本节主题：

单链表基本操作的实现

ADT ——线性表

ADT List
{
数据对象：
D = {a_i | $a_i \in$ ElemType, i=1,2,...,n, n≧0 } //ElemType为类型标识符
数据关系：
R = {<a_{i-1}, a_i> | a_{i-1}, $a_i \in$ D, i=2,3,...,n }
数据操作：
（1）初始化线性表**InitList(&L)**：构造一个空的线性表L
（2）销毁线性表**DestroyList(&L)**：释放线性表L占用的内存空间
（3）判线性表是否为空表**ListEmpty(L)**：若L为空表，则返回真，否则返回假
（4）求线性表的长度**ListLength(L)**：返回L中元素个数
（5）输出线性表**DispList(L)**:当线性表L不为空时，顺序显示L中各节点的值域
（6）求线性表L中指定位置的某个数据元素**GetElem(L,i,&e)**：用e返回L中第 i 个元素的值
（7）查找元素**LocateElem(L,e)**：返回线性表L中第1个与e相等的序号，找不到返回0
（8）插入元素**ListInsert(&L, i, &e)**：在线性表L中的第i个位置插入元素e；
（9）删除元素**ListDelete(&L, i, &e)**：在线性表L中删除第i个元素，有e返回删除的值；
}

初始化线性表InitList(L)

- 任务
  - 该运算建立一个空的单链表，即创建一个头节点。

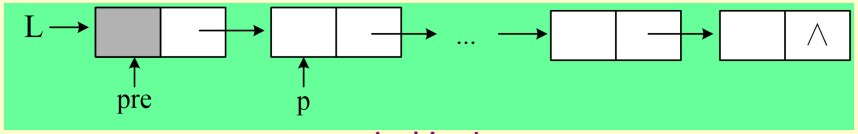
- 算法

```
void InitList(LinkList *&L)
{
    L=(LinkList *)malloc(sizeof(LinkList));
    L->next=NULL;
}
```

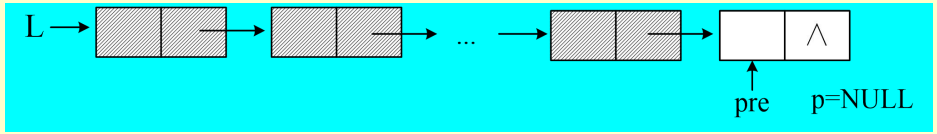
销毁线性表DestroyList(L)

- 任务
  - 释放单链表L占用的内存空间。即逐一释放全部节点的空间。
- 算法

```
void DestroyList(LinkList *&L)
{
    LinkList *pre=L,*p=L->next;
    while (p!=NULL)
    {
        free(pre);
        pre=p;
        p=pre->next;
    }
    free(pre);
}
```



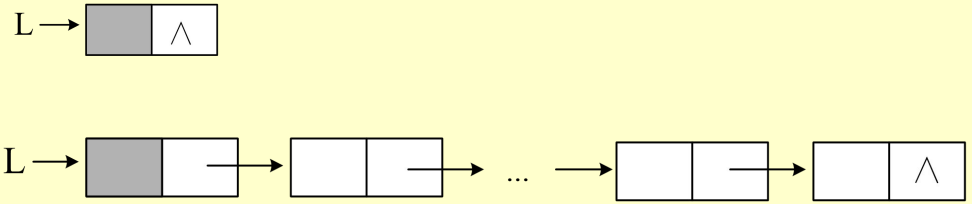
初始时



销毁最后一个节点

判线性表是否为空表ListEmpty(L)

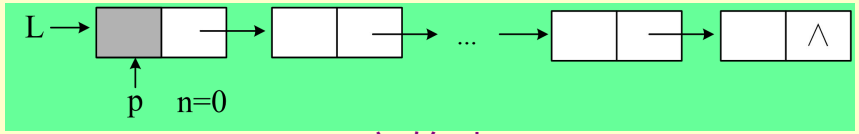
```
bool ListEmpty(LinkList *L)
{
    return(L->next==NULL);
}
```



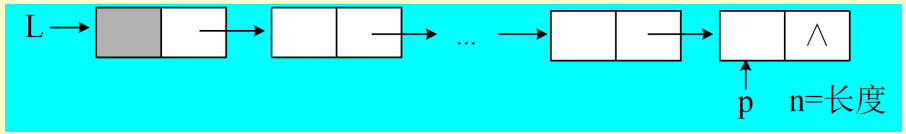
求线性表的长度ListLength(L)

- 返回单链表L中数据节点的个数。

```
int ListLength(LinkList *L)
{
    int n=0;
    LinkList *p=L;
    while (p->next!=NULL)
    {
        n++;
        p=p->next;
    }
    return(n);
}
```



初始时

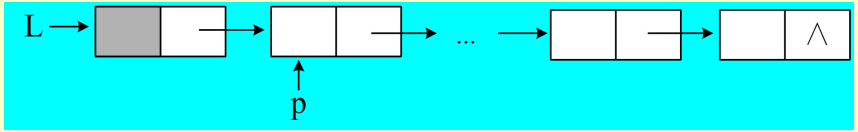


结束时

输出线性表DispList(L)

- 逐一扫描单链表L的每个数据节点，并显示各节点的data域值

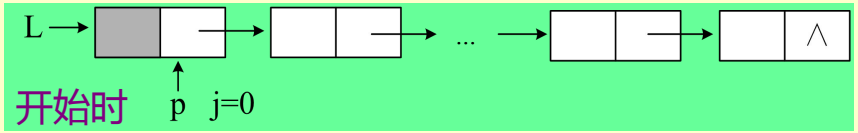
```
void DispList(LinkList *L)
{
    LinkList *p=L->next;
    while (p!=NULL)
    {
        printf("%d ",p->data);
        p=p->next;
    }
    printf("\n");
}
```

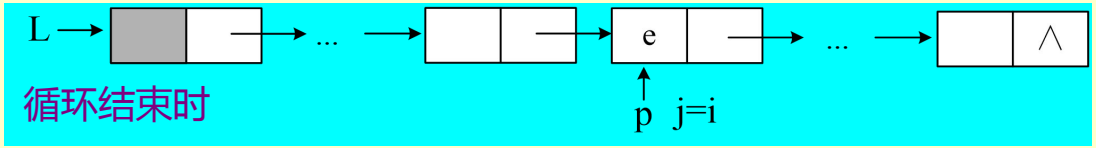


求线性表L中指定位置的某个数据元素GetElem(L,i,&e)

- 思路：在单链表L中从头开始找到第i个节点，若存在第i个数据节点，则将其data域值赋给变量e。

```
bool GetElem(LinkList *L,int i,ElemType &e)
{
    int j=0;
    LinkList *p=L;
    while (j<i && p!=NULL)
    {
        j++;
        p=p->next;
    }
    if (p==NULL)
        return false;
    else
    {
        e=p->data;
        return true;
    }
}
```

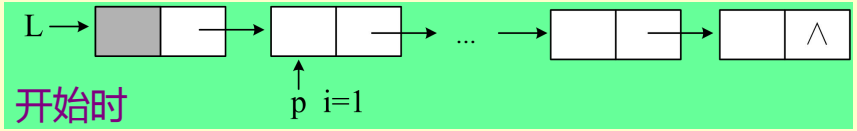


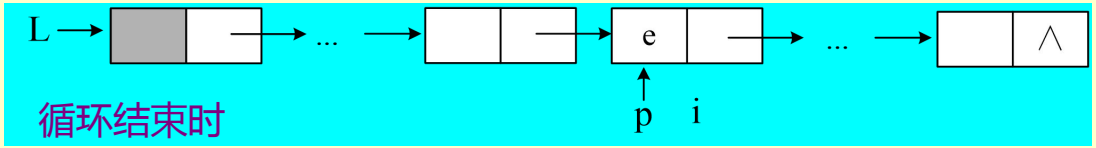


按元素值查找LocateElem(L,e)

- 思路：从头开始找第1个值域与e相等的节点，若存在这样的节点，则返回位置，否则返回0。

```
int LocateElem(LinkList *L,ElemType e)
{
    int i=1;
    LinkList *p=L->next;
    while (p!=NULL && p->data!=e)
    {
        p=p->next;
        i++;
    }
    if (p==NULL)
        return(0);
    else
        return(i);
}
```

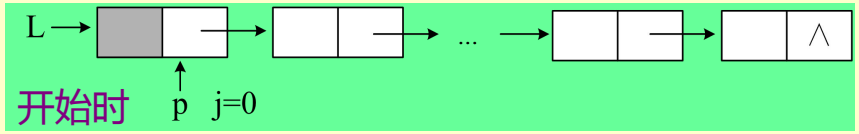


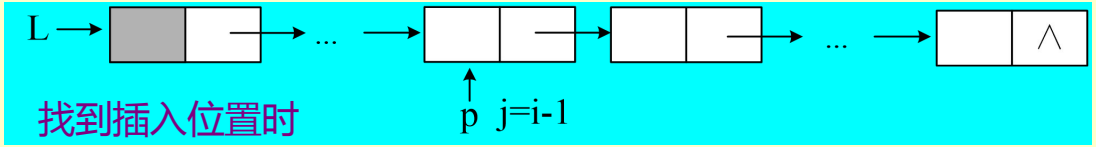


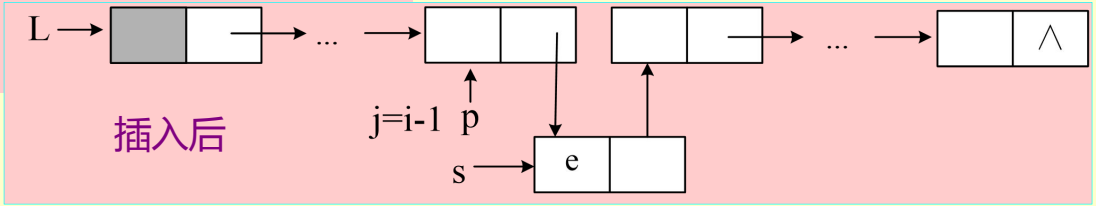
插入数据元素ListInsert(&L,i,e)

- 思路：先在单链表L中找到第i-1个节点 *p，若存在这样的节点，将值为e的节点*s插入到其后。

```
bool ListInsert(LinkList *&L,int i,ElemType e){
    int j=0;
    LinkList *p=L,*s;
    while (j<i-1 && p!=NULL){
        j++;
        p=p->next;
    }
    if (p==NULL)
        return false;
    else{
        s=(LinkList *)malloc(sizeof(LinkList));
        s->data=e;
        s->next=p->next;
        p->next=s;
        return true;
    }
}
```







删除数据元素ListDelete(&L,i,&e)

- 思路：先找到第i-1个节点*p，若存在这样的节点，且也存在后继节点，则删除该后继节点。

```
bool ListDelete(LinkList *&L,int i,ElemType &e){
    int j=0;
    LinkList *p=L,*q;
    while (j<i-1 && p!=NULL) {
        j++;
        p=p->next;
    }
    if (p==NULL)
        return false;
    else{
        q=p->next;
        if (q==NULL)
            return false;
        e=q->data;
        p->next=q->next;
        free(q);
        return true;
    }
}
```