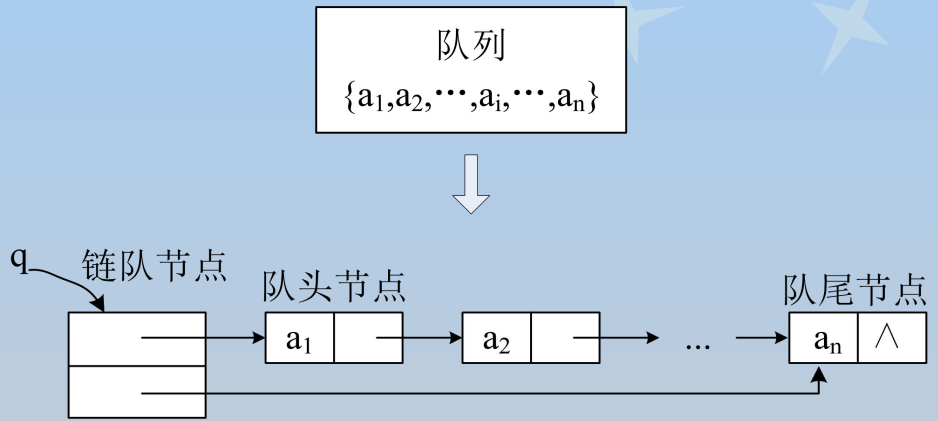
本节主题：

队列的链式存储结构及其基本运算的实现

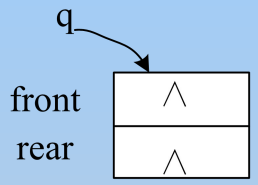
方案

- 链队组成
  （1）存储队列元素的单链表
  （2）指向队头和队尾指针的链队头节点
- 存储结构定义

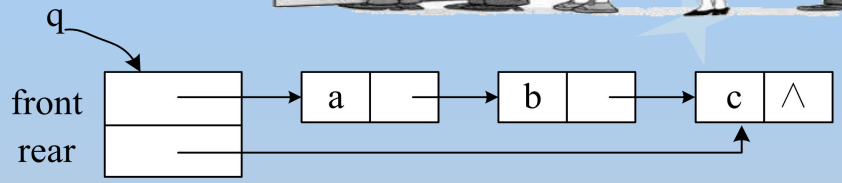
```
typedef struct qnode  //数据节点
{
   ElemType data;
   struct qnode *next;
} QNode;
typedef struct   //链队节点
{
   QNode *front;
   QNode *rear;
} LiQueue;
```



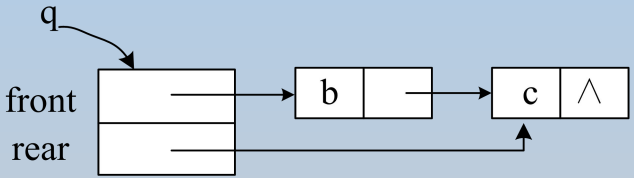
链队的4要素



(1) 空队列



(2) a b c 入队



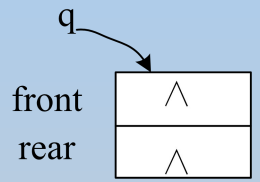
(3) 出队一次

- 队空条件： front=rear=NULL
- 队满条件： 不考虑
- 进队e操作： 将包含e的节点插入到单链表表尾
- 出队操作： 删除单链表中首个数据节点

初始化队列InitQueue(q)

- 构造一个空队列，即只创建一个链队头节点，其front和rear域均置为NULL，不创建数据元素节点。

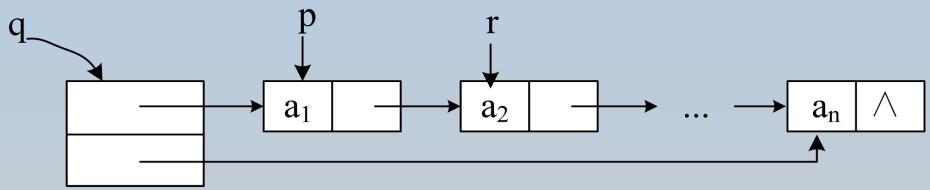
```
void InitQueue(LiQueue *&q)
{
    q=(LiQueue *)malloc(sizeof(LiQueue));
    q->front=q->rear=NULL;
}
```



销毁队列DestroyQueue(q)

- 释放队列占用的存储空间,包括链队头节点和所有数据节点的存储空间。

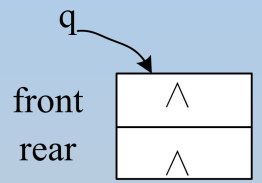
```
void DestroyQueue(LiQueue *&q)
{
    QNode *p=q->front,*r;
    if (p!=NULL)
    {
        r=p->next;
        while (r!=NULL)
        {
            free(p);
            p=r;
            r=p->next;
        }
    }
    free(p);
    free(q);
}
```



判断队列是否为空QueueEmpty(q)

- 若链队节点的rear域值为NULL，表示队列为空，返回true；否则返回false。

```
bool QueueEmpty(LiQueue *q)
{
    return(q->rear==NULL);
}
```



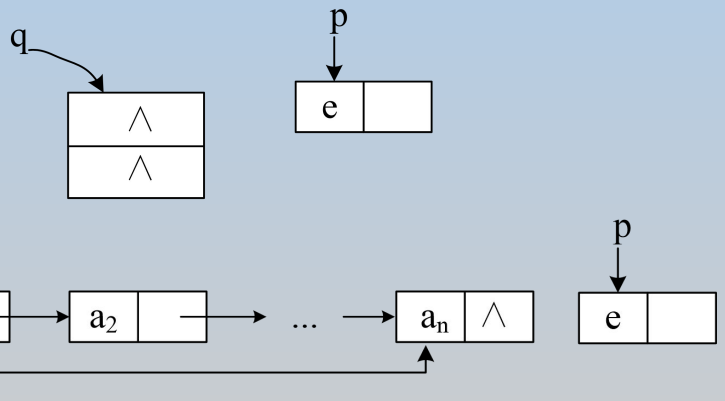
入队enQueue(q,e)

```
void enQueue(LiQueue *&q,ElemType e)
{
    QNode *p;
    p=(QNode *)malloc(sizeof(QNode));
    p->data=e;
    p->next=NULL;
    if (q->rear==NULL)
        q->front=q->rear=p;
    else
    {
        q->rear->next=p;
        q->rear=p;
    }
}
```

若链队为空,新节点是队首节点又是队尾节点

将*p节点链到队尾,并将rear指向它

- 操作
  - 创建data域为e的数据节点*p。
  - 若原队列为空，则将链队节点的两个域均指向*p节点，否则将*p链到单链表的末尾，并让链队节点的rear域指向它。



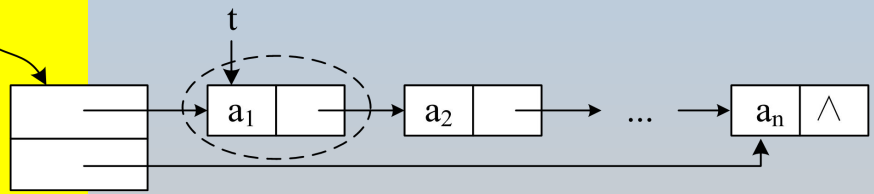
出队deQueue(q,e)

```
bool deQueue(LiQueue *&q,ElemType &e)
{
    QNode *t;
    if (q->rear==NULL)
       return false;
    t=q->front;
    if (q->front==q->rear)
        q->front=q->rear=NULL;
    else
        q->front=q->front->next;
    e=t->data;
    free(t);
    return true;
}
```

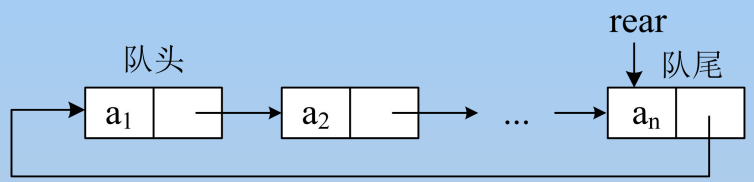
队列中只有一个节点时

队列中有多个节点时

- 操作
  - 若原队列不为空，则将第一个数据节点的data域值赋给e,并删除之。
  - 若出队之前队列中只有一个节点，则需将链队节点的两个域均置为NULL，表示队列已为空。



拓展：只保存尾结点的队列



链队可以通过尾节点指针rear唯一标识

链队的4要素

- 队空条件：rear=NULL
- 队满条件：不考虑
- 进队e操作：将包含e的节点插入到单链表表尾
- 出队操作：删除单链表首节点

```
//初始化队的算法
void initQueue(LinkList *&rear)
{
    rear=NULL;
}
```

```
//判断队空的算法
bool queueEmpty(LinkList *rear)
{
    return(rear==NULL);
}
```

```
//进队的算法
void enQueue(LinkList *&rear,ElemType x)

{
    LinkList *p;
    p=(LinkList *)malloc(sizeof(LinkList));
    p->data=x;
    if (rear==NULL)
    {
        p->next=p;
        rear=p;
    }
    else
    {
        p->next=rear->next;
        rear->next=p;
        rear=p;
    }
}
```

```
//出队的算法
bool deQueue(LinkList *&rear,ElemType &x)
{
    LinkList *q;
    if (rear==NULL)
        return false;
    else if (rear->next==rear)
    {
        x=rear->data;
        free(rear);
        rear=NULL;
    }
    else
    {
        q=rear->next;
        x=q->data;
        rear->next=q->next;
        free(q);
    }
    return true;
}
```