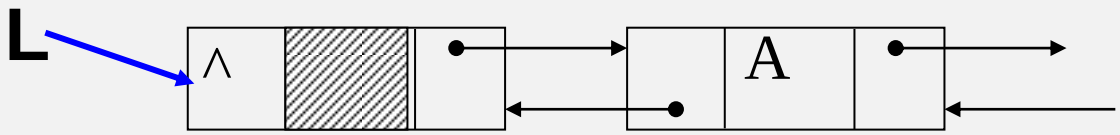
第四章　线性表的链式存储2(续)

- 双向链表的基本算法实现
- 顺序存储与链式存储的比较
- 链式存储的应用举例

4.5 双向链表的基本算法实现

带头结点的双向链表：



双向链表为“空”的条件：

Head->next == NULL

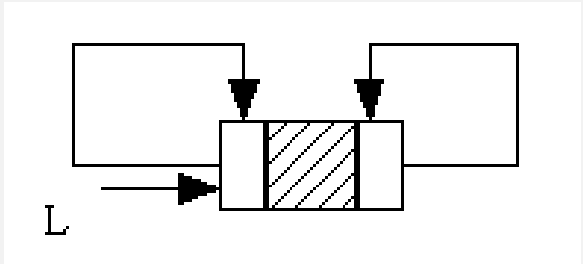
或者 **Head->prior == NULL**

带头结点的双向循环链表：

判断为空表的条件：

Head->next == Head

或者 **Head->prior == Head**



特点：**p->prior->next == p->next->prior == p**

双向链表的C语言定义

- 双向环链表的数据定义：

```
typedef  int  ElemType;   //元素类型定义
typedef struct Dnode {    //Dnode为结点类型预定义符
        ElemType  data;                 //数据域
        struct  Dnode *prior;           //前驱指针域
        struct  Dnode *next;            //后继指针域
} Dnode;                                //结点类型符
typedef struct {                //定义一个双向链表的表结构
        Dnode  *head;                   //表头指针
        Dnode   *tail;          //表尾指针
        int       length;               //当前表长度
}  DLinkStruct, *DLinkList;     //表结点类型和指针类型符
DLinkStruct   DList; //定义一个双向链表为DList的变量
```

带头结点的双向环链表 (建空表)

1、构造一个空的双向链表（带头结点的双向链表）

参数采用变参专递要被初始化的表结构指针。

```
bool  Init_DLinkList(DLinkList  DL )
{   DNode   *p;
    p = (DNode *) malloc( sizeof(DNode) );
    if  (p == NULL ) return FALSE;
    p->next = NULL; p->prior=NULL;
    DL->head = p;
    DL->tail   = p;
    DL->length = 0;
    return  TRUE;
 }
```

调用方式：Init_DLinkList(&DList);　　// Dlist是一个双向链表的表结点

算法复杂度为：O(1)。

带头结点的双向环链表 (表头插入)

2、插入一个元素（在双向链表的表头插入）

```
bool Insert_First(DLinkList DL, ElemType x) // 双向表结构指针，待插元素值
{   DNode  *p;                              //定义临时的元素结点指针变量
    p = ( DNode *)malloc( sizeof(DNode) ); // 动态分配一个结点存储空间
    if （ p==NULL）return FALSE;           // 返回结点分配失败
    p->data = x;                           // 给新结点的数据域填值
    p->next = DL->head->next;              // 填写p的后继指针，指向原来的首结点
    p->prior = DL->head;                   // 填写p的前驱指针，指向头结点
    if  (DL->head->next != NULL)  //插入前是非空表
          DL->head->next->prior = p; //修改首结点的前驱指针
    else   DL->tail = p;                   // 插入前是空表，修改表尾指针
    DL->head->next = p;                    //修改头结点的后继指针
    DL->length++;
    return TURE;
}
```

调用方式： Insert_First(&DList, 80)　其中：Dlist双向链表的表结构变量

算法复杂度为： O(1)

带头结点的双向环链表(删除值x)

3、删除指定元素（查找值为x的元素，并删除之）

```
bool Delete_DList(DLinkList DL, ElemType x)  //表结构指针，待删除元素值
{   Dnode  *p;                    //定义临时的元素结点指针变量
    p = DL->head->next;   // 让p指向第一个结点；
    while( p != NULL）{                  // 链表未结束，则继续
          if ( p->data == x ) break;   // 检查p指向元素的值是否为要查找值
          else  p = p->next;              //  当前不是，则q指向p，p指针后移
    }
    if (p == NULL )  return FALSE;              // 未找到待删除元素，返回失败
    else  { p->prior->next=p->next;             //找到的是链表中元素,摘除链
          if  (p->next) p->next->prior = p->prior;
          else DL->tail=prior;
    }
    free(p);  DL->length--;                 //删除结点，元素计数器减1
    return  TRUE;              // 删除结束，返回成功
}
```

调用方式： Delete_DList(&DList, 80)   其中：Dlist双向链表的表结构变量

算法复杂度为：O(n) ---计算循环的执行频度

带头结点的双向环链表(计算表长)

4、计算线性表的长度(带头结点的双向链表)

```
int  Length_DList(DLinkList  DL)  //表结构指针
{
    return  DL->length;      // 返回双向链表的长度
}
```

调用方式： Length_DList(DList)　其中： Dlist双向链表的表结构变量

算法复杂度为：O(1)

带头结点的双向环链表(表头遍历)

5、遍历线性表元素（从表头开始带头结点的双向链表）

```
void  Show_Dlist_Head(DLinkStruct  &L)   //表结构间接变量
{   DNode  *p;                  //定义临时的元素结点指针变量
    p = L.head->next;                    // 让p指向第一个结点
    printf("\n 线性表元素如下: \n");
    if ( p == NULL) printf("\n 空表！ " );
    while( p != NULL）{                  // 链表未结束，则继续
          printf("  %d ", p->data );    // 输出当前元素的值
          p = p->next;                   //  p指针后移
    }
}
```

调用方式： Show_Dlist_Head(&DList) 其中：Dlist双向链表的表结构变量

算法复杂度为：O(n)

带头结点的双向环链表(表尾遍历)

6、遍历线性表元素（从表尾开始带头结点的双向链表）

```
void  Show_Dlist_Tail(DLinkStruct  &L)     //表结构间接变量
{   DNode  *p;                  //定义临时的元素结点指针变量
    p = L.tail;                 // 让p指向最后一个结点
    printf("\n 线性表元素如下: \n");
    if ( p == L.head ) printf("\n 空表！ " );
    while( p != L.head）{               // 链表未结束，则继续
          printf(“  %d “, p->data );   // 输出当前元素的值
          p = p->prior;                //  p指针前移
    }
}
```

调用方式： Show_Dlist_Tail(&DList)　其中： Dlist双向链表的表结构变量

算法复杂度为：O(n)

4.6 顺序存储与链式存储的比较

链式存储结构的优缺点：

- Advantanges :
  - 存储数据不受位置限制；
  - 插入和删除数据不需要移动其他数据；
  - 可以方便扩充线性表的大小。
- Disadvantages
  - 必须动态分配和释放空间；
  - 不能直接访问数据，因此在需要有效访问元素的算法中，元素不宜用链表存储。

顺序存储结构的优缺点：

优点：

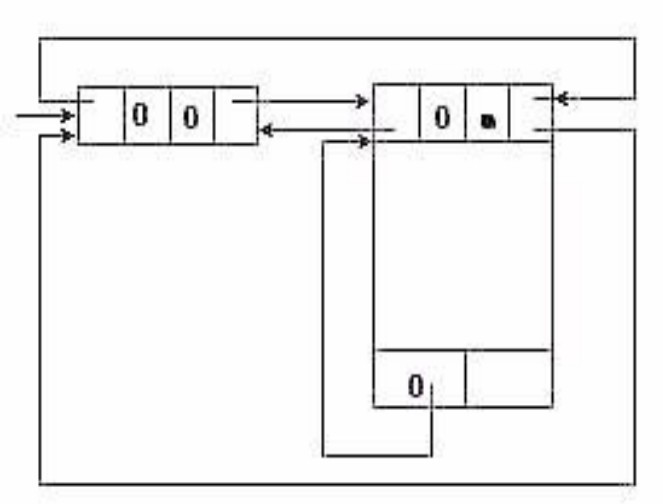
- 数据读取方便，可实现随机存取；
- 操作简单、易于编程实现。

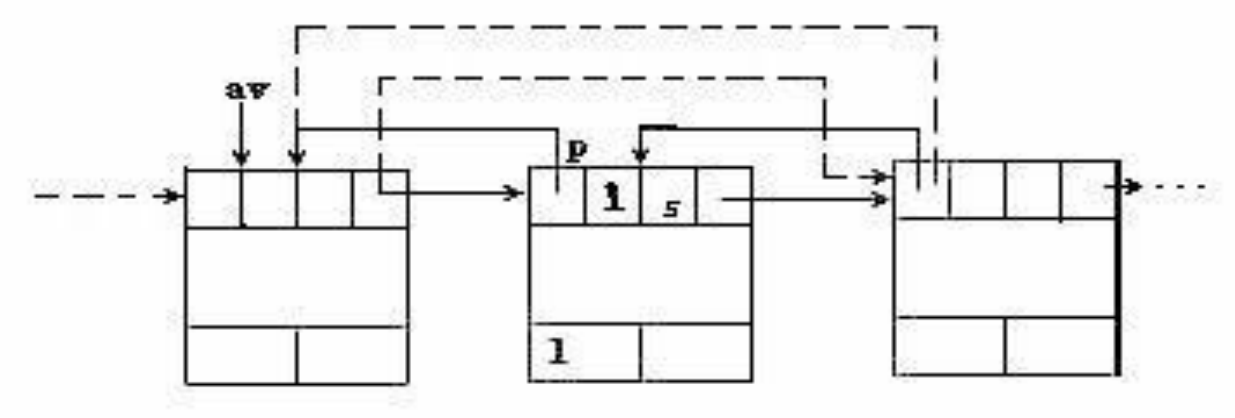
缺点：

- 插入、删除时需要移动数据，效率低；
- 线性表空间有限制，而自动扩展的代价高。

4.7 链式存储的应用举例

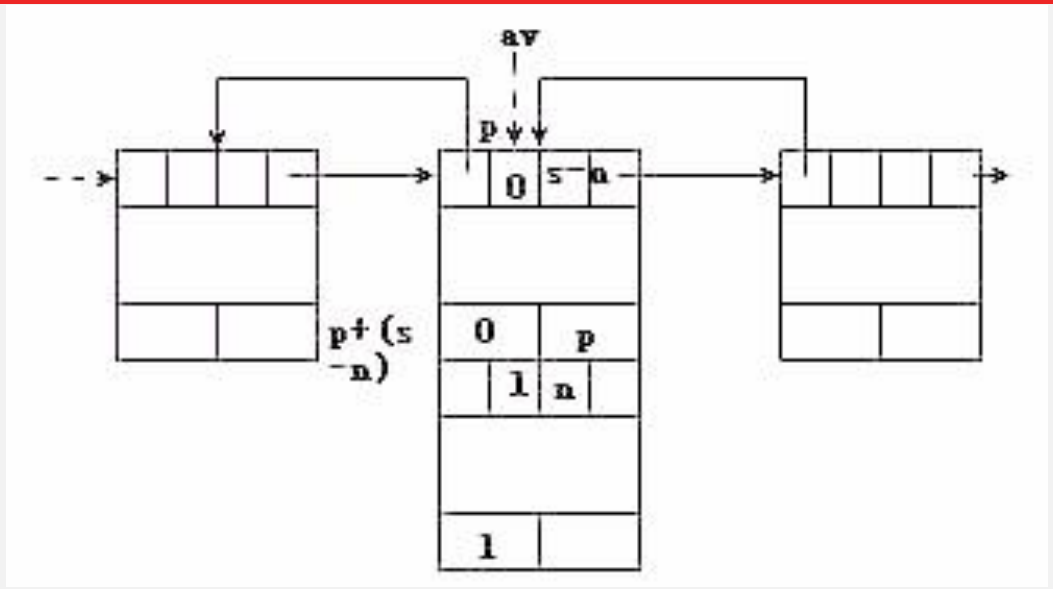
计算机内存空间的管理：

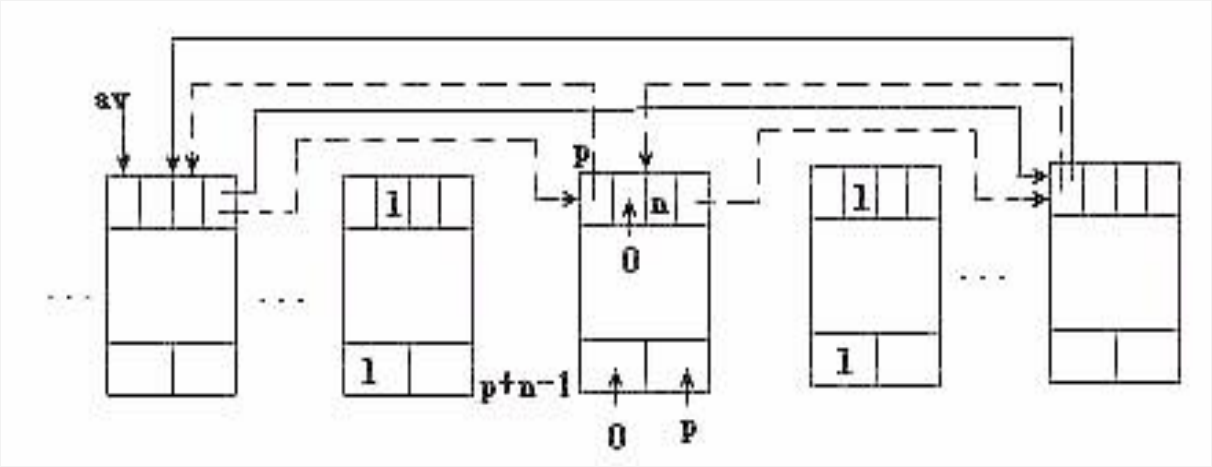




将整个块给用户的（注:虚线为分配后的情况）

4.7 链式存储的应用举例





释放块前、后为分配块的回收情况（注：虚线表示回收后链接的情况。）