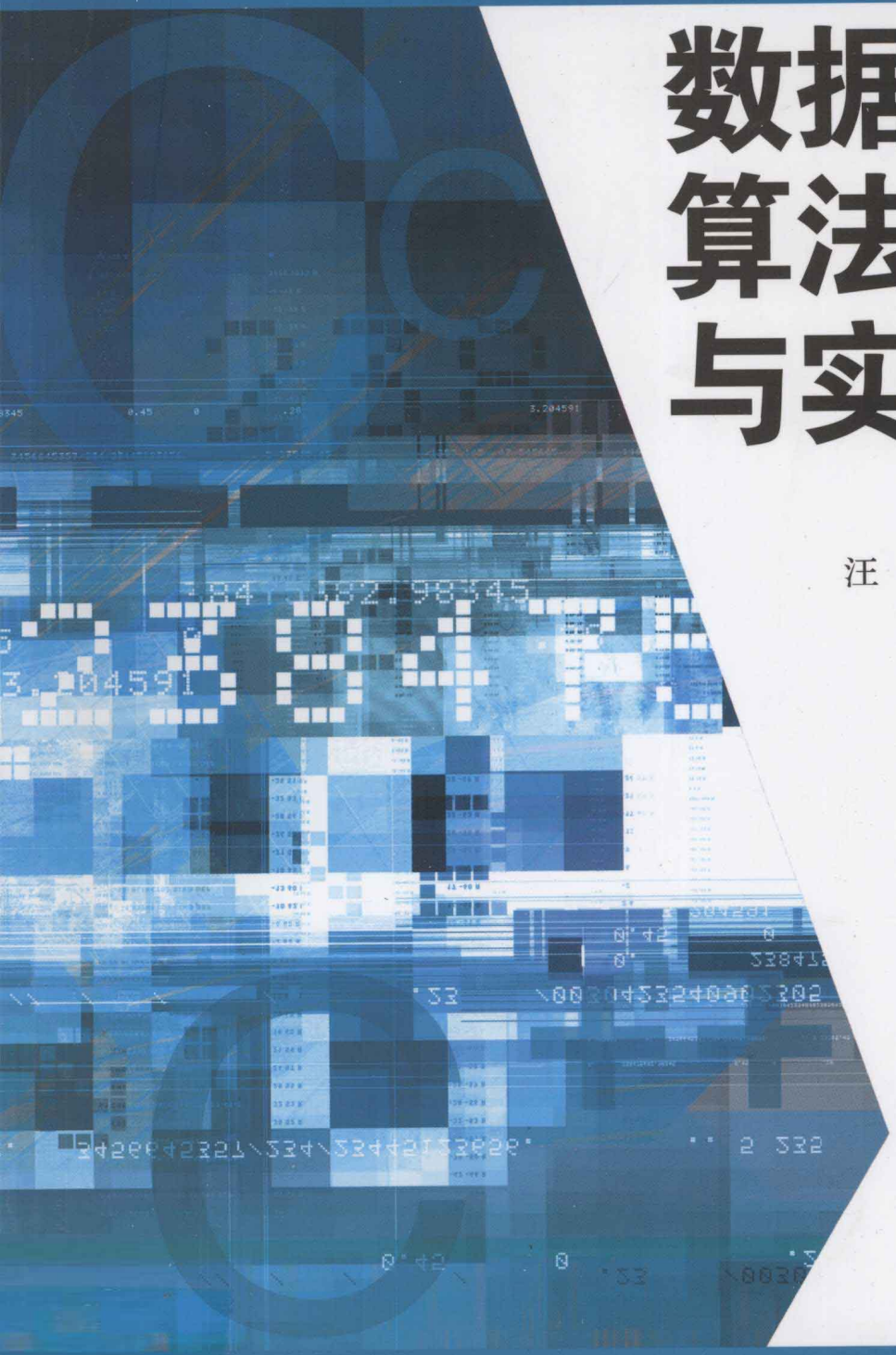
The Problem and Experiment of Data Structure and Algorithm



数据结构与算法的问题与实验

汪 萍 陆正福 彭 程 编著

云南大学出版社
YUNNAN UNIVERSITY PRESS

The Problem and Experiment of Data Structure and Algorithm

数据结构与算法的问题与实验

责任编辑：徐 曼
封面设计：刘 雨

上架建议：计算机

ISBN 978-7-5482-0574-6

9 787548 205746 >

定价：35.00元

数据结构与算法的问题与实验

汪　萍　陆正福　彭　程　编著

云南大学出版社

图书在版编目(CIP)数据

数据结构与算法的问题与实验/汪萍等编著. --昆明:云南大学出版社,2011

ISBN 978-7-5482-0574-6

Ⅰ.①数… Ⅱ.①汪… Ⅲ.①数据结构—高等学校—教学参考资料 ②算法分析—高等学校—教学参考资料 Ⅳ.①TP311.12

中国版本图书馆 CIP 数据核字(2011)第 183088 号

数据结构与算法的问题与实验

汪 萍 陆正福 彭 程 编著

策划组稿：徐 曼
责任编辑：徐 曼
封面设计：刘 雨
出版发行：云南大学出版社
印 装：云南科技印刷厂
开 本：850mm×1168mm 1/16
印 张：18
字 数：508 千
版 次：2011 年 9 月第 1 版
印 次：2011 年 9 月第 1 次印刷
书 号：ISBN 978-7-5482-0574-6
定 价：35.00 元

地 址：云南省昆明市翠湖北路 2 号（邮编：650091）
电 话：0871-5031071 5033244
网 址：http://www.ynup.com E-mail：market@ynup.com

内容简介

本书围绕数据结构与算法内容汇集了约60个综合问题，各问题中包含了若干子问题，内容包括线性表、栈和队列、串、线性表的查找、二叉树、树表的建立与查找、图、数组与广义表；书中还选择了一批基础性问题，它们源自相关的后继课程或研究工作，作为数据结构与算法内容的扩展与延伸。全书内容大多采用了“问题描述”、“问题分析”、“算法步骤”、“程序代码”以及“运行结果”的形式给出，给读者提供了从问题分析到代码实现的一个相对完整过程。

本书可作为高等院校学习数据结构与算法的本科生辅导教材以及从事算法类研究工作的低年级研究生参考书，亦适合于相关工程技术人员参考。

前　言

《数据结构与算法》是信息类专业的一门重要的理论和技术基础课程，也是数学类的信息与计算科学本科专业的核心专业基础课程，在专业课程体系中起着承前启后的作用。该课程的主要目标是使学习者较为全面地理解数据结构与算法的基本概念，掌握数据的各种逻辑结构、物理结构及其实现方法，能够分析和比较不同数据结构与算法，设计和应用相关知识解决后继课程（如信息论基础、操作系统、数据库系统原理、计算机网络、算法图论、编译原理、软件工程、符号计算与计算机代数等）中的基本问题或实际问题。通过课程学习，学生至少应达到运用计算机解决较复杂问题的中级编程能力，能够有效地组织、存储和处理较复杂数据并进行相应的算法设计。

本课程的实践性很强，解题和编程实现对理解理论概念、巩固理论知识、强化算法实现技能，特别是训练学生对非线性数据结构的运用和处理能力是不可或缺的，解题和编程也是培养学生专业基本技能、科学工作作风和创新精神的重要途径。总结多年《数据结构与算法》课程的教学情况，特别是高校扩招的近十年，我们发现学生中普遍存在“听课容易实践难”的问题。学生们普遍反映理论课的内容不难听懂，但要自己动手实践，特别是编程时问题较多。为了帮助学生解决编程练习和实验中的问题，巩固理论课程的内容，拓宽视野，我们编写了《数据结构与算法的问题与实验》。本书围绕数据结构与算法的主要内容编写和收录了约60个相关的综合问题（许多问题中包含了若干子问题）、算法以及一些与后续课程或研究工作有关的问题扩展，给出了C程序编码并经过上机调试（第9章含有部分C++、Java代码）。

在内容安排上，本书采取了以三大数据结构（线性结构、树结构、复杂结构）为线索的组织方式。按照融算法于数据结构中的原则，第1章到第3章是线性结构，第4章是线性表的查找；第5章是树结构，第6章则是树结构的查找；第7、8章讨论复杂结构（图、数组及广义表）问题；第9章给出了数据结构与算法的一些延展问题，其内容涉及操作系统、计算机网络、信息安全与现代密码学等后续课程中的基础问题。书中基本以“问题描述”、“问题分析”、“算法步骤或描述”、“程序代码”以及“运行结果”的形式给出，提供了从问题分析到代码实现的一个相对完整过程，力图为读者提供解题和编程实验环节的一些具体指导和参考，对课堂教学内容进行补充与延伸，帮助学生在课外有针对性地复习和学习，特别是从实验的多角度（不同的存储结构、不同的解题思路、不同的算法设计、不同的程序实现）提供解决问题与代码编写中的分析及解答，从而更好地理解《数据结构与算法》课程的内涵，以期解决“听课容易实践难”的问题。作为云南大学精品课程“数据结构与算法”，数学与统计学院教学改革研究项目“信息论基础中的算法与数据结构”的建设任务之一，希望本书给读者带来更深刻的思考、更宽广的视野、更坚定的学习信心、更浓厚的学习兴趣，成为学好《数据结构与算法》的朋友。

本书的出版要感谢参与云南大学数据结构与算法精品课程建设项目立项的赵越、李源、何敬等各位同仁，感谢参与了本书1－8章代码调试工作的信息与计算科学专业的本科生禹农、伏树林、张路涛、韩泽楷、王学庆、李建强、李佳，感谢参与了本书第9章代码调试工作的计算数学、运筹与控制专业的研究生杨春尧、杜飞、黄豪杰、宋丽、胡江涛、周青婷、侯颖、崔敏、刘永幸、王锦芳、单宝玉，还要感谢数学与统计学院、云南大学出版社、云南大学教务处的支持。

由于时间和我们的专业水平所限，本书的错误和不妥在所难免，敬请同仁和广大读者不吝指正。

作　者

2011年6月于云南大学

目　录

第1章　线性表

问题1　编写C程序，实现顺序表的插入、删除和求给定元素在表中的位序等运算

【问题分析】

线性表的顺序存储是指用一组地址连续的存储单元依次存储线性表的每一个数据元素，使逻辑关系相邻的两个元素在物理存储位置上也相邻，因此用下标可随机地存取任意元素。

在顺序表中的指定位置i插入元素时，首先要排除插入位置i出错以及表内没有插入位置的问题，其次要先将插入位置及其后面的元素向表尾方向后移，再在位置i插入元素，并将表长加1。

在指定位置i删除元素时，首先要排除删除位置i出错以及表空的问题，其次要将插入位置后面的元素向表首方向前移，并将表长减1。

求给定元素在表中的位序时，可循环比较，在顺序表中找到给定元素满足比较条件（等于或小于或大于）的第一个元素并返回其位序，否则返回0。

本问题先用子函数实现顺序表的插入、删除、求位序、初始化、撤销以及打印输出，通过主函数的调用来实现对各个子函数的测试。具体的功能设计如下：

（1）初始化建立空顺序表L；

（2）插入元素（可按输入的插入元素和插入位置插入多个元素），并输出插入后的顺序表L；

（3）删除元素（可按选择的删除位置删除多个元素），并输出删除后的顺序表L；

（4）找出表中与给定元素e满足大于/小于/等于关系的第一个元素的位序；

（5）释放顺序表L；

（0）结束程序。

【程序代码】

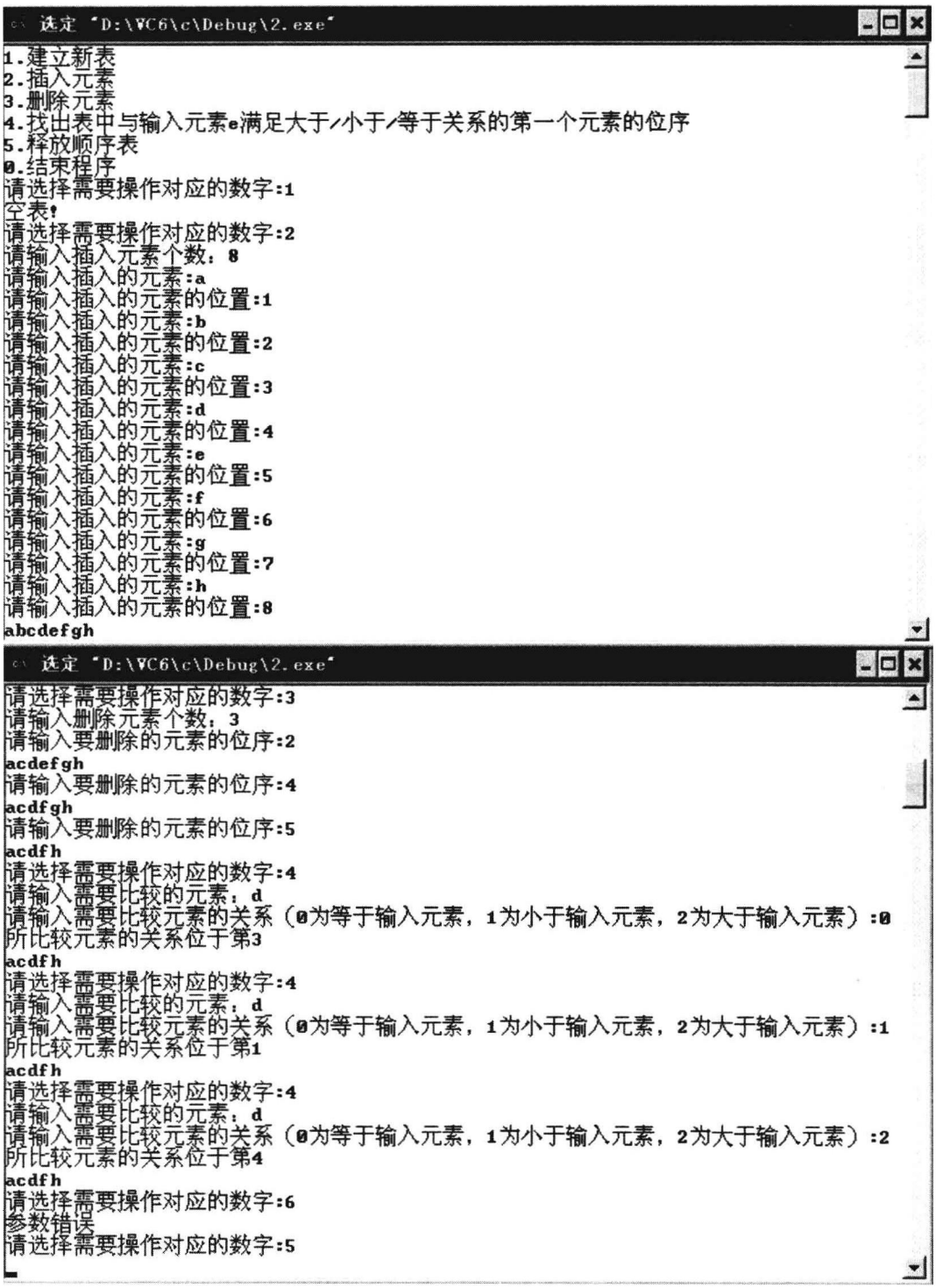
```
/*=============================== ProgramDescription ===============================*/
/* 程序名称：1.c                                                                    */
/* 程序目的：在顺序表中插入和删除元素                                               */
/*==================================================================================*/
#include <stdio.h>
#include <malloc.h>
#define LIST_SIZE 100
#define LISTINCREMENT 10
typedef char ElemType;
typedef struct
{ElemType *elem;
 int length;
```

```
 int listsize;
}SqList ;                          /*定义顺序表类型*/
SqList L;
/*-------------------------------------------------*/
/*初始化顺序表 L.                                  */
SqList *InitiaList()
{L.elem = (ElemType *)
          malloc(LIST_SIZE * sizeof(ElemType));
  if(! L.elem)
   { printf("溢出,扩展空间失败!"); return 0; }
  L.length =0;
  L.listsize = LIST_SIZE;
  return &L;
}
/*-------------------------------------------------*/
/*撤销顺序表 L.                                    */
void DestroyList(SqList *L)
  { free(L);}
/*-------------------------------------------------*/
/*在顺序表 L 中第 i 个位置插入元素 e.              */
int ListInsert(SqList *L,int i,ElemType e)
{ int j;
  ElemType *newbase;
  if(i<1 ||i>L->length+1)  /*位置不合法*/
    { printf("位置不合法"); return 0; }
  if(L->length==L->listsize) /*表满,申请额外空间*/
    { newbase = (ElemType *)realloc(L->elem,
                (L->listsize + LISTINCREMENT)
                 * sizeof(ElemType));
   if(! newbase)
   {printf("溢出,扩展空间失败!"); return 0; }
    L->elem = newbase;
    L->listsize += LISTINCREMENT;
  }
  i--;        /*将位序参数转为数组下标*/
  for(j=L->length-1;j>i;j--)
     L->elem[j+1]=L->elem[j];   /*后移元素*/
  L->elem[i]=e;                 /*插入元素*/
  L->length++;                  /*表长加1*/
  return 1;
}
/*-------------------------------------------------*/
/*在顺序表 L 中删除第 i 个元素.                    */
int ListDelete(SqList *L,int i)
{ int j;
  if(L->length==0)        /*判断是否为空表*/
   { printf("这是个空表,无法删除");
     return 0;
   }
  else
    if(i<1 ||i>L->length)     /*位置不合法*/
     { printf("输入位置不合法");
       return 0;
     }
   i--;           /*将位序参数转化为数组下标*/
   for(j=i;j<L->length-1;j++)     /*前移元素*/
       L->elem[j]=L->elem[j+1];
   L->length--;               /*表长减1*/
   return 1;
}
/*-------------------------------------------------*/
/*输出顺序表 L.                                    */
void ListTraverse(SqList *L)
{ int i;
  if(L->length==0) printf("空表! \n"); /*是否空表*/
  else
  { for(i=0;i<L->length;i++)      /*输出顺序表L*/
        printf("%c",L->elem[i]);
    printf("\n");
  }
}
/*-------------------------------------------------*/
/*在线性表 L 中确定第一个与参数 e 等于/小于/大于
的元素位序,f=0 小于,f=1 小于,f=2 大于.          */
void LocatElem(SqList *L,ElemType e,int f)
{ int i;
```

```
  switch(f)
{ case 0:
        for(i=0;i<L->length;i++)/*0 为等于 e*/
            if(L->elem[i]==e) {++i; break;}
        break;
   case 1:
        for(i=0;i<L->length;i++)  /*1 为小于 e*/
            if(L->elem[i]<e) {++i; break;}
        break;
   case 2:
        for(i=0;i<L->length;i++)  /*2 为大于 e*/
            if(L->elem[i]>e) {++i; break;}
        break;
  default: printf("参数错误\n");
}
if (i<=L->length)
    printf("所比较元素的关系位于第%d\n",i);
else
    printf("不能比较\n");
}
/*-------------------------------------------------*/
/*根据参数 a 的值选择子函数执行.                   */
void chose (int a)
{ int n,i,m;
  ElemType e;
  switch(a)
  { case 1: InitiaList();
            ListTraverse(&L);
            break;
    case 2: printf("请输入插入元素个数:");
            scanf("%d",&n);
            for(i=0;i<n;i++)
              {printf("请输入插入的元素:");
               scanf("%c",&e);
               printf("请输入插入的元素的位置:");
               scanf("%d",&m);
               ListInsert(&L,m,e);
              }
            ListTraverse(&L);
            break;
   case 3:printf("请输入删除元素个数:");
          scanf("%d",&n);
          for(i=0;i<n;i++)
          {printf("请输入要删除的元素的位序:");
           scanf("%d",&m);
           ListDelete(&L,m);
          }
          ListTraverse(&L);
          break;
   case 4: printf("请输入需要比较的元素:");
           scanf("%c",&e);
           printf("请输入需要比较元素的关系(0 为等于输 入元素,1 为小于输入元素,2 为大于输入元素):");
           scanf("%d",&m);
           LocatElem(&L,e,m);
           ListTraverse(&L);
           break;
   case 5: DestroyList(&L);
           ListTraverse(&L);
           break;
   default:printf("参数错误 \n");
  }
}
/*-------------------------------------------------*/
/*主函数:测试上述各函数的功能进行.                */
void main()
{ int a;
  printf("请输入需要操作对应的数字\n");
  printf("1. 建立新表\n");
  printf("2. 插入元素\n");
  printf("3. 删除元素\n");
  printf("4. 找出表中与输入元素 e 满足大于/小于/等于关系的第一个元素的位序\n");
  printf("5. 释放顺序表\n");
  printf("0. 结束程序\n");
```

```
for(a=1;a!=0;)                                    }
{ scanf("%d",&a);                                 }
    chose(a);
```

【运行结果】



问题2 编写C程序，实现在带表头结点的单链表中的12个基本运算

【问题描述】

先用子函数实现带表头结点单链表的各种基本运算（见下面各子函数的说明），再以主函数调用这些子函数测试其功能，主函数设计如下：

（1）初始化，构造一个空的单链表L；

（2）在上述L表上依次插入元素a，b，c，d，e，生成有5个元素的单链表；

（3）输出上述单链表L；

（4）求上述单链表L的长度，并输出之；

（5）判断单链表L是否为空；

（6）输出上述单链表L的第3个元素；

（7）输出上述单链表中元素a的位置；

（8）输出上述单链表中元素b的前驱元素；

（9）输出上述单链表中元素b的后继元素；

（10）在上述单链L表中第4个位置上插入元素f，并输出插入后的表L；

（11）删除上述单链表L的第3个元素，并输出删除后的表L；

（12）将非空单链表L置为空表，并再判断L是否为空；

（13）释放单链表L。

【问题分析】

链式存储结构是用一组地址任意的存储单元（这些单元可以连续、也可以不连续）存储线性表中的各个元素。每一个链结点由一个元素（数据）域和用以指示该元素的直接后继结点存储位置的指针域组成。

所设计的单链表带头结点，故初始化即是生成只有一个头结点的空表。

在指定位置i插入或删除操作时，先要找到插入/删除位置的前驱结点位置i－1以方便操作，还应检查该位置i是否合法，若合法再进行操作。对于插入需要生成一个新结点，然后修改指针完成插入；而删除则只需修改指针即可实现。

对于求给定元素（cur_e）的前驱和后继元素操作，需要定义一个工作指针p，初始时指向第一个结点，并循环地通过p->next->data（求前驱）或p->data（求后继）与元素cur_e的比较，以及p=p->next不断向后移动查找，直到找到元素cur_e，再提取其前驱或后继元素；还要注意第一元素无前驱、最后一个元素无后继的问题。

【程序代码】

```
/*==========ProgramDescription==========*/
/*程序名称:2.c                          */
/*程序目的:单链表的12个基本操作的实现.  */
/*======================================*/
```

```
# include < stdio. h >
# include < malloc. h >
typedef char ElemType;
typedef struct LNode
{ ElemType data;
  struct LNode *next;
}LinkList;          /* 定义单链表的结点类型 */
/*-------------------------------------------------*/
/* 1. 初始化一个空的单链表.                        */
LinkList *InitiaList( )
{ LinkList *L;
  L=(LinkList *) malloc( sizeof( LinkList) ) ;
  L -> next = NULL;
  return L;
}
/*-------------------------------------------------*/
/* 2. 释放单链表 L.                                */
void DestroyList1 ( LinkList *L)
{LinkList *p = L,*q = p -> next;
  while( q! = NULL)
    {free( p) ;
     p = q;
     q = p -> next;
    }
  free( p) ;          /* 释放表中最后一个结点 */
}
/*-------------------------------------------------*/
/* 3. 将 L 置为空表.                               */
voidSetNull( LinkList *L)
{ L -> next = NULL; }
/*-------------------------------------------------*/
/* 4. 若 L 为空表则返回 TRUE,否则返回 FALSE        */
int ListEmpty1 ( LinkList *L)
  {return( L -> next == NULL) ; }
/*-------------------------------------------------*/
/*  5. 求单链表 L 中元素的个数并返回之.            */
int ListLength1 ( LinkList *L)
{LinkList *p = L;
  int i =0;
  while( p -> next! = NULL)
    { i ++ ;
      p = p -> next;
    }
  return( i) ;
}
/*-------------------------------------------------*/
/* 6. 返回 L 的第 i 个元素的值,若操作失败则返回 F. */
ElemType GetElem1 ( LinkList *L, int i)
{int j =0;
 LinkList *p = L;
 ElemType e;
 while( j < i && p! = NULL)
      {j ++ ;
       p = p -> next;
      }
   if( p == NULL) return( ' F ') ;
   else{ e = p -> data; return e; }
}
/*-------------------------------------------------*/
/* 7. 在单链表 L 中查找指定元素 e,并返回它的位序,
   若 L 中没有元素 e,则返回 0. */
int LocateElem1 ( LinkList *L, ElemType e)
{ LinkList *p = L -> next;
  int n = 1;
  while( p! = NULL && p -> data! = e)
    { p = p -> next;
      n ++ ;
    }
  if( p == NULL) return(0) ;
  else return( n) ;
}
/*-------------------------------------------------*/
/* 8. 若 cur_e 是 L 中的数据元素且不是第一个,则返
   回它的前驱元素,否则操作失败返回 F.  */
ElemType PriorElem1 ( LinkList *L, ElemType cur_e)
{LinkList *p = L -> next;
```

```
 ElemType pre_e;
 if(p -> data == cur_e)
   return('F');  /* 第一元素没有前驱元素 */
 else
  { while(p -> next -> data! = cur_e)
    /* 用 p -> next -> data 比较,前驱元是 p -> data */
        p = p -> next;
    pre_e = p -> data;
    return(pre_e);
  }
}
/*-------------------------------------------------- */
/* 9. 若 cur_e 是 L 中的数据元素且不是最后一个,则
返回它的后继元素,否则操作失败返回 F.  */
ElemType NextElem1(LinkList *L,ElemType cur_e)
{LinkList *p = L -> next,*q;
 ElemType next_e;
 if(p == NULL)
   return('F');      /* 表空 */
 else
   {while(p -> data! = cur_e)
        p = p -> next;
    q = p -> next;
    if(q! = null)          /* 后继元素存在 */
      {next_e = q -> data; /* 提取后继元素 */
       return(next_e);
      }
    else
    return('F');   /* 表尾元素没有后继 */
  }
}
/*-------------------------------------------------- */
/* 10. 在 L 中第 i 个位置之前插入新的元素 e,插入成
   功返回 1,否则返回 0. */
   int ListInsert1(LinkList *L,int i,ElemType e)
{int j = 0;
 LinkList *p = L,*s;
 while(j < i - 1 && p! = NULL)  /* 查找第 i - 1 结点 */
   { j++;
     p = p -> next;
   }
 if(p == NULL || j > i - 1)
   return(0); /* 未找到 i - 1 结点,即 i < 1 或大于表长 */
 else          /* 找到第 i - 1 个结点 */
   {s = (LinkList *)malloc(sizeof(LinkList));
    s -> data = e;
    s -> next = p -> next;
    p -> next = s;  /* 将 s 插入到第 i - 1 个结点之后 */
    return(1);
   }
}
/*-------------------------------------------------- */
/* 11. 删除 L 中第 i 个元素,并返回其值.  */
ElemType ListDelete1(LinkList *L,int i)
{int j = 0;
 LinkList *p = L,*q;
 ElemType e;
 while(j < i - 1 && p! = NULL)    /* 查找第 i - 1 结点 */
 { j++;
   p = p -> next;
 }
 if(p == NULL || j > i - 1)
     return('F');   /* 删除位置 i 不合理 */
 else                /* 找到第 i - 1 个结点 */
 { q = p -> next;    /* q 指向要删除的结点 */
   if(q == NULL)
       return('F');  /* 被删结点不存在 */
   e = q -> data;
   p -> next = q -> next;  /* 删除 q 结点 */
   free(q);            /* 释放 q 结点 */
   return(e);          /* 返回被删结点的值 */
 }
}
/*-------------------------------------------------- */
/* 12. 输出单链表中所有元素的值.  */
void ListTraverse1(LinkList *L)
```