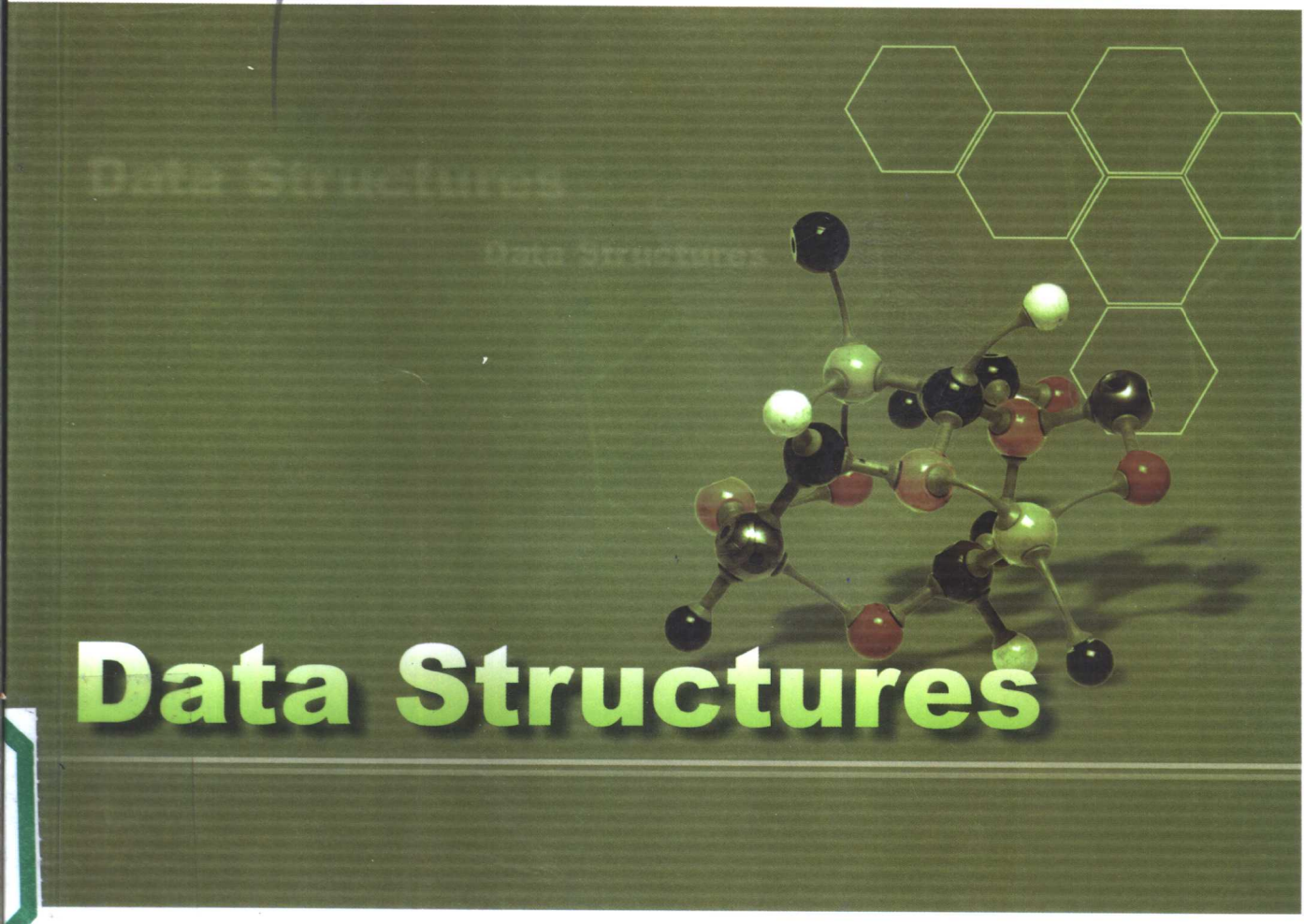
《数据结构》算法实现及解析

——配合严蔚敏、吴伟民编著的《数据结构》（C语言版）

■ 高一凡 编著



西安电子科技大学出版社
http://www.xduph.com

《数据结构》算法实现及解析

——配合严蔚敏、吴伟民编著的《数据结构》(C语言版)

□ 高一凡　编著

西安电子科技大学出版社

2002

内 容 简 介

本书是根据作者的授课讲义整理而成的。

本书为清华大学出版社出版的、由严蔚敏和吴伟民编著的《数据结构》（C 语言版）的学习辅导书。主要内容包括：教材中的每一种数据存储结构的图示；教材中每一种存储结构的基本操作函数及调用这些基本操作的主程序和程序运行结果。有些在教材中一带而过的存储结构（如第 2 章的静态链表和第 6 章的二叉树的三叉链表），本书也提供了完整的基本操作函数及主程序和程序运行结果。本书配有光盘，光盘中包括书中所有程序及用标准 C 语言改写的程序。所有程序均在计算机上运行通过。

本书适用于使用该教材的大中专学生和自学者。书中的基本操作函数也可供从事计算机工程与应用工作的科技人员参考和采用。

图书在版编目（CIP）数据

《数据结构》算法实现及解析 / 高一凡编著.
—西安：西安电子科技大学出版社，2002.10
ISBN 7-5606-1176-1

Ⅰ. 数… Ⅱ. 高… 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. TP311.12

中国版本图书馆 CIP 数据核字（2002）第 072889 号

责任编辑 马武装
出版发行 西安电子科技大学出版社（西安市太白南路 2 号）
电 话 (029)8227828 邮 编 710071
http://www.xduph.com E-mail: xdupfxb@pub.xaonline.com
经 销 新华书店
印 刷 西安文化彩印厂
版 次 2002 年 10 月第 1 版 2002 年 10 月第 1 次印刷
开 本 787 毫米×1092 毫米 1/16 印张 28
字 数 668 千字
印 数 1～4 000 册
定 价 32.00 元
ISBN 7-5606-1176-1 / TP・0608

XDUP 1447001-1

前　言

“数据结构”并非一门纯数学课程。它要求学生能根据所学的“数据结构”理论完成较复杂的程序设计。而程序设计能力的提高有个学习、观摩、借鉴和实践的过程。

学生在学习“数据结构”课程时，虽然已学过 C 语言，但仅是初学，并不精通。对于抽象的数据类型、动态分配存储空间等概念，在理解上还是有一定困难的。如何理解数据存储结构、消化算法，将算法转化成 C 语言的函数并能编写出运行该函数的主程序，往往是摆在他们面前的一道难关。

作者多次讲授“数据结构”课，所用教材为清华大学出版社出版的严蔚敏、吴伟民编著的《数据结构》（C 语言版）（以下简称为教材）。该教材内容较全面，但在叙述一些基本概念和算法时过于精炼，使学生在理解上有一定的困难。作者根据多年的授课经验，编写了教材中各种数据存储结构示意图，并给出了基本操作函数以及调用这些基本操作的主程序。作者力图把抽象的问题具体化，使学生深刻、透彻地理解教材中的各种存储结构和算法，掌握数据结构基本操作函数的编写和应用，并在此基础上，进一步达到能针对具体的工程问题选择甚至创建恰当的数据存储结构，正确应用基本操作函数编程解决之。

本书内容包括：

教材中的每一种数据存储结构的图示；

教材中每一种存储结构的基本操作函数及调用这些基本操作的主程序和程序运行结果。有些在教材中一带而过的存储结构（如第 2 章的静态链表和第 6 章的二叉树的三叉链表），本书也提供完整的基本操作函数及主程序和程序运行结果；

实现教科书中每一个算法的函数及调用该函数的主程序和程序运行结果。

本书附带包含书中所有程序的光盘。所有程序（在光盘的\BC 子目录下）都在 Borland C++ Version 3.1 和 Microsoft Visual C++ 6.0 下运行通过。为了方便使用标准 C 语言的读者，光盘中也附有用标准 C 语言改写的所有程序（在光盘的\TC 子目录下）。用标准 C 语言改写的所有程序都在 Turbo C 2.0 下运行通过。

本书紧密配合教材，故在章节编排上与教材保持一致，以便读者对照查找。在引用教材中的算法和基本操作时，尽量与其保持一致，不做修改。有些章节内容因易于理解和掌握，故未提供学习指导，只保留了章节目录。

本书曾以讲义的形式印过两次（第一次仅包括前 7 章），受到学生的欢迎和好评。学生普遍反映本书对于理解教材内容很有帮助，有的学生还建议正式出版，正是学生对本书的认可给了我极大的鼓励，谨在此对他们表示深深的感谢。

作者对于学习方法的建议：

对于每一种数据类型，要注重主要结构的基本操作。如第 2 章，要注重顺序表和单链表的基本操作。有余力再看次要结构的基本操作。

对于每一个程序，不应仅仅满足于运行出结果，应根据自己的研究目的修改主程序，或在函数中加一些输出语句，以便更好地理解各函数。

各种数据类型的结构都有相通之处，可多做对比。

尽管作者尽了最大努力，但限于水平，书中疏漏之处在所难免，希望读者不吝赐教，以便再版时修订。作者 Email：gyfan@xahu.edu.cn。读者也可通过出版社与我取得联系。

作　者

2002 年 6 月

目　录

```
#include<io.h> // eof().
#include<math.h> // floor(),ceil(),abs()
#include<process.h> // exit()
#include<iostream.h> // cout,cin
// 函数结果状态代码
#define TRUE 1
#define FALSE 0
#define OK 1
#define ERROR 0
#define INFEASIBLE -1
// #define OVERFLOW -2 因为在math.h中已定义OVERFLOW的值为3,故去掉此行
typedef int Status; // Status是函数的类型,其值是函数结果状态代码，如OK等
typedef int Boolean; // Boolean是布尔类型,其值是TRUE或FALSE
```

例 1-7　抽象数据类型 Triplet 的表示和实现。

例 1-7 实际上是教材第 2~7 章中各种存储结构的一个范例：定义一种存储结构及建立在这种存储结构上的一组基本操作，并给出基本操作的实现。

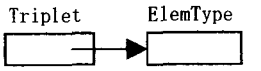


图 1-1　采用动态分配的顺序存储结构

采用动态分配的顺序存储结构（图 1-1）的头文件名为 c1-1.h。

```
// c1-1.h 采用动态分配的顺序存储结构
typedef ElemType *Triplet; // 由InitTriplet分配三个元素存储空间
// Triplet类型是ElemType类型的指针，存放ElemType类型的地址
```

在 c1-1.h 中我们遇到 ElemType（元素类型），在后面的章节中我们还会遇到 SElemType（栈元素类型）、QElemType（队列元素类型）和 TElemType（树元素类型）等，在头文件（诸如 c1-1.h）中，它们是抽象的数据类型。也就是说，还没有确定它们是 C 语言的哪一种具体的类型。

bo1-1.cpp 是有关抽象数据类型 Triplet 和 ElemType 的 8 个基本操作函数。这 8 个函数返回值的类型都是 Status，即函数返回值只能是头文件 c1.h 中定义的 OK、ERROR 等：

```
// bo1-1.cpp 抽象数据类型Triplet和ElemType(由c1-1.h定义)的基本操作(8个)
Status InitTriplet(Triplet &T,ElemType v1,ElemType v2,ElemType v3)
{ // 操作结果:构造三元组T,依次置T的三个元素的初值为v1,v2和v3（图1-2）
  if(!(T=(ElemType *)malloc(3*sizeof(ElemType))))
    exit(OVERFLOW);
  T[0]=v1,T[1]=v2,T[2]=v3;
  return OK;
}
Status DestroyTriplet(Triplet &T)
{ // 操作结果：三元组T被销毁（图1-3）
  free(T);
  T=NULL;
  return OK;
}
```

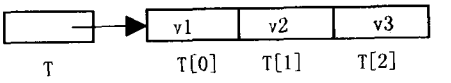


图 1-2　构造三元组 T

图 1-3　三元组 T 被销

```
Status Get(Triplet T,int i, ElemType &e)
{ // 初始条件：三元组T已存在，1≤i≤3。操作结果：用e返回T的第i元的值
  if(i<1||i>3)
    return ERROR;
  e=T[i-1];
  return OK;
}
Status Put(Triplet T,int i,ElemType e)
{ // 初始条件：三元组T已存在，1≤i≤3。操作结果：改变T的第i元的值为e
  if(i<1||i>3)
    return ERROR;
  T[i-1]=e;
  return OK;
}
Status IsAscending(Triplet T)
{ // 初始条件：三元组T已存在。操作结果：如果T的三个元素按升序排列，返回1，否则返回0
  return(T[0]<=T[1]&&T[1]<=T[2]);
}
Status IsDescending(Triplet T)
{ // 初始条件：三元组T已存在。操作结果：如果T的三个元素按降序排列，返回1，否则返回0
  return(T[0]>=T[1]&&T[1]>=T[2]);
}
Status Max(Triplet T,ElemType &e)
{ // 初始条件：三元组T已存在。操作结果：用e返回T的三个元素中的最大值
  e=T[0]>=T[1]?T[0]>=T[2]?T[0]:T[2]:T[1]>=T[2]?T[1]:T[2];
  return OK;
}
Status Min(Triplet T,ElemType &e)
{ // 初始条件：三元组T已存在。操作结果：用e返回T的三个元素中的最小值
  e=T[0]<=T[1]?T[0]<=T[2]?T[0]:T[2]:T[1]<=T[2]?T[1]:T[2];
  return OK;
}
```

main1-1.cpp 是检验 bo1-1.cpp 中的各基本操作函数是否正确的主函数。在 main1-1.cpp 中可根据需要定义抽象数据类型 ElemType 为 int、double 或其它数值型类型（只能是数值类型，因为其中有几个函数是比较大小的），而不需改变基本操作 bo1-1.cpp，这使得 bo1-1.cpp 的通用性大大增强。在 main1-1.cpp 中，有些基本操作的实参（ElemType 类型）要根据 ElemType 的类型做相应改变，而另一些基本操作的实参（Triplet 类型）不需改变。

```
// main1-1.cpp 检验基本操作bo1-1.cpp的主函数
#include"c1.h" // 要将程序中所有#include命令所包含的文件拷贝到当前目录下
// 以下2行可根据需要选用一个（且只能选用一个），而不需改变基本操作bo1-1.cpp
typedef int ElemType; // 定义抽象数据类型ElemType在本程序中为整型
//typedef double ElemType; // 定义抽象数据类型ElemType在本程序中为双精度型
#include"c1-1.h" // 在此命令之前要定义ElemType的类型
#include"bo1-1.cpp" // 在此命令之前要包括c1-1.h文件（因为其中定义了Triplet）
void main()
{
```

```
  Triplet T;
  ElemType m;
  Status i;
  i=InitTriplet(T,5,7,9);
  //i=InitTriplet(T,5.0,7.1,9.3); // 当ElemType为双精度型时,可取代上句
  printf("调用初始化函数后，i=%d(1:成功) T的三个值为：",i);
  cout<<T[0]<<' '<<T[1]<<' '<<T[2]<<endl;
  // 为避免ElemType的类型变化的影响，用cout取代printf()。注意结尾要加endl
  i=Get(T,2,m);
  if(i==OK)
    cout<<"T的第2个值为："<<m<<endl;
  i=Put(T,2,6);
  if(i==OK)
    cout<<"将T的第2个值改为6后，T的三个值为："<<T[0]<<' '<<T[1]<<' '<<T[2]<<endl;
  i=IsAscending(T); // 此类函数实参与ElemType的类型无关,当ElemType的类型变化时,实参不需改变
  printf("调用测试升序的函数后，i=%d(0:否 1:是)\n",i);
  i=IsDescending(T);
  printf("调用测试降序的函数后，i=%d(0:否 1:是)\n",i);
  if((i=Max(T,m))==OK) // 先赋值再比较
    cout<<"T中的最大值为："<<m<<endl;
  if((i=Min(T,m))==OK)
    cout<<"T中的最小值为："<<m<<endl;
  DestroyTriplet(T); // 函数也可以不带回返回值
  cout<<"销毁T后，T="<<T<<"(NULL)"<<endl;
}
```

程序运行结果：

```
调用初始化函数后，i=1(1:成功) T的三个值为：5 7 9
T的第2个值为：7
将T的第2个值改为6后，T的三个值为：5 6 9
调用测试升序的函数后，i=1(0:否 1:是)
调用测试降序的函数后，i=0(0:否 1:是)
T中的最大值为：9
T中的最小值为：5
销毁T后，T=0x0000(NULL)
```

main1-1.cpp 是我们运行的第 1 个程序。通过运行 main1-1.cpp，要掌握这样几点：首先，main1-1.cpp 是提供本书程序风格的一个例子。它是教科书中例 1-7 的完整实现。通过它，我们把数据的存储结构、建立于此结构上的基本操作函数以及调用这些函数的主程序结合到一起了；其次，抽象的数据类型如何具体化；第三，函数类型 Status 的应用；最后，熟悉 C 语言中 malloc 函数的使用。在学习 C 语言时，malloc 函数使用得并不多，但在“数据结构”中它却几乎是使用得最多的函数。

在 bo1-1.cpp 中，有些基本函数的形参带有“&”，如第一个基本函数的声明：

```
Status InitTriplet(Triplet &T,ElemType v1,ElemType v2,ElemType v3);
```

其中形参T前带有“&”,说明形参T是引用类型的。引用类型是C++语言特有的。引用类型的变量，其值若在函数中发生变化，则变化的值会带回主调函数中。程序 f.cpp 说明了函数中引用类型变量和非引用类型变量的区别：

```
// f.cpp 变量的引用类型和非引用类型的区别
#include<stdio.h>
void fa(int a) // 在函数中改变a，将不会带回主调函数(主调函数中的a仍是原值)
{
  a=5;
  printf("在函数fa中：a=%d\n",a);
}
void fb(int &a) // 由于a为引用类型，在函数中改变a，其值将带回主调函数
{
  a=5;
  printf("在函数fb中：a=%d\n",a);
}
void main()
{
  int n=1;
  printf("在主程中，调用函数fa之前：n=%d\n",n);
  fa(n);
  printf("在主程中，调用函数fa之后,调用函数fb之前：n=%d\n",n);
  fb(n);
  printf("在主程中，调用函数fb之后：n=%d\n",n);
}
```

程序运行结果：

```
在主程中，调用函数fa之前：n=1
在函数fa中：a=5
在主程中，调用函数fa之后,调用函数fb之前：n=1
在函数fb中：a=5
在主程中，调用函数fb之后：n=5
```

由于标准C语言中没有引用类型，把C++中含有引用类型的函数转换为标准C语言可执行的语句的方法详见附录A。为了方便使用标准C语言的读者，书后所附光盘中TC\子目录下有本书中所有程序的标准C语言版。

1.4 算法和算法分析

1.4.1 算法

1.4.2 算法设计的要求

1.4.3 算法效率的度量

同样是计算 1-1/x+1/x*x…，algo1-1.cpp（algo 代表算法）的语句频度表达式为(1+n)*n/2，它的时间复杂度 $T(n)=O(n^2)$；而 algo1-2.cpp 的语句频度表达式为 n，它的时间

复杂度 T(n)=O(n)。从两个程序的运行结果可以看出：当输入数据一样时，计算结果是一样的。但运行时间的差别很大。在算法正确的前提下，我们自然要选算法效率高的了。

```
// algo1-1.cpp 计算1-1/x+1/x*x…
#include<stdio.h>
#include<sys/timeb.h>
void main()
{
  timeb t1,t2;
  long t;
  double x,sum=1,sum1;
  int i,j,n;
  printf("请输入x n: ");
  scanf("%lf%d",&x,&n);
  ftime(&t1); // 求得当前时间
  for(i=1;i<=n;i++)
  {
    sum1=1;
    for(j=1;j<=i;j++)
      sum1=sum1*(-1.0/x);
    sum+=sum1;
  }
  ftime(&t2); // 求得当前时间
  t=(t2.time-t1.time)*1000+(t2.millitm-t1.millitm); // 计算时间差
  printf("sum=%lf 用时%ld毫秒\n",sum,t);
}
```

程序运行结果（其中用时与计算机的配置有关，带下划线的字符为键盘输入）：

```
请输入x n: 123 10000↙
sum=0.991935 用时5440毫秒
```

```
// algo1-2.cpp 计算1-1/x+1/x*x…的更快捷的算法
#include<stdio.h>
#include<sys/timeb.h>
void main()
{
  timeb t1,t2;
  long t;
  double x,sum1=1,sum=1;
  int i,n;
  printf("请输入x n: ");
  scanf("%lf%d",&x,&n);
  ftime(&t1); // 求得当前时间
  for(i=1;i<=n;i++)
  {
    sum1*=-1.0/x;
    sum+=sum1;
  }
```

```
    ftime(&t2); // 求得当前时间
    t=(t2.time-t1.time)*1000+(t2.millitm-t1.millitm); // 计算时间差
    printf("sum=%lf 用时%ld毫秒\n",sum,t);
}
```

程序运行结果（其中用时与计算机的配置有关）：

```
请输入x n: 123 10000↙
sum=0.991935 用时0毫秒
```

第2章 线 性 表

2.1 线性表的类型定义

算法 2.1 在 algo2-1.cpp 和 algo2-12.cpp 中，算法 2.2 在 algo2-2.cpp 和 algo2-13.cpp 中。

2.2 线性表的顺序表示和实现

顺序表存储结构容易实现随机存取线性表的第 i 个数据元素的操作，但在实现插入、删除的操作时要移动大量数据元素。所以，它适用于数据相对稳定的线性表，如职工工资表、学生学籍表等。

c2-1.h 是动态分配的顺序表存储结构，bo2-1.cpp 是基于顺序表的基本操作。由于C++函数可重载，故去掉 bo2-1.cpp 中算法 2.3 等函数名中表示存储类型的后缀_Sq：

```
// c2-1.h 线性表的动态分配顺序存储结构（图2-1）
#define LIST_INIT_SIZE 10 // 线性表存储空间的初始分配量
#define LISTINCREMENT 2 // 线性表存储空间的分配增量
struct SqList
{
  ElemType *elem; // 存储空间基址
  int length; // 当前长度
  int listsize; // 当前分配的存储容量(以sizeof(ElemType)为单位)
};
```

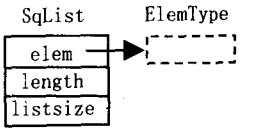


图 2-1 动态分配顺序存储结构

```
// bo2-1.cpp 顺序表示的线性表(存储结构由c2-1.h定义)的基本操作(12个)
Status InitList(SqList &L) // 算法2.3
{ // 操作结果：构造一个空的顺序线性表（图2-2）
  L.elem=(ElemType*)malloc(LIST_INIT_SIZE*sizeof(ElemType));
  if(!L.elem)
    exit(OVERFLOW); // 存储分配失败
  L.length=0; // 空表长度为0
  L.listsize=LIST_INIT_SIZE; // 初始存储容量
  return OK;
}
Status DestroyList(SqList &L)
{ // 初始条件：顺序线性表L已存在。操作结果：销毁顺序线性表L（图2-3）
  free(L.elem);
  L.elem=NULL;
  L.length=0;
  L.listsize=0;
```

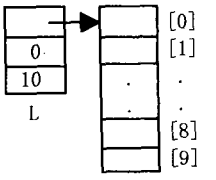


图 2-2 构造一个空的顺序线性表 *L*

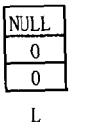


图 2-3 销毁顺序线性表 *L*

```
  return OK;
}
Status ClearList(SqList &L)
{ // 初始条件：顺序线性表L已存在。操作结果：将L重置为空表
  L.length=0;
  return OK;
}
Status ListEmpty(SqList L)
{ // 初始条件：顺序线性表L已存在。操作结果：若L为空表，则返回TRUE，否则返回FALSE
  if(L.length==0)
    return TRUE;
  else
    return FALSE;
}
int ListLength(SqList L)
{ // 初始条件：顺序线性表L已存在。操作结果：返回L中数据元素个数
  return L.length;
}
Status GetElem(SqList L,int i,ElemType &e)
{ // 初始条件：顺序线性表L已存在，1≤i≤ListLength(L)
  // 操作结果：用e返回L中第i个数据元素的值
  if(i<1||i>L.length)
    exit(ERROR);
  e=*(L.elem+i-1);
  return OK;
}
int LocateElem(SqList L,ElemType e,Status(*compare)(ElemType,ElemType))
{ // 初始条件：顺序线性表L已存在，compare()是数据元素判定函数(满足为1,否则为0)
  // 操作结果：返回L中第1个与e满足关系compare()的数据元素的位序。
  //           若这样的数据元素不存在，则返回值为0。算法2.6
  ElemType *p;
  int i=1; // i的初值为第1个元素的位序
  p=L.elem; // p的初值为第1个元素的存储位置
  while(i<=L.length&&!compare(*p++,e))
    ++i;
  if(i<=L.length)
    return i;
  else
    return 0;
}
Status PriorElem(SqList L,ElemType cur_e,ElemType &pre_e)
{ // 初始条件：顺序线性表L已存在
  // 操作结果：若cur_e是L的数据元素，且不是第一个，则用pre_e返回它的前驱，
  //           否则操作失败，pre_e无定义
  int i=2;
  ElemType *p=L.elem+1;
  while(i<=L.length&&*p!=cur_e)
  {
    p++;
    i++;
  }
```