同濟大學
TONGJI UNIVERSITY

项目说明文档

银行业务

指导老师：张颖

1851009 沈益立

目录

目录
1.分析
1.1 背景分析
1.2 功能分析
2.设计
2.1 数据结构设计
2.2 成员与操作设计
2.3 成员和操作设计
2.4 系统设计
3.实现
3.1 队列压入的实现
3.1.1 队列压入功能流程图
3.1.2 队列压入功能核心代码实现
3.2 队列弹出功能的实现
3.2.1 队列弹出功能流程图
3.2.2 队列弹出功能核心代码实现
3.3 得到数据功能的实现
3.3.1 得到数据功能流程图
3.3.2 得到数据核心代码实现
4.测试
4.1 功能测试
4.1.1 一般情况测试-A窗口人多
4.1.2 一般情况测试-B窗口人多
4.1.3 最小N
4.1.4 较大批量数据测试
4.1.5 错误人数输入

1.分析

1.1 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，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，输出的编号用空格隔开，且要注意到最后输出的编号后不得有空格。

1.2 功能分析

输入为一串正整数，其中第一个数字表示本次处理的总人数。要求实现的功能比较容易理解，即调整顾客之间的顺序，按照优先级的顺序调整重新输出。

2.设计

2.1 数据结构设计

由题意，队列这一数据结构本身就是对于排队这个过程的抽象化，因此很容易令人联想到用队列的方式来处理。队列先进先出的数据结构特点很匹配本题的相关要求。考虑到本次处理业务最多有1000名顾客，用静态的数组来存放顾客信息大部分情况下会比较浪费空间，因此我采用了链式队列作为储存顾客信息的数据结构。

本题未采用STL，而使用了自己设计的链式队列类`LinkedQueue<T>`。

链式队列一定会含有链表节点的元素，因此另外设计了一个数据结构`LNode<T>`作为储存节点的数据结构。

此外，为了加强代码的复用性和可读性，设计了一个`BankQueue`类作为答案求解的类，实现基本的IO和求解答案。

2.2 成员与操作设计

由于作业的相关规定，实现了自己定义的用数组模拟的可变长的栈`Stack<T>`，以及表示二元数对的`MyPair<T1, T2>`。为简化IO、增加代码复用性和美观性，同时设计了类`BankQueue`，用于输入、解决和输出结论。

2.3 成员和操作设计

链式队列类`LinkedQueue<T>`

公有操作：

```
LinkedQueue();              //默认的构造函数
~LinkedQueue();             //析构函数，执行所有链表节点内存的释放
void pushBack(T& val);      //推入元素
T popFront();               //弹出元素
T getFront();               //得到队首元素的值
bool isEmpty();             //返回该队列是否为空的布尔值
void makeEmpty();           //清空队列，释放内存
void printQueue();          //将队列的值输出到屏幕上
```

私有成员：

```
LNode<T>* m_rear = nullptr; //队尾指针，默认为空
LNode<T>* m_front = nullptr;//队头指针，默认为空
```

链表节点类`LNode<T>`

公有操作：

```
LNode() {};                          //默认无参数构造函数
LNode(int val, LNode<T>* next)   //带值和下一个指针域的构造函数
        :m_val(val), m_next(next) {};
```

公有成员：

```
T m_val;                    //节点值
LNode<T>* m_next;           //下一个指针域
```

银行队列类BankQueue

公有操作：

```
void getQueueData();     //通过IO实现队列的初始化
void solveByPrior();     //解答，将顾客按次序输出
```

私有成员：

```
int m_totalNum;          //顾客总人数
int m_failToSolve = 0;   //是否能正确
LinkedQueue<int> m_queueA;//A窗口的模拟队列
LinkedQueue<int> m_queueB;//B窗口的模拟队列
LinkedQueue<int> m_resQueue;//最后存放结论的容器
```

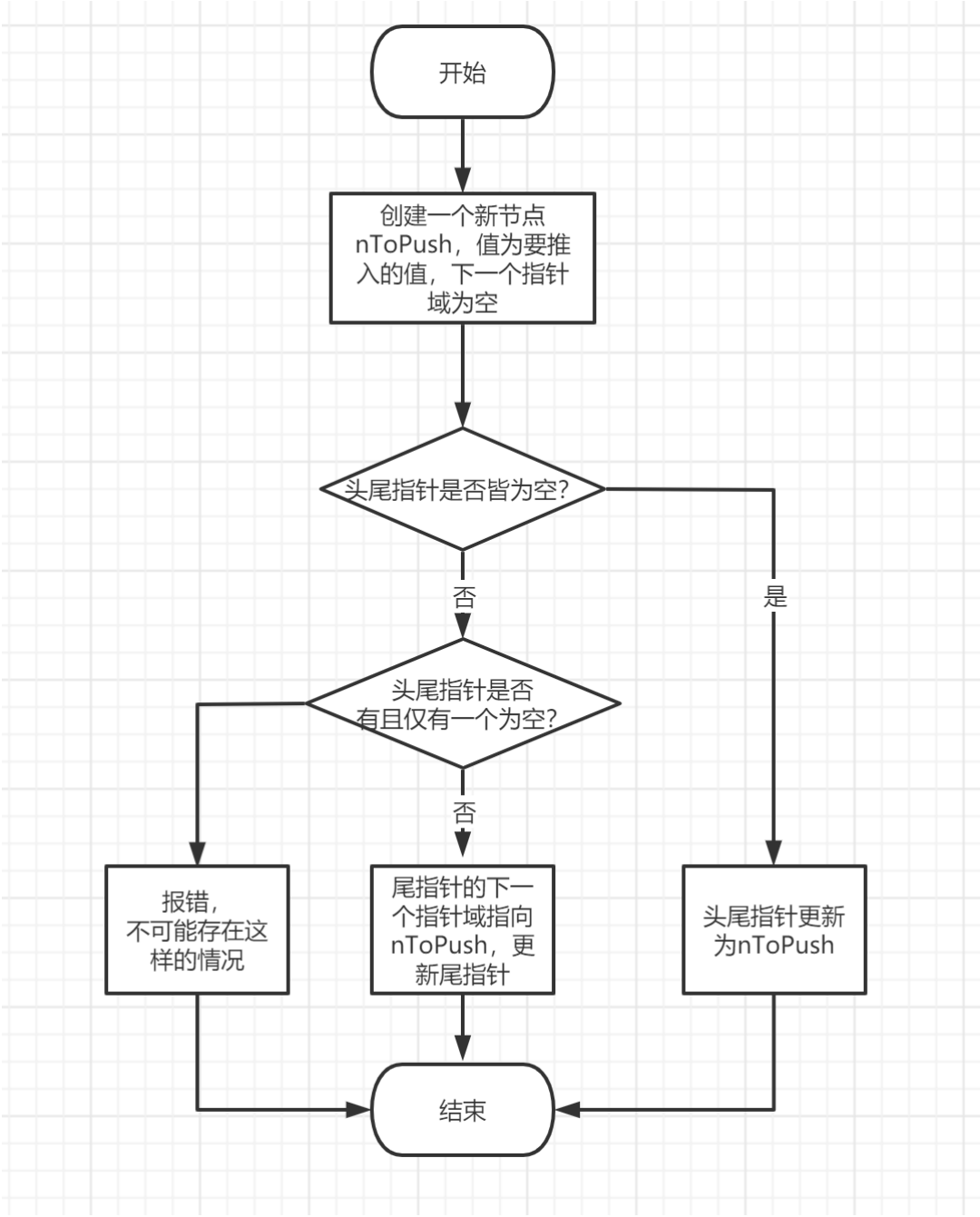
2.4 系统设计

程序运行后，系统会自动创建一个`BankQueue`的实体化对象，随后调用`getQueueData()`用一系列基础的IO来实现对于顾客人数和顾客序列的录入，通过顾客编号奇偶性的判断将他们推入`m_queueA`和`m_queueB`中。随后，通过调用`solveByPrior()`方法，实现对他们的重排，压入`m_resQueue`中，并将最终结果的编号序列输出到屏幕上。

3.实现

3.1 队列压入的实现

3.1.1 队列压入功能流程图



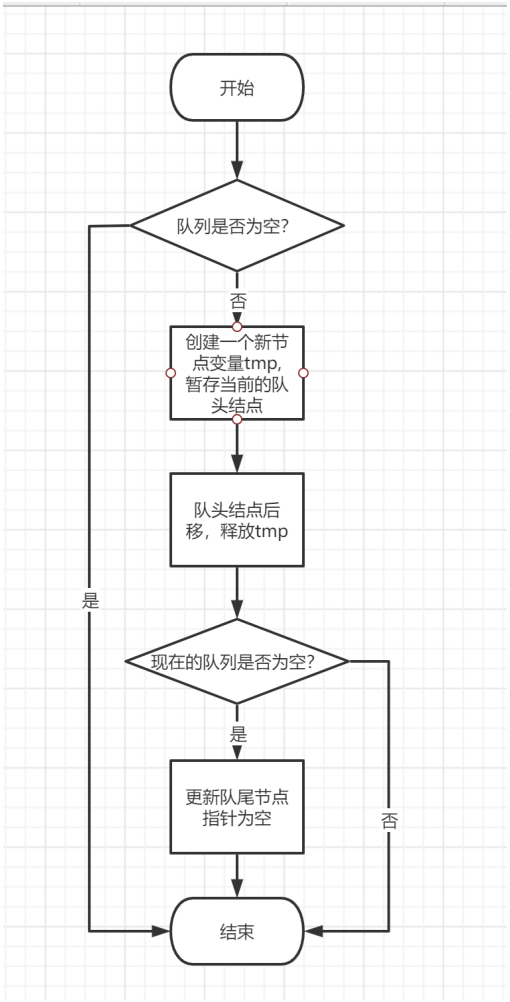
3.1.2 队列压入功能核心代码实现

```
template<class T>
void LinkedQueue<T>::pushBack(T& val) {
    LNode<T>* nToPush = new LNode<T>(val, nullptr);
```

```
    if (!m_rear && !m_front) {//空集判定，若空则将头尾指针都设为其值
        m_rear = nToPush;
        m_front = nToPush;
        return;
    }
    if (!m_rear || !m_front) {
        cerr << "指针错误" << endl;
        return;
    }
    m_rear->m_next = nToPush;
    m_rear = nToPush;
    return;
}
```

3.2 队列弹出功能的实现

3.2.1 队列弹出功能流程图

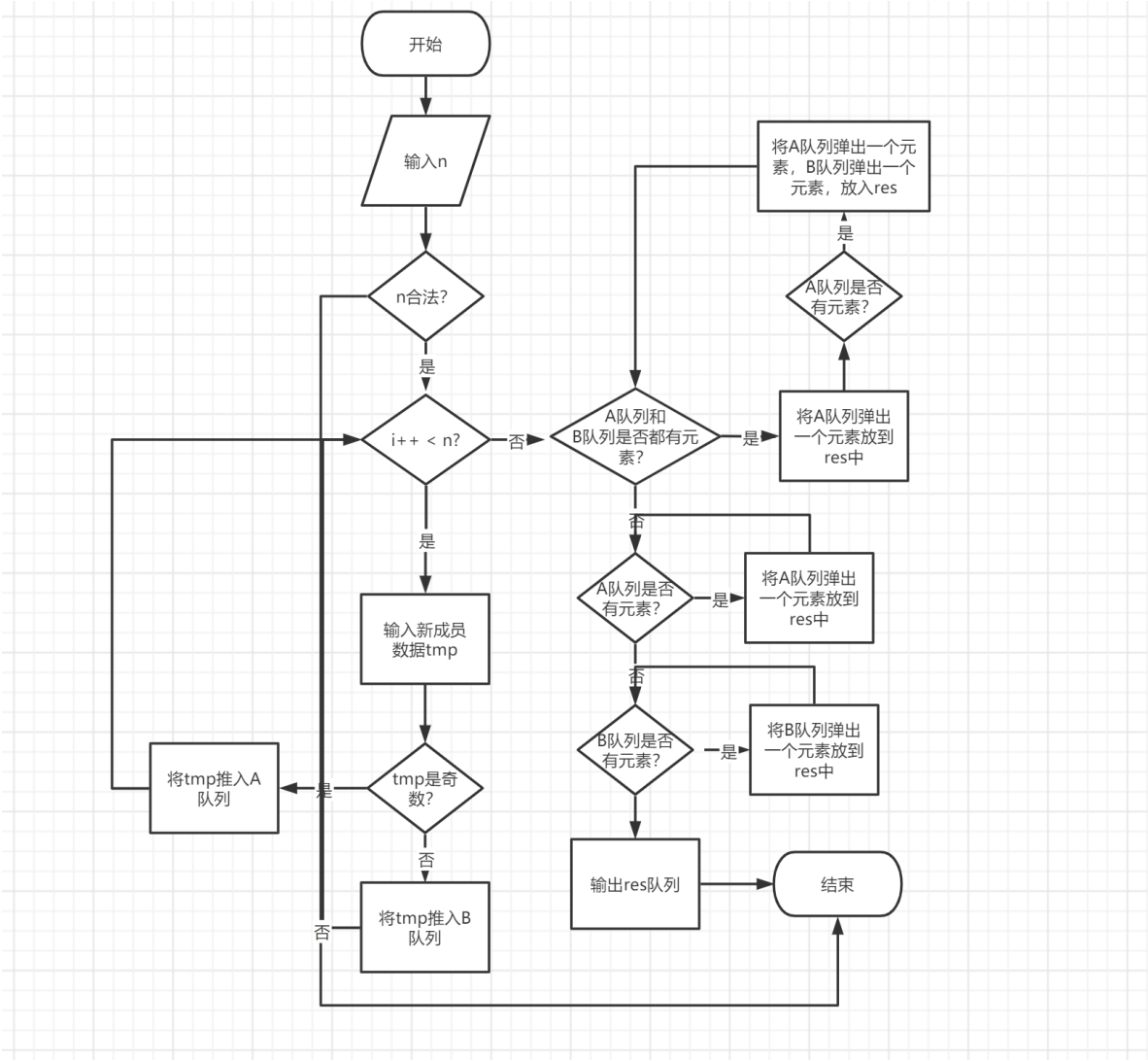


3.2.2 队列弹出功能核心代码实现

```
template<class T>
T LinkedQueue<T>::popFront(){
    if (isEmpty()) {
        cerr << "空队列，不可弹出" << endl;
        return T();
    }
    LNode<T>* tmp = m_front;
    T retVal = tmp->m_val;
    m_front = m_front->m_next;
    if (m_front == nullptr) {//空集情况判定
        m_rear = nullptr;
    }
    delete tmp;//Tips: 内存泄漏
    return retVal;
}
```

3.3 得到数据功能的实现

3.3.1 得到数据功能流程图



3.3.2 得到数据核心代码实现

```
void BankQueue::getQueueData() {
    cout << "请输入一个[1, 1000]内的总人数N并且输入其顺序: " <<
endl;
    cin >> m_totalNum;
    if (m_totalNum <= 0 || m_totalNum > 1000) {
        cerr << "总人数有问题" << endl;
        m_failToSolve = 1;
        return;
    }
    for (int i = 0; i < m_totalNum; i++) {
        int tmp;
        cin >> tmp;
        if (tmp % 2) {
            m_queueA.pushBack(tmp);
        }
```

```
        else {
            m_queueB.pushBack(tmp);
        }
    }
}

void BankQueue::solveByPrior() {
    if (m_failToSolve) {
        return;
    }
    while (!m_queueA.isEmpty() && !m_queueB.isEmpty()) {
        int tmp = m_queueA.popFront();
        m_resQueue.pushBack(tmp);
        if (m_queueA.isEmpty()) {
            break;
        }
        tmp = m_queueA.popFront();
        m_resQueue.pushBack(tmp);
        tmp = m_queueB.popFront();
        m_resQueue.pushBack(tmp);
    }
    while (!m_queueA.isEmpty()) {
        int tmp = m_queueA.popFront();
        m_resQueue.pushBack(tmp);
    }
    while (!m_queueB.isEmpty()) {
        int tmp = m_queueB.popFront();
        m_resQueue.pushBack(tmp);
    }
    cout << "顾客办理业务最终的顺序为：" << endl;
    m_resQueue.printQueue();
}
```

4.测试

4.1 功能测试

4.1.1 一般情况测试-A窗口人多

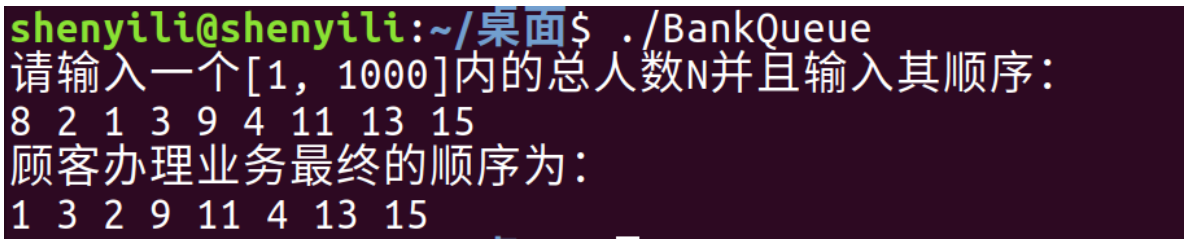
测试用例：

```
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：



4.1.2 一般情况测试-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：

```
shenyili@shenyili:~/桌面$ ./BankQueue
请输入一个[1, 1000]内的总人数N并且输入其顺序:
8 2 1 3 9 4 11 12 16
顾客办理业务最终的顺序为:
1 3 2 9 11 4 12 16
```

4.1.3 最小N

测试用例:

```
1 6
```

预期结果:

```
6
```

实验结果:

```
shenyili@shenyili:~/桌面$ ./BankQueue
请输入一个[1, 1000]内的总人数N并且输入其顺序:
1 6
顾客办理业务最终的顺序为:
6
```

4.1.4 较大批量数据测试

测试用例:

```
30 1 3 5 7 9 12 34 55 43 32 11 33 67 88 76 75 74 80 99 13
53 54 60 31 36 37 97 98 96 87
```

预期结果:

```
1 3 12 5 7 34 9 55 32 43 11 88 33 67 76 75 99 74 13 53 80
31 37 54 97 87 60 36 98 96
```

实验结果:

```
shenyili@shenyili:~/桌面$ ./BankQueue
请输入一个[1, 1000]内的总人数N并且输入其顺序:
30 1 3 5 7 9 12 34 55 43 32 11 33 67 88 76 75 74 80 99 13 53 54 60 31 36 37 97 9
8 96 87
顾客办理业务最终的顺序为:
1 3 12 5 7 34 9 55 32 43 11 88 33 67 76 75 99 74 13 53 80 31 37 54 97 87 60 36 9
8 96
```

4.1.5 错误人数输入

测试用例:

```
-1
```

预期结果:

```
提示总人数有误
```

实验结果:

```
shenyili@shenyili:~/桌面$ ./BankQueue
请输入一个[1, 1000]内的总人数N并且输入其顺序:

-1
总人数有问题
```