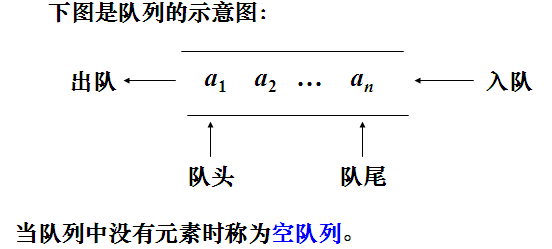
循环队列的实现

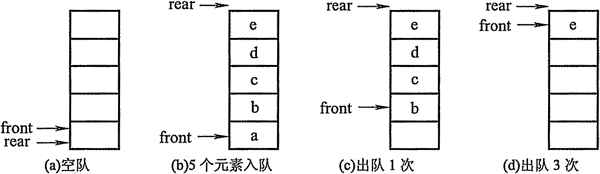
生活中有很多队列的案例，比如打饭排队，买火车票排队问题等，可以说与时间或顺序相关的问题，一般都会涉及到队列问题；从生活中，可以抽象出队列的概念，队列就是一个能够实现“先进先出”的存储结构。



队列可以用链表或者数组来实现。用链表实现队列，需要比较复杂的内存管理。而用数组实现比较简单，因此基于数组的队列，常在计算机操作系统的内核中或嵌入式系统中得到应用。

**1.队列的顺序存储**

队列的顺序实现是指分配一块连续的存储单元存放队列中的元素，并附设两个指针front 和rear分别指示队头元素和队尾元素的位置。设队头指针指向队头元素，队尾指针指向队尾 元素的下一个位置（也可以让rear指向队尾元素，front指向队头元素的前一个位置，对于 这种设置方法，以下图为例思考出队和入队后这两个指针的变化）。

  队列的操作

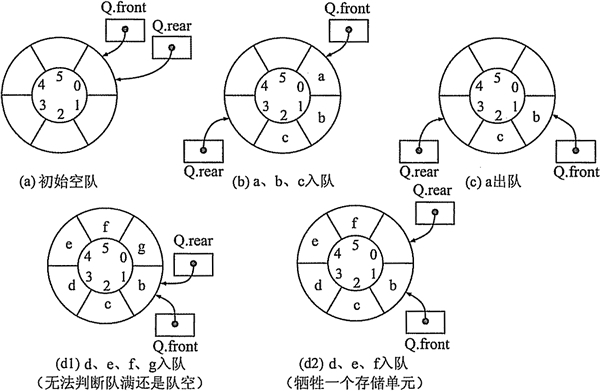
队列的顺序存储类型可描述为：

1. #define MaxSize 50 //定义队列中元素的最大个数
2. **typedef** **struct**{
3. ElemType data[MaxSize]; //存放队歹I]元素
4. int front, rear; //队头指针和队尾指针
5. }SqQueue;

初始状态（队空条件）：Q.front==Q.rear==0。  
  
进队操作：队不满时，先送值到队尾元素，再将队尾指针加1。  
  
出队操作：队不空时，先取队头元素值，再将队头指针加1。  
  
如图3-6 (a)所示为队列的初始状态，有Q.front==Q.rear==0成立，该条件可以作为队 列判空的条件。但能否用Q.rear==MaxSize作为队列满的条件呢？显然不能，图3-6(d)中， 队列中仅有1个元素，但仍满足该条件。这时入队出现“上溢出”，但这种溢出并不是真正 的溢出，在data数组中依然存在可以存放元素的空位置，所以是一种“假溢出”。

**2.循环队列**

前面已指出了顺序队列的缺点，这里我们引出循环队列的概念。将顺序队列臆造为一个 环状的空间，即把存储队列元素的表从逻辑上看成一个环,称为循环队列。当队首指针Q.ftont =MaxSiZe-1后，再前进一个位置就自动到0，这可以利用除法取余运算(%)来实现。  
  
初始时：Q.front=Q.rear=0  
  
队首指针进 1：Q.front=(Q.front+1)%MaxSize  
  
队尾指针进 1：Q.rear=(Q.rear+1)%MaxSize  
  
队列长度：(Q.rear+MaxSize-Q.front)%MaxSize  
  
出队入队时：指针都按顺时针方向进1 (如图3-7所示)。  
  
那么，循环队列队空和队满的判断条件是什么呢？显然，队空的条件是Q.front==Q.rear。 如果入队元素的速度快于出队元素的速度，队尾指针很快就赶上了队首指针，如图3-7(d1)，此时可以看出队满时也有Q.front==Q.rear。循环队列出入队示意图如图3-7所示。

  
图3-7  循环队列出入队示意图

为了区分队空还是队满的情况，有三种处理方式：本题选择了第一种方式。  
  
1) 牺牲一个单元来区分队空和队满，入队时少用一个队列单元，这是一种较为普遍的 做法，约定以“队头指针在队尾指针的下一位置作为队满的标志”，如图3-7(d2)所示。  
队满条件为：(Q.rearfl)%MaxSize==Q.front。  
队空条件仍为：Q.front==Q.rear。  
队列中元素的个数：(Q.rear-Q.front+MaxSize)%MaxSize  
  
2) 类型中增设表示元素个数的数据成员。这样，则队空的条件为Q.Size==0，队满的条 件为 Q.size==MaxSize。这两种情况都有 Q.front=Q.rear。  
  
3) 类型中增设tag数据成员，以区分是队满还是队空。tag等于0的情况下，若因删除导 致Q.front==Q.rear则为队空；tag等于1的情况下，若因插入导致Q.ftont==Q.rear则为队满。