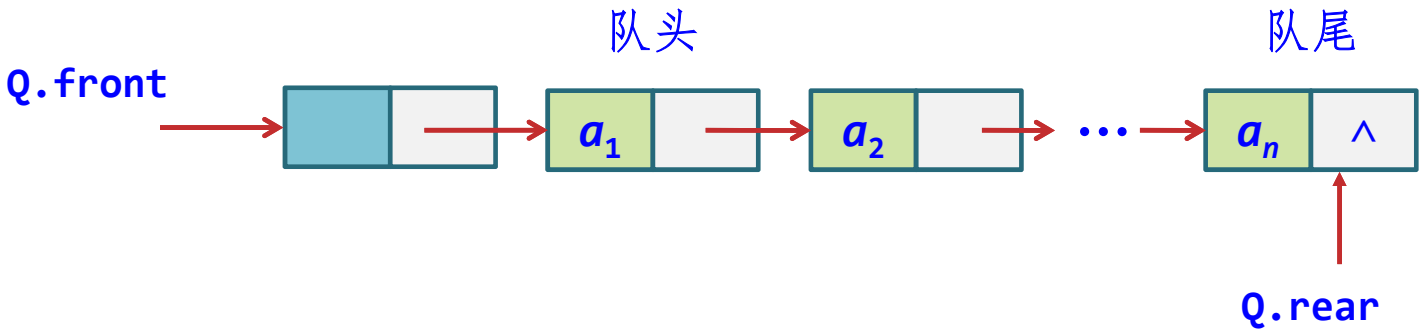
3.4.2 链队列—队列的链式表示和实现

- 队列的链式存储结构简称为链队列。
  - 这里采用的链队列是一个同时带有队头指针front和队尾指针rear的单链表。
- 队头指针指向头结点，队尾指针指向队尾结点即单链表的尾结点，并将队头和队尾指针结合起来构成链队结点，

链队列的基本结构：



数据结点

```
typedef struct QNode
{   ElemType data;          //存放队中元素
    struct QNode *next;     //指向下一个结点
} QNode, *QueuePtr;         //数据结点类型
```

链队结点

```
typedef struct
{   QNode *front;           //队头指针
    QNode *rear;            //队尾指针
} LinkQueue;                //链队结点类型
```

归纳起来，链队列的4个要素如下：

- 队空条件：Q.front->next == NULL;
- 队满条件：不考虑（因为每个结点是动态分配的）；
- 进队操作：创建结点*p*，将其插入到队尾，并由Q.rear指向它；
- 出队操作：删除队头的结点。

在链队列上实现队列基本运算如下：

（1）初始化队列算法

主要操作：创建链队列头结点，并置rear和front指向头结点，且指针域为NULL。

```
void InitQueue(LinkQueue &Q)
{
    // 构造一个空队列Q
    if(!(Q.front = Q.rear = (QueuePtr)malloc(sizeof(QNode))))
        exit(OVERFLOW);
    Q.front->next = NULL;
}
```

（2）销毁队列算法

依次释放单链表的结点即可。

```
void DestroyQueue(LinkQueue &Q)
{    // 销毁队列Q(无论空否均可)
    while(Q.front) {
        Q.rear = Q.front->next;
        free(Q.front);
        Q.front = Q.rear;
    }
}
```

（3）入队列算法

主要操作：创建一个新结点，将其链接到链队的末尾，并由rear指向它。

```
void EnQueue(LinkQueue &Q, QElemType e)
{  // 插入元素e为Q的新的队尾元素
    QueuePtr p;
    if(!(p = (QueuePtr)malloc(sizeof(QNode)))) //存储分配失败
        exit(OVERFLOW);
    p->data = e;
    p->next = NULL;
    Q.rear->next = p;
    Q.rear = p;
}
```

（4）出队列算法

主要操作：将front->next结点的data域值赋给e，并删除该结点。？

```
Status DeQueue(LinkQueue &Q, QElemType &e)
{  // 若队列不空，删除Q的队头元素，用e返回其值，并返回OK，
   // 否则返回ERROR
    QueuePtr p;
    if(Q.front == Q.rear)
        return ERROR;
    p = Q.front->next;
    e = p->data;
    Q.front->next = p->next;
    if(Q.rear == p)
        Q.rear = Q.front;
    free(p);
    return OK;
}
```

（5）取队头元素算法

主要操作：将对头结点的data域值赋给e。

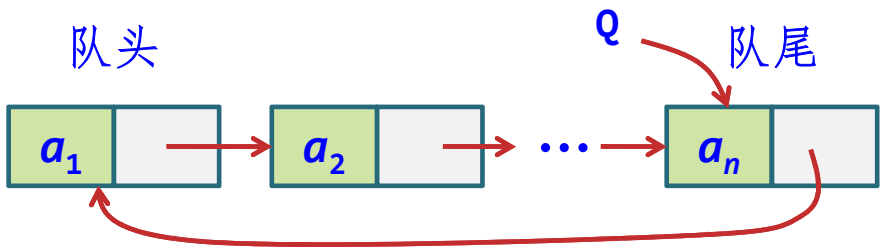
```
Status GetHead(LinkQueue Q, QElemType &e)
{  // 若队列不空，则用e返回Q的队头元素，并返回OK，
   // 否则返回ERROR
    QueuePtr p;
    if(Q.front == Q.rear)
        return ERROR;
    p = Q.front->next;
    e = p->data;
    return OK;
}
```

（6）判断队空算法

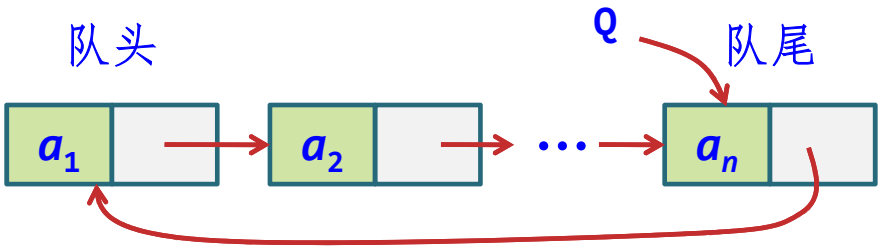
主要操作：若链队为空，则返回1；否则返回0。

```
Status QueueEmpty(LinkQueue Q)
{  // 若Q为空队列，则返回TRUE，否则返回FALSE
    if(Q.front->next == NULL)
        return TRUE;
    else
        return FALSE;
}
```

【示例】 若使用不带头结点的循环链表来表示队列，Q是这样的链表中尾结点指针。试基于此结构给出队列的入队列、出队列算法。



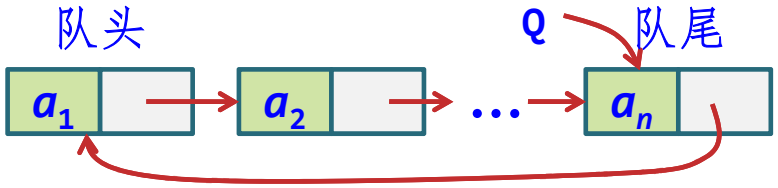
由Q唯一标识链表（链队）



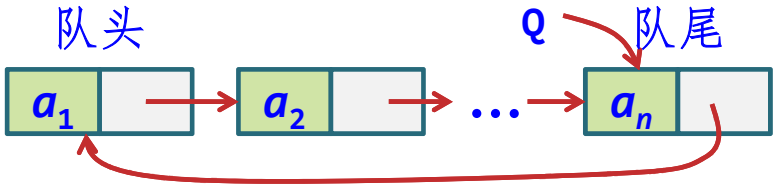

```
typedef struct node
{   ElemType data;          //数据域
    struct node *next;      //指针域
} QNode;                    //链队数据结点类型
```

解：这里使用的循环链表不带头结点。

- Q始终指向队尾结点，Q->next即为队头结点。
- 当Q==NULL时队列为空。
- Q->next==Q表示队列中只有一个结点。




```
//----入队列算法----
void EnQueue(QNode *&Q,ElemType x)
{  QNode *s;
   s=(QNode *)malloc(sizeof(QNode));
   s->data=x;                    //创建存放x的结点s
   if (Q==NULL)                  //原为空队
   {  Q=s;
      Q->next=Q;                 //构成循环单链表
   }
   else                          //原队不空,结点s插到队尾
   {  s->next=Q->next;
      Q->next=s;
      Q=s;          //Q指向结点s
   }
}
```




```
//-----出队列算法-----
int DeQueue(QNode *&Q,ElemType &x)
{  QNode *s;
   if (Q==NULL) return 0;          //原为队空
   if (Q->next==Q)                 //原队只有一个结点
   {  x=Q->data;
      free(Q);
      Q=NULL;
   }
   else                            //原队有两个或以上的结点
   {    s=Q->next;                 //将Q之后的结点s删除
        x=s->data;
        Q->next=s->next;
        free(s);
   }
   return 1;
}
```

END