2.3.3 双向链表和双向循环链表

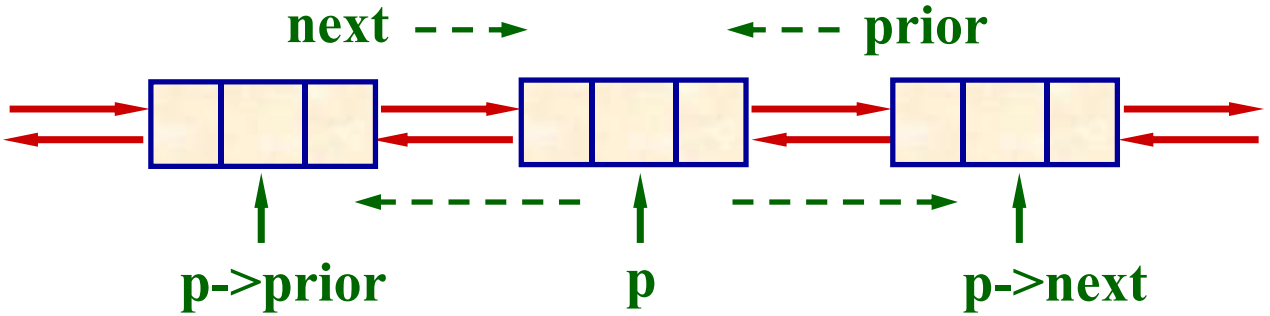
2.3.3.1 双向链表的定义

- 双向链表是指在前趋和后继方向都能遍历的线性链表。双向链表每个结点的结构为：

前驱方向 ← → 后继方向

- 双向链表中用两个指针表示结点间的逻辑关系。
- 指向其前驱结点的指针域prior。
- 指向其后继结点的指针域next。

- 结点指向
  - p->prior 指示结点 p 的前趋结点；
  - p->next 指示结点 p 的后继结点；
  - p->prior->next指示结点 p 的前趋结点的后继结点，即结点 p 本身；
  - p->next->prior指示结点 p 的后继结点的前趋结点，即结点 p 本身。



p->prior->next == p == p->next->prior　　固有特性

- 假设数据元素的类型为ElemType。双链表的类型声明如下：

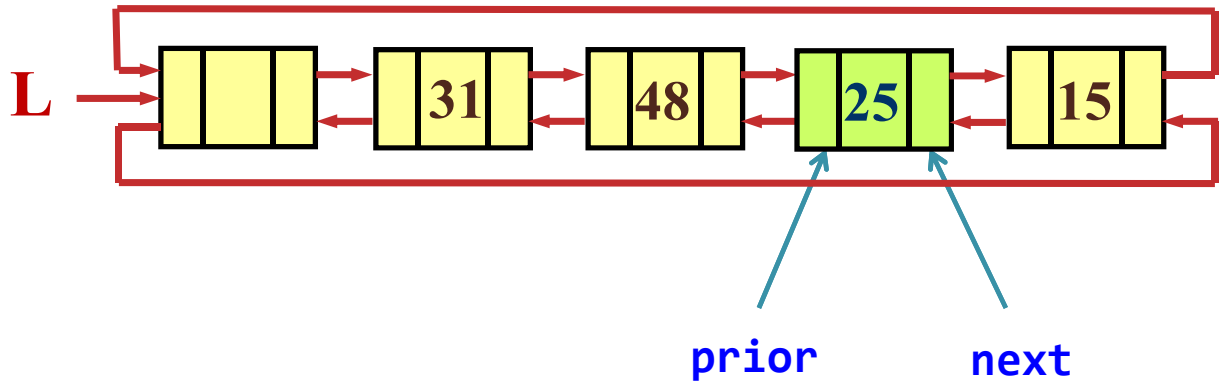
```
typedef struct DuLNode {
  ElemType data;
  DuLNode *prior, *next;
} DuLNode, *DuLinkList;
```

- 与单链表一样，双向链表也分为非循环双向链表（简称为双向链表）和双向循环链表两种。
- 本章所指的双向链表均指带头结点的双向循环链表。

2.3.3.2 双向循环链表上的算法实现

- 在带头结点的双向循环链表中，通常头结点的数据域可以不存储任何信息。

➢ 如下图所示是一个带头结点的双向循环链表。



（1）双向循环链表长度算法

其设计思路与单循环链表的求表长算法完全相同。

```
int ListLength(DuLinkList L)
{
    // 初始条件：L已存在。操作结果：返回L中数据元素个数
    int i = 0;
    DuLinkList p = L->next; // p指向第一个结点
    while(p != L) {    // p没到表头
        i++;
        p = p->next;
    }
    return i;
}
```

（2）双向循环链表插入算法

先在双向循环链表中查找到第i-1个结点，若成功找到该结点（由p所指向），创建一个以e为值的新结点s，将s结点插入到p之后即可。

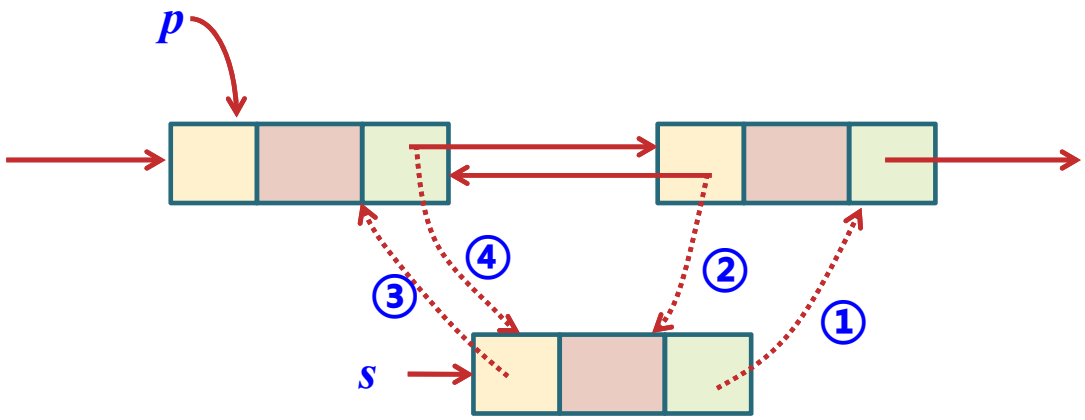
在双向循环链表中*p*结点之后插入*s*结点的操作步骤如下：

① 将结点s的next域指向结点p的下一个结点（ s->next=p->next）。

② 若p不是最后结点（若p是最后结点，只插入s作为尾结点），则将p之后结点的prior域指向s（p->next->prior=s）。

③ 将s的prior域指向p结点（s->prior=p）。

④ 将p的next域指向s（p->next=s）。



在双向循环链表中，可以通过一个结点找到其前驱结点，所以插入操作也可以改为：在双链表中找到第*i*个结点*p*，然后在*p*结点之前插入新结点。

```
Status ListInsert(DuLinkList L, int i, ElemType e)
{
    DuLinkList p, s;
    if(i < 1 || i > ListLength(L) + 1) // i值不合法
        return ERROR;
    p = GetElemP(L, i - 1);          // 在L中确定第i个元素前驱的位置指针p
    if(!p) return ERROR;
    s = (DuLinkList)malloc(sizeof(DuLNode));
    if(!s)
        exit(OVERFLOW);
    s->data = e;
    s->prior = p;                    // 在第i-1个元素之后插入
    s->next = p->next;
    p->next->prior = s;
    p->next = s;
    return OK;
}
```

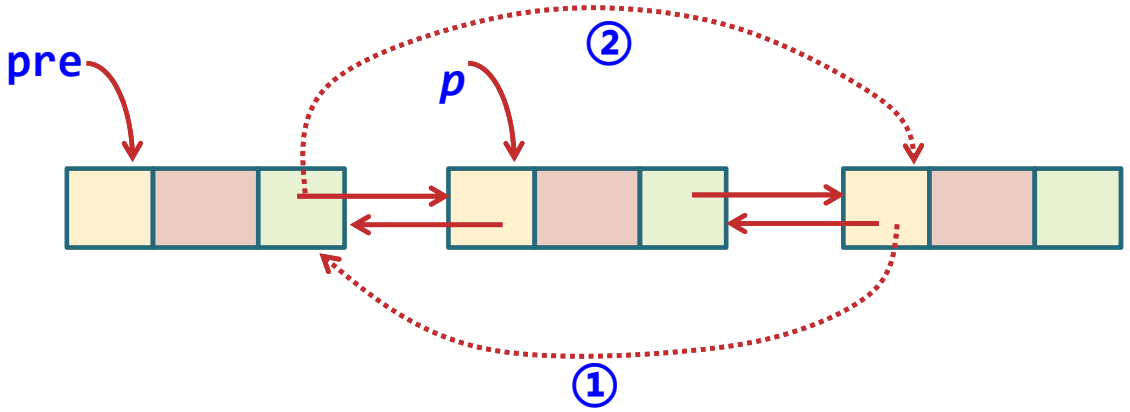
（3）删除结点运算算法

先在双链表中查找到第i个结点，若成功找到该结点（由p所指向），通过前驱结点和后继结点的指针域改变来删除p结点。

在双向循环链表中删除p结点（其前驱结点为pre）的操作：

① 若*p*不是尾结点，则将其后继结点的prior域指向pre结点（*p*->next->prior=pre）。

② 将pre结点的next域改为指向*p*结点的后继结点（pre->next=*p*->next）。



```
Status ListDelete(DuLinkList L, int i, ElemType &e)
{
    DuLinkList p;
    if(i < 1)                          // i值不合法
        return ERROR;
    p = GetElemP(L, i);                // 在L中确定第i个元素的位置指针p
    if(!p)                             // p=NULL,即第i个元素不存在
        return ERROR;
    e = p->data;
    p->prior->next = p->next;
    p->next->prior = p->prior;
    free(p);
    return OK;
}
```

END