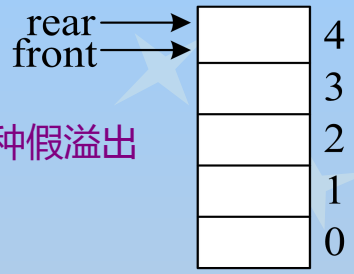
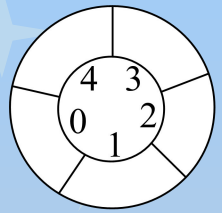
本节主题：

环形队列的存储及基本操作

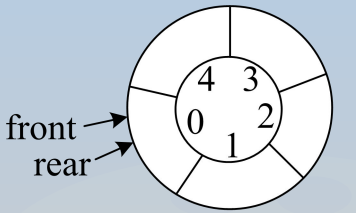
顺序队列的问题及环形队列的提出

- 顺序队列的问题
  - 用rear==MaxSize-1作为队满的条件有缺陷
  - 可能队列为空，但仍满足该条件，进队时出现“上溢出”是一种假溢出

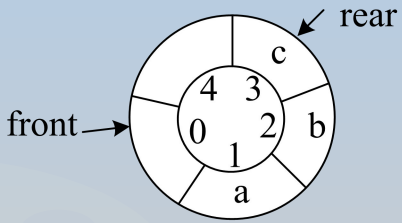




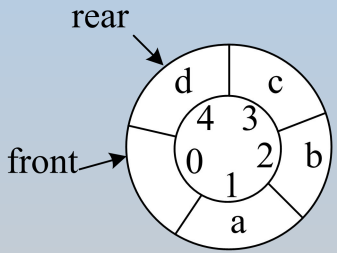
- 环形队列的对策
  - 充分地使用数组中的存储空间,把数组的前端和后端连接起来，形成一个环形的顺序表，即把存储队列元素的表从逻辑上看成一个环
  - 得到环形队列或循环队列



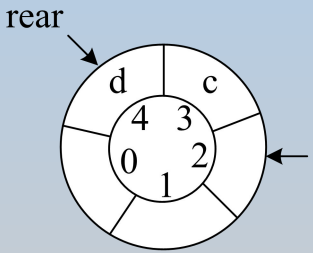
(1) 空队列



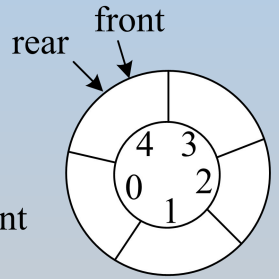
(2) a b c 入队



(3) d入队

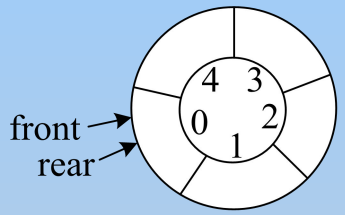


(4) 出队二次

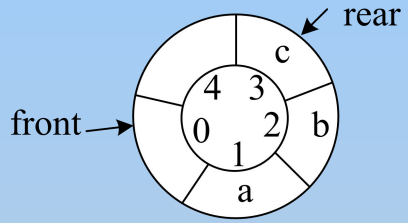


(5) 出队二次

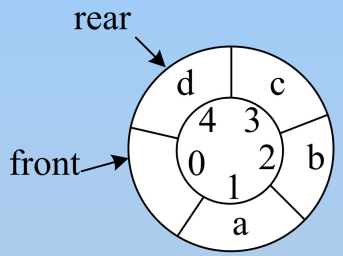
环形队列的四要素



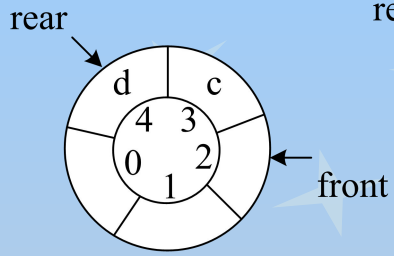
(1) 空队列



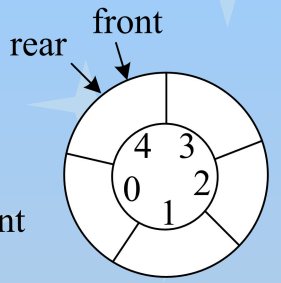
(2) a b c 入队



(3) d入队，队满



(4) 出队二次



(5) 出队二次

- 队空条件：front=rear
- 队满条件：(rear+1)%MaxSize=front

只存储MaxSize-1个元素

- 进队e操作：rear=(rear+1)%MaxSize; 将e放在rear处
- 出队操作：front=(front+1)%MaxSize;取出front处元素e;

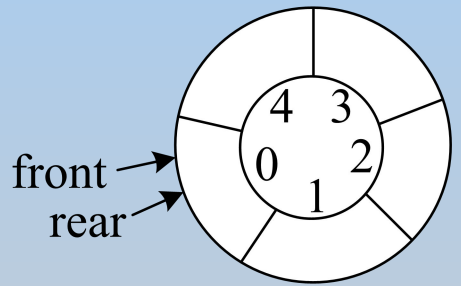
```
typedef struct
{
    ElemType data[MaxSize];
    int front,rear;
} SqQueue;
```

- rear指向队尾元素
- front指向队头元素的前一个位置

初始化队列InitQueue(q)

- 构造一个空队列q。将front和rear指针均设置成初始状态即0值。

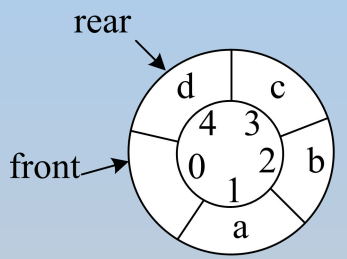
```
void InitQueue(SqQueue *&q)
{
    q=(SqQueue *)malloc (sizeof(SqQueue));
    q->front=q->rear=0;
}
```



销毁队列ClearQueue(&q)

- 释放队列q占用的存储空间。

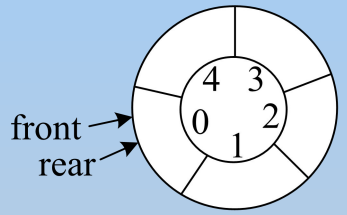
```
void DestroyQueue(SqQueue *&q)
{
    free(q);
}
```

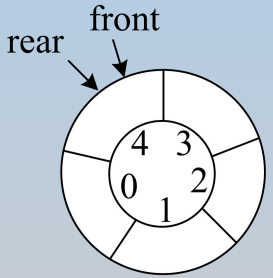


判断队列是否为空QueueEmpty(q)

- 若队列q满足q->front==q->rear条件，则返回true；否则返回false。

```
bool QueueEmpty(SqQueue *q)
{
    return(q->front==q->rear);
}
```

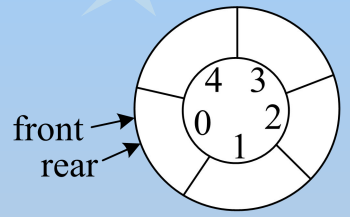




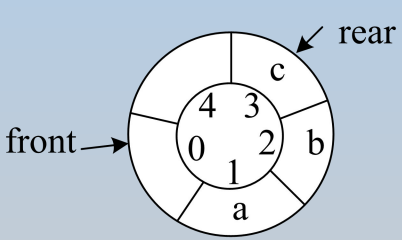
进队列enQueue(q,e)

- 在队列不满的条件下，先将队尾指针rear循环增1，然后将元素添加到该位置。

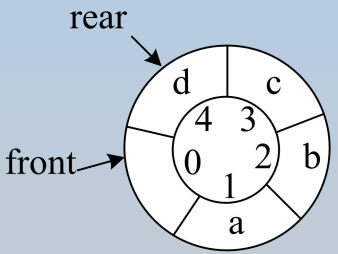
```
bool enQueue(SqQueue *&q,ElemType e)
{
    if ((q->rear+1)%MaxSize==q->front)
        return false;//队满上溢出
    q->rear=(q->rear+1)%MaxSize;
    q->data[q->rear]=e;
    return true;
}
```



(1) 空队列



(2) a b c 入队

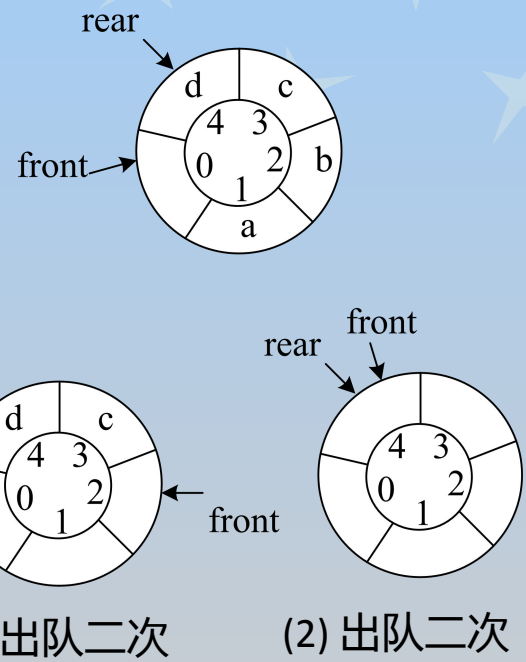


(3) d入队

出队列deQueue(q,e)

- 在队列q不为空的条件下，将队首指针front循环增1，并将该位置的元素值赋给e。

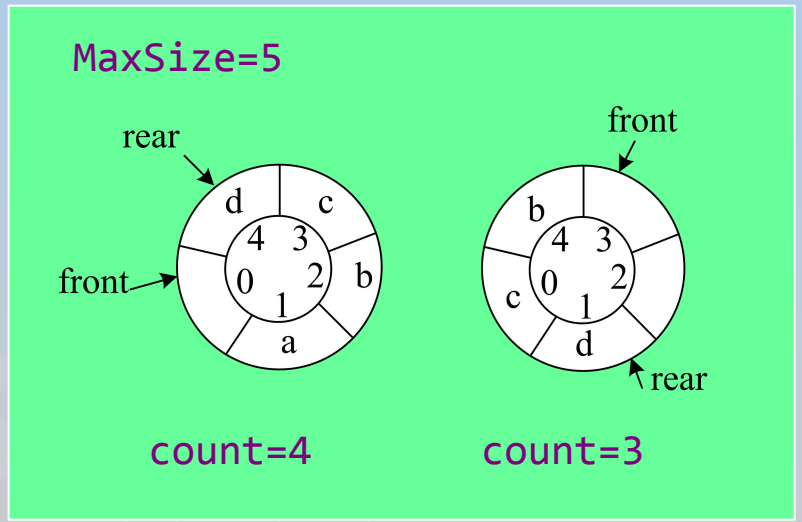
```
bool deQueue(SqQueue *&q,ElemType &e)
{
    if (q->front==q->rear)   //队空下溢出
        return false;
    q->front=(q->front+1)%MaxSize;
    e=q->data[q->front];
    return true;
}
```



(1) 出队二次　　(2) 出队二次

拓展：环形队列的另一种设计

- 问题
  - 如果知道环形队列队头指针和队列中元素个数，则可以计算出队尾指针。
  - 存储环形队列时，可以记录队列中队头指针front和队列中元素个数count，而不记录队尾指针rear。
  - 这也是一种可行的设计，设计这种环形队列的初始化、入队、出队和判空算法。

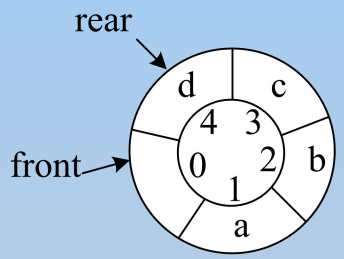


front、rear、count，三者择其二：

- 已知front、rear，求队中元素个数：
  count=(rear-front+MaxSize)%MaxSize
- 已知front、count，求rear：
  rear=(front+rear)%MaxSize
- 已知rear、count，求front：
  front=(rear-count+MaxSize)%MaxSize

只记录队头front和元素个数count的方案

```
typedef struct
{
    ElemType data[MaxSize];
    int front;
    int count;
} QuType;
```



- 环形队列的四要素
  - 队空条件：count = 0
  - 队满条件：count = MaxSize

可存储MaxSize个元素！

  - 进队e操作：rear=(rear+1)%MaxSize; 将e放在rear处（rear由front和count定）
  - 出队操作： front=(front+1)%MaxSize;取出front处元素e;

算法

```
//初始化队运算算法
void InitQueue(QuType *&qu)
{
   qu=(QuType *)malloc(sizeof(QuType));
   qu->front=0;
   qu->count=0;
}
```

```
//出队运算算法
bool DeQueue(QuType *&qu,ElemType &x)
{
   if (qu->count==0)
      return false;
   else
   {
       qu->front=(qu->front+1)%MaxSize;
       x=qu->data[qu->front];
       qu->count--;
       return true;
   }
}
```

```
//判队空运算算法
bool QueueEmpty(QuType *qu)
{
   return(qu->count==0);
}
```

```
//进队运算算法
bool EnQueue(QuType *&qu,ElemType x)
{
   int rear;
   if (qu->count==MaxSize)
       return false;
   else
   {
       rear=(qu->front+qu->count)%MaxSize;
       rear=(rear+1)%MaxSize;
       qu->data[rear]=x;
       qu->count++;
       return true;
   }
}
```