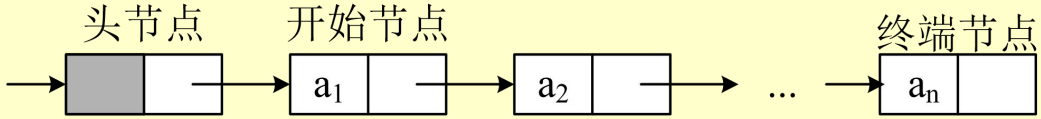
本节主题：

建立单链表

单链表的存储结构




```
typedef char ElemType;  //数据域可以为其他类型
typedef struct LNode   //定义单链表节点类型
{
    ElemType data;     //数据域
    struct LNode *next; //指针域，指向后继节点
} LinkList;
```

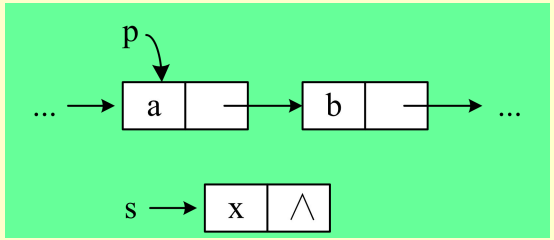
- 在单链表中，由于每个节点只包含有一个指向后继节点的指针，所以当访问过一个节点后，只能接着访问它的后继节点，而无法访问它的前驱节点。

插入节点操作

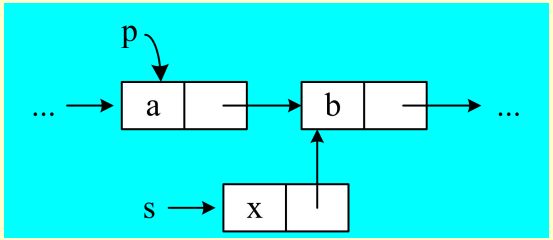
- 要求
  - 将s指向的结点，插入到链表中p指向的结点之后
- 实现方法

```
s->next=p->next;
p->next=s;
```

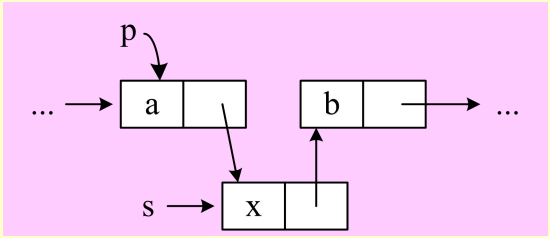
- 特点
  - 只需修改相关节点的指针域，不需要移动节点
- 应用要求
  - 在插入前，应该找到要插入的位置



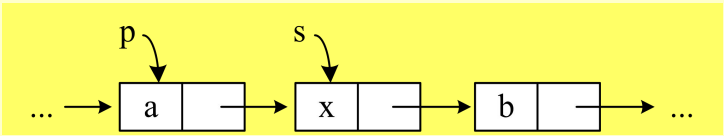
（1）插入前



（2）s->next=p->next



（3）p->next=s



（4）插入后

删除节点操作

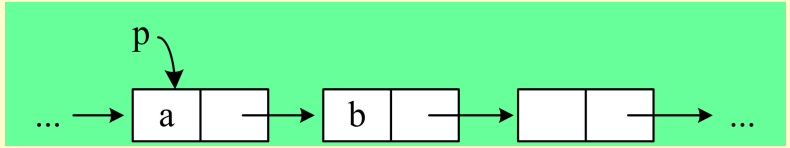
- 要求
  - 将p指向的节点的下一个节点删除
- 实现方法

```
p->next=p->next->next;
```

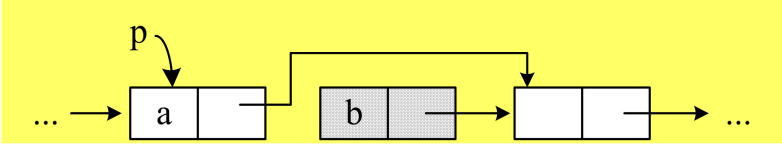
  - 另外的写法

```
q = p->next;
p->next = q->next;
free(q);
```

- 特点
  - 只需修改相关节点的指针域，不需要移动节点
- 应用要求
  - 在删除前，应该找到要删除节点的前一个节点



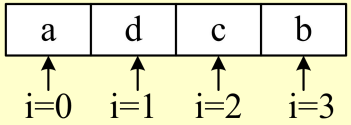
删除前



删除后

头插法建表

- 任务
  - 通过一个含有n个数据的数组来建立单链表
- 头插法
  - 从一个空表开始，读取字符数组a中的字符，生成新节点；
  - 将读取的数据存放到新节点的数据域中，然后**将新节点插入到当前链表的表头上**，直到结束。
- 过程



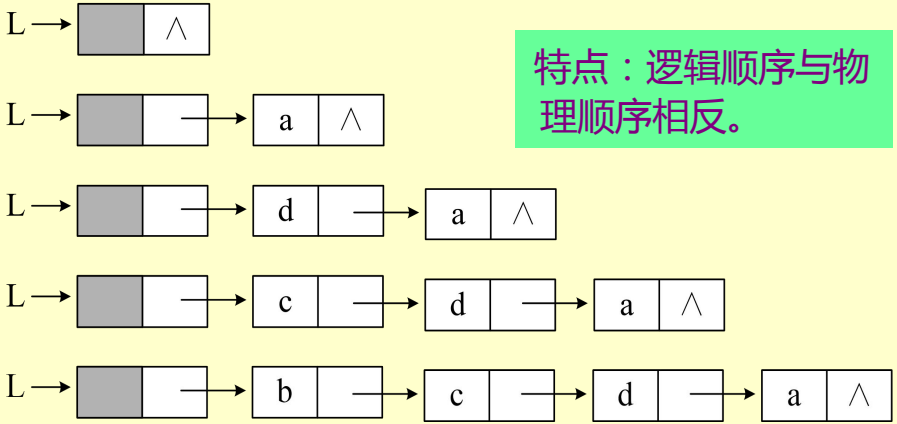
第1步:建头节点

第2步:i=0,新建a节点,插入到头节点之后

第3步:i=1,新建d节点,插入到头节点之后

第4步:i=2,新建c节点,插入到头节点之后

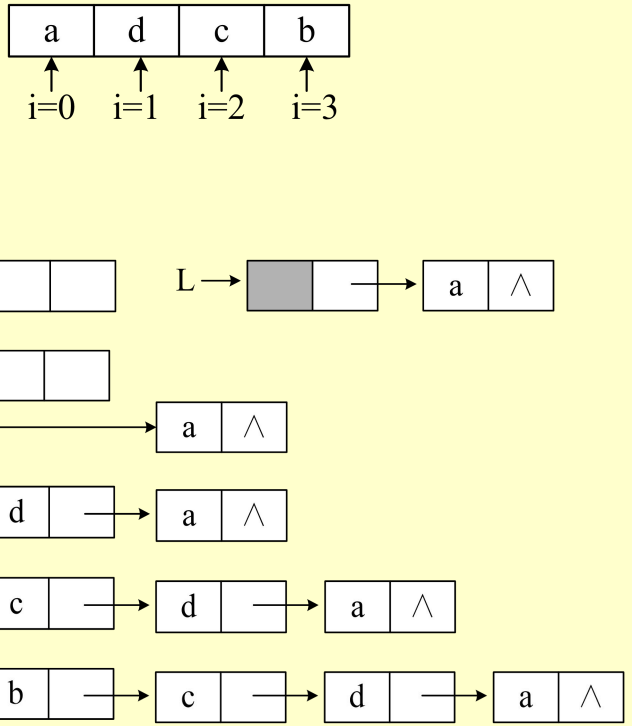
第5步:i=3,新建b节点,插入到头节点之后



特点：逻辑顺序与物理顺序相反。

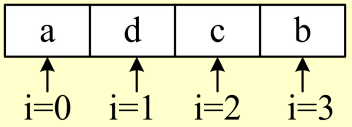
算法实现

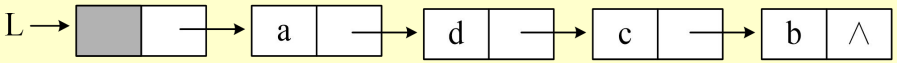
```
void CreateListF(LinkList *&L,ElemType a[],int n)
{
   LinkList *s;
   int i;
   L=(LinkList *)malloc(sizeof(LinkList));
   L->next=NULL;
   for (i=0; i<n; i++)
   {
      s=(LinkList *)malloc(sizeof(LinkList));
      s->data=a[i];
      s->next=L->next;
      L->next=s;
   }
}
```



尾插法建表

- 方法
  - 将新节点插到当前链表的表尾上
- 策略
  - 增加一个尾指针r，使其始终指向当前链表的尾节点。





特点：逻辑顺序与物理顺序相同。

```
void CreateListR(LinkList *&L,ElemType a[],int n)
{
   LinkList *s,*r;
   int i;
   L=(LinkList *)malloc(sizeof(LinkList));
   r=L;
   for (i=0; i<n; i++)
   {
      s=(LinkList *)malloc(sizeof(LinkList));
      s->data=a[i];
      r->next=s;
      r=s;
   }
   r->next=NULL;
}
```