# 2 模块 2 线 性 表

**知识目标**

(1)了解线性表的定义,熟练掌握线性表的操作。

(2)掌握线性表的顺序存储。

(3)掌握线性表的链式存储。

线性表是数据结构中最常用和最简单的一种结构,就像幼儿园的老师组织小朋友到教室外活动时,每次都能看到老师带着小朋友们,一个拉着另一个的衣服,依次从教室出来,为了保障小朋友的安全,避免漏掉小朋友,所以给他们安排了出门的次序,事先规定好了谁在谁的前面,谁在谁的后面。这样养成习惯后,如果有谁没有到位,他前面和后面的小朋友就会主动报告老师某人不在。即使在外出到公园或博物馆等情况下,老师也可以很快地清点人数,万一有人走丢,也能在最快时间知道,及时去寻找。这种排好队的组织方式,其实就是我们要介绍的数据结构——线性表。

## 案例导入

在数学上,一个一元多项式 $P_n(x)$可以写成以下表达式。

$$P_n(x)=P_0+P_1(x)+P_2(x^2)+\cdots+P_n(x^n)$$

可以完成相加、相减、相乘等基本运算,那么要想在计算机中完成这些基本运算,首先要确定如何在计算机中表示一元多项式。该一元多项式是由 $n+1$ 个系数唯一确定的,因此,在计算机中,它可用一个线性表 $P$ 来表示:$P=(p_0,p_1,p_2,\cdots,p_n)$。

每一项的指数 $i$ 隐含在其系数 $P_i$ 的序号中。

方法 1:采用顺序存储结构直接表示。数组各分量对应一元多项式各项的系数 $a_i$。

例如,一元多项式 $P_1(x)=3+4x+2x^3+x^5$ 可以表示为

| 下标 | 0 | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|---|
| a[i] | 3 | 4 | 0 | 2 | 0 | 1 |

假设 $Q_m(x)$是一元 $m$ 次多项式，同样可用线性表表示为 $Q=(q_0,q_1,q_2,\cdots,q_m)$，不失一般性。若 $m<n$，则两个多项式相加的结果 $R_n(x)=P_n(x)+Q_m(x)$可用线性表 $R$ 表示为$R=(p_0+q_0,p_1+q_1,p_2+q_2,\cdots,p_m+q_m,p_{m+1},\cdots,p_n)$。

显然，采用顺序存储结构，使得多项式相加的算法定义十分简洁。然而，在通常的应用中，多项式的次数可以很高且变化很大，这样使得顺序存储结构的最大长度很难确定。

例如，$P_2(x)=x-5x^{3000}$可以表示为

| 下标 | 0 | 1 | 2 | 3 | … | 3 000 |
|---|---|---|---|---|---|---|
| a[i] | 0 | 1 | 0 | 0 | 0 | −5 |

显然，在处理这样的多项式时，就要用长度为 3001 的线性表来表示，表中仅有两个非零元素，这就会造成巨大的空间浪费，这种对内存空间的浪费是应当避免的。

方法 2：采用顺序存储结构表示非零项。每个非零项 $p_ix^i$ 包括两个信息：系数 $p_i$ 和指数 $x^i$，用结构数组来表示。

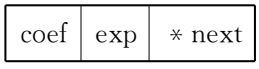
例如，一元多项式 $P_1(x)=3+4x+2x^3+x^5$ 可以表示为

| 下标 | 0 | 1 | 2 | 3 |
|---|---|---|---|---|
| 系数 | 3 | 4 | 2 | 1 |
| 指数 | 0 | 1 | 3 | 5 |

而 $P_2(x)=x-5x^{3\,000}$可以表示为

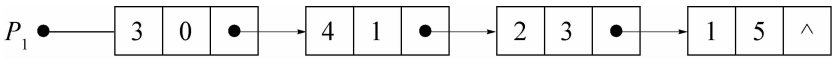
| 下标 | 0 | 1 |
|---|---|---|
| 系数 | 1 | −5 |
| 指数 | 0 | 3 000 |

方法 3：采用链表结构存储表示非零项。在链表中，每个结点包括系数和指数两个数据域，还要包括一个指针域(链域)。

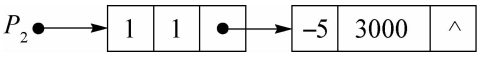



```
Typedef struct PolyNode{
  int coef;   //系数
  int exp;    //指数
  struct PolyNode * next;   //指针域，指向下一个结点
}
```

例如，一元多项式 $P_1(x)=3+4x+2x^3+x^5$ 可以表示为



而 $P_2(x)=x-5x^{3000}$ 可以表示为




## 案例分析

通过案例导入可以看出，对于同一个问题可以有不同的存储方法，如对于一元多项式可以采用顺序存储结构和链式存储结构，在实际的应用程序设计中如何选择，则要视多项式做何种运算而定。若只对多项式进行“求值”等不改变多项式的系数和指数的运算，则采用类似于顺序表的顺序存储结构即可，否则应采用链式存储表示。但不管采用哪种方法，它们都有一个共性的问题：我们的目标都是实现有序线性序列的组织与管理，这些问题都可以抽象为线性表的问题，本模块将学习线性表的相关知识。

## 相关知识

线性表是线性结构的一种表现形式。它的特点是数据的元素在非空有限集{$\Phi$}中存在唯一的一个被称为“第一个”的数据元素，并存在唯一的一个被称为“最后一个”的数据元素。除第一个元素之外，集合中的每个数据元素均只有唯一的一个前驱；除最后一个元素之外，集合中的每个数据元素均只有唯一的一个后继。

# 2.1 线性表的定义与操作

### 2.1.1 线性表的定义

**定义** 一个线性表是 $x$ 个数据元素的有限序列，如$(m_1,m_2,m_3,m_4,\cdots,m_i,\cdots,m_x)$。

**【例 2-1】** 阿拉伯数字表(0,1,2,3,4,5,6,7,8,9)。

**【例 2-2】** 图书管理系统中的图书信息表如表 2-1 所示。

表 2-1 图书信息表

| 序　　号 | 图书编号 | 图书名称 | 出 版 社 |
|---|---|---|---|
| 201801 | 9787517051718 | C 语言程序设计 | 中国水利水电出版社 |
| 201802 | 9787517051497 | C 语言程序设计习题与实验指导 | 中国水利水电出版社 |
| …… | …… | …… | …… |

从上面的例子中不难看出，线性表的特征如下。

元素的个数 $x$ 表示线性表的长度 $S$。

(1)当 $x=0$ 时，表示当前线性表为空。

(2)当 $1<i<x$ 时，表示在当前线性表中，$m_1$ 是线性表的头结点，$m_x$ 是线性表的尾结

点。也就是说，$m_1$ 没有前驱结点，$m_x$ 没有后继结点。而 $m_i$ 的前驱结点为 $m_{i-1}$，$m_i$ 的后继结点为 $m_{i+1}$。

## 2.1.2 线性表的操作

对于线性表的操作，我们可以分为线性表的基本操作、线性表的引用型操作（线性表的结构不发生变化）及线性表的加工型操作（线性表的结构发生变化）。下面分别针对不同的操作进行讲解。

### 1. 基本操作

（1）初始化操作：InitList(&L)。

操作结果：创造一个空的线性表 L。

（2）结构清空操作：DestroyList(&L)。

初始条件：线性表 L 已经存在。

操作结果：清空线性表 L 的线性结构。

### 2. 引用型操作（线性表的结构不发生变化）

（1）判断线性表是否为空：ListEmpty(L)。

初始条件：线性表 L 已经存在。

操作结果：若线性表 L 为空表，则返回 T，否则返回 F。

（2）求线性表的长度：ListLength(L)。

初始条件：线性表 L 已经存在。

操作结果：返回线性表 L 中数据元素的个数。

（3）求数据元素的前驱：PriorElem(L, cur_n, &pre_n)。

初始条件：线性表 L 已经存在。

操作结果：若 cur_n 是线性表 L 中的数据元素，但不是第一个，则用 pre_n 返回它的前驱结点，否则操作失败，pre_n 没有定义。

（4）求数据元素的后继：NextElem(L, cur_n, &next_n)。

初始条件：线性表 L 已经存在。

操作结果：若 cur_n 是线性表 L 中的元素，但不是最后一个，则用 next_n 返回它的后继结点，否则操作失败，next_n 无定义。

（5）求线性表中某个位置的数据元素：GetElem(L,x, &n)。

初始条件：线性表 L 已经存在并且 $1 \leqslant x \leqslant$ LengthList(L)。

操作结果：用 $n$ 返回线性表 L 中的第 $x$ 个数据元素的值。

（6）定位数据元素函数：LocateElem(L, $x$)。

初始条件：线性表 L 已经存在，$x$ 为给定值。

操作结果：返回 L 中第 1 个与 $x$ 满足关系的元素的位置序列。若与 $x$ 数据元素相同的数据元素不存在，则返回值为 0。

（7）遍历线性表：ListTraverse(L)。

初始条件：线性表 L 已经存在。

操作结果：按顺序依次输出线性表 L 中的每个数据元素。

### 3. 加工型操作（线性表的结构发生变化）

(1)线性表置空：ClearList(&L)。

初始条件：线性表 L 已经存在。

操作结果：将 L 重置为空表。

(2)数据元素赋值：PutElem(&L，x，&n)。

初始条件：线性表 L 已经存在。

操作结果：给线性表 L 当中第 $x$ 个元素赋值为 $n$ 的值。

(3)插入数据元素：ListInsert(&L，x，n)。

初始条件：线性表 L 已经存在。

操作结果：在线性表 L 的第 $x$ 个元素之前插入新的数据元素 $n$，线性表 L 的长度增 1。

(4)删除数据元素：ListDelete(&L，x，&n)。

初始条件：线性表 L 已经存在且非空，$1 \leqslant x \leqslant$ LengthList(L)。

操作结果：删除线性表 L 的第 $x$ 个数据元素，并用 $n$ 来返回其数值，线性表 L 的长度减 1。

可以根据上面对于线性表的定义，去实现其他更为复杂的设计。

**【例 2-3】** 假设存在两个集合 $Q$ 和 $P$，分别用两个线性表 $L_Q$ 和 $L_P$ 表示，即线性表中的数据元素即为集合中的成员。现要求一个新的集合 $Q=Q \cup P$。

这个问题的思路为：首先要扩大线性表 $L_Q$ 的存储空间，将存在于线性表 $L_P$ 中而不存在于线性表 $L_Q$ 中的每一个数据元素插入线性表 $L_Q$ 中去。

对线性表进行如下操作。

(1)从线性表 $L_P$ 中依次对比每个数据元素：GetElem(LP，i)→X。

(2)依次在线性表 $L_Q$ 中进行查访：LocateElem(LQ，X，equal())，其中，equal()函数用来实现元素的比较操作。

(3)若不存在相同元素，则插入：ListInsert(LQ，N+1，X)。

算法如下。

```
void listadd(List &LQ, List LP){
    LQ_len = ListLength(LQ);     // 求线性表 LQ 的长度
    LP_len = ListLength(LP);     // 求线性表 LP 的长度
    for(n=1;n<=LP_len;n++)
    {
        GetElem(LP, n, x); // 取线性表 LP 中第 i 个数据元素赋给 x
        if (!LocateElem(LQ, x))
          ListInsert(LQ, ++LQ_len, x);//如果线性表 LQ 中不存在和线性表 LP 中的
                                      //数据元素 x 相同的数据元素，则插入线性表 LQ 中
      }
}
```

# 2.2 线性表的顺序存储

## 2.2.1 顺序表

用一组地址连续的存储单元依次存放线性表中的数据元素，该表就称为顺序表。顺序表示意图如图 2-1 所示。

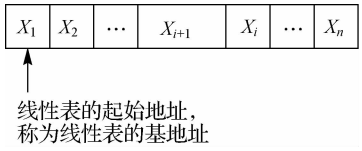



图 2-1 顺序表示意图

顺序表的描述如下。

```
#define LIST_INIT_SIZE 100          //给线性表的存储空间分配初始大小
#define LISTINCREMENT 20            //给线性表的存储空间定义分配自动增长量
typedef struct{
   ElemType *elem;                  //定义存储空间基地址
   int length;                      //定义当前长度
   int listsize;                    //定义当前分配的存储容量
}SqList;                            //顺序表结构定义
```

## 2.2.2 顺序表上基本运算的实现

(1)初始化操作：InitList(&L)。

初始条件：顺序表 L 不存在。

操作结果：创造一个空的顺序表 L。

(2)查找数据元素函数：LocateElem(L，x)。

初始条件：顺序表 L 已经存在，$x$ 为给定值。

操作结果：返回 L 中第 1 个与 $x$ 满足关系的元素的位置序列。若与 $x$ 数据元素相同的数有多个，则只返回第一个相同的数据元素的位置。

(3)插入数据元素：ListInsert(&L，x，n)。

初始条件：顺序表 L 已经存在。

操作结果：在顺序表 L 的第 $x$ 个元素之前插入新的数据元素 $n$，顺序表 L 的长度增 1。

(4)删除数据元素：ListDelete(&L，x，&n)。

初始条件：顺序表 L 已经存在且非空，$1 \leqslant x \leqslant$ LengthList(L)。

操作结果：删除顺序表 L 的第 $x$ 个数据元素，并用 $n$ 来返回其数值，顺序表 L 的长度减 1。

## 2.2.3 顺序表基本运算的算法

**1. 初始化操作**

```
void InitList_Sq(SqList& L){                          //构造一个空的线性表
    L.elem=new ElemType[LIST_INIT_SIZE];
    if(!L.elem)
        exit(ERROR);
    L.length=0;                                       //定义顺序表的当前长度
    L.listsize=LIST_INIT_SIZE;
    L.incrementsize=LISTINCREMENT;                    //自动增加补充空间容量
}
```

**2. 查找顺序表中的数据元素函数**

```
int LocateElem_Sq(SqList L, ElemType x){
                                  //在顺序表中查询第一个满足给定条件的数据元素
                                  //若存在,则返回它的位序,否则返回 0
    n = 1;                        //n 的初值为顺序表中第 1 个元素的位置序列
    p = L.elem;                   //p 的初值为顺序表中第 1 个元素的存储位置
    while (n<= L.length && *p++!=x)
        ++i; //依次对顺序表中的每个数据元素进行判定
    if(i <= L.length)
        return i;
    else
        return 0;
}
```

**3. 插入数据元素：ListInsert(&L, x, n)**

```
void ListInsert_Sq(SqList &L, int n, ElemType x){
                                  //在顺序表 L 中的第 n 个元素之前插入新的元素 x
                                  //n 的合法范围为 1≤n≤L.length+1
    q = &(L.elem[i-1]);           //q 指示插入位置
    for(p=&(L.elem[L.length-1]);p>= q;--p)
    *(p+1) = *p;                  //顺序表中插入位置之后的数据元素右移
    *q = x;                       //插入数据元素 x
    ++L.length;                   //顺序表长度增 1
}
```

**4. 删除数据元素：ListDelete(&L, n, &x)**

```
void ListDelete_Sq(SqList &L, intn, ElemType &x){
    if(n<1)||n>L.length))
```

```
        return ERROR;              //删除位置不合法
    p=&(L.elem[i-1]);              //*p 为被删除元素的位置
    x=*p;                          //被删除的数据元素的值赋给 x
    q=L.elem+L.length-1;           //顺序表中表尾元素的位置
    for(++p;p<=q;++p)
        *(p-1)=*p;                 //顺序表中被删除的数据元素之后的元素左移
    --L.length;                    //顺序表长度减 1
}
```

## 2.3 线性表的链式存储

上一节讲了线性表中的一种存储表达形式——顺序表，现在来研究线性表的另一种存储表达形式——链表。

首先来看看现实世界里有没有链表的实际应用。

如今流行的区块链就是链式存储的一种形式。从广义上来讲，区块链技术是利用链式数据结构来验证与存储数据，并利用分布式结点共识算法来生成和更新数据的。

### 2.3.1 线性单链表

用一组地址任意的存储单元存放线性表中的数据元素的线性表称为链表。单链表的每个结点由一个数据域和一个指针域组成，结点中存放数据元素信息的域称为数据域，存放其后继地址的域称为指针域。单链表单个结点的存储结构如图 2-2 所示。

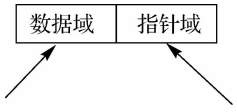



图 2-2 单链表单个结点的存储结构

从图 2-2 可知，单链表的存储结构由两部分组成，也就是一个结点应该由两部分组成，即

（数据元素的映像）+（指示后继数据元素的存储地址）=结点

在单链表中，我们发现，两两连接的单链表都需要用指针来实现其操作，在单链表中头指针指向链表中第一个结点（头结点或无头结点时的开始结点）的指针。头结点就是在开始结点之前附加的一个结点。在这里需要注意的是，开始结点是在链表中，存储第一个数据元素（$a_1$）的结点。单链表头结点结构如图 2-3 所示。

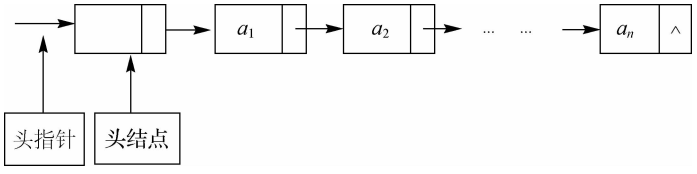



图 2-3 单链表头结点结构

在线性表中存放第一个数据元素 $a_1$ 的存储地址的结点，称为线性表的头结点。但是要注意的是：由于顺序表中的每个元素至多只有一个前驱结点和一个后继结点，即数据元素之间是一对一的逻辑关系。所以当进行链式存储时，一种最简单也是最常用的方法是：在每一个数据结点中包括两个部分，除了一个是数据域之外，还有一个是指针域，用来指向其后面链接的数据结点。这样构成的链接表称为线性表的单向链接表，简称单链表。另一种可以采用的方法是：在每个数据结点中除了包含数据域之外，还包含两个指针域，分别放在数据域的前、后，用于指向前驱结点和后继结点，这样构成的链接表称为线性双向链接表，简称双向链表。双向链表结构如图 2-4 所示。

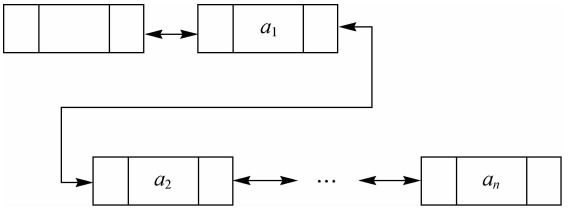



图 2-4　双向链表结构

在单链表中，因为每一个数据结点只包含一个指向后继结点的指针，所以当访问过一个结点之后，只能接着访问其本身的后继结点，却无法再次访问其本身的前驱结点。而双链表则可以实现这个功能，因为每个数据结点既包含一个指向其本身的后继结点的指针，还包含一个指向其本身的前驱结点的指针。所以当访问过一个结点，还要继续再次访问该数据结点之前的数据结点时，就可以轻松地依次访问到，也可以继续访问该数据结点之后的每一个数据结点。

在线性表的链式存储方式中，存储第一个数据元素的结点称为表头结点（简称头结点），存储最后一个数据元素的结点称为表尾结点（简称尾结点）。其他均称为中间结点。每个链表都需要设置一个指针用于指向表头结点，我们称该指针为表头指针。虽然表头指针只指向表头结点，但从表头指针开始，沿着数据结点的链可以访问到每一个数据结点，所以通常就以表头指针命名一个链表。如图 2-5 所示，单链表的表头指针为 head，双链表的表头指针为 dhead，故分别称它们为 head 单链表和 dhead 双链表。

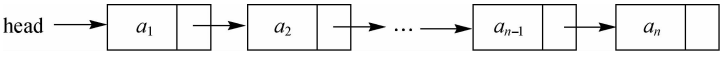



图 2-5　单链表

## 2.3.2　线性表上基本运算的实现

### 1. 单链表的基本操作

(1)求线性表的长度：ListLength_L。

初始条件：链表 L 存在。

操作结果：返回单链表 L 的长度值。

(2)查找元素：LocateElem_L。

初始条件:链表 L 存在。

操作结果:查找链表 L 中与指定的数据元素相同的第一个值。若存在,则返回它在链表中的位置,否则返回 NULL。

(3)插入结点:ListInsert_L。

初始条件: 链表 L 存在。

操作结果:在指定位置插入新的数据元素结点。

(4)删除结点:ListDelete_L。

初始条件:链表 L 存在。

操作结果:删除指定位置的数据结点。

**2. 单链表基本运算的算法**

(1)求线性表的长度:ListLength_L。

```
int ListLength_L(LinkList L){
    p=L;
    k=0;
    while(p){
        k++;
        p=p->next;          //k 计非空结点数
    }
    return k;
}
```

(2)查找元素:LocateElem_L。

```
LNode * LocateElem_L(LinkList L,ElemType e){
    p=L;
    while(p && p->data != e) p=p->next;
    return p;
}
```

(3)插入结点:ListInsert_L。

```
void ListInsert_L(LinkList &L, Lnode * p,Lnode * s){
    if(p == L)
        s->next=L;L=s;
    else{
        q=L;
        while(q->next !=p) q=q->next;     //查找 p 的前驱结点 q
        q->next = s;s->next=p;  //在链表中的 q 结点之后插入 s 结点
    }
}
```

(4)删除结点:ListDelete_L。

```
void ListDelete_L(LinkList &L, Lnode * p, ElemType &e){
```

```
    if(p == L)
        L = p->next;
    else{
        q = L;
        while(q->next != p) q = q->next;      //查找 p 结点的前驱结点 q
        q->next = p->next;                    //修改 q 结点的指针域
    }
    e = p->data;delete p;
}
```

在单链表中删除数据结点应该注意如下几个问题。

(1)单链表为空时不能进行删除，系统提示相应信息。

(2)单链表不为空但删除的是头结点时，不仅要删除该结点，还要修改单链表的表头指针位置，即 head=head<-next。

(3)单链表不为空但删除的是尾结点时，由指针 p 指向单链表中的倒数第二个结点，并把该结点的地址指针设置为 NULL，即 p->next=NULL。

### 2.3.3 其他形式的链表

#### 1. 循环链表

循环链表是单链表的一种变形形式，其和单链表的差别在于：单链表中的最后一个结点，即后继结点的判断条件是地址域为空，而循环链表判断链中最后一个结点的条件不再是“后继结点的地址域是否为空”，而是后继结点的地址域是否为头结点的地址。

为了方便记忆，我们可以理解为：最后一个数据结点的指针域的指针又重新指回第一个数据结点的链表，如图 2-6 所示。

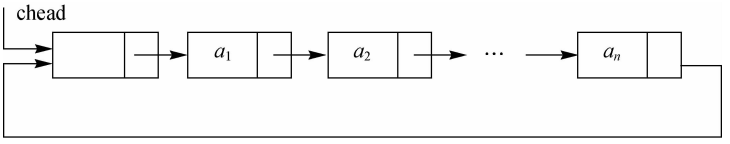



图 2-6 循环链表表示形式

循环链表的基本操作如下。

(1)新建循环链表：ListLength_C。

初始条件：循环链表 C 不存在。

操作结果：建立循环链表 C。

(2)插入结点：ListInsert_C。

初始条件：循环链表 L 存在。

操作结果：在指定位置插入新的数据元素结点。

(3)删除结点：ListDelete_L。

初始条件：循环链表 L 存在。

操作结果：删除指定位置的数据结点。

循环链表的基本运算的算法如下。

(1)新建循环链表：ListInsert_C。

```
void InitList(LinkList—L * &L) //新建一个只含头结点的空循环链表
{
    L=(LinkList * )malloc(sizeof(LinkList));
    L—>data=0;
    L—>next=L;
}
```

(2)插入结点：ListInsert_C。

```
void ListInsertTail(LinkList—L * &L,int x){        //采用尾插法插入数据 x
    LinkList * p=NULL, * q=NULL;
    q=(LinkList * )malloc(sizeof(LinkList));
    p=L;
    if(L—>data==0)//只有一个空结点时
    {
        L—>data=x;
        return ;
    }
    while(p—>next!=L){
        p=p—>next;
    }
    q—>data=x;
    q—>next=L;
    p—>next=q;
}
```

(3)删除结点：ListDelete_L。

```
void DestroyList(LinkLisL—lt * &L){
    LinkList * p=NULL;
    while(L—>next!=L)
    {
        p=L—>next;
        L—>next=L—>next—>next;
        free(p);
    }
    free(L);
}
```

### 2. 双向链表

双向链表中的每个数据结点包括两个地址域指针和一个数据域指针。两个地址域指针分别指向其直接后继结点和直接前驱结点，如图 2-7 所示。

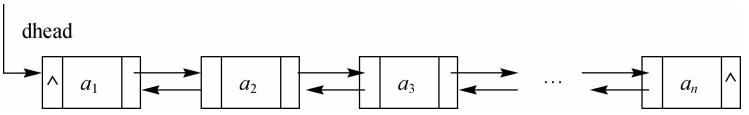



图 2-7　双向链表

由于设置了双向链表的前驱指针，如果需要前驱的位置信息，可直接使用，避免了遍历操作时的时间消耗。在双向链表中插入一个数据元素的操作如图 2-8 所示。

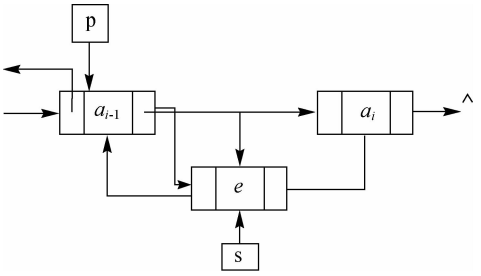



图 2-8　双向链表的插入操作

在双向链表中，运用指针来实现数据插入操作。其核心语句如下。

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

如果要删除一个数据元素，那么又该如果操作呢？

如图 2-9 所示，在双向链表中，运用指针来实现对指定数据的删除操作。其核心语句如下。

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

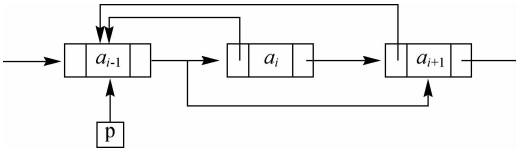



图 2-9　双向链表的删除操作

## 案例实施

用线性表实现一元多项式的存储，然后完成两个一元多项式的运算，如$(6+4x-7x^2)+(23-16x+5x^2)$。类似地，还有减法、乘法和除法运算。多项式结构比较合适。

以加法为例，对于两个一元多项式中所有指数相同的项，对应系数相加，若和不为零，则构成“和多项式”中的一项；对于两个一元多项式中所有指数不相同的项，则分别插入和“多项式”中。对于产生的系数为 0 的项，则舍弃。因此，一元多项式的加法运算就是有序链表的合并问题。

对于数据结构设计，一个一元多项式的每一个子项都由系数和指数两部分组成，因此可将其抽象为由系数 coef、指数 exp、指针域 next 构成的链式线性表。将两个多项式分别存放在两个线性表中，然后经过相加后将所得多项式存放在一个新的线性表中，但是不用再开辟新的存储空间，只依靠结点的移动来构成新的线性表，其间可以将某些不需要的空间回收。基于这样的分析，可以采用不带头结点的单链表来表示一个一元多项式。具体数据类型定义如下。

```
typedef struct LNode{
  float coef;                    //系数
  int exp;                       //指数
  struct LNode * next;
}LNode, * NodeList;
```

## 案例总结

用线性表的存储形式实现两个一元多项式的运算，首先要考虑多项式的存储形式。若只对多项式进行“求值”等不改变多项式系数和指数的运算，则可以采用顺序表形式实现，否则采用链式存储表示；但是在加法、减法、乘法等运算过程中，经常要增加新项，删除系数为 0 的项，所以采用链式结构比较合适。

## 思考与练习

**1. 选择题**

(1)对于线性表最常用的操作是查找指定序号的元素和在末尾插入元素，则选择(　　)最节省时间。

A. 顺序表

B. 带头结点的双循环链表

C. 单链表

D. 带尾结点的单循环链表

(2)若长度为 $n$ 的线性表采用顺序存储结构，在其第 $i$ 个位置插入一个新元素的算法时间复杂度为(　　)$(1\leqslant i\leqslant n+1)$。

A. $O(0)$

B. $O(1)$

C. $O(n)$

D. $O(n^2)$

（3）双向链表中有两个指针域，prior 和 next，分别指向前驱及后继，设 p 指向链表中的一个结点，q 指向一待插入结点，现要求在 p 前插入 q，则正确的插入为（　　）。

A. p->prior=q;q->next=p;p->prior->next=q;q->prior=p->prior;

B. q->prior=p->prior;p->prior->next=q;q->next=p;p->prior=q->next;

C. q->next=p;p->next=q;p->prior->next=q;q->next=p;

D. p->prior->next=q;q->next=p;q->prior=p->prior;p->prior=q;

（4）在一个具有 $n$ 个结点的有序单链表中插入一个新结点并仍然保持有序的时间复杂度是（　　）。

A. $O(n\log_2 n)$

B. $O(1)$

C. $O(n)$

D. $O(n^2)$

（5）在一个以 h 为头指针的单循环链中，p 指针指向链尾结点的条件是（　　）。

A. p->next==NULL

B. p->next==h

C. p->next->next==h

D. p->data==-1

（6）对于一个具有 $n$ 个结点的线性表，建立其单链表的时间复杂度是（　　）。

A. $O(n)$

B. $O(1)$

C. $O(n\log 2n)$

D. $O(n^2)$

（7）在双向链表存储结构中，删除 p 所指的结点时须修改指针（　　）。

A. p->prior->next=p->next　　p->next->prior=p->prior;

B. p->prior=p->prior->prior　　p->prior->next=p;

C. p->next->prior=p　　p->next=p->next->next;

D. p->next=p->prior->prior　　p->prior=p->next->next;

（8）线性表采用链式存储时，其元素地址（　　）。

A. 必须是连续的

B. 一定是不连续的

C. 部分是连续的

D. 连续与否均可

**2. 填空题**

（1）线性表 L=($a_1$,$a_2$,…,$a_n$)用数组表示，假定删除表中任一元素的概率相同，则删除一个元素平均需要移动的元素的个数是_____。

（2）在单链表中设置头结点的作用是_____。

(3)线性表的顺序存储结构通过_____来反映元素之间的逻辑关系,而链式存储结构则通过_____来反映元素之间的逻辑关系。

(4)当对一个线性表经常进行存取操作,而很少进行插入和删除操作时,采用_____存储结构最节省时间;相反地,当经常进行插入和删除操作时,则采用_____存储结构最节省时间。

(5)对于一个具有 $n$ 个结点的单链表,在已知的结点 *p 后插入一个新结点的时间复杂度为_____,在给定值为 $x$ 的结点后插入一个新结点的时间复杂度为_____。

(6)对于双向链表,在两个结点之间插入一个新结点需修改_____个指针,单链表需修改_____个。

(7)循环单链表的最大优点是_____。

(8)若要在一个不带头结点的单链表的首结点 *p 之前插入一个 *s 结点,则可执行下列操作。

s->next=_____;

p->next=s;

t=p->data;

p->data= _____;

s->data=_____;

(9)某线性表采用顺序存储结构,每个元素占据 4 个存储单元,首地址为 100,则下标为 11(第 12 个)的元素的存储地址为_____。

(10)带头结点的双循环链表 $L$ 中只有一个元素结点的条件是_____。

**3. 判断题**

(1)取线性表的第 $i$ 个元素的时间同 $i$ 的大小有关。 ( )

(2)线性表的特点是每个元素都有一个前驱结点和一个后继结点。 ( )

(3)顺序存储方式的优点是存储密度大,且插入、删除运算效率高。 ( )

(4)线性表采用链表存储时,结点的存储空间可以是不连续的。 ( )

(5)链表是采用链式存储结构的线性表,进行插入、删除操作时,在链表中比在顺序存储结构中效率高。 ( )

(6)顺序存储方式只能用于存储线性结构。 ( )

(7)顺序存储结构的主要缺点是不利于插入或删除操作。 ( )

(8)顺序存储方式插入和删除时效率太低,因此它不如链式存储方式好。 ( )

**4. 程序设计题**

(1)设顺序表 va 中的数据元素递增有序。试设计一个算法,将 $x$ 插入顺序表的适当位置,以保持该表的有序性。

(2)设 A=($a_1$,$a_2$,…,$a_m$)和 B=($b_1$,$b_2$,…,$b_n$)均为顺序表,试设计一个比较 A、B 大小的算法(注意,在算法中,不要破坏原表 A 和 B)。

(3)已知指针 ha 和 hb 分别指向两个单链表的头结点,并且已知两个链表的长度分别为 $m$ 和 $n$。试设计一个算法将这两个链表连接在一起(令其中一个表的首元结点连在另一个

表的最后一个结点之后），假设指针 $hc$ 指向连接后的链表的头结点，并要求算法以尽可能短的时间完成连接运算。

（4）试设计一个算法，在无头结点的动态单链表上实现线性表操作 INSERT(L,i,b)，并与在带头结点的动态单链表上实现相同操作的算法进行比较。

（5）已知线性表中的元素以值递增有序排列，并以单链表作为存储结构。试设计一个高效的算法，删除表中所有值大于 mink 且小于 maxk 的元素（若表中存在这样的元素），同时释放被删结点空间（注意，mink 和 maxk 是两个给定的参变量。它们的值可以与表中的元素相同，也可以不同）。

（6）已知线性表中的元素以值递增有序排列，并以单链表作为存储结构。试设计一个高效的算法，删除表中所有值相同的多余元素（使得操作后的线性表中所有元素的值均不相同），同时释放被删除结点空间。

（7）试设计一个算法，对带头结点的单链表实现就地逆置。

（8）设线性表 A=$(a_1,a_2,\cdots,a_m)$和 B=$(b_1,b_2,\cdots,b_n)$，试设计一个按下列规则合并 A、B 为线性表 C 的算法，即使得

当 $m\leqslant n$ 时，C=$(a_1,b_1,\cdots,a_m,b_m,b_{m+1},\cdots,b_n)$；

或

当 $m>n$ 时，C=$(a_1,b_1,\cdots,a_n,b_n,a_{n+1},\cdots,a_m)$。

线性表 A、B 和 C 均以单链表作为存储结构，且 C 表由 A 表和 B 表中的结点空间构成。注意：单链表的长度值 $m$ 和 $n$ 均未显式存储。

（9）假设有两个按元素值递增有序排列的线性表 A 和 B，均以单链表作为存储结构，请设计一个算法将 A 表和 B 表归并成一个按元素值递减有序（非递增有序，允许表中含有值相同的元素）排列的线性表 C，并要求利用原表（A 表和 B 表）的结点空间构造 C 表。

（10）已知 A、B 和 C 为三个递增有序的线性表，现要求对 A 表进行如下操作：删去那些既在 B 表中出现、又在 C 表中出现的元素。试对顺序表编写实现上述操作的算法，并分析算法的时间复杂度（注意，题中没有特别指明同一表中的元素值各不相同）。

（11）设 L 为单链表的头结点地址，其数据结点的数据都是正整数且不相同，试设计利用直接插入的原则把该链表整理成数据递增的有序单链表的算法。

（12）设有一个双向循环链表，每个结点中除有 prior、data 和 next 三个域外，还增设了一个访问频度域 freq。在链表被起用前，频度域 freq 的值均初始化为零，而每当对链表进行一次 LOCATE(L,X)操作后，被访问的结点（元素值等于 X 的结点）中的频度域 freq 的值便增 1，同时调整链表中结点之间的次序，使其按访问频度非递增的次序顺序排列，以便始终保持被频繁访问的结点总是靠近表头结点。试编写符合上述要求的 LOCATE 操作的算法。

（13）已知三个带头结点的线性链表 A、B 和 C 中的结点均依元素值自小至大非递减排列（可能存在两个以上值相同的结点），编写算法对 A 表进行如下操作：使操作后的链表 A 中仅留下三个表中均包含的数据元素的结点，且没有值相同的结点，并释放所有无用结点。限定算法的时间复杂度为 $O(m+n+p)$，其中，$m$、$n$ 和 $p$ 分别为三个表的

长度。

(14)设 head 为一单链表的头指针,单链表的每个结点由一个整数域 data 和指针域 next 组成,整数在单链表中是无序的。编写一个函数,将 head 链中结点分成一个奇数链和一个偶数链,分别由 $p$、$q$ 指向,每个链中的数据按由小到大排列。程序中不得使用 malloc 申请空间。