数据结构课程设计项目说明文档

——银行业务

作 者 姓 名： 马威

学 号： 2151294

指 导 教 师： 张颖

学院、 专业： 软件学院 软件工程

****

目 录

[1 项目分析 1](#_Toc495668153)

[1.1 项目背景 1](#_Toc495668154)

[1.2 项目要求 1](#_Toc495668154)

[1.3 项目需求分析 1](#_Toc495668155)

[2 项目设计 3](#_Toc495668156)

[2.1 数据结构设计 3](#_Toc495668157)

[2.2 类设计 3](#_Toc495668158)

2.2.1 结点类（QueueNode） 3

2.2.2 链式栈类（LinkedQueue） 3

2.3 算法设计 5

2.3.1 算法思路 5

2.3.2 性能评估 5

2.3.3 准备工作部分 6

·流程图表示 6

·代码实现 6

2.3.4 求解序列部分 8

·流程图表示 8

·代码实现 8

[3 项目测试 1](#_Toc495668161)0

[3.1 正常测试，A窗口人多 1](#_Toc495668162)0

[3.2 正常测试，B窗口人多 1](#_Toc495668166)0

[3.3 正常测试，两窗口人数一样多 1](#_Toc495668170)0

[3.4 最小N 1](#_Toc495668174)1

[3.5 顾客人数输入错误的情况 1](#_Toc495668178)1

[3.6 顾客编号输入错误的情况 1](#_Toc495668182)1

**1.项目分析**

1.1 项目背景

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个顾客时，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。

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。

测试用例：

|  |  |  |  |
| --- | --- | --- | --- |
| 序号 | 输入 | 输出 | 说明 |
| 1 | 8 2 1 3 9 4 11 13 15 | 1 3 2 9 11 4 13 15 | 正常测试，A窗口人多 |
| 2 | 8 2 1 3 9 4 11 12 16 | 1 3 2 9 11 4 12 16 | 正常测试，B窗口人多 |
| 3 | 1 6 | 6 | 最小N |

1.3 项目需求分析

对于完成银行业务问题的程序，需考虑以下需求：

**·正确性**

程序应当能够按照要求将顾客序列输出，没有遗漏。每位顾客编号之间要用空格隔开，末尾不能有多余空格。

**·健壮性**

程序应当对顾客人数、顾客编号的错误输入进行处理。

**2.项目设计**

2.1 数据结构设计

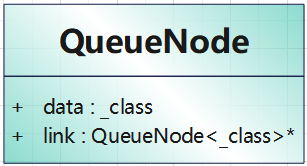
由项目分析可以得出，无论顾客是编号为奇数还是偶数，先到的顾客会先处理自己的业务，在程序中表现为编号被打印出来，这与日常生活中排队的逻辑是一致的：先到先得。因此，该项目需要一种像排队一样先进先出的数据结构，而队列可以满足这样的需求。

2.2 类设计

由项目分析可以得出，本项目需要用到一个队列，其本质上通过链表实现，其包括两个抽象数据类型（ADT）——链表结点类（ListNode）与链表类（List），而两个类之间的耦合关系可以采用嵌套、继承等多种关系。为了实现代码的复用性，本系统实现了一个链表为形式的栈。采用struct描述链表结点类（QueueNode），这样使得链式栈类（LinkedQueue）可以直接访问链表结点而不需要定义友元关系。

**2.2.1 结点类（QueueNode）**

链表结点存储了结点数据、后继节点位置，其UML图如下：

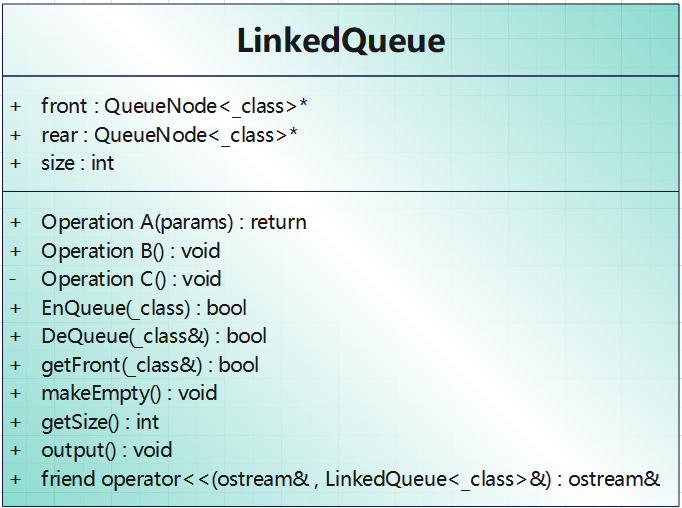


**2.2.2链式队列类（LinkedQueue）**

链式队列类本质上仍是单链表，保存了链表的基本结构。由于我们在使用队列时仅仅利用其“先进先出”的特性，所以其提供的操作并不像一般链表操作那样齐全，而是仅仅有入队和出队。

由于对队列的插入、删除元素等操作的位置比较固定，为了节省空间，并未采用附加头结点。

其UML图如下：



其中，主要函数如下：

//入队

bool EnQueue(const \_class x);

//出队

bool DeQueue(\_class& x);

//取队列第一个元素

bool getFront(\_class& x)const;

//清空队列

void makeEmpty();

//判断队列是否为空

bool IsEmpty()const;

//取队列中元素个数

int getSize()const;

//输出队列中所有元素

void output()const;

2.3 算法设计

**2.3.1 算法思路**

首先定义两个队列，表示要去到A窗口的人和要去到B窗口的人，在输入顾客序号的同时即可根据编号的奇偶将顾客分配到不同的窗口。

分配完成后即可开始处理业务（即队列开始出队）。可以按每个周期依次循环来处理，在这样一个周期中，A口最多出2名顾客，B口最多出1名顾客。由于同时处理A口先出，故先对A口处理，再对B口处理。

{

先考虑A口情况：

（1）A口人数大于等于2，直接出2人

（2）A口人数等于1，直接出1人

（3）A口已空，不做任何操作

再考虑B口情况：

（4）B口不为空，直接出1人

（5）B口已空，不做任何操作

}

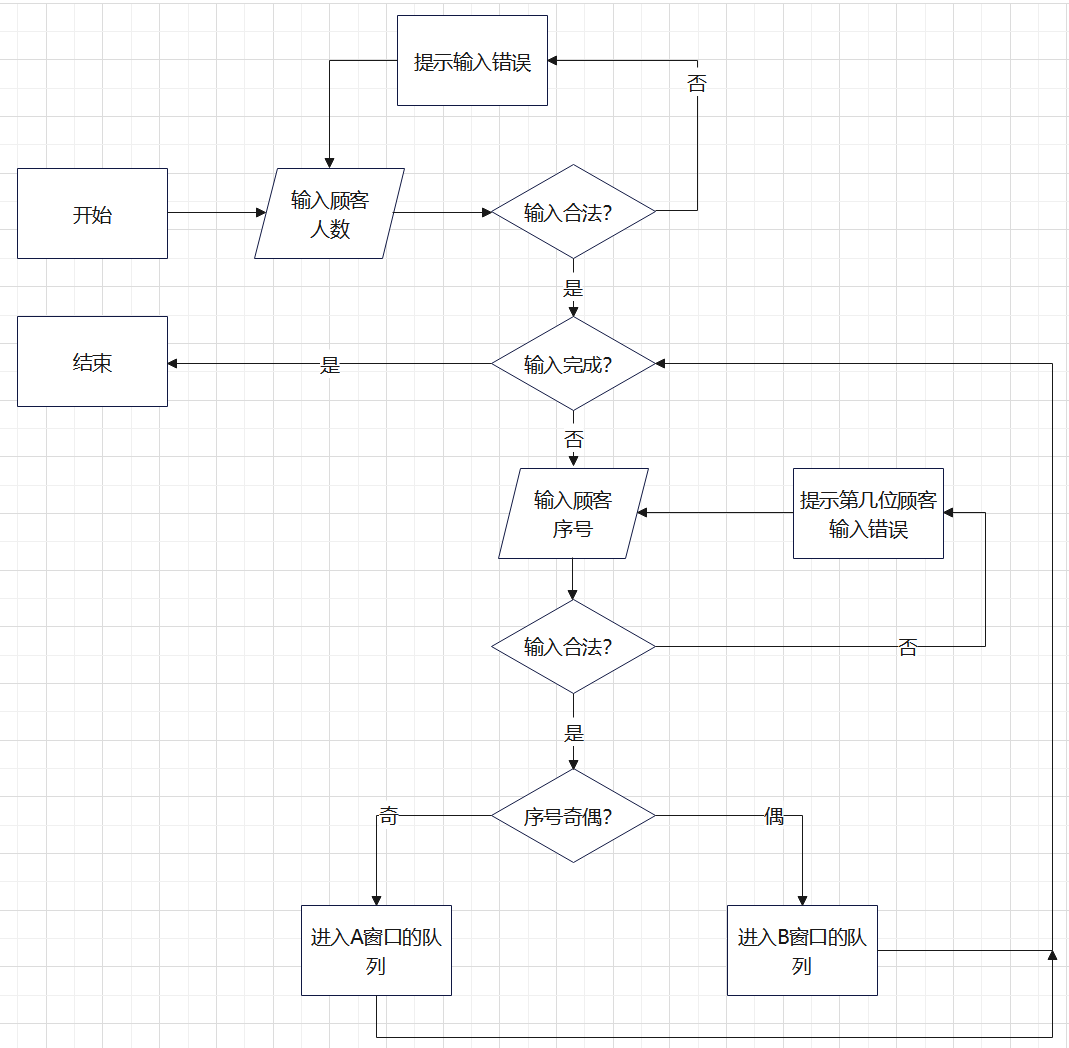
继续以上循环直至两个队列均为空，所求的顾客序列也就自然得到了。

**2.3.2 性能评估**

设输入总人数为n，所有顾客入队出队操作各1次，故总的时间复杂度为O(n)。

**2.3.3 准备工作部分**

·流程图表示

****

·代码实现

LinkedQueue<int> A, B;  /\*A口队列、B口队列\*/

    int total, num;   /\*顾客总数、顾客编号\*/

    int customer[3];  /\*三人一组\*/

    /\*1、输入顾客总人数\*/

    while (1) {

        cout << "请输入顾客总人数以及顾客编号，每个数字间用空格分开" << endl;

        cin >> total;

        if (cin.good() && total > 0 && total <= MAX)

            break;

        cin.clear();

        cin.ignore(INT\_MAX, '\n');

        cout << "总人数输入错误，请重新输入！" << endl;

    }

    /\*2、输入顾客编号\*/

    for (int i = 0; i < total; i++) {

        bool if\_enqueue = false;

        while (1) {

            cin >> num;

            if (cin.good() && num > 0 && num <= INT\_MAX) {

                if\_enqueue = true;

                break;

            }

            cin.clear();

            cin.ignore(INT\_MAX, '\n');

            cout << "第" << i + 1 << "位顾客编号输入错误，请重新输入！" << endl;

        }

        if (if\_enqueue) {  /\*根据编号的奇偶进入不同的队列\*/

            if (num % 2 == 0)

                B.EnQueue(num);

            else

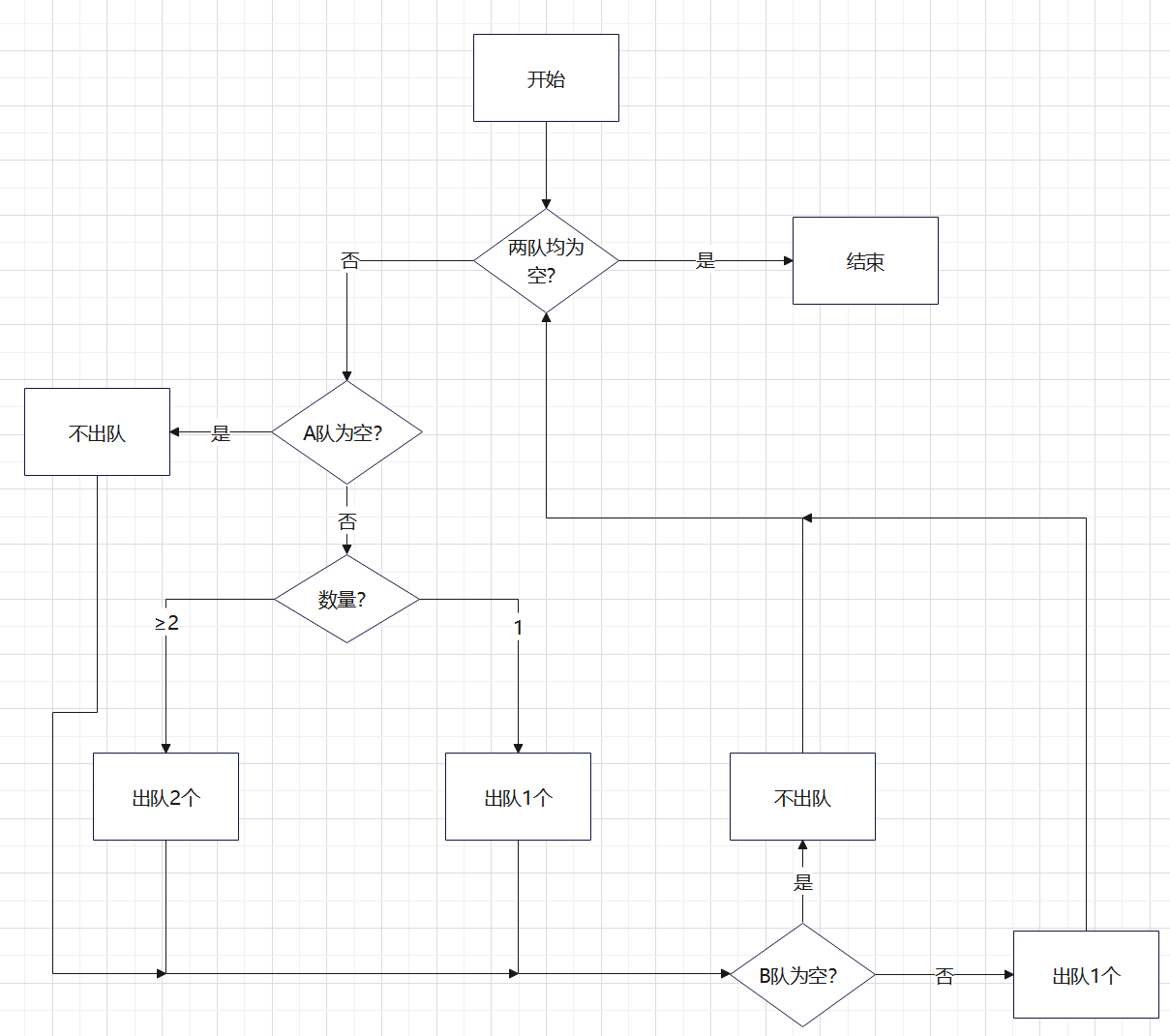
                A.EnQueue(num);

        }

    }

**2.3.4 求解序列部分**

·流程图表示



·代码实现

/\*3、按要求出队列\*/

    while (!A.IsEmpty() || !B.IsEmpty()) {

        if (A.getSize() >= 2) {  /\*若A队数量大于2，则直接出队2个\*/

            A.DeQueue(customer[0]);

            A.DeQueue(customer[1]);

            if (A.IsEmpty() && B.IsEmpty())

                cout << customer[0] << ' ' << customer[1];

            else

                cout << customer[0] << ' ' << customer[1] << ' ';

        }

        else if (A.getSize() == 1) {  /\*若A队数量为1，则直接出队1个\*/

            A.DeQueue(customer[0]);

            if (A.IsEmpty() && B.IsEmpty())

                cout << customer[0];

            else

                cout << customer[0] << ' ';

        }

        if (B.getSize()) {  /\*若B队不为空，则出队1个\*/

            B.DeQueue(customer[0]);

            if (A.IsEmpty() && B.IsEmpty())

                cout << customer[0];

            else

                cout << customer[0] << ' ';

        }

    }

    cout << endl;

    return 0;

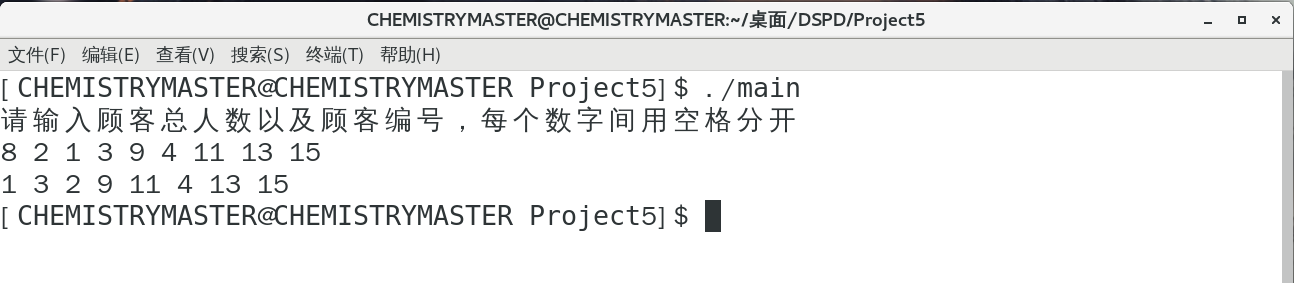
**3.项目测试**

3.1 正常测试，A窗口人多

输入内容：8 2 1 3 9 4 11 13 15

预期结果：1 3 2 9 11 4 13 15

测试结果：

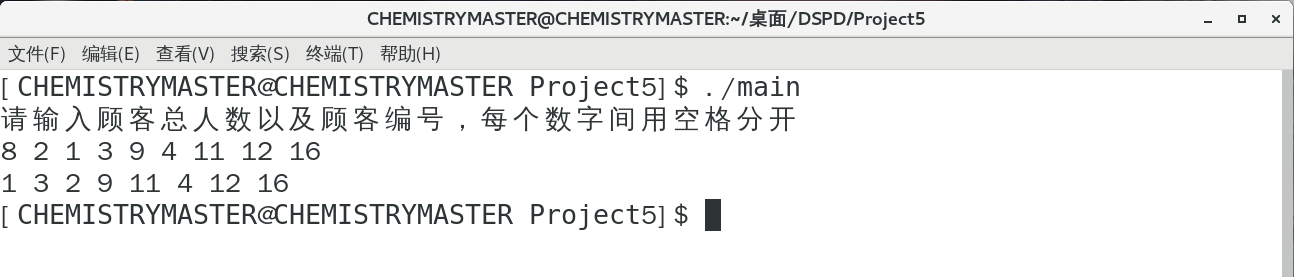


3.2 正常测试，B窗口人多

输入内容：8 2 1 3 9 4 11 12 16

预期结果：1 3 2 9 11 4 12 16

测试结果：

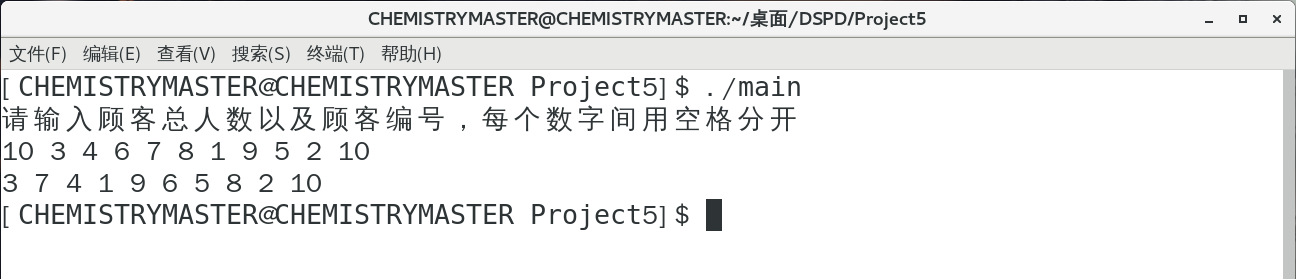


3.3 正常测试，两窗口人数一样多

输入内容：10 3 4 6 7 8 1 9 5 2 10

预期结果：3 7 4 1 9 6 5 8 2 10

测试结果：

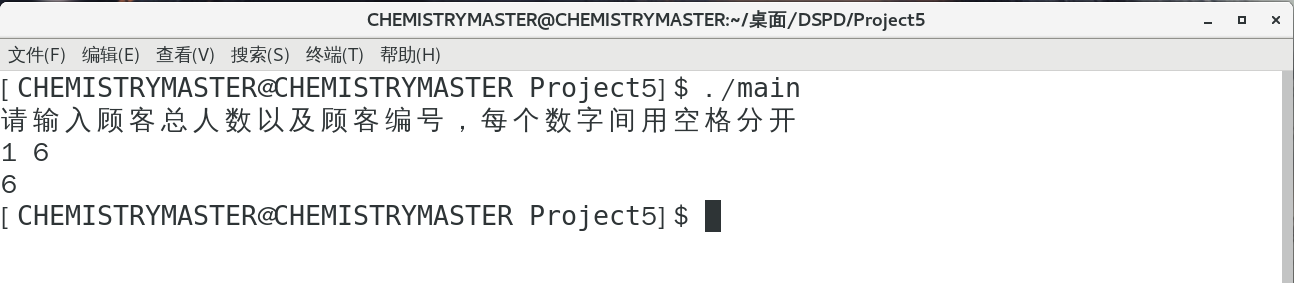


3.4 最小N

输入内容：1 6

预期结果：6

测试结果：

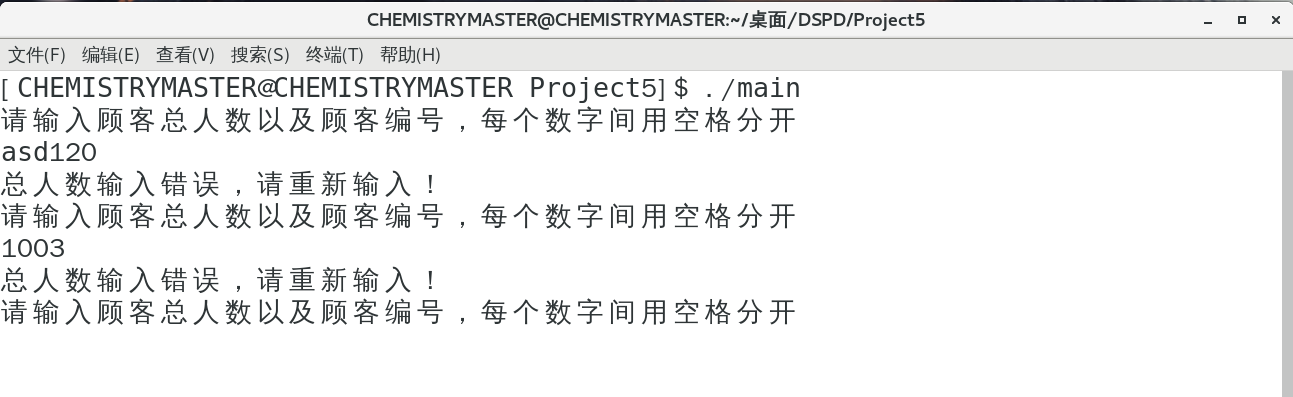


3.5 顾客人数输入错误的情况

输入内容：asd120 1003

预期结果：（提示错误，直至输入正确为止）

测试结果：



3.6 顾客编号输入错误的情况

输入内容：10 1 3 5 7 9 a 4 6 8 10

预期结果：（提示第几位顾客编号输入错误，需从该顾客开始继续输入）

测试结果：

