工业和信息化普通高等教育“十二五”规划教材立项项目

COMPUTER

数据结构项目实训

Date Structure of Project Training

戴文华 赵君喆 主编
卢社阶 闻彬 副主编

- 精选项目实训内容
- 配备齐全代码资源
- 构思新颖理实结合

人民邮电出版社
POSTS & TELECOM PRESS

TP311.12/148C2
2012

工业和信息化普通高等教育“十二五”规划教材立项项目

COMPUTER

数据结构项目实训

Date Structure of Project Training

戴文华 赵君喆 主编
卢社阶 闻彬 副主编

READ
北方工业大学图书馆
C00309382

人 民 邮 电 出 版 社
北 京

图书在版编目（CIP）数据

数据结构项目实训 / 戴文华，赵君喆主编. -- 北京
：人民邮电出版社，2012.9
ISBN 978-7-115-29075-5

Ⅰ. ①数… Ⅱ. ①戴… ②赵… Ⅲ. ①数据结构
Ⅳ. ①TP311.12

中国版本图书馆CIP数据核字(2012)第179472号

内 容 提 要

本书可作为严蔚敏编著的《数据结构（C语言版）》（清华大学出版社出版）一书的配套实训教程。

本书由11章和1个附录组成，其中第0章给出所有项目的总体实训规范，第1～10章描述各种数据结构的实训项目，各实训项目由结构特点总结、项目实训具体要求、核心代码提示和实训拓展等几部分组成，附录提供了标准化代码规范参考。配套源代码中包含了所有实训项目的完整参考代码。

本书内容丰富、实践性强，不仅可作为大专院校数据结构课程的配套教材，也可作为广大工程技术人员和自学读者的辅助学习资料。

数据结构项目实训

◆ 主　　编　戴文华　赵君喆
副 主 编　卢社阶　闻　彬
责任编辑　韩旭光
◆ 人民邮电出版社出版发行　　北京市崇文区夕照寺街14号
邮编　100061　　电子邮件　315@ptpress.com.cn
网址　http://www.ptpress.com.cn
大厂聚鑫印刷有限责任公司印刷
◆ 开本：787×1092　1/16
印张：19.5　　2012年9月第1版
字数：470千字　　2012年9月河北第1次印刷

ISBN 978-7-115-29075-5

定价：45.00元

读者服务热线：(010)67132746　印装质量热线：(010)67129223
反盗版热线：(010)67171154

前　言

计算机程序由数据结构加算法构成。在计算机科学理论方面，数据结构揭示了信息的逻辑结构、存储结构和相应操作；在编程实践方面，数据结构是程序设计的技术基础。因此，数据结构这门学科既具有理论性，又具有很强的实践性。它的实践性体现在工程开发的多个阶段，在实际软件工程开发过程中，能正确并灵活地运用数据结构，是衡量数据结构实践教学质量的一个重要标准。

传统的数据结构实践教材往往立足于离散知识点，并针对各种数据结构和算法设计小规模验证实验。这样的实践方式由于缺乏系统性，而显得较为枯燥，难以激发学习者对数据结构实践应用的兴趣，也难以使学习者体会数据结构在软件工程项目开发中所扮演的角色。

本书打破了传统数据结构实践教材的格局，以工程项目思想贯穿始终，将所有数据结构及相关知识点包装成一个个完整的工程项目，并明确项目成果要求和标准化项目开发过程要求。本书的学习者将站在项目开发者的角度进行数据结构的应用和实践，同时遵循标准化软件开发流程，以实训的方式体验工程项目的实现过程。本书所构造的所有项目均具有较高的实用性，项目成果还可应用于学习者今后的学习和工作中。

全书由 11 章和 1 个附录组成，各章节的主要内容为：第 0 章给出所有项目的总体实训规范（项目开发流程的公共要求）；第 1～7 章对各种基本数据结构（顺序表、栈、队列、串、广义表、树、图）的特性进行了总结，并针对各种数据结构设计了相应的项目实训和拓展；第 8 章设计了多种动态存储结构的项目实训和拓展；第 9 章设计了各种查找表和算法的项目实训和拓展；第 10 章设计了各种排序算法的项目实训和拓展。附录提供了标准化代码规范参考。本书项目实训的完整参考代码可在教学资源网下载或联系编者（邮箱：daiwenh@163.com）索取。

本书第 0、1、2、9 章由戴文华编写，第 7、8、10 章由赵君喆编写，第 3、6 章由闻彬编写，第 4、5 章由卢社阶编写。由于编者水平有限，难免存在不当之处，恳请广大读者批评指正。

编者

2012 年 5 月

目　录

此为试读，需要完整PDF请访问：www.ertongbook.com

第0章 项目总体实训规范

数据结构是一门实践性非常强的课程，不但要深入理解各种结构的原理，还要能在解决实际问题的过程中灵活运用。因此，我们从项目实践入手，在商业项目的标准化运作过程中提炼核心开发流程，结合软件工程学的知识体系，针对各种数据结构设计专门的实训项目。

我们对本书的所有的实训项目做出以下总体性实训规范：

一、实训要求

1. 程序必须严格按照标准代码规范进行编写；
2. 依据软件标准开发流程完成各种文档；
3. 按照教师要求，独立完成实训项目。

二、实训环境

操作系统 Windows XP 以上，C/C++编译环境 VC6.0 以上。

三、实训安排

1. 准备阶段

实训前准备空白文档：

◆ 项目计划及进度控制表；
◆ 需求规格说明表；
◆ 功能模块结构说明表；
◆ 测试计划表；
◆ 缺陷记录表。

2. 计划、设计阶段

（1）估计项目规模及开发时间，针对项目细节做出详细时间安排，并跟踪完成情况；
（2）对项目进行需求分析，完成需求规格说明表；
（3）设计各程序模块，完成功能模块结构说明表。

3. 编码调试阶段

（1）遵循标准代码规范（参考附录）书写规整的代码；
（2）根据设计文档编码实现程序；
（3）编译调试程序，使程序能够顺利运行。

4. 项目测试阶段

（1）完成项目测试计划表，根据项目功能填写测试用例，构造测试数据；
（2）依据测试计划表对程序进行测试；

（3）修复程序缺陷，填写缺陷记录表。

5. 文档提交阶段

整理并汇总所有文档以及可顺利编译运行的源程序，提交任课教师。

四、文档规范

1. 项目计划及进度控制表（见表 0.1）

针对项目开发流程中的每一个阶段，制订准确的时间约束，在项目进行中跟踪每一个开发环节，每天更新各环节的完成进度以及任务延迟情况。

表 0.1　　项目计划及进度控制表

开发阶段	起始时间	终止时间	延迟时间	完成量（%）	备　注
计划制订					
需求分析					
设计					
编码					
测试					
整理打包					
总计					

2. 需求规格说明表（见表 0.2）

对项目的功能而言，应站在数据结构使用者的角度进行分析和设计，在需求分析的过程中要着重考虑结构操作的实用性、健壮性和可扩展性。对于一些常用的结构，甚至可以将其设计成一个具有实用意义的库，以供今后的开发使用。

表 0.2　　需求规格说明表（以顺序表结构为例）

功能需求	详细说明
用户操作菜单	用户可以参照菜单输入操作编号来选择对结构的特定操作
创建顺序表	用户可选择以文件的方式输入结构初始数据，也可采用实时的方式输入数据
按位置查询元素	如果输入的位置超过表长，则提示用户输入错误，不会对表越界访问
删除元素	如果表中无任何元素，则直接提示空表
…	…

3. 功能模块结构说明表（见表 0.3）

程序的功能模块的说明是项目的重要资料，这些资料是程序的实用者以及程序的维护者的主要参考依据。因此，针对程序中每一个重要的函数模块都应该给出详细的接口信息以及内部运作逻辑。

表 0.3　　功能模块结构说明表（以顺序表结构为例）

函数名	参数说明	返回值说明	操作行为说明
InitList	参数 1：传入顺序表结构引用	无返回值	为顺序表申请缺省长度的内存空间，并将表置空
LoopCommand	参数 1：传入顺序表结构引用	返回逻辑值，表示是否结束循环命令输入状态	让用户循环输入操作代号，依据代号调用不同操作函数，其中一个代号代表退出程序

续表

函　数　名	参　数　说　明	返回值说明	操作行为说明
GetElem	参数 1：传入顺序表结构变量 参数 2：传入查询元素位置 参数 3：外部变量引用，传出查到的元素值	返回状态码	判断参数 2 是否在表长度之内，如果不在则返回错误值（负值）；将参数 3 赋值为查询到的元素值，并返回正确值（0）
…	…	…	…

4. 测试计划表（见表 0.4）

项目编码和调试结束之后，应该执行严格的测试流程以尽可能发现程序的缺陷。测试计划设计的好坏直接决定着测试的效果，因此尽可能从多种角度设计测试项目和测试步骤。而一份完善的数据结构测试计划表，可作为将来结构修改或维护时的质量标准。

表 0.4　　测试计划表（以顺序表结构为例）

测　试　项　目	测　试　步　骤	期　望　结　果	测　试　目　的
顺序表插入元素测试	1. 插入元素，输入合理插入位置 2. 列出表信息，观察结果	元素正确插入到特定位置	正面测试，测试插入功能的正常使用情况
顺序表插入元素测试	1. 插入元素，输入不合理的插入位置 2. 观察程序输出状态	程序提示输入位置不合理而不会崩溃	负面测试，测试插入位置错误情况下的程序健壮性
顺序表插入元素测试	1. 销毁顺序表 2. 在合理的位置插入元素，观察结果	程序提示表已销毁，不能插入元素	负面测试，测试在表销毁情况下插入元素的程序健壮性
…	…	…	…

5. 缺陷记录表（见表 0.5）

对项目的测试应该严格依据之前制订的测试计划表来进行，在测试的过程中每发现一个程序缺陷都应该详细记录。在修复缺陷之后，还要进行回归测试，并跟踪每一个缺陷的修复状态。

表 0.5　　缺陷记录表（以顺序表结构为例）

序号	程序缺陷说明	修　复　情　况	备　　注
1	空表删除元素导致程序崩溃	已修复	严重 bug，必须修复
2	表中元素数量达到表容量后，插入元素不会自动扩充表容量	待修复	可采用 2 倍递增的方式扩充空间
…	…	…	…

第1章 抽象数据类型项目实训

对于一些基本数据类型无法描述的信息结构，我们可以将其逻辑特性抽象出来，形成一组数据以及对这些数据的一组逻辑操作，这称为**抽象数据类型**。

逻辑特征抽象层次越高则抽象数据类型的复用性程度也越高；因此有必要掌握如何定义一个抽象数据类型，以及如何用代码来实现该数据类型。

一、本章实训目的

1. 用 C 或 C++语言实现一个抽象数据类型；
2. 实现一个用户操作界面来验证该数据类型；
3. 运行程序并对其进行测试。

二、实训项目要求

编写程序实现三元组抽象数据类型 Triplet，并实现一个主函数调用三元组的所有函数操作。

```
ADT Triplet {
    数据对象：D = {e1, e2, e3 | e1, e2, e3 ∈ Elemset}
    数据关系：R1 = {<e1, e2>, <e2, e3>}
    基本操作：
        InitTriplet(&T, v1, v2, v3)
        操作结果：构造三元组 T，元素 e1、e2、e3 分别赋值为 v1、v2、v3。
        DetroyTriplet(&T)
        操作结果：三元组 T 被销毁。
        Get(T, i, &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 元值。
        Put(&T, i, 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 元值为 e。
        IsAscending(T)
        初始条件：三元组 T 已存在。
        操作结果：若 T 的 3 元素按升序排列，则返回 1，否则返回 0。
        IsDescending(T)
        初始条件：三元组 T 已存在。
        操作结果：若 T 的 3 元素按降序排列，则返回 1，否则返回 0。
        Max(T, &e)
```

```
        初始条件：三元组 T 已存在。
        操作结果：用 e 返回 T 的三个元素最大值。
        Min(T, &e)
        初始条件：三元组 T 已存在。
        操作结果：用 e 返回 T 的三个元素最小值。
} ADT Triplet
```

三、重要代码提示

使用连续的空间来存储三元组中的元素，如果是动态三元组的内存，则只用定义三元组类型 Triplet 为其元素类型的指针。

```
// Triplet 类型是 ElemType 类型的指针，存放 ElemType 类型的地址
typedef ElemType *Triplet; // 由 InitTriplet 分配 3 个元素存储空间
```

三元组初始化函数 InitTriplet 传入已定义的 Triplet 变量的引用 T（因此函数要将新申请的空间指针赋予三元组变量，所以参数必须传入三元组变量的引用），同时传入三元组的 3 个初值 *v*1、*v*2、*v*3。在函数中首先为 T 申请 3 个 ElemType 类型的空间，如果申请失败则直接退出程序，否则将 *v*1、*v*2、*v*3 分别赋值给 T[0]、T[1]、T[2]，参考如下代码。

```
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 3 个元素的初值为 v1、v2 和 v3
    if (!(T = (ElemType *) malloc(3 * sizeof(ElemType))))
    {
        exit(OVERFLOW);
    }
    T[0] = v1, T[1] = v2, T[2] = v3;
    return OK;
}
```

因为三元组 T 是连续分配的空间，所以销毁三元组的函数 DestroyTriplet 只需要直接释放 T 所指向的空间，随机将 T 置为空即可。

```
Status DestroyTriplet(Triplet &T)
{
    // 操作结果：三元组 T 被销毁
    free(T);
    T = NULL;
    return OK;
}
```

获取三元组元素值的函数 Get 通过参数 *i* 传入元素顺序，通过引用 *e* 传出元素值。首先判断 *i* 的合法性，因为是三元组，所以 *i* 值只能在 1～3。因为下标编号是从 0 开始，所以 T[*i*−1] 是三元组第 *i* 元的值，最后将 T[*i*−1]的值赋予 *e*。

```
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;
}
```

修改三元组元素值的函数 Put 和函数 Get 类似，只是将要修改的值通过参数 *e* 传递进来，并将其赋值给 T[*i*−1]，在此之前同样要对 *i* 的合法性进行判断，见如下代码。

```
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;
}
```

判断三元组是否升序排列的函数 IsAscending，实现起来很简单，只需要比较相邻的两元素是不是后者比前者大，如果后者都比前者大则返回 1，否则返回 0。

```
Status IsAscending(Triplet T)
{
    // 初始条件：三元组 T 已存在
    // 操作结果：如果 T 的 3 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]);
}
```

判断降序排列的函数 IsDescending 和 IsAscending 类似，代码略去。

从三元组中找出最大元素的函数 Max，首先比较 T[0]和 T[1]，将大的赋值给引用参数 *e*，然后再比较 *e* 和 T[2]，将大的赋值给 *e* 即可。找寻最小数的函数 Min 和函数 Max 类似，具体代码不再赘述。

```
Status Max(Triplet T, ElemType &e)
{
    // 初始条件：三元组 T 已存在
    // 操作结果：用 e 返回指向 T 的最大元素的值
    e = T[0] >= T[1] ? T[0] : T[1];
    e = e >= T[2] ? e : T[2];
    return OK;
}
```


此为试读，需要完整PDF请访问：www.ertongbook.com

第 2 章
线性表项目实训

线性表是具有相同属性的数据元素的一个有限序列，是最简单最常用的一种数据结构，其基本特点是结构中的元素之间满足线性关系。

一、线性表的基本操作

1. 创建表；
2. 求表长度；
3. 查找元素；
4. 插入元素；
5. 删除元素；
6. 遍历元素。

二、线性表的实现形式

1. 顺序存储形式（顺序表）；
2. 链式存储形式（单链表、双向链表、循环链表）；
3. 静态存储形式（静态链表）。

三、本章实训目的

1. 用 C 或 C++语言实现本章所学的各种线性表结构；
2. 编写线性表的基本操作函数（求长度、查找、插入、删除、遍历等）；
3. 实现一个对线性表进行各种操作的用户界面（如图 2.1 所示）；

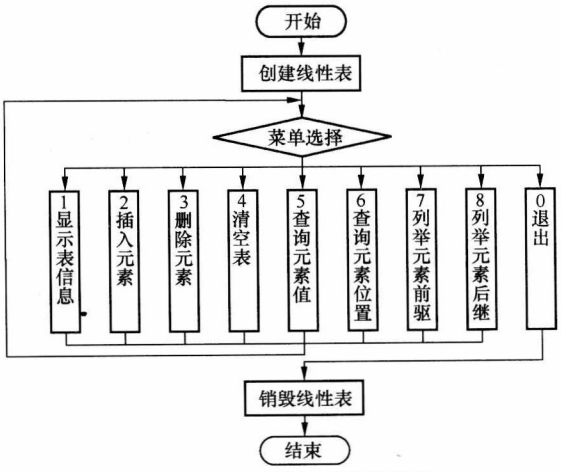

图 2.1　线性表操作程序流程图

4. 运行程序并对其进行测试。

2.1 顺 序 表

一、顺序表结构特点

1. 采用一组地址连续的存储单元来存储数据；
2. 表的存储容量长度不易改变；
3. 方便随机查找；
4. 适应于频繁访问元素的应用；
5. 不适用于频繁增删元素的应用。

顺序表存储结构如图 2.2 所示。

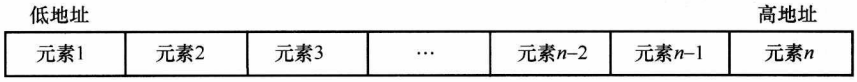

图 2.2 顺序表存储结构示意图

二、实训项目要求

开发一个顺序表的操作程序，要求程序至少具备以下顺序表的操作接口：

- InitList（顺序表初始化函数）；
- DestroyList（顺序表销毁函数）；
- ClearList（顺序表清空函数）；
- ListEmpty（判断是否为空表的函数）；
- ListLength（计算表长的函数）；
- GetElem（依据位置查询元素值的函数）；
- LocateElem（判断元素位置的函数）；
- PriorElem（查询前驱函数）；
- NextElem（查询后继函数）；
- ListInsert（元素插入函数）；
- ListDelete（元素删除函数）；
- ListTraverse（元素遍历函数）。

要求程序还应具有任用户选择操作的菜单，并支持以下菜单项：

1. 列举表元素；
2. 插入元素；
3. 删除元素；
4. 清空表；
5. 查询元素值；
6. 查询元素位置；
7. 列举前驱元素；
8. 列举后继元素；

9. 退出程序。

三、重要代码提示

在构造结构之前应该确定顺序表的元素数据类型、表初始存储容量以及每次扩大顺序表的容量时所分配的增量，参考以下代码。

```
typedef int ElemType;           // 用户指定数据类型
#define LIST_INIT_SIZE 10       // 线性表存储空间的初始分配量
#define LIST_INCREMENT 2        // 线性表存储空间的分配增量
```

一个完整的顺序表结构必须拥有的属性包括存储空间的起始地址、顺序表的数据长度以及存储容量。因此，在顺序表结构定义中包含了 3 个成员变量以对应 3 个必要的顺序表属性。参考如下代码，定义一个顺序表结构 SqList，其中存储空间起始指针变量 elem 必须定义为顺序表元素的数据类型 ElemType 的指针，而表容量 listsize 则以 sizeof(ElemType)为单位。

```
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
};
```

下面给出了构造空线性表的参考函数 InitList，此处采用 SqList 变量的引用作为函数参数传递进函数进行处理，因此必须要有明确定义的 SqList 变量才能调用该函数。需要注意的是，顺序表的空间是通过动态内存分配申请的，因此在销毁顺序表的时候必须要调用 free 函数释放内存。

```
void InitList(SqList &L)   // 创建顺序表
{
    // 初始条件：顺序表 L 已定义
    // 操作结果：构造一个空的顺序表 L
    L.elem = (ElemType*) malloc(LIST_INIT_SIZE * sizeof(ElemType));
    if (!L.elem)
    {
        exit(OVERFLOW);     // 存储分配失败则退出程序
    }
    L.length = 0;           // 空表长度为 0
    L.listsize = LIST_INIT_SIZE; // 初始存储容量
}
```

对于顺序表而言，依据元素的位置查找元素值实现起来非常方便，下面的代码实现了随机查找顺序表元素的函数 GetElem。利用 ElemType 型变量 e 的引用传递出查询到的元素值，因此调用函数前要先定义变量 e，而对于传入元素编号的变量 i 一定要在函数中判断其值是否在顺序表长度以内，以免对顺序表的越界访问。

```
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)
```

```
    {
        return ERROR;
    }
    e = *(L.elem + i - 1);
    return OK;
}
```

根据元素值满足的条件来查询元素编号的函数 LocateElem 可以设计得比较灵活，因为元素值和条件比较结果是一个二元逻辑值（真和假），所以可以约定一个回调函数形式来比较元素值和参考值，并约定回调函数的 Status 类型返回值来判定元素值和参考值之间的比较结果。所以用户可以自由地设计回调函数，并将函数指针通过 compare 参数传递进 LocateElem 函数。

下面给出了参考代码，依次将顺序表的元素和参数 *e* 调用 compare 所指向的函数进行比较处理，如果结果为 1 则代表查询成功，否则查询失败，查询到的元素编号通过函数返回。

```
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
    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;
    }
}
```

顺序表的元素插入函数 ListInsert 的关键操作在于插入元素之前要先判定插入位置 *i* 的合法性，以及顺序表是否还有足够的空间以供元素的插入。参考如下代码，若线性表的存储空间已满，则调用 realloc 函数增加 LIST_INCREMENT 个元素的存储空间，同时将顺序表的容量值增加 LIST_INCREMENT。由于顺序表的元素在内存中是连续存储，所以插入元素之前，要从第 *i* 个元素开始，将后续所有元素后移一个单位；注意，元素插入成功之后，一定要将顺序表的长度值增加 1。

```
Status ListInsert(SqList &L, int i, ElemType e)
{
    // 初始条件：顺序线性表 L 已存在，1≤i≤ListLength(L)+1
    // 操作结果：在 L 中第 i 个位置之前插入新的数据元素 e，L 的长度加 1
    ElemType *newbase, *q, *p;
```

```
    if (i < 1 || i > L.length + 1)        // i值不合法
    {
        return ERROR;
    }
    if (L.length >= L.listsize)           // 当前存储空间已满,增加分配
    {
        if (!(newbase = (ElemType *)realloc(L.elem,
                  (L.listsize + LIST_INCREMENT) * sizeof(ElemType))))
        {
            exit(OVERFLOW);               // 存储分配失败
        }
        L.elem = newbase;                 // 新基址
        L.listsize += LIST_INCREMENT;     // 增加存储容量
    }
    q = L.elem + i - 1;                   // q为插入位置
    for (p = L.elem + L.length - 1; p >= q; --p)
    {
        // 插入位置及之后的元素右移
        *(p + 1) = *p;
    }
    *q = e;                               // 插入e
    ++L.length;                           // 表长增1
    return OK;
}
```

类似于插入元素，顺序表在删除第 i 个元素时，也是先判定 i 是否合法，然后将第 i 个元素之后的每个元素向前移动一个单位，并将表长度减 1 即可。即使顺序表之前在插入元素时扩大过存储容量，在删除一些元素之后也不需要缩小顺序表的容量，函数 ListDelete 的实现参考如下代码。

```
Status ListDelete(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：删除L的第i个数据元素，并用e返回其值，L的长度减1
    ElemType *p, *q;
    if (i < 1 || i > L.length)            // i值不合法
    {
        return ERROR;
    }
    p = L.elem + i - 1;                   // p为被删除元素的位置
    e = *p;                               // 被删除元素的值赋给e
    q = L.elem + L.length - 1;            // 表尾元素的位置
    for (++p; p <= q; ++p)                // 被删除元素之后的元素左移
    {
        *(p - 1) = *p;
    }
    L.length--;                           // 表长减1
    return OK;
}
```

获取顺序表某个元素的前驱元素或后继元素也是一个比较重要的操作。此类操作首先要

判定参考值 cur_e 是否为顺序表 L 的元素，如果是 L 的元素则获取其前驱或后继，需要特别考虑的是线性表第 1 个元素没有前驱，最后一个元素没有后继。下面给出获取前驱元素的函数 PriorElem，获取后继元素的函数 NextElem 与之类似。

```
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 1 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++;
    }
    if (i > L.length)
    {
        return INFEASIBLE;           // 操作失败
    }
    else
    {
        pre_e = *--p;
        return OK;
    }
}
```

对顺序表元素进行遍历的函数是一个重要的函数，应该设计得比较灵活，因此同样采用用户自由定义的回调函数来处理顺序表中的每一个元素值。为了能让回调函数能够改变顺序表的元素值，所以回调函数所约定的参数应该是 ElemType 类型的引用。顺序表元素遍历函数 ListTraverse 函数的定义如下。

```
void ListTraverse(SqList L, void(*vi)(ElemType&))
{
    // 初始条件：顺序线性表 L 已存在
    // 操作结果：依次对 L 的每个数据元素调用函数 vi()，
    //           vi()的形参加'&'，表明可通过调用 vi()改变元素的值
    ElemType *p;
    int i;
    p = L.elem;
    for (i=1; i<=L.length; i++)
    {
        vi(*p++);
    }
}
```

对顺序表进行操作的主函数可以设计如下，首先定义顺序表变量 sqList，然后将其引用传入函数 IniteList 构造一个空的顺序表，接下来循环调用菜单选择函数 LoopCommand，让用户自由选择操作项对顺序表进行操作，直到函数返回 0 为止，最后调用 DestroyList 函数销毁顺序表。

此为试读，需要完整PDF请访问：www.ertongbook.com