# 1 需求

## 项目背景

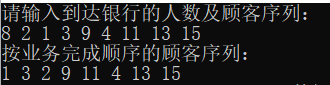
在现实的银行业务办理中，银行会根据业务办理的时长分设各种不同的窗口办理对应业务。现假定不考虑顾客信后到达的时间间隔，对现实问题建模为：设某银行有A，B两个业务窗口，且处理业务的速度不一样，其中A窗口处理速度是B窗口的2倍----即当A窗口每处理完2个顾客是，B窗口处理完1个顾客。给定到达银行的顾客序列，按照业务完成的顺序输出顾客序列。假定不考虑顾客信后到达的时间间隔，并且当不同窗口同时处理完2个顾客时，A窗口的顾客优先输出。

本项目根据如上假设，拟开发一个业务办理模拟系统，用户输入到达银行的顾客次序，系统输出相应的业务完成顺序，以此模拟整个业务办理的流程。当存在多个业务办理窗口且人数较多时，本程序可以很好地模拟人流情况，帮助银行模拟出最优策略，高效完成工作。

1. 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。
2. 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。

## 1.2 功能分析

本项目核心功能主要为读取用户输入的序列并进行错误处理、根据序列将顾客序号压入窗口A和窗口B对应的队列、根据窗口A与窗口B的处理业务速度依次弹出队列元素并打印结果。以上功能在程序中体现为元素入队列、元素出队列等，其核心是建立队列的数据结构并完成数据的存储和读取。

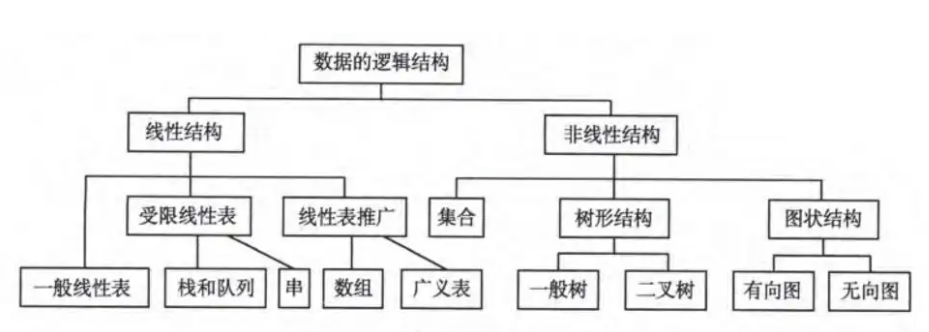


# 2 设计

## 2.1 数据结构设计

### 2.1.1 逻辑结构设计

本项目的核心功能是对顾客办理业务顺序进行模拟。由于每个顾客的信息相对独立，各个元素之间不存在明显的偏序关系或层次关系，也不存在“一对多”与“多对一”的映射，所以可以采用线性结构进行存储。



从1.2功能分析可知顾客办理业务顺序模拟对元素的读取和插入是满足“先进先出”规则的，即队列的“First In First Out”形式，因此可以建立受限线性表队列对顾客序列进行存储。而由于通常的队列实现在入队出队过程中容易造成空间的浪费，且有超出存储范围的风险，本项目在建立队列时采用了循环队列作为主要逻辑结构。

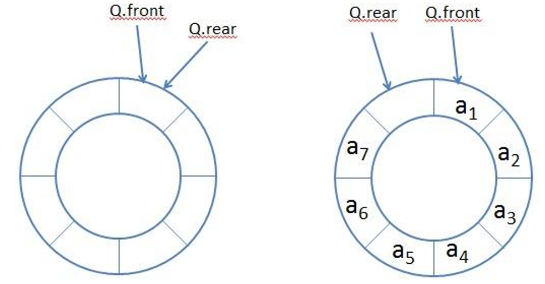
### 2.1.2 存储结构设计

本项目对序列元素的操作主要为插入与删除，每次操作位置固定，在队首读取数据，在队尾插入数据，若使用顺序存储方式实现队列的push与pop操作时间复杂度均为O(1)，且顺序存储的空间开销较小，故本项目选用顺序存储方式。

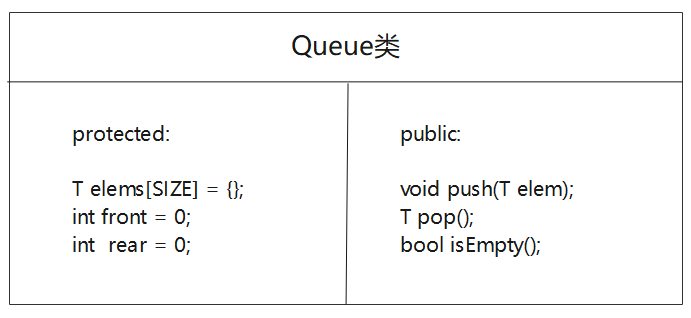
## 2.2 类设计

本项目共设计了一个队列类Queue，其protected元素有存储数组elems、队首指针front、队尾指针rear。Queue的public属性主要用于存放操作，包括元素入队列、元素出队列、判断队列是否为空等。

和通常的队列实现不同的是，循环队列将顺序队列臆造成一个环状的空间，即把存储队列元素的[顺序表](https://so.csdn.net/so/search?q=%E9%A1%BA%E5%BA%8F%E8%A1%A8&spm=1001.2101.3001.7020" \t "https://blog.csdn.net/cnds123321/article/details/_blank)从逻辑上视为一个环，但实际仍为线性存储。当队头指针 queue.front==MAXSIZE-1，即到数组的最后一个位置时，再前进一个位置就自动到 0，这里通过除法取余运算实现。



Queue类图示如下：



Queue类定义代码如下：

//循环队列实现

**template** <**typename** T>

**class** Queue {

**protected**:

    T elems[SIZE] = {};    //存储数组

**int** front = 0, rear = 0;    //队首指针与队尾指针

**public**:

    //元素入队列

**void** push(T elem) {

        elems[rear] = elem;

        rear = (rear + 1) % SIZE;

    }

    //元素出队列

    T pop() {

**int** temp = front;

        front = (front + 1) % SIZE;

**return** elems[temp];

    }

    //判断队列是否为空

**bool** isEmpty() {

**if** (rear == front)

**return** **true**;

**return** **false**;

    }

};

## 2.3 函数设计

本项目主要定义了两个函数，一个是输入函数input用于输入顾客总数及顾客编号，同时对用户的输入进行错误处理；一个是输出函数output,将之前求出的结果序列依次输出。

**void** input(Queue<**int**> windowA, Queue<**int**> windowB,**int** &step,**int** result[]);

void output(int step,int result[]);

程序对于队列的操作主要封装于input函数中，其先定义了两个存放int类型元素的队列windowA、windowB，将用户输入的顾客序列按奇偶交替压入两个队列。之后通过while循环不断弹出两个队列中的元素并将结果储存在result数组中。循环过程通过int类型变量p计数，当p%3!=2时windowA弹出元素存入result中，当p%3==2时windowB弹出元素存入result中，该过程反复进行直至其中一个队列为空，最后将存在剩余的队列所有元素依次存入结果数组。

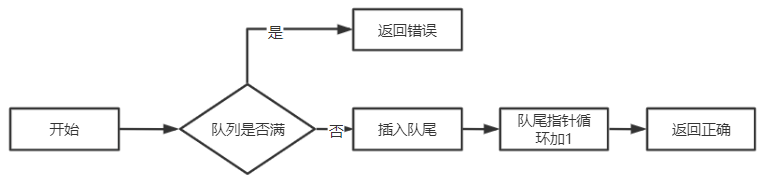
# 3 实现

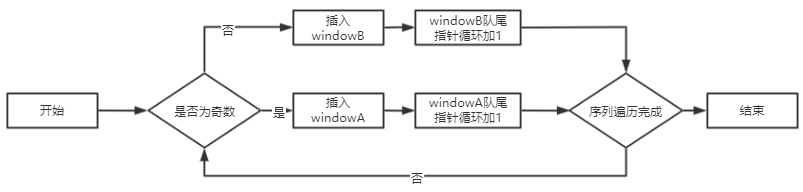
## 3.1 入队列过程的实现

### 3.1.1 入队列过程实现思路

入队列功能封装在队列类Queue的push函数中，程序读取待入队列数，将其赋值给存储数组的栈顶元素，同时队尾指针加1并对maxSize取余。在模拟顾客办理业务过程中，假定奇数编号顾客在A窗口办理，偶数顾客在B窗口办理，则应按照奇偶顺序交替让顾客序列压入两个循环队列windowA、windowB，方便之后出队列过程的模拟。

### 3.1.2 入队列过程实现流程图





### 3.1.3 入队列过程实现代码

cout << "请输入到达银行的人数及顾客序列：" << endl;

//输入与错误处理

**while** (1) {

    cin >> sum;

**if** (cin.fail()) {

        cout << "输入错误，请重新输入：";

        cin.clear();

**char** t;

**while** ((t = cin.get()) != '\n');

    }

**else** **if** (sum > 1000 || sum < 0) {

        cout << "顾客总数应在0-1000范围内，请重新输入：";

        cin.clear();

**char** t;

**while** ((t = cin.get()) != '\n');

    }

**else**

**break**;

}

**for** (**int** i = 0; i < sum; i++) {

**int** number;

**while** (1) {

        cin >> number;

**if** (cin.fail()) {

            cout << "输入错误，请重新输入整段序列：";

            cin.clear();

**char** t;

**while** ((t = cin.get()) != '\n');

            i = 0;

**while** (!windowA.isEmpty())

                windowA.pop();

**while** (!windowB.isEmpty())

                windowB.pop();

        }

**else**

**break**;

    }

**if** (number % 2 == 1)

        windowA.push(number);

**else**

        windowB.push(number);

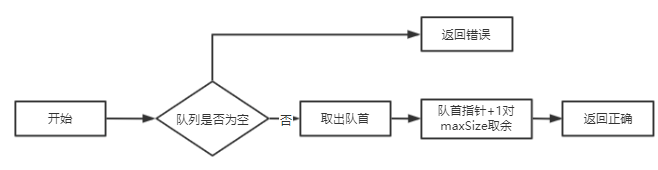
}

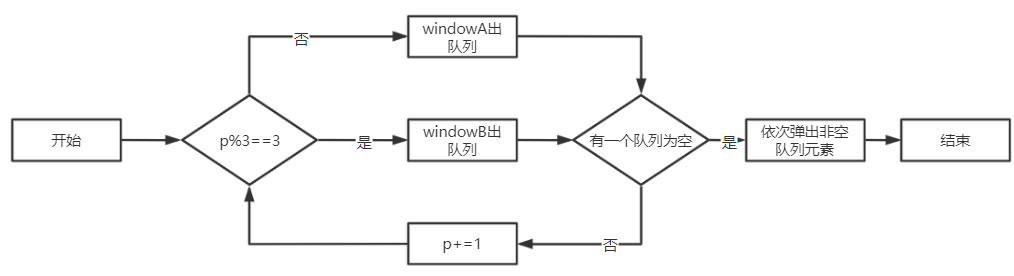
## 3.2 出队列过程的实现

### 3.2.1 出队列过程实现思路

出队列功能封装在队列类Queue的pop函数中，函数首先创建一个临时变量存储当前的队首指针，存储完成后队首指针加1并对maxSize取余，最后返回之前的队尾指针指向的队列元素。在模拟顾客办理业务过程中，假定不同窗口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。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。因此可通过循环模拟windowA、windowB出队列，通过int型变量p计数，p%3!=2时windowA弹出元素存入result数组中，p%3==2时windowB弹出元素存入result数组中，该过程反复进行直至其中一个队列为空，最后将存在剩余的队列所有元素依次存入result数组。result数组中存放的顾客顺序即为业务完成顺序。

### 3.2.2 出队列过程实现流程图





### 3.2.3 出队列过程实现代码

//按顺序输出队列元素

**while** (!windowA.isEmpty() && !windowB.isEmpty()) {

**if** (p % 3 != 2 && !windowA.isEmpty()) {

        result[step] = windowA.pop();

        step += 1;

        p += 1;

    }

**if** (p % 3 == 2 && !windowB.isEmpty()) {

        result[step] = windowB.pop();

        step += 1;

        p += 1;

    }

}

//如果队列有剩余直接输出

**while** (!windowA.isEmpty()) {

    result[step] = windowA.pop();

    step += 1;

}

**while** (!windowB.isEmpty()) {

    result[step] = windowB.pop();

    step += 1;

}

## 3.3 input函数与output函数完整代码

//输入顾客总数及顾客编号

**void** input(Queue<**int**> windowA, Queue<**int**> windowB,**int** &step,**int** result[]) {

**int** sum;

    cout << "请输入到达银行的人数及顾客序列：" << endl;

    //输入与错误处理

**while** (1) {

        cin >> sum;

**if** (cin.fail()) {

            cout << "输入错误，请重新输入：";

            cin.clear();

**char** t;

**while** ((t = cin.get()) != '\n');

        }

**else** **if** (sum > 1000 || sum < 0) {

            cout << "顾客总数应在0-1000范围内，请重新输入：";

            cin.clear();

**char** t;

**while** ((t = cin.get()) != '\n');

        }

**else**

**break**;

    }

**for** (**int** i = 0; i < sum; i++) {

**int** number;

**while** (1) {

            cin >> number;

**if** (cin.fail()) {

                cout << "输入错误，请重新输入整段序列：";

                cin.clear();

**char** t;

**while** ((t = cin.get()) != '\n');

                i = 0;

**while** (!windowA.isEmpty())

                    windowA.pop();

**while** (!windowB.isEmpty())

                    windowB.pop();

            }

**else**

**break**;

        }

**if** (number % 2 == 1)

            windowA.push(number);

**else**

            windowB.push(number);

    }

**int** p = 0;

    //按顺序输出队列元素

**while** (!windowA.isEmpty() && !windowB.isEmpty()) {

**if** (p % 3 != 2 && !windowA.isEmpty()) {

            result[step] = windowA.pop();

            step += 1;

            p += 1;

        }

**if** (p % 3 == 2 && !windowB.isEmpty()) {

            result[step] = windowB.pop();

            step += 1;

            p += 1;

        }

    }

    //如果队列有剩余直接输出

**while** (!windowA.isEmpty()) {

        result[step] = windowA.pop();

        step += 1;

    }

**while** (!windowB.isEmpty()) {

        result[step] = windowB.pop();

        step += 1;

    }

}

//输出业务完成顺序

**void** output(**int** step,**int** result[]) {

    cout << "按业务完成顺序的顾客序列：" << endl;

**for** (**int** i = 0; i < step; i++) {

        cout << result[i];

**if** (i != step - 1)

            cout << " ";

    }

}

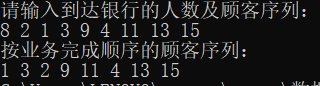
# 4 测试

## 4.1 正常测试，A窗口人多

输入序列：8 2 1 3 9 4 11 13 15

预取输出：1 3 2 9 11 4 13 15

实际结果：

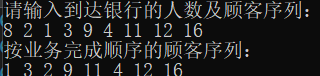


## 4.2 正常测试，B窗口人多

输入序列：8 2 1 3 9 4 11 12 16

预取输出：1 3 2 9 11 4 12 16

实际结果：

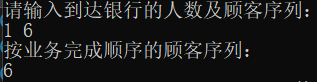


## 4.3 最小N

输入序列：1 6

预取输出：6

实际结果：

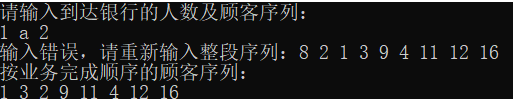


## 4.4 错误测试—输入非法

输入序列：1 a 2

预取输出：报错

实际结果：



## 4.5 错误测试—输入人数过多

输入序列：10000 1 2 3

预取输出：报错

实际结果：

