第 2 章 线 性 表

学习目标

- 理解顺序表的定义和特点；
- 掌握顺序表的查找、插入和删除；
- 掌握单链表的定义；
- 掌握单链表的查找、插入和删除；
- 理解循环链表及双向链表的定义及实现；
- 掌握链表的应用方法。

2.1 项 目 导 入

线性表是最简单、最常用的一种数据结构，其数据元素之间是一种线性结构，而且也是其他数据结构的基础。线性表的特点是：在数据元素的非空有限集合中，有且只有一个开始结点，有且只有一个终端结点，其余的内部结点都有且仅有一个前趋和一个后继。

线性结构可以解决的问题，以单链表作为存储结构，设计和实现某班某门课程成绩管理系统。程序实现功能如下：

① 创建成绩链表，学生数据包括学生的学号、姓名和成绩。

② 可以在指定学号学生前插入学生成绩数据。

③ 可以删除指定学号的学生数据。

④ 可以计算学生的总数。

⑤ 可以按学号和姓名查找学生。

⑥ 可以显示所有学生的成绩。

⑦ 可以把学生成绩按从高到低的顺序排序。

实现效果如图 2 – 1 所示。

图 2－1　首页面

按任意键进入项目目录预览，效果如图 2－2 所示。

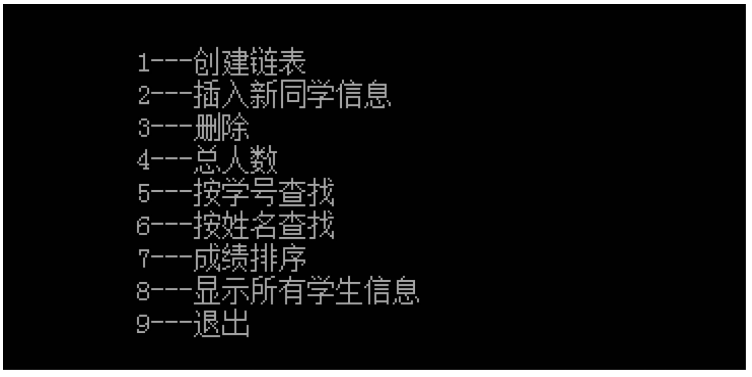

图 2－2　目录预览

图 2－2 中选择 1，执行创建链表信息，具体如图 2－3 所示。

图 2－3　创建链表

创建链表后，选择显示所有学生的信息，预览效果如图 2－4 所示。

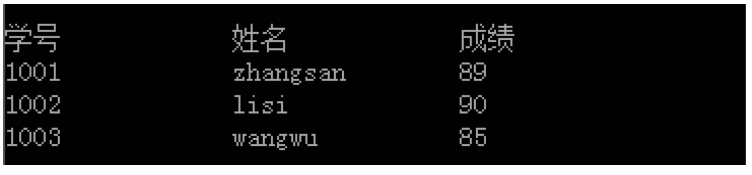

图 2－4　显示信息

在第 2 个位置插入新同学信息，如图 2 -5 所示。

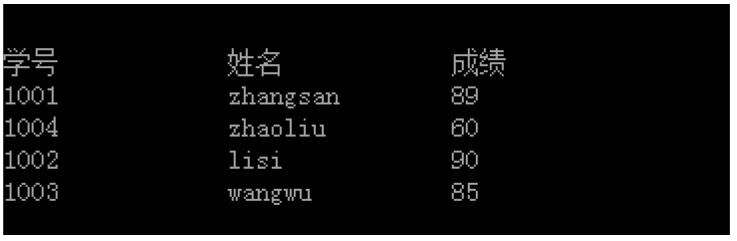

图 2 -5　插入信息后

目前学生总数，如图 2 -6 所示。

图 2 -6　显示总数

根据学号查找学生信息，输入要查找的学生信息，如图 2 -7 所示。

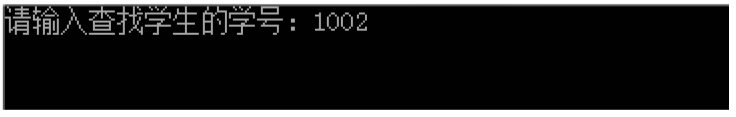

图 2 -7　输入学号

查找到的学生信息情况，如图 2 -8 所示。

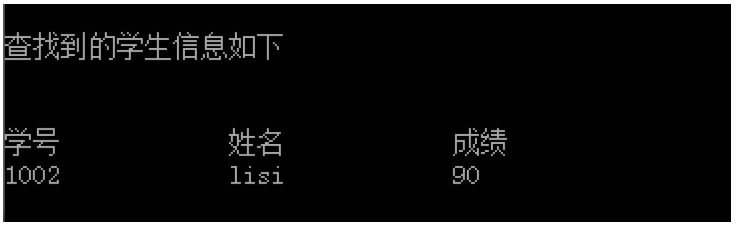

图 2 -8　查找到学生信息

2.2　知 识 概 述

2.2.1　线性表及其基本操作

1. 线性表的定义

线性表是最基本、最简单、也是最常用的一种数据结构。

线性表是数据结构的一种，一个线性表是 n 个具有相同特性的数据元素的有限序列。数据元素是一个抽象的符号，其具体含义在不同的情况下一般不同。

在稍复杂的线性表中，一个数据元素可由多个数据项组成，此种情况下常把数据元素

称为记录，含有大量记录的线性表又称文件。

线性表中的个数 n 定义为线性表的长度，n = 0 时称为空表。在非空表中每个数据元素都有一个确定的位置，如用 a_i 表示数据元素，则 i 称为数据元素 a_i 在线性表中的位序。

线性表中数据元素之间的关系是一对一的关系，即除了第一个和最后一个数据元素之外，其他数据元素都是首尾相接。

线性表的相邻元素之间存在着对应关系。如用（a_1，…，a_{i-1}，a_i，a_{i+1}，…，a_n）表示一个顺序表，则表中 a_{i-1} 领先于 a_i，a_i 领先于 a_{i+1}，称 a_{i-1} 是 a_i 的直接前驱元素，a_{i+1} 是 a_i 的直接后继元素。当 i = 1，2，…，n - 1 时，a_i 有且仅有一个直接后继，当 i = 2，3，…，n 时，a_i 有且仅有一个直接前驱。

综上所述，线性表是由 n（n≥0）个数据元素 a_1，a_2，…，a_n 组成的一个有限序列，表中的每一个数据元素，除第一个外，有且只有一个前驱元素，除最后一个外，有且只有一个后继元素。

线性表的逻辑结构简单，便于实现和操作。因此，线性表这种数据结构在实际应用中是广泛采用的一种数据结构。

2. 线性表的逻辑特征

一个线性表是由 n 个数据元素组成的有限序列。若 n = 0，则表示一个空表，即没有任何数据元素的线性表；若 n > 0，则除 a_1 和 a_n 外，有且仅有一个直接前驱和一个直接后继，即 a_i（其中 1 < i < n）是线性表中第 i 个数据元素，在 a_i 之前仅有一个数据元素 a_{i-1}，而在 a_i 之后也仅有一个数据元素 a_{i+1}。则 a_1 称为起始结点，a_n 称为终端结点，i 称为 a_i 在线性表中的序号或位置。线性表中结点之间的关系就是上述的邻接关系，由于该关系是线性的，称线性表是一种线性结构。

3. 线性表的特性

① 数据元素在线性表中是连续的，表的长度（即数据元素的个数）可根据需要增加和减少，但调整后的线性表中，数据元素仍然必须是连续的，即线性表是一种线性结构。

② 线性表有确定的最大长度，即线性表的容量，表内元素的个数是线性表的当前长度。根据表的长度，线性表可以分为空表、满表或有若干个元素的表。

③ 数据元素在线性表中的位置仅取决于它们自己在表中的序号，并由该元素的数据项中的关键字 KEY 加以标识。

④ 线性表中所有数据元素的同一数据项，其属性相同，它们的数据类型也是一致的。

4. 线性表的基本操作

对于线性表，根据其性质、结构特点以及在实际应用中的需要，有如下几种基本的运算或操作。

（1）线性表初始化

格式：InitList(L)。

初始条件：线性表 L 不存在。

操作结果：构造一个空的线性表 L。

（2）求线性表的长度

格式：LengthList(L)。

初始条件：线性表 L 存在。

操作结果：返回线性表 L 中所有元素的个数。

(3) 取表元

格式：GetList(L，i)。

初始条件：线性表 L 存在，且 1≤i≤LengthList(L)。

操作结果：返回线性表 L 的第 i 个元素(a_i)的值。

(4) 按值查找

格式：LocateList(L，x)。

初始条件：线性表 L 存在，x 有确定的值。

操作结果：在线性表 L 中查找值为 x 的数据元素，并返回该元素在 L 中的位置。若 L 中有多个元素的值与 x 相同，则返回首次找到的元素的位置；若 L 中没有值为 x 的数据元素，返回一个特殊值(通常为 -1)表示查找失败。

(5) 插入操作

格式：InsertList(L，i，x)。

初始条件：线性表 L 存在，i 为插入位置(1≤i≤n+1，n 为插入前的表长)。

操作结果：在线性表 L 的第 i 个元素(a_i)位置上插入值为 x 的新元素，原序号为 i，i+1，…，n 的数据元素的序号插入后变为 i+1，i+2，…，n+1，插入后表长 = 原表长 +1。

(6) 删除操作

格式：DeleteList(L，i)。

初始条件：线性表 L 存在，i(1≤i≤n)为给定的待删除元素的位置值。

操作结果：在线性表 L 中删除序号为 i 的数据元素(a_i)，删除后，原序号为 i+1，i+2，…，n 的数据元素的序号变为 i，i+1，…，n-1，删除后表长 = 原表长 -1。

2.2.2 顺序表及其基本操作

1. 线性表的顺序存储结构

在计算机中存放线性表，一种最简单的方法是顺序存放，也称为顺序分配。

把线性表按顺序存储方法，即把线性表的结点按逻辑次序依次存放在一组地址连续的存储单元里。用这种方法存储的线性表简称为顺序表，或称为向量，用 V 来表示向量，用 V(i)来表示向量 V 的第 i 个分量，用 V[n]来表示含有 n 个数据元素的顺序表。

线性表的顺序存储结构具有以下两个基本特点：

① 线性表中所有元素所占的存储空间是连续的。

② 线性表中各数据元素在存储空间中是按逻辑顺序依次存放的。

由此可以看出，在线性表的顺序存储结构中，其前后两个元素在存储空间是紧邻的，且前驱元素一定存储在后继元素的前面。

在线性表顺序存储结构中，如果线性表中各数据元素所占的存储空间(字节数)相等，则要在该线性表中查找某一个元素是很方便的。

2. 顺序表中数据元素存储地址的计算

假设每个数据元素在存储器中占用 d 个字节的存储单元，如果第一个元素 a_1 的存储地址(也称为基地址)设为 $LOC(a_1)$，则第 i 个元素 a_i 的存储地址 $LOC(a_i)$ 可表示为

$$LOC(a_i) = LOC(a_1) + (i-1) \times d$$

由此，只要知道基地址和每个数据元素的存储长度，就可以求出任一数据元素的存储地址，也就可以随机地访问顺序表的每一个元素。因此，顺序表是一种随机存取结构。

3. 顺序表的描述

顺序表可用如下两种方法来描述。

(1) 通过确定元素和表长描述顺序表

```
#define MAXSIZE 100
typedef int DataType;
DataType data[MAXSIZE];                    //数组 data 用于存放顺序表的结点
int length;                                //当前表的表长
```

说明：

① MAXSIZE 是数组 data 中元素个数的最大值，即顺序表的容量，其值的大小应足够大，这里定义顺序表的容量为 100。

② DataType 是定义的一个新的类型标识符，用来表示顺序表中各结点的类型，这里假设结点的类型为 int 类型。

③ 顺序表的结点 a_1，a_2，…，a_n，从 data[0]开始依次顺序存放，由于结点个数可能未达到 MAXSIZE 个，所以需要使用变量 length 来记录当前顺序表含结点的个数，即当前顺序表的表长，一旦顺序表的每个结点及表长确定了，那么这个顺序表也就被确定了。

注：也可以采用其他方法来确定顺序表，比如通过一个整型变量 last 记录当前顺序表中最后一个结点在数组中的位置来确定顺序表。

使用上述方法来描述顺序表，顺序表的结点和长度不能用一个统一的变量来表示，结构比较松散。因此，我们常使用下面的方法来描述顺序表。

(2) 使用结构体描述顺序表

```
#define MAXSIZE 100                        //假定表容量为 100
typedef int DataType;                      //假定 DataType 代表的类型为 int 型
typedef struct{
DataType data[MAXSIZE];                    //数组 data 用于存放表结点
int length;                                //当前表的长度(结点数)
} SeqList;
```

说明：

① 该方法是在前述方法的基础上，使用结构体将结点和表长封装在一起。

② SeqList 是新定义的结构体类型标识符，用来定义顺序表，可使用语句 SeqList L。定义一个顺序表 L，这样，顺序表的结点和表长都可用 L 来表示，顺序表中的各个结点依次可表示为 L. data[0]，L. data[1]，…，L. data[L. length-1]，表长可表示为 L. length。

③ 也可使用语句 SeqList * L；定义一个指向顺序表的指针 L，为叙述方便就把 L 称为顺序表，这样，顺序表的结点和表长都可用 L 表示为 L -> data[0]，L -> data[1]，…，L -> data[L -> length - 1]和 L -> length。建议使用结构体来描述顺序表，并通过此方法来定义顺序表，但要注意这样定义的顺序表 L 在使用前必须要确定 L 的指向。

有了顺序表的描述方法之后，就可以实现对顺序表的操作。

4. 顺序表的基本操作的实现

（1）顺序表的初始化

```
void InitList(SeqList*L)
{                                    //将当前表 L 的表长 L - > length 置为 0
        L - > length = 0;
}
```

（2）求顺序表的长度

```
int LengthList(SeqList *L)
{                                    //返回顺序表 L 的表长 L - > length
        return L - > length;
}
```

（3）取表元

```
DataType GetList(SeqList *L, int i)
{                                    //返回顺序表 L 的第 i 个结点的值 L - > data[i - 1]
        return L - > data[i - 1];
}
```

（4）插入操作

线性表的插入是指在表 L 的第 i($1 \leqslant i \leqslant n+1$)个位置插入一个值为 x 的新元素，插入后使原长度为 n 的表

$$(a_1, a_2, \cdots, a_{i-1}, a_i, a_{i+1}, \cdots, a_n)$$

成为长度为 n + 1 的表

$$(a_1, a_2, \cdots, a_{i-1}, x, a_i, a_{i+1}, \cdots, a_n)$$

步骤：

第一步：由于是插入运算，首先要检查是否有插入位置，即顺序表是否已满，若顺序表已满(L -> last == MAXSIZE - 1)，则产生溢出错误，返回插入失败。

第二步：在有插入位置的前提下，还要检查插入位置是否合适，即是否满足 $1 \leqslant i \leqslant n+1$，若不满足条件，则插入位置错误，返回插入失败。

第三步：实现后移操作。

第四步：插入新结点。

第五步：将表长 +1，返回插入成功。

```
int InsertList(SeqList*L, DataTypet, int i)
{                                    //将 t 插入顺序表 L 的第 i 个结点的位置上
```

```
    int j;
    if (L->length >= ListSize - 1)
    { printf("表满不能插入"); return 0; }
    if (i < 1 || i > L->length + 2)
    { printf ("插入位置错"); return 0; }
    for(j = L->length; j >= i - 1; j--)
        L->data[j + 1] = L->data[j];          //结点依次后移
    L->data[i - 1] = t;                       //插入 t
    L->length++;                              //表长加 1
    return 1;
}
```

（5）删除操作

线性表的删除运算是指从线性表中删除第 i（1≤i≤n）个元素，删除后使原表长为 n 的线性表

$$(a_1, a_2, \cdots, a_{i-1}, a_i, a_{i+1}, \cdots, a_n)$$

变成长度为 n - 1 的线性表

$$(a_1, a_2, \cdots, a_{i-1}, a_{i+1}, \cdots, a_n)$$

```
int DeleteList(SeqList *L, int i)
{                                             //从顺序表 L 中删除第 i 个结点
    int j;
    if (i < 1 || i > L->length + 1)
    { printf ("删除位置错"); return 0;}
    if (L->length == -1)
    { printf ("空表不能删除"); return 0;}
    for(j = i; j <= L->length; j++)
        L->data[j - 1] = L->data[j];          //结点依次前移
    L->length--;                              //表长减 1
    return 1;
}
```

（6）按值查找

从顺序表 L 中查找值为 t 的结点，找到返回位置值 i，否则返回 -1。

```
int SearchList(SeqList*L, DataType t)
{   int i = 1;
    while (i <= L->length&& L->data[i - 1]! = t)
        i++;
    if(L->data[i - 1] == t)
        return i;
```

```
    else
        return - 1;
}
```

2.2.3 单链表及其基本操作

1. 链表的有关概念

(1) 结点的组成

线性表中的数据元素及元素之间的逻辑关系可由结点来表示。结点由两部分组成：一部分是用来存储数据元素的值的数据域，另一部分是用来存储元素之间逻辑关系的指针域，指针域存放的是该结点的直接后继结点的地址。结点的结构如图 2-9 所示。

图 2-9 结点结构

(2) 单链表

用链式存储结构表示的线性表称为链表，即用一组任意(可以连续也可以不连续)的存储单元来存放线性表的结点。把每个结点只有一个指针域的链表称为单链表。

(3) 开始结点、尾结点、头结点和头指针

在链表中存储第一个数据元素(a_1)的结点称为开始结点，存储最后一个数据元素(a_n)的结点称为尾结点。由于尾结点没有直接后继，所以尾结点指针域的值为 NULL，NULL 在表示链表的示意图中经常用'^'来代替。在开始结点之前附加的一个结点称为头结点，指向链表中第一个结点(头结点或无头结点时的开始结点)的指针称为头指针。

4. (链表)结点的描述

```
typedef char DataType;                    //定义结点的数据域类型
typedef struct node{                      //结点类型定义
DataType data;                            //结点的数据域
struct node *next;                        //结点的指针域
}ListNode;                                //结构体类型标识符
typedef ListNode *LinkList;
ListNode *p;                              //定义一个指向结点的指针
LinkList head;                            //定义指向链表的头指针
```

注意：

① LinkList 和 ListNode * 是不同名字的同一个指针类型，各有专门的用途以示区别。

② LinkList 类型的指针变量 head 表示它是单链表的头指针。

③ ListNode * 类型的指针变量 p 表示它是指向某一结点的指针。

(5) 结点的生成与释放

当需要建立新的结点时，可以使用 C 语言为提供的动态存储分配函数 malloc，向系统申请一个指定大小和类型的存储空间来生成一个新结点，新结点必须要用指向结点的指针

来指向。例如：p = (ListNode *) malloc(sizeof(ListNode))。

当由用户申请的某个存储空间(比如 p 指向的空间)不需要时，可使用 free(p)；释放 p 指向的结点空间，以便其他应用程序使用，不至于造成空间的浪费。

(6) 单链表示意图

单链表见图 2 - 10。

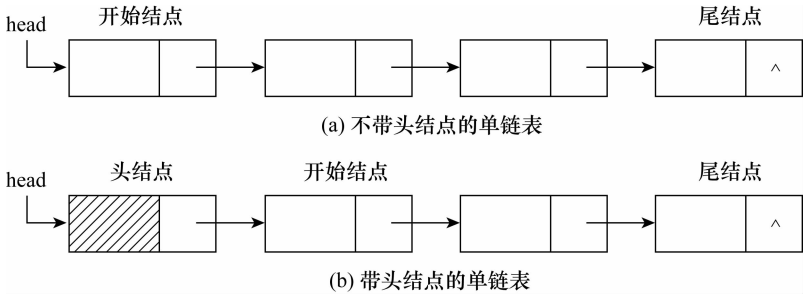

图 2 - 10 单链表示意图

2. 链表的基本操作

(1) 链表的建立

链表的建立是一种动态生成的存储结构，链表中的每个结点占用的存储空间不是预先分配的，而是运行时用户根据需求向系统申请而生成的。

① 头插法建单链表

头插法建单链表是从一个空表开始，重复读入数据，生成新结点，将读入数据存放在新结点的数据域中，然后将新结点插入当前链表的表头上，直到读入结束标志为止。具体实现时需要顺序完成 4 个操作：

向系统申请新阶段存储空间 s = (ListNode *) malloc(sizeof(ListNode));

填入新结点数据域的值 s -> data = ch;

给新结点的指针域赋值为头指针 s -> next = head;

头指针指向新结点 head = s;

算法步骤如下：

第一步 将头指针置为 NULL，如图 2 - 11 所示。

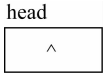

图 2 - 11 头指针

第二步 生成一个新结点 s(即由 s 指向)，如图 2 - 12 所示。

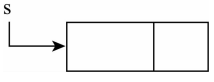

图 2 - 12 新结点 s

第三步　将结点的值写入数据域，如图 2－13 所示。

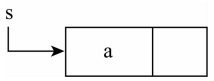

图 2－13　新结点 s 写入数据

第四步　将 head 的值写入结点的指针域，如图 2－14 所示。

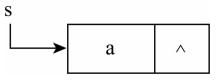

图 2－14　新结点 s 写入指针域

第五步　将新结点的地址 s 赋给 head，如图 2－15 所示。

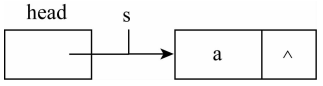

图 2－15　赋给 head

重复进行第二步至第五步，便可建立一个含有多个结点的不带头结点的单链表，如图 2－16 所示。

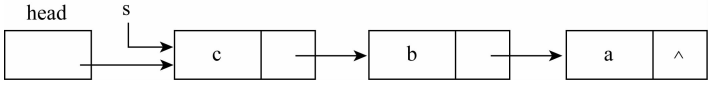

图 2－16　不带头结点的单链表

头插法建单链表算法的实现：

```
LinkList CreatListF(void)
{                                    //返回单链表的头指针
    DataType ch;
    LinkList head;                   //头指针
    ListNode *s;                     //工作指针
    head = NULL;                     //链表开始为空
    printf("请输入链表各结点的数据(字符型):\n");
    while((ch = getchar( ))! = '\n')
    {
        s = (ListNode *)malloc(sizeof(ListNode ));
        s -> data = ch;
        s -> next = head;
        head = s;
    }
    return head;
}
```

② 尾插法建带头结点的单链表

头结点：是在链表的开始结点之前附加一个结点。

先建立一个头结点，使头指针指向头结点，产生一个带头结点的空表。从这一空表开始，重复读入数据，生成新结点，将读入数据存放在新结点的数据域中，然后将新结点插入当前链表的表尾上，直到读入结束标志为止。算法步骤如下：

第一步　建立一个头结点，将头指针和指向尾结点的指针指向头结点，如图2－17所示。

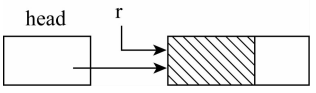

图2－17　头结点

第二步　生成一个新结点s(即由s指向)，并向结点的数据域写数据，如图2－18所示。

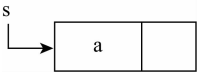

图2－18　生成新结点

第三步　将新结点插入表尾，如图2－19所示。

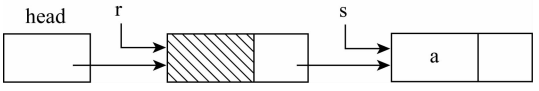

图2－19　新结点插入表尾

第四步　使尾指针指向新表尾，如图2－20所示。

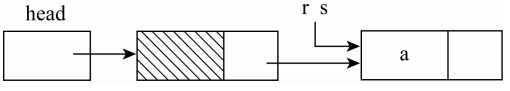

图2－20　新表尾

重复以上第二步至第四步，建立含有多个结点的链表，但此时尾结点的指针域还没有写数据。

第五步　将尾结点的指针域写入NULL。这样，便可建立一个含有多个结点的带头结点的单链表，如图2－21所示。

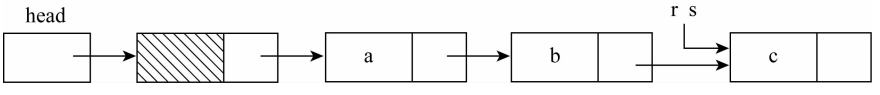

图2－21　尾插法建带头结点单链表

尾插法建立带头结点的单链表算法的实现：

```
LinkList CreatListRH(void)
{                                    //用尾插法建立带头结点的单链表
    DataType ch;
    LinkList head;
```

```
    ListNode *s, *r;                    //工作指针
    head = (ListNode *)malloc(sizeof(ListNode));
    r = head;                           //尾指针初值也指向头结点
    printf("请输入链表各结点的数据(字符型):\n");
    while((ch = getchar( ))! = '\n'){
        s = (ListNode *)malloc(sizeof(ListNode));
        s -> data = ch;
        r -> next = s;                  //将新结点插到链表尾
        r = s;                          //尾指针指向新表尾
    }
    r -> next = NULL;
    return head;
}
```

（2）求表长

① 求带头结点的单链表的表长

设置一个计数器 j 并置初值为 0 和一个移动指针 p 并把 head 赋给 p，使 p 指向头结点。如果 p 指向结点的下一个结点存在，即 p -> next! = NULL，就把 p -> next 赋给 p(使 p 指向下一个结点)，同时计数器 j 加 1，直到 p -> next == NULL 为止，计数器 j 的值就是表长。

具体算法实现：

```
int LengthListH(LinkList head)
{                                       //求带头结点的单链表的表长
  ListNode *p = head;                   //p 指向头结点
  int j = 0;
  while(p -> next){
      p = p -> next;                    //使 p 指向下一个结点
      j ++ ;
  }
  return j;
}
```

② 求不带头结点的单链表的表长

当表不空时与带头结点的单链表基本相同，只是开始 p 指向的是开始结点，所以 j 的初值应设置为 1；当表空时直接返回 0。

具体算法实现：

```
int LengthList (LinkList head)
{                                       //求不带头结点的单链表的表长
  ListNode *p = head;                   //p 指向开始结点
```

```
    int j;
    if(p == NULL)                    //处理空表
        return 0;
    j = 1;                           //处理非空表
    while(p -> next){
        p = p -> next;               //使 p 指向下一个结点
        j ++ ;
    }
    return j;
}
```

(3) 链表的查找

在链表中，即使知道被访问结点的序号 i，也不能像顺序表中那样直接按序号 i 访问结点，而只能从链表的头指针出发，顺着链域 next 逐个结点往下搜索，直至搜索到第 i 个结点为止。因此，链表不是随机存取结构。

① 按序号在带头结点的单链表中查找

设置一个计数器 j，并置初值为 0，从指针 p 指向链表的头结点开始顺着链扫描。当 p 扫描下一个结点时，计数器 j 相应地加 1。当 j == i 时，指针 p 所指的结点就是要找的第 i 个结点；而当指针 p 的值为 NULL 且 j≠i 时，则表示找不到第 i 个结点。

具体算法实现：

```
ListNode *GetNode(LinkList head, int i)
{                                    //在带头结点的单链表 head 中查找第 i 个结点,若找
                                       到(0≤i≤n),则返回该结点的存储地址,否则返
                                       回 NULL
    int j = 0;
    ListNode *p = head;              //从头结点开始扫描
    while(p -> next! = NULL&&j < i){
        p = p -> next;
        j ++ ;
    }
    if(i == j)
        return p;                    //找到了第 i 个结点
    else return NULL;
}
```

② 按值在带头结点的单链表中查找

从开始结点出发，顺着链逐个将结点的值和给定值 key 作比较，若有结点的值与 key 相等，则返回首次找到的其值为 key 的结点的存储地址；否则返回 NULL。

具体算法实现如下：

```
ListNode *LocateNode (LinkList head, DataType key)
```

```
{                                          //在带头结点的单链表 head 中查找其值为 key 的结点
    ListNode *p = head -> next;
    while(p&&p -> data! = key)        //p 等价于 p! = NULL
        p = p -> next;
    return p;
}
```

（4）链表的插入

将值为 x 的新结点 *s 插入到带头结点的单链表 head 的第 i 个结点 a_i 的位置上。

从开始结点出发，顺着链查找第 i－1 个结点，使指针变量 p 指向第 i－1 个结点，即实现以下操作：p = GetNode(head，i－1)；

如图 2－22 所示。

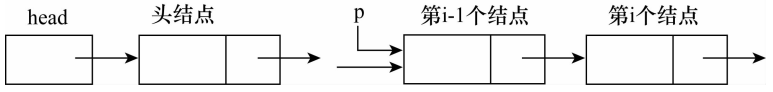

图 2－22　找到第 i 个结点

生成一个数据域为 x 的新结点 *s，即实现以下操作：

s = (ListNode *)malloc(sizeof(ListNode))；s -> data = x；

如图 2－23 所示。

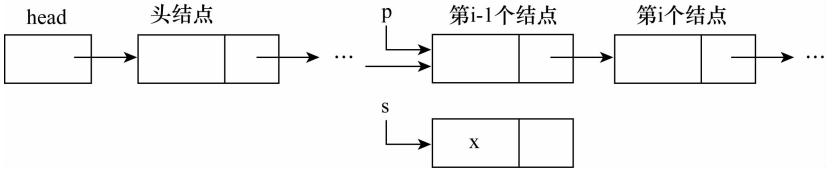

图 2－23　新建结点

使新结点的指针域指向结点 i，即实现以下操作：s -> next = p -> next；

如图 2－24 所示。

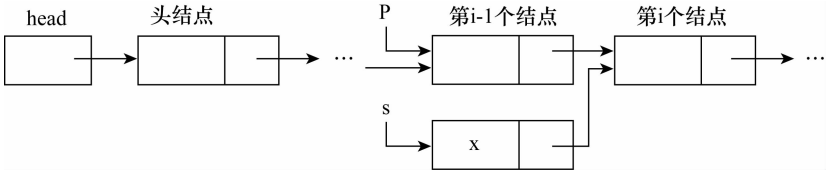

图 2－24　将 s 指向第 i 个结点

使结点 *p 的指针域指向新结点，即实现以下操作：p -> next = s；

如图 2－25 所示。

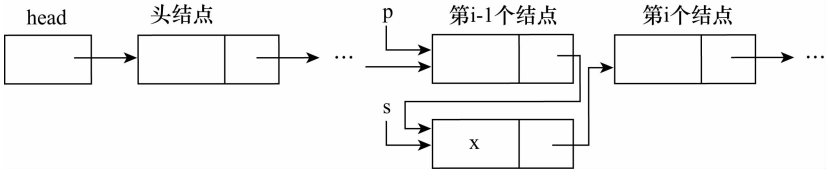

图 2－25　完成插入

```
int InsertList(LinkList head, DataType x, int i)
{                                    //将值为 x 的新结点插入到带头结点的单链表 head 的
                                       第 i 个结点的位置上
    ListNode *p, *s;
    p = GetNode(head, i - 1);        //寻找第 i - 1 个结点
    if(p == NULL)
    {
        printf("未找到第%d 个结点", i - 1);
        return 0;
    }
    s = (ListNode *)malloc(sizeof(ListNode));
    s -> data = x;
    s -> next = p -> next;
    p -> next = s;
    return 1;
}
```

5. 链表的删除

删除带头结点的单链表 head 上的第 i 个结点。

从开始结点出发，顺着链查找第 i－1 个结点，使指针变量 p 指向第 i－1 个结点，即实现以下操作：p = GetNode(head, i－1);

如图 2－26 所示。

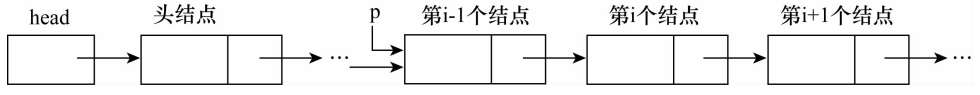

图 2－26 找到第 i－1 个结点

使 r 指向第 i 个结点(被删除的结点)，即实现以下操作：r = p -> next;

如图 2－27 所示。

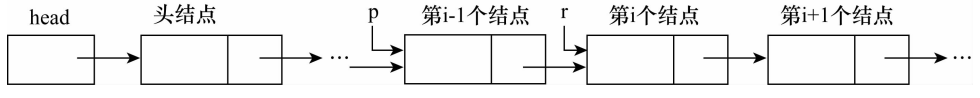

图 2－27 r 指向第 i 个结点

使 p 的指针域指向被删除结点的直接后继，即实现以下操作：p -> next = r -> next;

如图 2－28 所示。

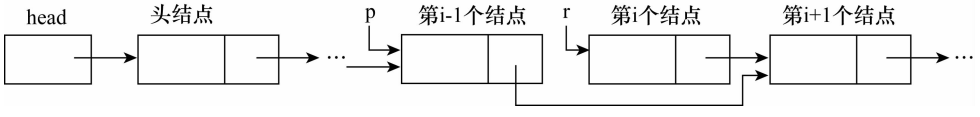

图 2－28 p 指向 r 的直接后继

释放被删除结点的空间即实现以下操作：free(r)；

如图 2－29 所示。

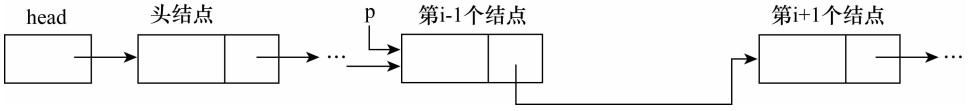

图 2－29　释放空间

具体算法实现

```
int DeleteList(LinkList head,int i)
{                                    //删除带头结点的单链表 head 上的第 i 个结点
    ListNode *p, *r;
    p = GetNode(head, i - 1);        //找到第 i - 1 个结点
    if(p == NULL || p -> next == NULL)
    {
        printf("未找到第%d 个结点", i - 1);
        return 0;
    }
    r = p -> next;                   //使 r 指向被删除的结点 ai
    p -> next = r -> next;           //将 ai 从链上摘下
    free(r);                         //释放结点 ai 的空间
    return 1;
}
```

2.2.4　双向链表及其基本操作

1. 循环链表

（1）循环链表

循环链表是一个首尾相接的链表，它是单链表的另一种形式。

（2）单循环链表

将单链表最后一个结点的指针域由 NULL 改为指向头结点或开始结点，所得到了单链形式的循环链表，称为单循环链表。

（3）带头结点的循环链表

为了使某些操作实现起来方便，在循环单链表中也可设置一个头结点。这样，空循环链表仅由一个自成循环的头结点表示，如图 2－30 所示。

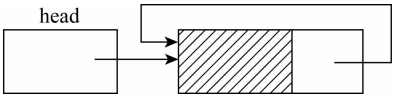

图 2－30　带头结点的空循环链表

（4）带头结点的循环链表的操作

带头结点的循环单链表的各种操作的实现算法与带头结点的单链表的实现算法类似，只需将相应算法中判断指针是否为 NULL 改为是否为 head。

（5）单循环链表的优点

从任意结点出发，都可访问到该链表的所有结点，而单链表只能从开始结点遍历整个链表。

（6）仅设尾指针的单循环链表

在单循环链表中附设尾指针有时比附设头指针会使操作变得更简单。如在用头指针表示的单循环链表中，找开始结点 a_1 的时间复杂度是 O(1)，而要找到终端结点 a_n，则需要从开始结点遍历整个链表，其时间复杂度是 O(n)。如果用尾指针 rear 来表示单循环链表，则查找开始结点和终端结点都很方便，它们的存储位置分别是 rear -> next -> next 和 rear，显然，查找时间复杂度都是 O(1)。因此，多采用尾指针表示单循环链表。

2. 双向链表

（1）单链表(循环链表)存在的缺点

在单链表中，从一已知结点出发，只能访问到该结点及其后续结点，无法找到该结点之前的其他结点。而在单循环链表中，虽然从任一结点出发都可访问到表中所有结点，但访问该结点的直接前驱结点的时间复杂度为 O(n)；另外，在单链表中，若已知某结点的存储位置 p，则将一新结点 * s 插入 * p 之前(称为前插)不如插入 * p 之后方便，因为前插操作必须知道 * p 的直接前驱的位置；同理，删除 * p 本身不如删除 * p 的直接后继方便。

（2）双向链表

在单链表的每个结点里再增加一个指向其直接前驱结点的指针域 prior，这样形成的链表中有两条方向不同的链，因此称为双向链表。

（3）双向链表的描述

```
typedef char DataType;                  //定义结点的数据域类型
typedef struct dlistnode{               //结点类型定义
    DataType data;                      //结点的数据域
    struct dlistnode *prior, *next;     //结点的指针域
}DListNode;                             //结构体类型标识符
typedef DListNode *DLinkList;           //定义新指针类型
DListNode *p, *s;                       //定义工作指针
DLinkList head;                         //定义头指针
```

（4）带头结点的双向链表

在双向链表中增加一个头结点，得到带头结点的双向链表。带头结点的双向链表能使某些运算变得方便。

（5）双向循环链表

将双向链表的头结点和尾结点链接起来构成的循环链表，称为双向循环链表。

（6）双向循环链表的对称性

如果 p 是当前结点 * p 的地址，那么 p -> prior 是结点 * p 的前驱结点的地址，p -> next 是结点 * p 的后继结点的地址。因此 p -> prior -> next == p == p -> next -> prior，即结点 * p 的地址存放在它的前驱结点 *（p -> prior）的（直接后继结点的）next 指针域中，也存放在它的后继结点 *（p -> next）的（直接前驱结点的）prior 指针域中。

（7）双向（循环）链表

如图 2 - 31 所示。

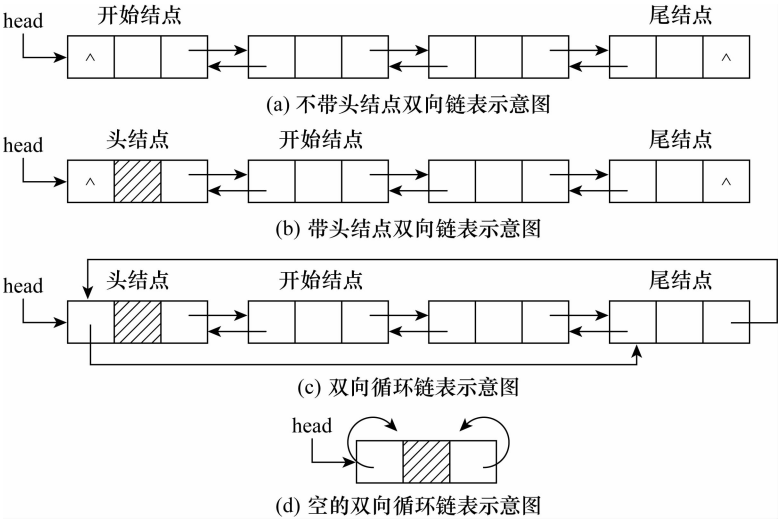

图 2 - 31　双向（循环）链表示意图

（8）双向链表的前插操作算法

在带头结点的双向链表中，将值为 x 的新结点插入结点 * p 之前，设 p≠NULL。

算法如下：

```
void DInsertBefore(DListNode *p, DataType x)
{
    DListNode *s = malloc(sizeof(DListNode));
    s -> data = x;
    s -> prior = p -> prior;
    s -> next = p;
    p -> prior -> next = s;
    p -> prior = s;
}
```

（9）双向链表的删除当前结点的算法

在带头结点的双向链表中，删除当前结点 * p，设 * p 为非终端结点。

算法如下：

```
void DDeleteNode(DListNode *p)
```

```
{
    p->prior->next = p->next;
    p->next->prior = p->prior;
    free(p);
}
```

2.2.5 线性表的应用

【例2.1】顺序表的划分

将顺序表(a_1，a_2，…，a_n)重新排列为以 a_1 为界的两部分：a_1 前面的值都比 a_1 小，a_1 后面的值都比 a_1 大(假设数据元素的类型具有可比性，不妨设为 int 类型)，这一操作称为划分，a_1 称为基准。

基本思想：从第二个元素开始到最后一个元素，逐一向后扫描。

当前数据元素 a_i 比 a_1 大时，表明它已经在 a_1 的后面，不必改变它与 a_1 之间的位置，继续比较下一个。

当前数据元素 a_i 比 a_1 小时，表明它应该在 a_1 的前面，此时将它上面的元素都依次向下移动一个位置，然后将它置入最上方。

顺序表的划分算法：

```
void Part(SeqList *L)
{
  int i, j;
  DataType x, y;                    //用于存放基准和当前小于基准的结点
  x = L->data[0];                   //将基准置入 x 中
  for(i = 1; i < L->length; i++)
      if(L->data[i] < x)
      {
          y = L->data[i];           //将当前小于基准的置入 y 中
          for(j = i - 1; j >= 0; j--)
              L->data[j + 1] = L->data[j];
          L->data[0] = y;
      }
}
```

【例2.2】有序顺序表的合并

有顺序表 A 和 B，其元素均按由小到大的升序排列。编写一个算法将它们合并成一个顺序表 C，要求 C 的元素也是由小到大的升序排列。

基本思想：均从第一个元素开始依次扫描 A 和 B，并进行比较，将较小的元素赋给 C，直到其中的一个顺序表扫描完毕，然后将另一个顺序表中的剩余元素赋给 C 即可。C 的容量应能够容纳 A、B 两个顺序表的所有元素。

顺序表的合并算法：

```
void Merge(SeqList A, SeqList B, SeqList *C)
{
   int i, j, k;
   i = 0; j = 0; k = 0;
   while(i < A. length&&j < B. length)
        if(A. data[i] < B. data[j])
             C - > data[k + + ] = A. data[i + + ];
        else
             C - > data[k + + ] = B. data[j + + ];
   while(i < A. length)
        C - > data[k + + ] = A. data[i + + ];
   while(j < B. length)
        C - > data[k + + ] = B. data[j + + ];
   C - > length = k;
}
```

2.3 项目实践

用顺序存储实现学生成绩管理系统。

本项目需要用到学生信息数据的保存，根据要求，可定义学生结构体类型如下：

```
//定义学生结构体
typedef struct{
     char num[10];                    //学生学号
     char name[20];                   //学生姓名
     int score;                       //学生成绩
}STUDENT;
```

当学生的学号为“#”时，表示数据输入的结束。顺序表中保存的数据元素均为STUDENT类型，则顺序表定义如下：

```
//定义顺序表
typedef struct{
     int last;
     STUDENT data[MAXSIZE];
}SeqList;
```

以下是顺序表作为存储结构实现的学生某门课程成绩管理的完整C语言程序。

C语言代码如下：

```
#include < stdio. h >
```

```
#include < conio. h >
#include < windows. h >
#include < string. h >
#define MAXSIZE 100
//定义学生结构体
typedef struct{
    char num[10];
    char name[20];
    int score;
}STUDENT;
//定义顺序表
typedef struct{
    int last;
    STUDENT data[MAXSIZE];
}SeqList;
void displayList(SeqList * L);
void displayOne(SeqList * L, int i);
//创建成绩链表,学生数据包括学生的学号、姓名和成绩
SeqList createSeqList( ){
    SeqList list;
    STUDENT s;
    char num[10];
    list. last = - 1;
    printf("请输入学生成绩,当学号为#时结束\n\n");
    while(1){
        printf("请输入学号:");
        scanf("%s", num);
        if(num[0] = = '#')
            break;
        strcpy(s. num, num);
        printf("请输入姓名:");
        scanf("%s", s. name);
        printf("请输入成绩:");
        scanf("%d", &s. score);
        list. last = list. last + 1;
        list. data[list. last] = s;
    }
```

```
    return list;
}
//在指定学号学生前插入学生成绩数据
int insertSeqList(SeqList *L, int i){
    int j, k, score;
    STUDENT stu;
    k = L->last;
    if(L->data == MAXSIZE - 1){
        printf("\n 已满,不能插入\n");
        return 0;
    }
    if(i < 1 || i > L->last + 2){
        printf("\n 插入位置有误\n");
        return 0;
    }
    for(j = k; j >= i - 1; j--){
        L->data[j + 1] = L->data[j];
    }
    printf("请输入学号:");
    scanf("%s", stu.num);
    printf("请输入姓名:");
    scanf("%s", stu.name);
    printf("请输入成绩:");
    scanf("%d", &stu.score);
    L->last = L->last + 1;
    L->data[i - 1] = stu;
     return 1;
}
//删除指定学号的学生数据
int deleteSeqList(SeqList *L, int i){
    int j;
    if(i < 1 || i > L->last + 1){
        printf("不存在第%d 个元素", i);
        return 0;
    }
    for(j = i; j <= L->last; j++)
        L->data[j - 1] = L->data[j];
```

```
    L->last = L->last - 1;
    return 1;
}
//计算学生的总数
int lengthSeqList(SeqList *L){
    return L->last + 1;
}
//按学号查找学生
void LocateNum(SeqList *L, char num_find[]){
    int i;
    for(i = 0; i <= L->last; i++)
        if(strcmp(L->data[i].num, num_find) == 0){
            displayOne(L, i);
            break;
        }
}
//按姓名查找学生
void LocateName(SeqList *L, char name_find[]){
    int i;
    for(i = 0; i <= L->last; i++)
        if(strcmp(L->data[i].name, name_find) == 0){
            displayOne(L, i);
            break;
        }
}
//显示所有学生的成绩
void displayList(SeqList *L){
    int i = 0, n;
    n = L->last;
    system("cls");
    printf("\n\n学号\t\t姓名\t\t成绩");
    while(i <= n){
        printf("\n%-6s\t\t", L->data[i].num);
        printf("%-12s\t", L->data[i].name);
        printf("%-4d", L->data[i].score);
        i++;
    }
```

```
}
//显示某个学生信息
void displayOne(SeqList *L, int i){
    printf("\n\n学号\t\t姓名\t\t成绩");
    printf("\n%-6s\t\t", L->data[i].num);
    printf("%-12s\t", L->data[i].name);
    printf("%-4d", L->data[i].score);
}
//把学生成绩按从高到低的顺序排序
void sortSeqList(SeqList *L){
    SeqList *L1 = (SeqList *)malloc(sizeof(SeqList));
    STUDENT temp;
    int len, i, j;
    len = L->last;
    for(i = 0; i <= len; i++){
        L1->data[i] = L->data[i]l
    }
    L1->last = L->last;
    for(i = 1; i <= len; i++){
    if(L1->data[i].score > L1->data[i-1].score){
        temp.score = L1->data[i].score;
        strcpy(temp.num,L1->data[i].num);
        strcpy(temp.name,L1->data[i].name);
        L1->data[i] = L1->data[i-1];
        for(j = i-2; (temp.score > L1->data[j].score)&&(j >= 0); j--){
            L1->data[j+1] = L1->data[j];
    }
    L1->data[j+1].score = temp.score;
    strcpy(L1->data[j+1].num,temp.num);
    strcpy(L1->data[j+1].name,temp.name);
    }
    }
}
int main(int argc, char *argv[])
{
    SeqList L;
    char temp[20], choose;
```

```
int i, fz = 1, res;
printf("\n\n\n\n");
printf("\t\t********************************\n");
printf("\t\t********************************\n");
printf("\t\t******学生成绩管理系统**********\n");
printf("\t\t********************************\n");
printf("\t\t********************************\n");
getch( );
system("cls");
while(fz){
    printf("\n\n");
    printf("\t\t1 - - - 创建链表\n");
    printf("\t\t2 - - - 插入新同学信息\n");
    printf("\t\t3 - - - 删除\n");
    printf("\t\t4 - - - 总人数\n");
    printf("\t\t5 - - - 按学号查找\n");
    printf("\t\t6 - - - 按姓名查找\n");
    printf("\t\t7 - - - 成绩排序\n");
    printf("\t\t8 - - - 显示所有学生信息\n");
    printf("\t\t9 - - - 退出\n");
    choose = getch( );
    system("cls");
    switch(choose){
        case '1':L = createSeqList( );
            displayList(&L);
            getch( );
            system("cls");
            break;
        case '2':
            printf("请输入插入位置:");
            scanf("%d", &i);
            res = insertSeqList(&L, i);
            if(res = = 1)    displayList(&L);
            break;
        case '3':
            printf("请输入删除位置:");
```

```
                scanf("%d", &i);
                res = deleteSeqList(&L, i);
                if(res == 1)    displayList(&L);
                break;
            case '4':
                printf("学生总数为:%d\n", lengthSeqList(&L));
                break;
            case '5':
                printf("请输入查找学生的学号:");
                scanf("%s", temp);
                printf("\n查找到的学生信息如下\n");
                LocateNum(&L, temp);
                break;
            case '6':
                printf("请输入查找学生的姓名:");
                scanf("%s", temp);
                printf("\n查找到的学生信息如下\n");
                LocateNum(&L, temp);
                break;
            case '7':
                sortSeqList(&L);
                displayList(&L);
                break;
            case '8':
                displayList(&L);
                break;
            case '9':
                printf("结束!\n");
                exit(0);
                break;
        }
    }
    system("PAUSE");
    return 0;
}
```

2.4 实作强化

项目一：结构体的建立

建立一个记录职工姓名信息的单链表的结构体结点类型和单链表类型。

首先定义结点类型如下：

```
typedef struct
{
    char data[10];                    //结点的数据域为字符串
    struct node *next;                //结点的指针域
}ListNode;
```

定义单链表类型如下：

```
typedef ListNode *LinkList;
```

项目二：应用项目一中建立的单链表

用头插法建立带头结点的单链表，并输入测试数据（如：wanghong，lili，xiaojun，xiaoxiao，#）。

设计及实现过程如下：

首先用头插入法建立带头结点的单链表。

```
LinkList CreatList(void)
{
    char ch[100];
    LinkList head, p;
    head = (LinkList)malloc(sizeof(ListNode));
    head -> next = NULL;
    while(1)
    {
        printf("输入#结束 \n ");
        printf("请输入具体的职工名:");
        scanf("%s", ch);
        if(strcmp(ch, "#"))
        {
            if(LocateNode(head, ch) == NULL)
            {
                strcpy(head -> data, ch);
                p = (LinkList)malloc(sizeof(ListNode));
                p -> next = head;
                head = p;
```

```
            }
        }
        else
            break;
    }
    return head;
}
```

设计 main 方法。

```
void main( )
{
    char ch[10], num[5];
    LinkList head;
    head = CreatList( );                    //用头插法建立单链表,返回头指针
}
```

项目三：在项目二的基础上设计输出显示所有姓名信息

首先设计打印链表信息的函数，具体如下：

```
void printlist(LinkList head)
{
    ListNode *p = head -> next;             //从开始结点打印
    while(p){
        printf("%s,    ", p -> data);
        p = p -> next;
    }
    printf("\n");
}
```

在项目二的基础上,在 main 函数中增加显示输出功能。

```
void main( )
{
    char ch[10], num[5];
    LinkList head;
    head = CreatList( );                    //用头插法建立单链表,返回头指针
    printlist(head);                        //遍历链表输出其值
}
```

项目四：实现删除链表中的指定结点的职工信息

在前三个项目基础上设计删除带头结点的单链表中的指定结点的功能，具体如下：

```
void DeleteList(LinkList head, char *key)
{
```

```
    ListNode *p, *r, *q = head;
    p = LocateNode(head, key);          //按 key 值查找结点
    if(p == NULL ) {                    //若没有找到结点,退出
        printf("position error");
        exit(0);
    }
    while(q -> next! = p)               //p 为要删除的结点,q 为 p 的前结点
        q = q -> next;
    r = q -> next;
    q -> next = r -> next;
    free(r);                            //释放结点
}
```

在 main 函数中,加入删除职工姓名的功能,具体 main 函数的变化如下:

```
void main( )
{
    char ch[10];
    LinkList head;
    head = CreatList( );                //用头插法建立单链表,返回头指针
    printlist(head);                    //遍历链表输出其值
    printf("请输入要删除的信息");
    scanf("%s", ch);                    //输入要删除的职工姓名
    DeleteList(head, ch);
    printlist(head);
}
```

项目五:实现查找某职工的姓名,找到则显示该结点的位置

按值查找结点,具体实现过程如下:

```
void LocateNode(LinkList head, char *key)
{   int count = 0;
    ListNode *p = head -> next;         //从开始结点比较
    while(p! = NULL && strcmp(p -> data, key)! = 0)
                                        //直到 p 为 NULL 或 p -> data 为 key 止
    {   p = p -> next;                  //扫描下一个结点
        count + + ;
    }
    return p;       //若 p = NULL 则查找失败,否则 p 指向找到的值为 key 的结点
    if(p == NULL)
    {
```

```
            printf("没有查找到该职工信息");
        }
        else{
            printf("%d", count);
        }
}
```

在 main 函数中，加入查找职工姓名的功能，具体 main 函数的变化如下：

```
void main( )
{
    char ch[10];
    LinkList head;
    head = CreatList( );                    //用头插法建立单链表,返回头指针
    printlist(head);                        //遍历链表输出其值
    printf("请输入要删除的信息");
    scanf("%s", ch);                        //输入要删除的职工姓名
    DeleteList(head, ch);
    printlist(head);
    scanf("%s", ch);                        //输入要查找的职工姓名
    LocateNode(head, ch)
}
```

2.5 精选练习

一、选择题

1. 关于线性表 $L=(a_1, a_2, \cdots, a_n)$，下列说法正确的是(　　)。

   A. 每个元素都有一个直接前驱和一个直接后继

   B. 线性表中至少有一个元素

   C. 表中元素的排列顺序必须是由小到大或由大到小

   D. 除第一个和最后一个元素外，其余每个元素都有且仅有一个直接前驱和一个直接后继

2. 下面关于线性表的叙述中，错误的是(　　)。

   A. 线性表若采用顺序存储，必须占用一片连续的存储单元

   B. 线性表若采用顺序存储，便于进行插入和删除操作

   C. 线性表若采用链接存储，不必占用一片连续的存储单元

   D. 线性表若采用链接存储，便于插入和删除操作

3. 在长度为 n 的顺序表的第 $i(1 \leqslant i \leqslant n+1)$ 个位置上插入一个元素，元素的移动次数为(　　)。

   A. $n-i+1$　　B. $n-i$　　C. i　　D. $i-1$

4. 删除长度为 n 的顺序表中的第 i(1≤i≤n)个位置上的元素，元素的移动次数为(　　)。

A. n - i + 1　　B. n - i　　C. i　　D. i - 1

5. 已知一个带头结点单链表 L，在表头元素前插入新结点 * s 的语句为(　　)。

A. L = s; s -> next = L;　　B. s -> next = L -> next; L -> next = s;

C. s = L; s -> next = L;　　D. s -> next = L; s = L;

6. 已知一个不带头结点单链表的头指针为 L，则在表头元素之前插入一个新结点 * s 的语句为(　　)。

A. L = s; s -> next = L;　　B. s -> next = L; L = s;

C. s = L; s -> next = L;　　D. s -> next = L; s = L;

7. 已知单链表上一结点的指针为 p，则在该结点之后插入新结点 * s 的正确操作语句为(　　)。

A. p -> next = s; s -> next = p -> next;　　B. s -> next = p -> next; p -> next = s;

C. p -> next = s; p -> next = s -> next;　　D. p -> next = s -> next; p -> next = s;

8. 已知单链表上一结点的指针为 p，则删除该结点后继的正确操作语句是(　　)。

A. s = p -> next; p = p -> next; free(s);

B. p = p -> next; free(p);

C. s = p -> next; p -> next = s -> next; free(s);

D. p = p -> next; free(p -> next);

9. 设一个链表最常用的操作是在表尾插入结点和在表头删除结点，则选用下列哪种存储结构效率最高？(　　)

A. 单链表　　B. 双链表

C. 单循环链表　　D. 带尾指针的单循环链表

10. 线性表的链接存储结构是一种(　　)存储结构。

A. 随机存取　　B. 顺序存取　　C. 索引存取　　D. 散列存取

11. 链表不具备的特点是(　　)。

A. 插入删除不需要移动元素　　B. 不必事先估计存储空间

C. 可随机访问任一结点　　D. 所需空间与其长度成正比

二、填空题

1. 在单链表 L 中，指针 p 所指结点有后继结点的条件是 _____。

2. 判断带头结点的单链表 L 为空的条件是 _____。

3. 顺序表和链表中能实现随机存取的是_____，插入、删除操作效率高的是 _____。

4. 对于一个具有 n 个结点的单链表，已知一个结点的指针 p，在其后插入一个新结点的时间复杂度为_____；若已知一个结点的值为 x，在其后插入一个新结点的时间复杂度为 _____。

5. 顺序表的存储密度_____，链表的存储密度 _____。

三、简答题

1. 比较顺序表和链表这两种线性表不同存储结构的特点。
2. 简述头结点的作用。
3. 写出单链表存储结构的 C 语言描述。

四、完善程序题

1. 设计一个算法，其功能为：向一个带头结点的有序单链表(从小到大有序)中插入一个元素 x，使插入后链表仍然有序。请将代码补充完整。

```
typedef int DataType;
typedef struct Node
{   DataType data;
        _________;                  //定义指向该结构类型的指针变量 next
}Linklist;
void insert(Linklist *L, DataType x)
{       Linklist *s, *p = L;
        while(p -> next && p -> next -> data < x)
        _________;                  //p 指针后移一步
        _________;                  //申请一个新结点空间,将其地址赋给变量 s
    s -> data = x;
    _________;_________;            //将*s 结点插入到*p 结点的后面
}
```

2. 设计一个函数功能为：在带头结点的单链表中删除值最小的元素。请将代码补充完整。

```
typedef int DataType;
typedef _________ Node              //定义结构体类型
{   DataType data;
    struct Node * next;
}LinkList;
void deleteMin(LinkList *L)
{
  LinkList *p = L -> next, *q;      //首先查找值最小的元素,指针 q 指向最小元素结点
    q = p;
    while(p)
    {   if( p -> data < q -> data)
        q = p;
            _________;              //p 指针后移一步,比较单链表中的每一个结点
    }
```

```
if(!q) return;                    //不存在最小结点(空表)时,直接退出
p = L;          //若存在最小结点,则先找到最小结点的前驱,即*q 的前驱
while _________
    p = p - > next;
    _________;                    //从单链表中删除最小元素结点(指针 q 所指结点)
    _________;                    //释放指针 q 所指结点的空间
}
```

五、算法设计题

1. 已知长度为 n 的线性表 A 中的元素是整数，写算法求线性表中值大于 item 的元素个数。分两种情况编写函数：

（1）线性表采用顺序存储；

（2）线性表采用单链表存储。

2. 试写一算法实现对不带头结点的单链表 H 进行就地(不额外增加空间)逆置。