复杂链表的基本操作

学习目标和要求

1.能够写出单向加尾链表的查找 的算法；

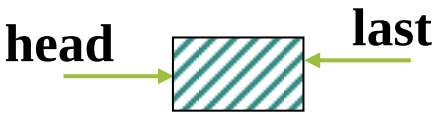
2.能够写出单向加头循环链表的删除算法

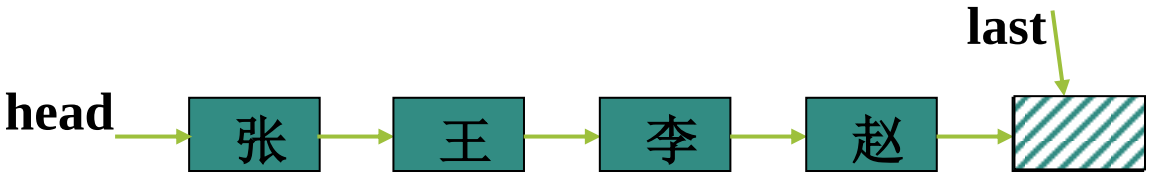
3.能够写出双向链表的插入和删除 算法

1.单向加尾链表的查找

```
ptr  searchB(ptr p,ptr last,int x)
{
1.    last->data=x;//设置监督元
2.    while(p->data!=x )
3.        p=p->next;//没找到，继续
4.    if(p!=last)return  p;//查找成功
5.    return  NULL;//查找不成功
}
```

主调语句：searchB(head, last, x);

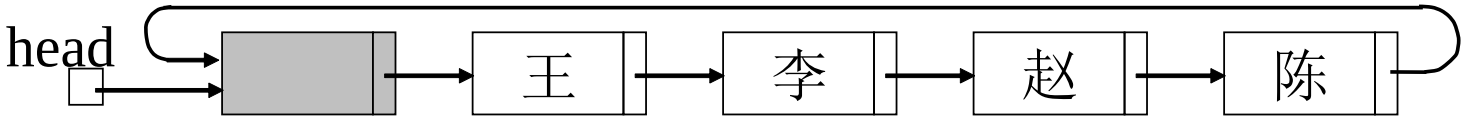




2.单向加头循环链表的删除算法

```
int  deleteBC(ptr h,element_type x)
{ ptr  f, p;
1.   f=h, p=h->next;   //置搜索指针初值
2.   h->data=x;              //置监督元
3.   while(p->data!=x)f=p, p=p->next;
4.   if(p==h)return  0;                  //删除不成功
5.   f->next=p->next;
6.   free(p);
7.   return 1;     //删除成功
 }
```

主调语句：deleteBC(head,x);



3.双向链表的插入和删除

结点含有两个链域的是**双向链表**，其中，左链域用于指向前驱结点，右链域用于指向后继结点。

结点类型定义为：

```
typedef  struct  dnode
{   element_type  data;                  //值域
    struct  dnode  *Llink,*Rlink;        //左右链域
}   dnode, *dptr;  //结构类型名dnode和指针类型名dptr
```

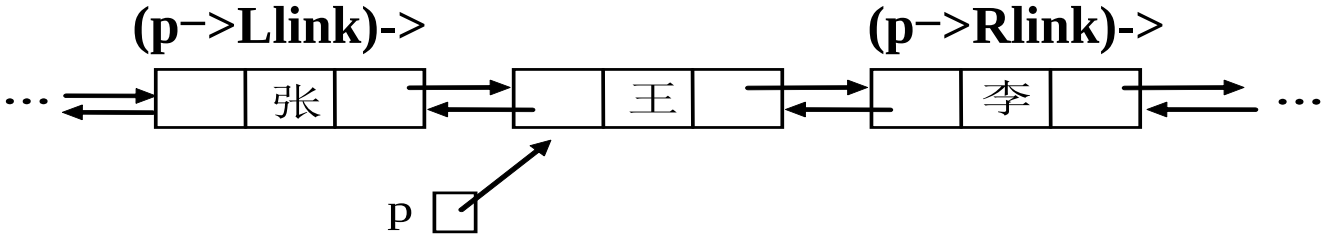
3.双向链表的插入和删除

（1）链接关系的特点

p的前驱后继都不空

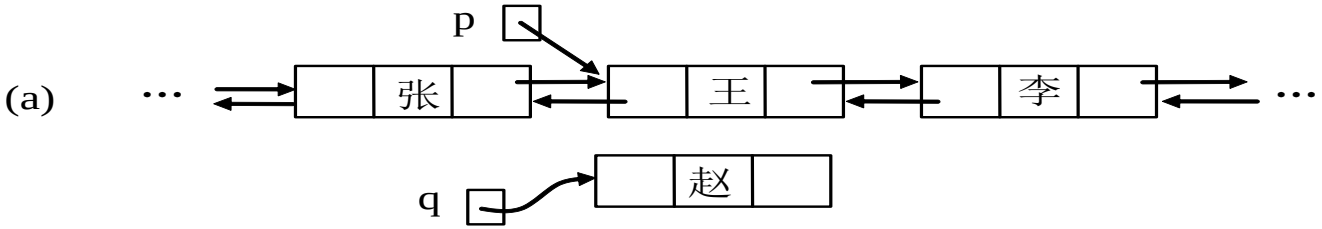
(p->Rlink)->Llink=p

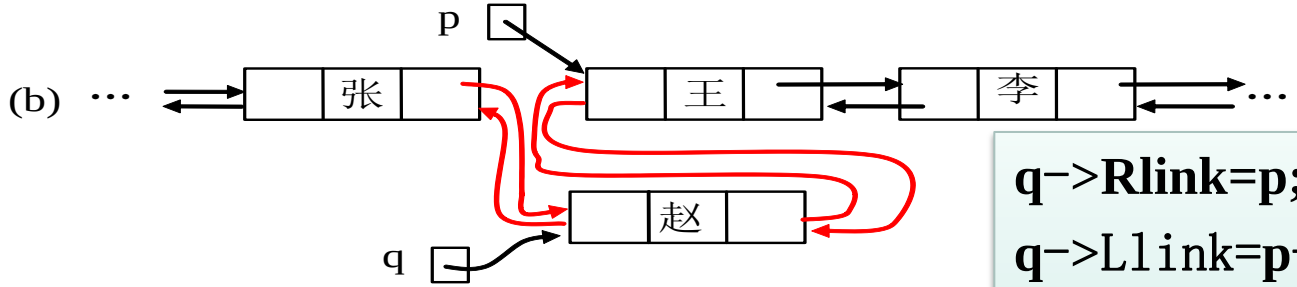
(p->Llink)->Rlink=p



3.双向链表的插入和删除

（2）插在p的左侧






```
q->Rlink=p;
q->Llink=p->Llink;
p->Llink=q;
q->Llink->Rlink=q;
```

插在p的右侧：“Llink”与“Rlink”交换

3.双向链表的插入和删除

（3）删除p所指结点

```
p->Rlink->Llink=p->Llink;
p->Llink->Rlink=p->Rlink;
free(p);
```