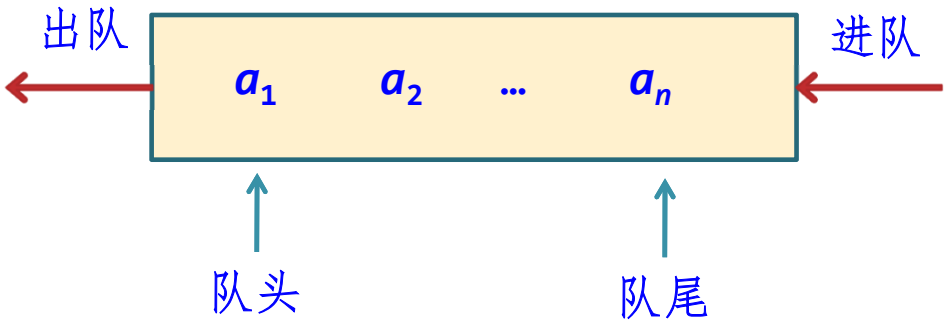
3.4 队　列

3.4.1 队列的基本概念

- 队列（简称队）也是一种运算受限的线性表，在这种线性表上，插入限定在表的某一端进行，删除限定在表的另一端进行。
  - 队列的插入操作称为进队，删除操作称为出队。
  - 允许插入的一端称为队尾，允许删除的一端称为队头。
  - 新插入的元素只能添加到队尾，被删除的只能是排在队头的元素。

队列的模型示意图



- 是一种先进先出(First In First Out， 简称FIFO)的线性表。

队列的基本运算

- 初始化队列InitQueue(Q)。建立一个空队Q。
- 销毁队DestroyQueue(Q)。释放队列Q占用的内存空间。
- 进队EnQueue(Q, *x*)。将*x*插入到队列Q的队尾。
- 出队DeQueue(Q, *x*)。将队列Q的队头元素出队并赋给*x*。
- 取队头元素GetHead(Q, *x*)。取出队列Q的队头元素并赋给*x*，但该元素不出队。
- 判断队空QueueEmpty(Q)。判断队列Q是否为空。

【示例-1】以下属于队列的基本运算的是（　　）。

A．对队列中的元素排序

B．取出最近进队的元素

C．在队列中某元素之前插入元素

D．删除队头元素

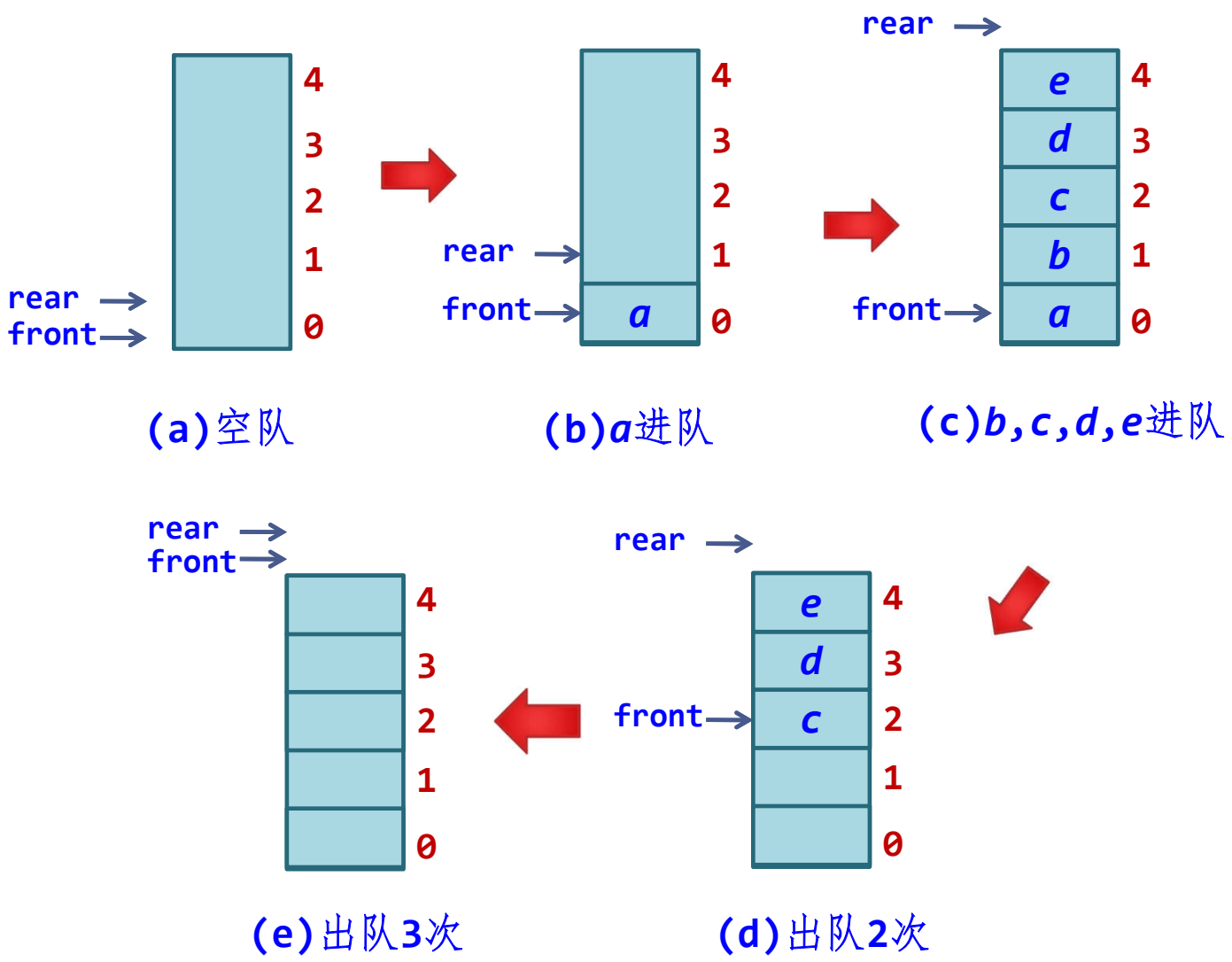
选择的答案是：D

3.4.2 循环队列 — 队列的顺序表示和实现

- 队列通常有两种存储结构，即顺序存储结构和链式存储结构。
- 队列的顺序存储结构简称为顺序队列，它由一个一维数组（用于存储队列中元素）及两个分别指示队头和队尾的变量组成，这两个变量分别称为“队头指针”和“队尾指针”。
- 通常约定：初始化建空队列时，令 **front=rear=0**。在非空队列中，头指针始终指向队列头元素，而尾指针始终指向**rear**队列尾元素的下一个位置。

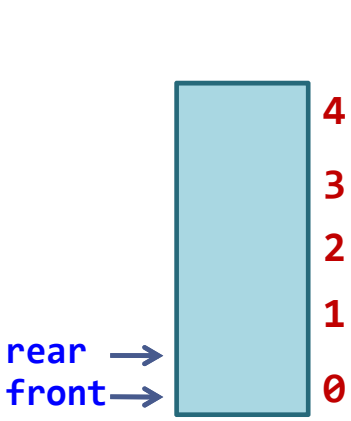
◆ 顺序队列的类型声明如下：

```
//----------队列的静态存储结构-------------
#define MaxSize 20          //指定队列的容量
typedef struct
{
   ElemType data[MaxSize];//保存队中元素
   int front,rear;         //队头和队尾指针
} SqQueue;
```

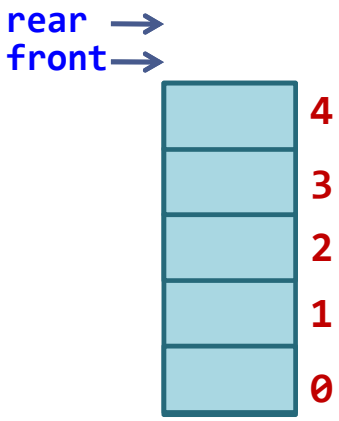

rear
front
4
3
2
1
0
(a)空队
rear
front
a
(b)a进队
rear
front
e
d
c
b
a
(c)b,c,d,e进队
rear
front
(e)出队3次
rear
front
e
d
c
(d)出队2次

顺序队的几种状态

从中可以看到，图 (a)和(e)都是队空的情况，均满足front==rear的条件，所以可以将front==rear作为队空的条件。



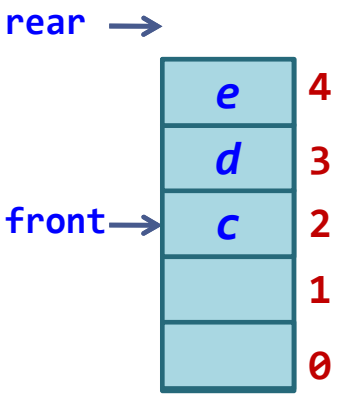
(a)空队



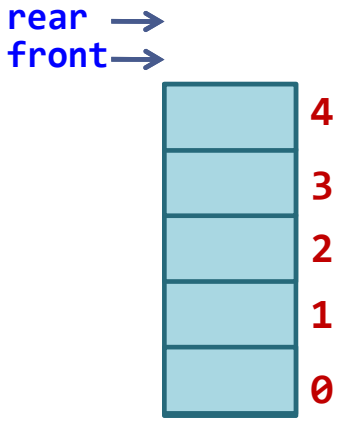
(e)出队3次

那么队满的条件如何设置呢？受顺序栈的启发，似乎很容易得到队满的条件为rear==MAXSIZE。？

- 显然这里有问题，因为图 (d)和(e)都满足这个“队满”的条件，而实际上队列并没有满。
- 这种因为队满条件设置不合理而导致的“溢出”称为**假溢出**，也就是说这种“溢出”并不是真正的溢出，尽管队满条件成立了，但队列中还有多个存放元素的空位置。

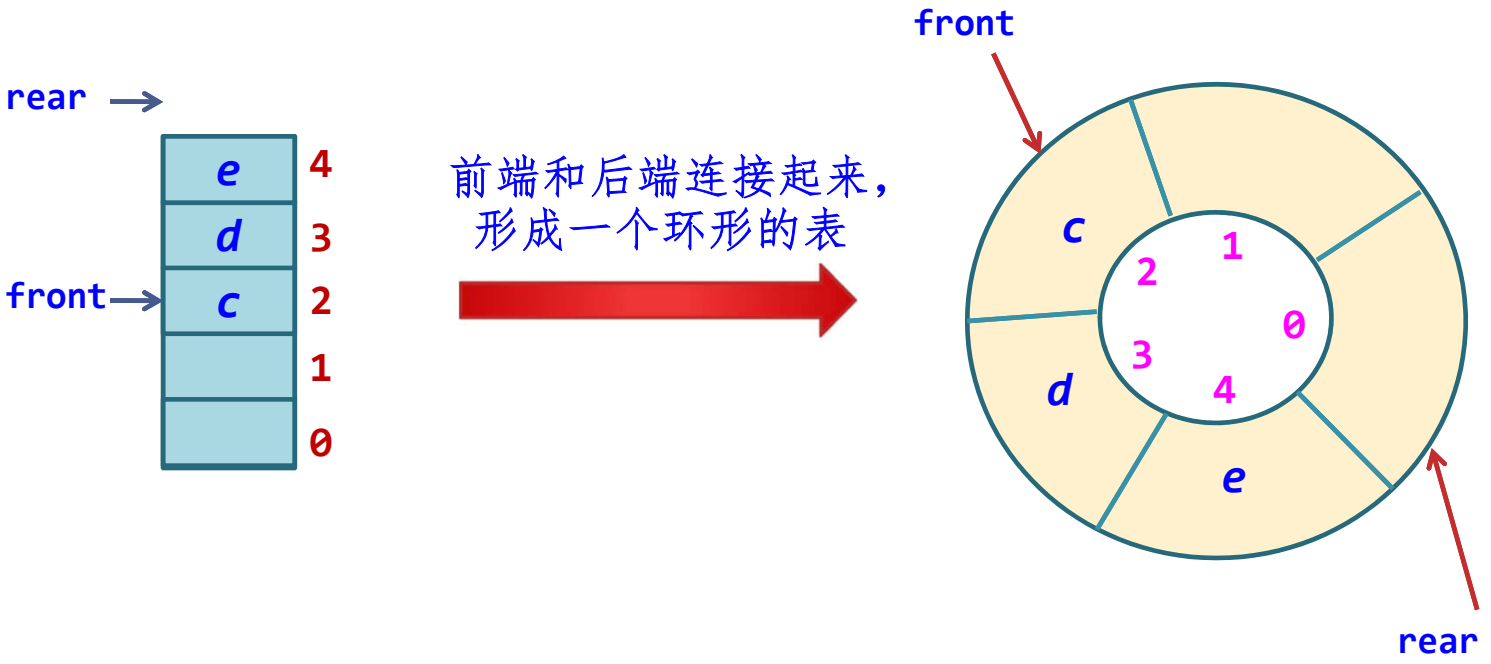


(d)出队2次



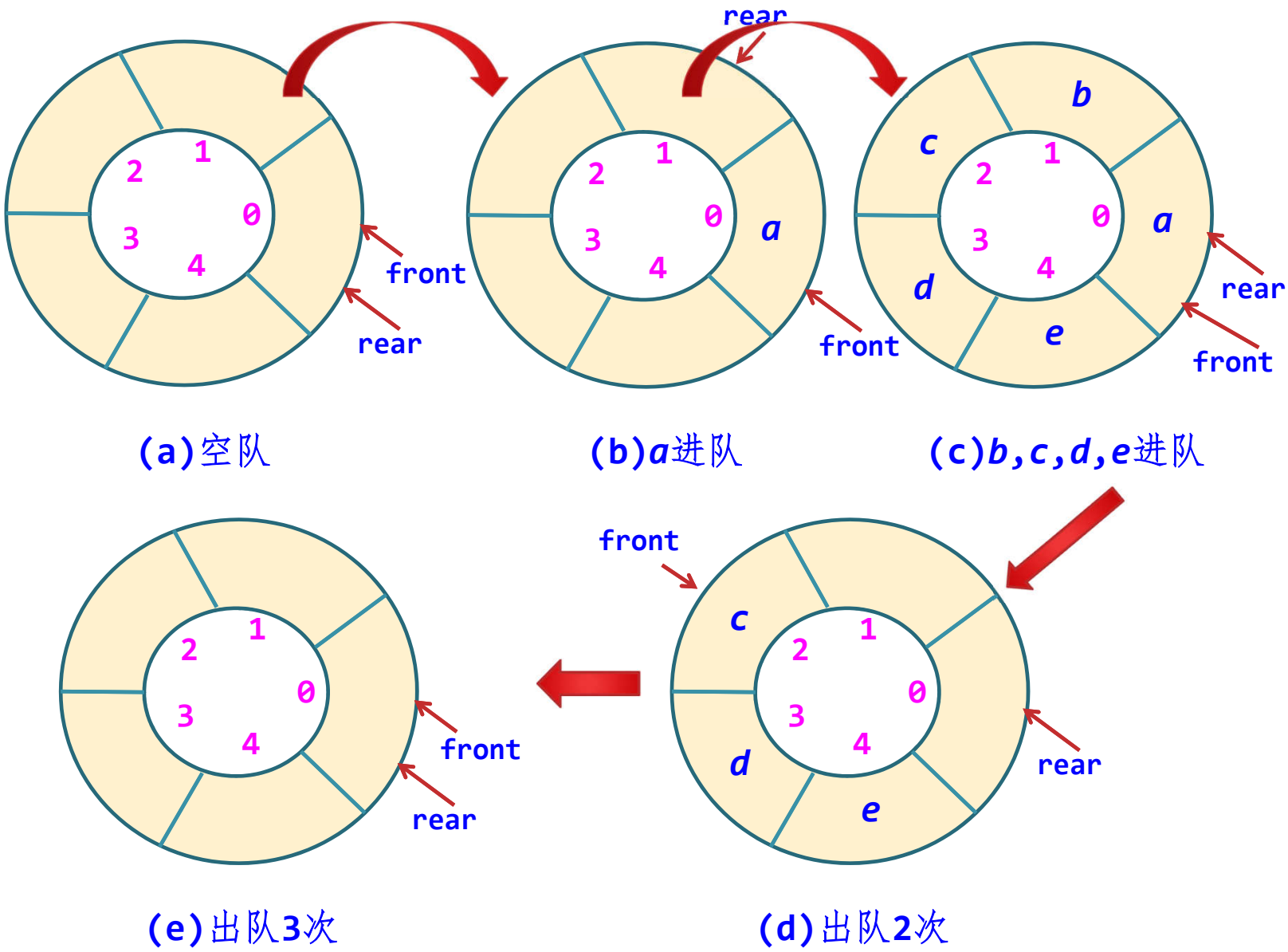
(e)出队3次

- 为了能够充分地使用数组中的存储空间，可以把数组的前端和后端“假想地”连接起来，形成一个环形的表，即把存储队列元素的表从逻辑上看成一个环。
- 这个环形的表叫做**循环队列**或**环形队列**。

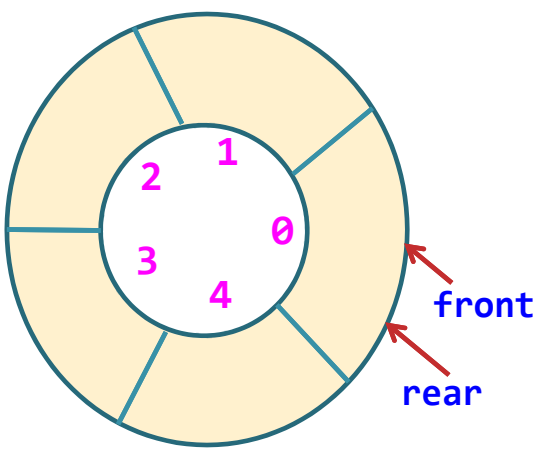

rear
e
d
front
c
4
3
2
1
0
前端和后端连接起来，
形成一个环形的表
front
c
d
e
1
2
0
3
4
rear

队头队尾指针加1的操作为：

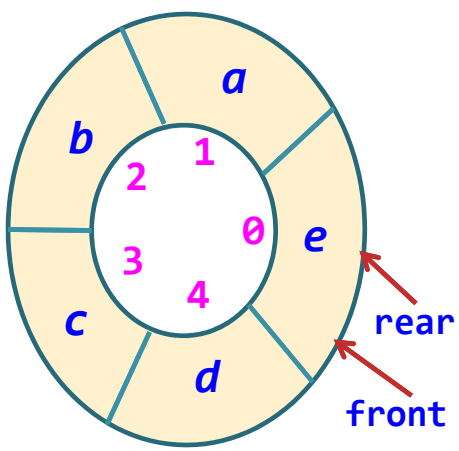
- 队头指针进1：front=(front+1) MOD MAXSIZE
- 队尾指针进1：rear=(rear+1) MOD MAXSIZE


rear
front
rear
front
rear
front
(a)空队
(b)a进队
(c)b,c,d,e进队
front
rear
front
rear
(e)出队3次
(d)出队2次

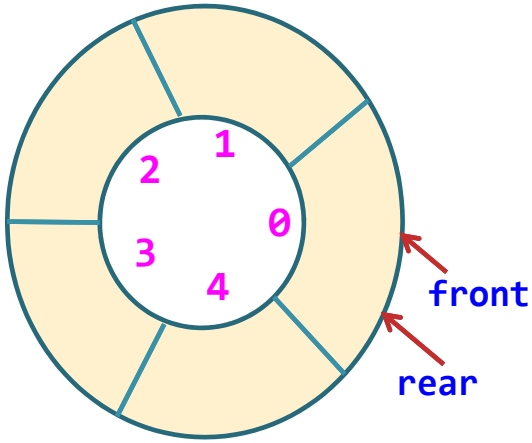
在循环队列中仍不能区分队空和队满，图(a)、(c)和(e)都满足条件front==rear。



(a)空队



(c)*b*,*c*,*d*,*e*进队



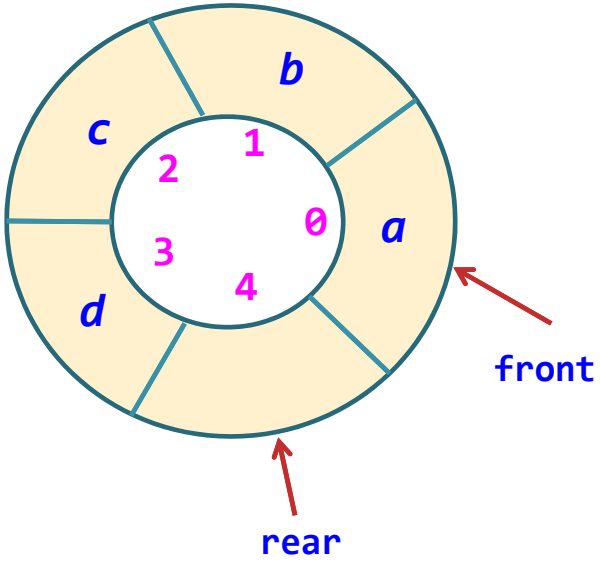
(e)出队3次

规定

- 仍设置队空条件为front==rear。
- 将队满条件设置为(rear+1) MOD MAXSIZE==front。
- 也就是说，当rear指到front的前一位置时就认为队列满了。

试探进队，若达到队头指针位置，
则认为队满！

- 显然在这样设置的队满条件下，队满条件成立时队中还有一个空闲单元，也就是说这样的队中最多只能进队MAXSIZE-1个元素。



MAXSIZE=5，最多只有4个元素在循环队列中，不会发生这样的情况

归纳起来，上述设置的循环队列Q的4个要素如下：

- 队空条件：Q.front==Q.rear
- 队满条件：(Q.rear+1)%MaxSize==Q.front
- 进队操作：Q.data[Q.rear]=x;
  Q.rear=(Q.rear+1)%MaxSize;
- 出队操作：x=Q.data[Q.front];
  Q.front=(Q.front+1)%MaxSize;

◆循环队列的类型声明如下：

```
//---------循环队列 --顺序存储结构---------------------
#define  MAXQSIZE  100    // 最大队列长度+1
typedef  struct {
   ElemType   *base;  // 初始化的动态分配存储空间
   int   front;   // 头指针，若队列不空，指向队列头元素
   int   rear;  // 尾指针，若队列不空，指向队列尾元素的下一个位置
} SqQueue ;
```

循环队列的基本算法

（1）初始化队列算法

主要操作：指定sq.front=sq.rear=0。

```
void InitQueue(SqQueue &Q)
{    // 构造一个空队列Q
    Q.base = (QElemType *)malloc(MAXQSIZE * sizeof(QElemType));
    if(!Q.base) // 存储分配失败
        exit(OVERFLOW);
    Q.front = Q.rear = 0;
}
```

（2）销毁队列运算算法

这里顺序队的内存空间是动态申请得到的，在不再需要时由申请者主动释放其空间。

具体算法的编制？

```
void DestroyQueue(SqQueue &Q)
{   // 销毁队列Q，Q不再存在
    if(Q.base)
        free(Q.base);
    Q.base = NULL;
    Q.front = Q.rear = 0;
}
```

（3）入队列算法

主要操作：先判断队列是否已满，若不满，在队尾指针位置存放*x*，然后循环加1。

```
Status EnQueue(SqQueue &Q, QElemType e)
{   // 插入元素e为Q的新的队尾元素
    if((Q.rear + 1) % MAXQSIZE == Q.front) // 队列满
        return ERROR;
    Q.base[Q.rear] = e;
    Q.rear = (Q.rear + 1) % MAXQSIZE;
    return OK;
}
```

（4）出队列算法

主要操作：先判断队列是否已空，若不空，将队头指针位置的元素值赋给x，然后对头指针循环加1。

```
Status DeQueue(SqQueue &Q, QElemType &e)
{  // 若队列不空，则删除Q的队头元素，用e返回其值，
   // 并返回OK；否则返回ERROR
    if(Q.front == Q.rear) // 队列空
        return ERROR;
    e = Q.base[Q.front];
    Q.front = (Q.front + 1) % MAXQSIZE;
    return OK;
}
```

（5）取队头元素运算算法

主要操作：先判断队列是否已空，若不空，将队头指针前一个位置的元素值赋给*x*。

```
Status GetHead(SqQueue Q, QElemType &e)
{   // 若队列不空，则用e返回Q的队头元素，并返回OK;
    // 否则返回ERROR
    if(Q.front == Q.rear) // 队列空
        return ERROR;
    e = Q.base[Q.front];
    return OK;
}
```

（6）判断队空算法

主要操作：若队列为空，则返回1；否则返回0。

```
Status QueueEmpty(SqQueue Q)
{
    // 若队列Q为空队列，则返回TRUE；否则返回FALSE
    if(Q.front == Q.rear) // 队列空的标志
        return TRUE;
    else
        return FALSE;
}
```

END