

综上，本程序要完成两组数据的整合，并以先进先出（FIFO）的原则进行输出。

# 设计

## 算法设计

本程序解决银行业务处理的基本思路是：依次遍历所给数据，并将它们依次分别放入队列A,B。待全部数据都遍历完之后，开始向外输出数据，其顺序即为业务办理的顺序。这里的思路是：每次从A中输出两个元素之后再从B中输出一个元素。

## 数据结构的设计

考虑到银行办理业务的原则是先来的人先办理，我们自然想到了数据结构中的先进先出（FIFO）原则。于是，本程序采用了符合这一特征的队列（Queue）这一数据结构。并且，考虑到程序的可扩展性，以及银行业务量比较大的特点，本程序采用的动态数组这一存储结构。

## 类成员与操作设计

### 队列类(Queue)

私有成员：

private:

T \*Array; //动态数组用于存储数据

int Capacity; //队列最大容量

int Front; //队列头位置

int Rear; //队列尾位置

int Sizes; //队列中对象的数量

公有成员：

public:

Queue(int QueueSize); //构造函数初始化各个参数

~Queue(); //析构函数

bool IsFull(); //判断队列是否满

bool IsEmpty(); //判断队列是否为空

void EnQueue(T &data); //入队

T DeQueue(); //出队

void print(); //输出队中元素

# 实现

## 链表类的实现

### 构造函数Queue（）

该函数主要功能是初始化队列各参数：

//构造函数初始化各个参数

template<typename T>

Queue<T>::Queue(int QueueSize)

{

Front = 0;

Rear = 0;

Sizes = 0;

Capacity = QueueSize;

Array = new T[QueueSize];

}

### 判断是否空队函数IsEmpty（）

该函数主要功能是判断队列是否空，即大小是否为0。基本思路是检查Sizes是否：

//判断是否为空

template<typename T>

bool Queue<T>::IsEmpty()

{

return Sizes == 0;

}

### 判断是否满队函数IsFull（）

该函数主要功能是判断队列是否已满，即大小是否超过容量。基本思路是检查Sizes是否等于最大容量：

//判断是否为满队列

template<typename T>

bool Queue<T>::IsFull()

{

return Sizes == Capacity;

}

### 入队函数EnQueue（）

入队函数流程图

开 始

抛出异常

Rear=(Rear+1)%Capacity;

++Size;

IsFull?

结 束

Array[Rear]=data;

否

是

该函数主要功能是将传入参数作为一个元素插入队列中，要求每次插入到队尾：

//入队函数

template<typename T>

void Queue<T>::EnQueue(T data)

{

if (IsFull())

{

throw("full");

}

Array[Rear] = data;

Rear = (Rear+1 ) % Capacity;

++Sizes;

}

### 出队函数DeQueue（）

出队函数流程图

开 始

抛出异常

return this->Array[Front++];

IsEmpty?

结 束

--Sizes;

否

是

## 主函数的实现

**主函数流程图**

开 始

按A,B优先顺序出队

结 束

分别入队

输入是否合法？

输入人数及编号

否

是

**本函数的基本功能是通过与用户交互，获得各窗口顾客数，然后通过队列这一数据结构最终输出结果：**

cout << "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:"<<endl;

int N = 0, CapacityOfA = 0, CapacityOfB = 0;

while (1) //增强健壮性，防止用户误操作使得程序崩溃

{

cin >> N;

if (N > 0)break;

cout << "输入有误！请输入正整数：";

}

int \*temp = new int[N]; //开辟temp动态数组，用来暂时存储所有数据

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

{

while (1) //增强健壮性，防止用户误操作使得程序崩溃

{

cin >> temp[i];

if (temp[i] > 0)break;

cout << "输入有误！请输入正整数：";

}

if (temp[i] % 2 == 1) { CapacityOfA++; } //获得A的人数

if (temp[i] % 2 == 0) { CapacityOfB++; } //获得B的人数

}

Queue<int>A(CapacityOfA);

Queue<int>B(CapacityOfB);

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

{

if (temp[i] % 2 == 1) { A.EnQueue(temp[i]); }

if (temp[i] % 2 == 0) { B.EnQueue(temp[i]); }

}

int j = 1;

while (!A.IsEmpty() && !B.IsEmpty())

{

if (j % 3 == 1 || j % 3 == 2) { cout<<" "<<A.DeQueue(); }

else { cout <<" "<< B.DeQueue(); }

j++;

}

if (A.IsEmpty()) { B.print(); }

if (B.IsEmpty()) { A.print(); }

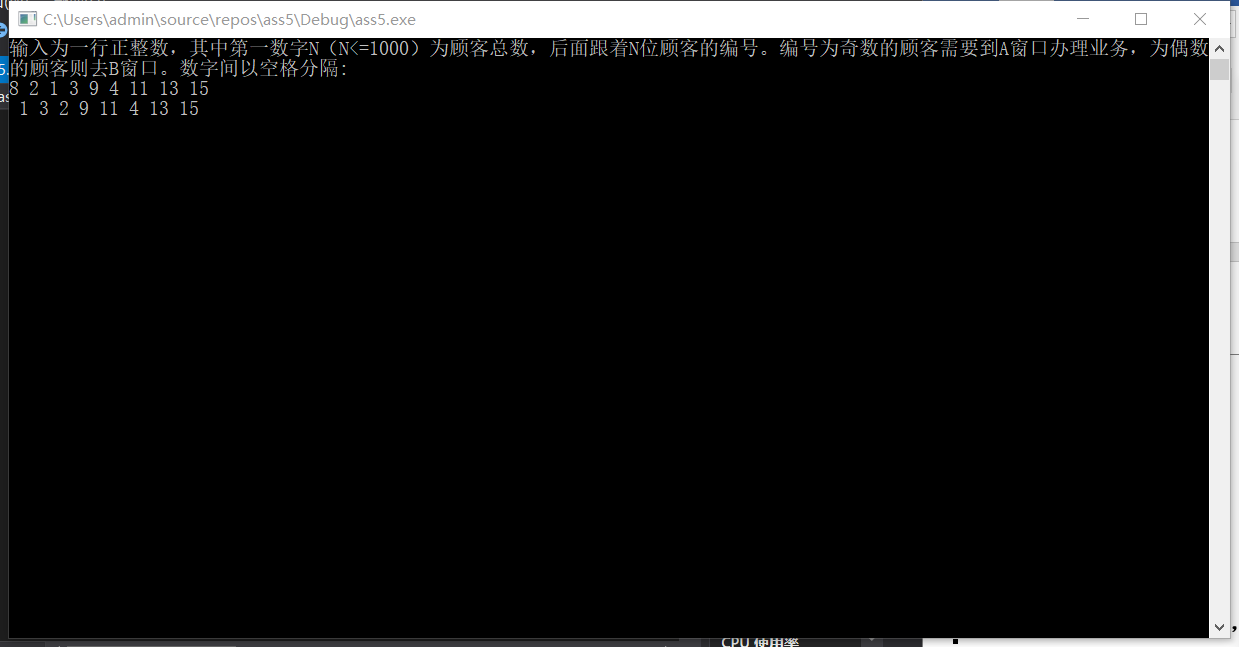
delete[] temp;

return 0;

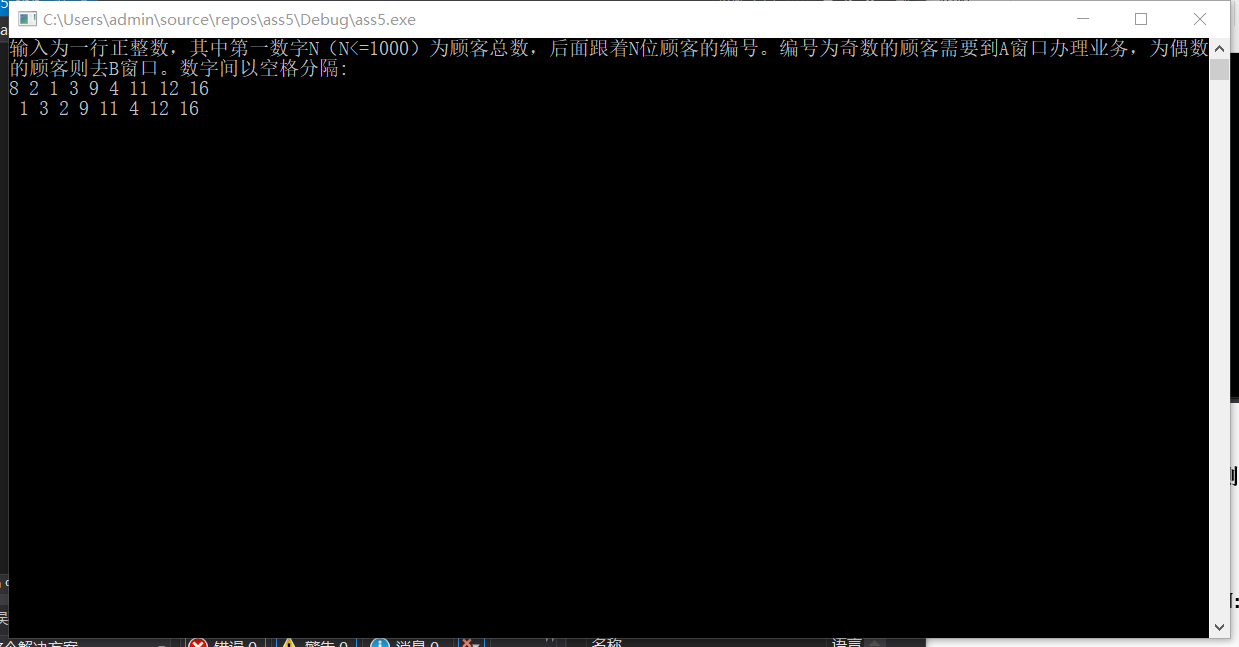
# ****测试****

## 功能性测试

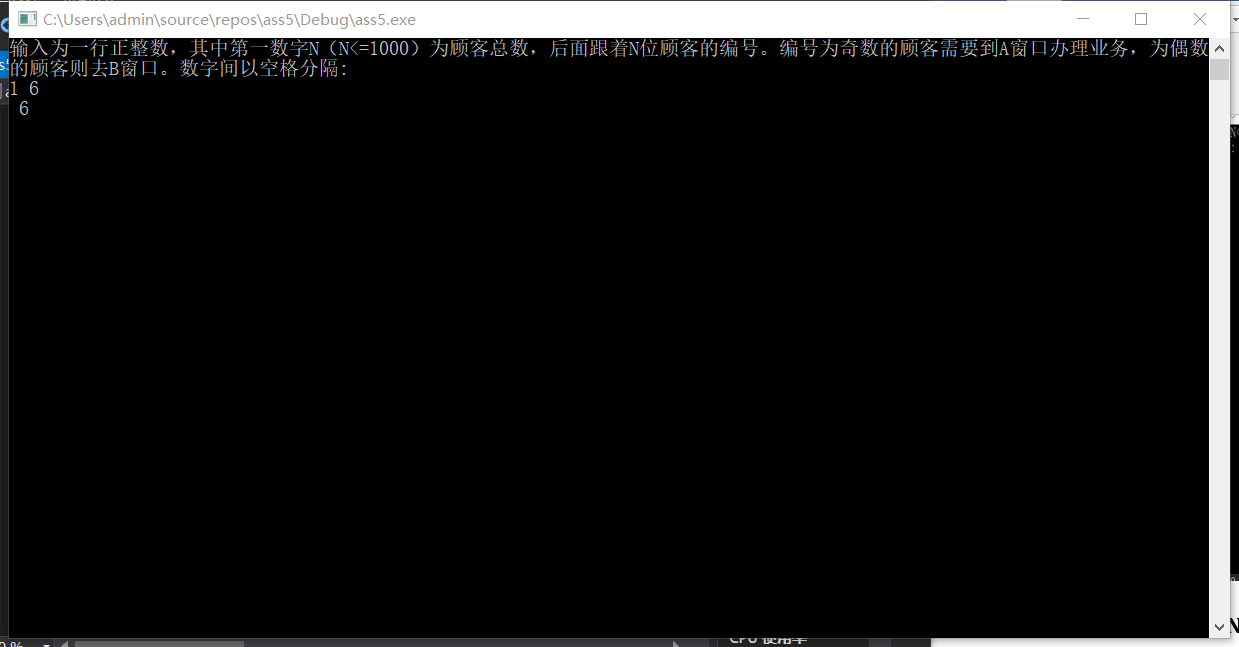
#### 正常测试，A窗口人多：



#### 正常测试，B窗口人多：



#### 最小N:



## 出错性测试

