队列（Queue）

和栈相反，队列是一种先进先出（first in first out，缩写为FIFO）的线性表

它只允许在表的一端进行插入，而在表的另一端删除元素；

在队列中

- 队尾（rear）；允许插入的一端叫做队尾
- 队头（front）；允许删除的一端叫做队头

抽象数据类型定义：

```
ADT Queue {
    数据对象：D={ai|ai∈ElementSet, i = 1,2,...,}

    数据关系：R1={<ai-1,ai>|ai-1,ai∈D,i=2,...,n}

    基本操作：

    //初始化队列
    Status InitQueue(Queue);

    //销毁队列
    Status DestoryQueue(Queue);

    //清空队列
    Status ClearQueue(Queue);

    //是否为空队列
    boolean isEmptyQueue(Queue);

    //获取队列长度
    int QueueLength(Queue);

    //获取队头
    ElementType GetHead(Queue);

    //入列
    Status EnQueue(Queue,
    ElementType element);

    //出列
    ElementType DeQueue(Queue);

    //遍历队列
    void TraverseQueue(Queue);
}
```

链式队列

用链表表示的队列简称为链队列

一个链队列显然需要两个分别指向队头和队尾的指针（分别称为队头指针和队尾指针）才能唯一确定；

空的链队的判决条件为头指针和尾指针均指向头结点

```
/**队列结点**/
typedef struct QueueNode {
    QueueElementType    data;   //数据
    struct QueueNode    *next;  //指针
} QueueNode, *QueuePointer;


/**队列**/
typedef struct {
    QueuePointer    front;      //队头指针
    QueuePointer    rear;       //队尾指针
} LinkedQueue;
```

初始化队列

```
/**
* 功能：初始化队列
* 参数：queue 队列;
* 返回值：初始化结果
*/
Status InitQueue(LinkedQueue &queue){
    //申请存储空间作为头结点
    queue.front = queue.rear = (QueuePointer)malloc(sizeof(QueueNode));

    //内存申请失败
    if(queue.front == NULL){
        exit(0);
    }

    //初始状态下，头结点的指针域为空
    q.front->next = NULL;

    return TRUE;
}
```

入队

```
/**
* 功能：入队；向队尾添加新元素
* 参数：queue 队列；element 要入队的元素；
* 返回值：入队结果
*/
Status EnQueue(LinkedQueue &queue, QueueElementType element){
    //申请存储空间作为新结点
    QueuePointer newNode = (QueuePointer)malloc(sizeof(QueueNode));

    //内存申请失败
    if(newNode == NULL){
        exit(0);
    }

    //设置该结点的数据域和指针域
    newNode->data = element;
    //因为队列先进先出的特性，新结点总是在最后没有后继，所以新结点指针域应该设置为NULL
    newNode->next = NULL;

    //原队尾的指针域连接到新结点
    queue.rear->next = newNode;
    //新结点成为队尾
    queue.rear = newNode;

    return TRUE;
}
```

出队

```
/**
* 功能：出队；取出队头元素
* 参数：queue 队列；element 通过此参数返回队头元素；
* 返回值：出队结果
*/
Status DeQueue(LinkedQueue &queue, QueueElementType &element){
    //队头指针和队尾指针相等，说明队空
    if(queue.front == queue.rear){
        return ERROR;
    }

    //获取队头元素
    LinkedQueue frontNode = queue.next;

    //通过引用参数返回元素
    element = frontNode->data;

    //让队头出队；
    //将头结点指针指向原队头的直接后继，将其作为新队头
    queue.front->next = frontNode->next;
```

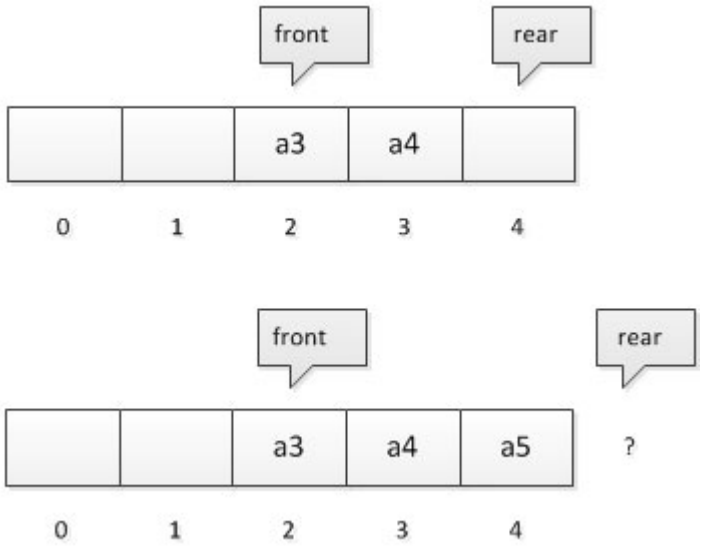
```
    //队列只有一个结点的特殊情况
    if(queue.rear == frontNode){
        //当队列中最后一个元素被删除后，队尾指针也丢失了
        //因此需要对队尾指针重新赋值（指向头节点）
        queue.rear = queue.front;

    }

    //将原队头所使用的内存空间释放
    free(frontNode);

    return TRUE;
}
```

循环队列（Circular Queue）

在队列的顺序存储结构中，除了用一组地址连续的存储单元依次存放从队列头到队列尾的元素之外，尚需附设两个指针front和rear分别指示队列头元素以及队列尾元素的位置；

因此，在非空队列中，头指针始终指向队列头元素，而尾指针始终指向队列尾元素的下一位置

一个大小为5的队列（下标从0开始），队头为2，头尾为4；虽然此时还有剩余空间，但是此时不能再继续插入新的队尾元素，否则会数组越界导致程序错误；



解决办法是臆造为一个环状的空间，称之为循环队列

```
typedef struct {
    QueueElementType    *base;  //初始化的空间
    int                 front;  //头指针；若队列不空，则指向队列头元素
    int                 rear;   //尾指针；若队列不空，则指向队尾元素的下一位置
} SequenceQueue;
```

牺牲一个存储空间构造的循环队列

- 队满 (rear + 1) % size == front
- 队空 front == rear
- 队列长度 (rear - front + size) % size