第 2 章　线性表

2.1　学习指南

（1）理解线性表的类型定义，掌握顺序表和链表的结构差别。
（2）熟练掌握顺序表的结构特性，熟悉顺序表的存储结构。
（3）熟练掌握顺序表的各种运算，并能灵活运用各种相关操作。
（4）熟练掌握链式存储结构特性，掌握链表的各种运算。

2.2　内容提要

线性表的特点：线性表由一组数据元素构成，表中元素属于同一数据对象。在线性表中，数据元素之间的相对位置是线性的,数据按照前后顺序进行有限排列，第一个数据元素有且仅有一个直接后继，最后一个数据元素有且仅有一个直接前驱，其他所有数据元素都有且仅有一个直接后继和一个直接前驱。

1. 线性表

（1）线性表 L 是由 n（n≥0）个数据元素 a_1，a_2，a_3，…，a_n 组成的有限序列，其中数据元素的个数 n 定义为线性表的长度，n=0 的线性表称为空表。

（2）线性表的逻辑结构：一个非空(n≠0)的线性表记为：L＝(a_1，a_2，a_3，…，a_n)。a_i（1≤i≤n）称作线性表的第 i 个数据元素，下标 i 为 a_i 元素在线性表中的位序，称其前面的元素 a_{i-1} 为 a_i(2≤i≤n)的直接前驱，称其后面的元素 a_{i+1} 为 a_i 的直接后继。

2. 线性表的顺序存储

（1）线性表的顺序存储特点：把逻辑相邻的数据元素存储在物理相邻的存储单元中，也就是在内存中用一组地址连续的存储空间顺序存放线性表的各元素。用顺序存储方法存储的线性表称为顺序表。第一个数据元素称为开始结点，最后一个数据元素称为终端结点。

线性表的顺序存储结构具有以下两个基本特点：

1）线性表的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，在顺序存储线性表结构中，其前后两个元素的存储空间是紧邻的，且前驱元素一定存储在后继元素的前面。

（2）线性表的顺序存储结构。

```
#define MaxLen 100                          /*定义线性表的容量为 100*/
typedef ElemType suitable data_type         /*数据元素类型*/
typedef struct{
ElemType data[MAXLEN];                      /*定义存储表中元素的数组*/
```

```
int length;                    /*线性表的实际长度 */
}sqlist;
```

（3）顺序表的基本运算主要包括：顺序表的插入、顺序表的删除、顺序表的查找等。

3. 线性表的链式存储

（1）线性表的链式存储特点。在链表存储方式中，任意两个在逻辑上相邻的数据元素在物理上不一定相邻，数据元素的逻辑次序是通过链表中的指针链接实现的。链表的长度是动态的、可扩充的，在链表中插入和删除时不需移动元素。链式存储结构适用于插入和删除频繁，存储空间需求不定的情形。

（2）线性表的链式存储结构。为了在链表中表示元素之间的线性关系，每一个数据元素在存储时除了存储自身的数据之外，还需要存储其后继或前驱元素的地址，因此数据元素需要一个复合的数据类型表示：

```
typedef struct LNode
{ ElemType data;          /*结点值*/
  Struct Lnode* next;     /*链接下一个结点的地址*/
}LNod,*LinkList;
```

（3）各种链表形式。

1）单链表：数据元素之间只存在单方向的指向，即可以由链表头直接查询到链表尾，但这种链表不能反方向访问。

2）循环链表：将单链表中最后一个结点的指针指向链表的头结点，使整个链表形成一个环形，称这种头尾相边的线性链表形式为循环链表，这样从表中任一结点出发，顺着指针链可以找到表中其他的任何结点。

3）双链表：用两个指针表示结点间的逻辑关系。

（4）线性链表的基本运算：链表的插入、链表的删除、链表的查找等。

2.3 实训概要

本章实训主要是在线性表的两类存储结构（顺序结构和链式结构）上实现基本操作。通过本章实训能够更好地理解和掌握线性表的相关知识。

2.3.1 实验 1：线性表的基本操作

1. 实验目的

（1）掌握线性表的基本运算，熟悉对线性表的一些基本操作和具体的函数定义。

（2）掌握顺序存储的概念，学会定义线性表的顺序存储类型。

（3）熟悉 C 语言程序的基本结构，掌握程序中的用户头文件、实现文件和主文件之间的相互关系及各自的作用。

（4）熟悉 C 语言环境的使用以及多文件程序的输入、编辑、调试和运行的全过程。

（5）加深对顺序存储结构的理解，逐步培养解决实际问题的编程能力。

2. 实验要求

（1）熟练掌握线性表的存储结构及其基本操作。

（2）理解实训案例中的算法，掌握线性表在实际中的应用。

（3）将上机程序全部调试通过。

（4）独立完成一至二个实训项目，保存程序运行结果，并结合程序进行分析。

3. 实验内容

（1）线性表的基本操作。

第一步：定义线性表的存储结构。

第二步：编写线性表操作的具体函数定义。

第三步：使用定义的线性表并调用线性表的一些操作，实现具体运算。

1）初始化线性表。

2）创建一个线性表。

3）在线性表中查找指定的元素。

4）在线性表中插入指定值的元素。

5）在线性表中删除指定位置的元素。

6）输出线性表。

注意：每完成一个步骤，必须及时输出线性表元素，以便于观察操作结果。

sqList.h

```
#define TRUE             1
#define FALSE            0
#define OK               1
#define ERROR            0
#define OVERFLOW         -2
#define ML 10
/*定义 ElemType 为 int 类型*/
typedef int ElemType;
/*线性表顺序存储类型*/
typedef struct sqList                /*定义线性表结构*/
  {
    ElemType  list[ML];              /*线性表元素，存储空间位址*/
    int size;                        /*当前线性表长度*/
    int MAXSIZE;                     /*当前 list 数组元素个数*/
}sqList;
```

sqList.c

```
#include <stdio.h>
#include <stdlib.h>
#include <alloc.h>
#include "sqList.h"

/*初始化线性表*/
sqList *Init_List(sqList *L,int ms)
{
    L=(sqList *)malloc(ms*sizeof(sqList));
    if(!L){
        printf("Memory allocation failure!\n");
        exit(OVERFLOW);
```

```
        }
else
        L->size=0;
    L->MAXSIZE=ms;
    return L;
}

/*输出一个线性表*/
void Disp_List(sqList *L)
{
int  i;
for(i=0;i<L->size;i++)
printf("%d\t",L->list[i]);
printf("\n");
}
/*在线性表中查找元素值为 x 的元素*/
int LocateElem_List(sqList *L, ElemType x)
/*寻找线性表元素，其中返回≥0 为元素位置，-1 为没找到*/
{
int  i=0;
for(i=0;i<=L->size;i++)
if (L->list[i]==x)                    /*找到相同的元素，返回位置*/
    return i;
if (i>L->size) return -1;             /*没找到*/
}

int Insert_List(sqList *L, ElemType x ,int mark)
/* x 为记录值,mark 为插入位置*/
{
int i=1;
if(L->size>=L->MAXSIZE)               /*线性表已满*/
   return -1;
if(mark>0){                           /*插入位置为表头*/
  for(i=L->size+1; i>=mark ; i--)     /*将线性表元素后移*/
    L->list[i+1]=L->list[i];
    L->list[i]=x;
}
else if(mark<0)
   L->list[L->size]=x;                /*插入位置为表尾*/
   L->size++;
return  FALSE ;
}

/*在线性表中删除指定元素值*/
int  Delete_List1(sqList *L , int item)
/*删除的线性表元素，返回≥0 为删除成功*/
```

```
{
    int i,j;
    for(i=0;i<L->size;i++)
      if(item==L->list[i])           /*找到相同的元素*/
        break;
    if(i<L->size){
      for(j=i+1;j<L->size-1;j++)
        L->list[j]=L->list[j+1];
      L->size--;
    return i;
    }
    return  FALSE;
}

/*删除指定位置的线性表元素*/
int Delete_List2(sqList *L,int mark)
{
int i, item;
    if(mark>0){
  item=L->list[mark];
       for(i=mark+1;i<L->size-1;i++)
         L->list[i]=L->list[i+1];
       L->size--;
      return i;
    }
  return FALSE;
}

sqListmain.c
#include"sqList.c"
void main ()
{
int p,n;
ElemType x=0;
sqList a,*b;
b=Init_List(&a,ML);
printf("listaddr=%p\tsize=%d\tMaxSize=%d\n",b->list,b->size,b->MAXSIZE);
while(1)
   {
     printf("\n 请输入元素值,0 为结束输入：");
     scanf("%d",&x);
     if(!x) break;
     printf("请输入插入位置：");
     scanf("%d",&p);
     Insert_List(b,x,p);
     printf("线性表为：\n");
```

```
      Disp_List(b);
    }
    while(1)
    {
      printf("请输入查找元素值，输入 0 结束查找操作：");
      scanf("%d",&x);
      if(!x)    break;
      n=LocateElem_List(b,x);
      if(n<0)    printf("没找到\n");
     else
      printf("有符合条件的元素,位置为：%d\n",n+1);
}
while(1)
{
    printf("请输入删除元素值,输入 0 结束查找操作：");
    scanf("%d",&x);
    if(!x)
      break;
    n= Delete_List1(b,x);
    if(n<0)
      printf("没找到\n");
    else{
      printf("删除元素成功，线性表为:");
    Disp_List(b);
    }
}
while(1)
{
 printf("请输入删除元素位置，输入 0 结束查找操作：");
    scanf("%d",&p);
    if(!p)    break;
    n= Delete_List2(b,p);
    if(p<0)   printf("位置越界\n");
    else{
      printf("线性表为：");
      Disp_List(b);
    }
}
}
```

2.3.2 实验 2：链表的操作

1. 实验目的

（1）学会单链表结点的定义。

（2）熟悉单链表的一些基本操作和具体的函数定义。

（3）加深对链表的理解，逐步培养解决实际问题的编程能力。

2. 实验内容

第一步：定义单链表的存储结构。

第二步：编写单链表操作的具体函数定义。

第三步：用定义单链表和调用单链表的一些操作，实现对单链表的具体运算。

（1）初始化单链表。

（2）创建一个单链表。

（3）在单链表中查找指定的元素。

（4）在单链表中删除指定值的元素。

（5）在单链表中删除指定位置的元素。

（6）输出单链表。

注意：每完成一个步骤，必须及时输出单链表元素，以便于观察操作结果

```
linkList.h
#define TRUE    1
#define FALSE   0
#define OK  1
#define ERROR   0
#define flag    0               /*按下 0 代表输入结束*/
typedef int ElemType;           /*定义 ElemType 为 int 类型*/
/*定义单链表存储结构*/
typedef struct linkList
{
ElemType   data;                /*结点值 */
struct linkList  *next;         /*指向下一个结点的指针 */
}LinkList;

linkList.c
#include<stdio.h>
#include<stdlib.h>
#include<alloc.h>
#include"linkList.h"

/*初始化线性表*/
void init_LinkList(LinkList *head)
{
head=NULL;
}

/*清空单链表*/
void cls_LinkList(LinkList *head)
{
LinkList  *cp ,*np;
cp=head;
while(cp!=NULL)
 {
```

```
    np=cp->next;
    free(cp);
    cp=np;
  }
head=NULL;
}

/*求单链表的长度*/
int length_LinkList(LinkList *head)
{
LinkList *p=head;
int i=0;
while(p!=NULL){
   i++;
   p=p->next;
}
return  i;
}

/*检查单链表是否为空*/
int Empty_LinkList(LinkList *head)
{
   if(head==NULL)
    return TRUE;
else
   return  FALSE;
}

/*用尾插法建立单链表*/
LinkList  *Create_Link_tail(LinkList  *head)
 {
LinkList *p,*q;                  /*q 指针变量用于指向链表的尾部*/
   int x;                        /*设数据元素的类型为 int*/
q=head;                          /*使指针 q 指向链表的尾部*/
printf("依次输入数据建立单链表,以 0 结束:\n");
scanf("%d",&x);
while (x!=flag)
{
  p=(LinkList *)malloc(sizeof(LinkList));
     p->data=x;
  q->next=p;                     /*将新结点链接到尾部*/
  q=p;                           /*使 q 指向新的尾结点*/
     scanf("%d",&x);
}
   q->next=NULL;                 /*最后结点的指针域设置一个空指针*/
  return  head;
}
```

```
/*在单链表第 i 个结点后插入一个新结点，其值为 x*/
int insertList_after(LinkList *head, int i, int x)
{
LinkList  *s,*p;
int j=0;
p=head;
while(p->next!=NULL&&j<i)           /*确定在链表中插入的位置*/
{
    p=p->next; j++;
}
if(!p||j>i)  return ERROR;
else                                /*为插入结点建立链接关系*/
{
    s=(LinkList *)malloc(sizeof(LinkList)) ;
    s->data=x;
    s->next=p->next;
    p->next=s;
}
 return OK;
}

/*在单链表中删除值为 x 的元素*/
LinkList  *delete_LinkList(LinkList  *head,  int  x)
{LinkList   *q, *p;
 p=head;
 while (p->next!=NULL && p->data!=x)    /*在链表中确定删除位置*/
   {q=p;
   p=p->next;}
if (p==NULL)
return  NULL;
else                                    /*修改指针指向*/
    {q->next=p->next;
    free(p);                            /*释放已删除的结点*/
}
return  head;
}

/*链表的查找*/
LinkList *Locate_Linklist(LinkList *head,int x)
/*在单链表中查找第一个值为 x 的结点，返回该结点的地址，如果不存在，则返回空指针*/
{ LinkList *p;
p=head->next;
while(p!=NULL && p->data!=x)
p=p->next;
return p;
```

```
 }

/*输出一个单链表*/
void disp_LinkList(LinkList *head)
{
LinkList *p=head;
    while(p!=NULL){
       printf("%4d",p->data);
       p=p->next;
}
}

LinkListmain.c
#include    "linkList.c"
void main( )
{
int i,x,Pos;
LinkList  a;
init_LinkList(&a);                /*初始化单链表*/
Create_Link_tail(&a);             /*用尾插法建立单链表*/
disp_LinkList(&a);
printf("\n 第 i 个结点后插入一个新结点,输入 i:\n");
scanf("%d",&i);
printf("输入插入值:\n");
scanf("%d",&x);
insertList_after(&a,i,x);
disp_LinkList(&a);
printf("\n 输入要查找的值:\n");
scanf("%d",&x);
Pos=Locate_LinkList(&a,x);
printf("结点位置为: ");
printf("%d\n",Pos);
printf("输入要删除的值:\n");
scanf("%d",&x);
delete_LinkList(&a, x);
disp_LinkList(&a);
}
```