高职高专精品课程规划教材 计算机系列

数据结构实训与案例分析

李 筠 姜学军 主 编
王艳梅 陈腾蛟 副主编

- 以国家级、省级优秀精品课程为基础
- 配有强大的网络教学资源：教学视频、案例、项目实践等
- 提供网上实践平台，可直接进行系统化、项目化实践
- 实现课程结构与内容实战化、职业化

清华大学出版社

高职高专精品课程规划教材　计算机系列

数据结构实训与案例分析

李　筠　姜学军　主　编
王艳梅　陈腾蛟　副主编

清华大学出版社
北 京

内 容 简 介

本书是高职高专精品课程系列教材《数据结构》的配套教材，适合计算机及相关专业数据结构课程设计或实训教学使用。

书中全面地介绍了不同数据结构的应用，从实际例子出发，分析逻辑结构和存储结构，分析如何应用数据操作解决实际问题和完成相应的功能。不同的数据结构配有实例，每个实例都从问题出发，分析数据结构和算法操作步骤，然后给出设计过程，并完成代码设计与调试。分析中配合了流程图、操作步骤分解图和测试结果图等，力求使学习者能深入理解并提升数据结构的设计和应用能力。

本书内容共有两篇，第一篇为实训篇，针对数据结构课程设计内容，叙述了几种不同数据结构的应用和实例，有线性表、栈、队列、二叉树、树、图、查找和排序这几种不同实例的设计过程。第二篇为习题分析与解答，针对《数据结构》教材中的习题做出全面分析和解答。

书中第一篇中的所有实例代码均调试通过，并配有电子版代码。

图书在版编目(CIP)数据

数据结构实训与案例分析/李筠，姜学军主编. --北京：清华大学出版社，2013
(高职高专精品课程规划教材　计算机系列)
ISBN 978-7-302-33495-8

Ⅰ. ①数… Ⅱ. ①李… ②姜… Ⅲ. ①数据结构—高等职业教育—教材 Ⅳ. ①TP311.12

中国版本图书馆 CIP 数据核字(2013)第 189080 号

责任编辑：桑任松
封面设计：刘孝琼
版式设计：杨玉兰
责任校对：周剑云
责任印制：杨　艳

出版发行：清华大学出版社
网　　址：http://www.tup.com.cn，http://www.wqbook.com
地　　址：北京清华大学学研大厦 A 座　　邮　　编：100084
社 总 机：010-62770175　　邮　　购：010-62786544
投稿与读者服务：010-62776969，c-service@tup.tsinghua.edu.cn
质 量 反 馈：010-62772015，zhiliang@tup.tsinghua.edu.cn
课 件 下 载：http://www.tup.com.cn,010-62791865

印 刷 者：北京世知印务有限公司
装 订 者：三河市溧源装订厂
经　　销：全国新华书店
开　　本：185mm×260mm　　**印　　张**：14.5　　**字　　数**：345 千字
版　　次：2013 年 9 月第 1 版　　**印　　次**：2013 年 9 月第1 次印刷
印　　数：1～3000
定　　价：28.00 元

产品编号：051194-01

《高职高专精品课程规划教材》丛书序

教育部于2003年开始在全国高等学校(包括高职高专院校)中启动高等学校教学质量与教学改革工程精品课程建设工作(简称精品课程建设)，随后许多省份和高校也启动了省级和校级精品课程建设工作。经过5年的发展，精品课程建设已经进入成熟期，网上涌现了大量的优质课程资源，充分利用这些资源，无论对学生学习课程还是教师的教学都有积极的促进意义。

系列化的优秀教材与精品课程相呼应非常有必要，不但可以使优质的教学资源以教材为载体最大限度地得到共享和利用，而且教材的沉淀、积累和推广也将反过来促进精品课程资源的不断完善。

现在各个高职高专院校都以就业为导向，把对学生的技能培养作为首要目标。因此本套丛书以“体现职业教育教学特点和强调现代教育技术应用”为原则，以提高课堂与实践的教学效率和效果为主旨，努力建设一套全新的、有实用价值的精品课程配套规划系列教材，并希望能够通过这套教材的出版和使用，促进优秀精品课程的发展，最大限度地发挥精品课程的“精品”与“网络课程资源立体化”的优势，使之成为一套适应社会需求，有利于培养高素质高技能人才的优秀系列教材。

本系列丛书具有以下特点：

- 以国家级、省级优秀精品课程为基础。
- 配有强大的网络教学资源——教学视频、案例、项目实践等。
- 提供网上实践平台，可直接进行系统化、项目化实践。
- 实现课程结构与内容实战化、职业化。

精品课程在各个方面都已经比较成熟，所以本丛书力求在实用性上更加突出，注重技术能力的培养，提倡动手实践。每个单元小节后都有必要的习题和实训案例。大部分教材还专门配有实验与实训指导。使读者在掌握基本知识的同时，还可以获得实际操作的基本技能。

每本教材都配有内容细致全面的网站，为教师免费提供电子教案、案例库、习题库；为教师和学生共同开设专题讨论网络空间，可实现更大范围的教与学互动，即时解决教学过程中遇到的问题。在帮助老师教学的同时，更能培养学生的学习兴趣，通过自己动手实践来提高专业技术能力。

本系列教材主要针对高职高专院校，以三年制高职教育为主，同时也适用于同等学历的职业教育。希望通过本系列教材的编写和推广应用，不仅能够有利于提高精品课程的整体水平，而且能够通过精品课程成熟的教学经验和丰富的网络教学资源，更有助于加快职业技术教育的改革步伐。

前　言

数据结构实训是数据结构课程和程序训练的重要环节。用计算机求解任何问题都离不开程序设计，而程序设计的实质是数据表示和数据处理。在程序的设计中，如何选择数据结构是设计考虑的基本因素，最优的数据结构的选择是系统实现的困难程度和构造的质量的关键问题，因此学会数据结构的应用是非常重要的。数据结构的理论抽象、难理解，算法复杂，使学习者感到学习困难，至于应用，更无从下手。因此以实际例子将数据结构应用的设计过程展示出来，不仅能深化对数据结构的理解，也能提升数据结构的应用能力。

数据的逻辑结构可分为线性结构、树型结构、图(网)和集合四种，其中线性结构又可分为线性表、栈、队列。当逻辑结构确定后，根据实际需要，可以采用顺序或链式存储结构存放数据，复杂的存储结构还可以采用两种不同存储结构的结合方式。当数据的逻辑结构和存储结构确定后，就可以编写算法实现，对数据进行操作。本书《数据结构实训与案例分析》是以引导数据结构的应用为导向，一般先提出一个实例，然后从逻辑结构、存储结构和算法三方面设计。先选择数据结构，然后是问题的分析与实现，最后给出完整可运行的源程序和测试结果，通过这样一种方式，将不同数据结构实例的设计过程展示出来。

本书是高职高专精品课程系列教材《数据结构》的配套教材，也是数据结构课程实训或软件实训的教学用书，是理想的软件方面实训的教材。利用数据结构实训过程，使学生能够以问题求解方法、程序设计方法及一些典型的数据结构算法为研究对象，学会分析数据对象的特征，掌握数据组织的方法和在计算机中的表示方法，培养良好的程序设计风格、程序设计的技能及创造性思维的方法，从而达到应用知识解决复杂问题的目的。

本书第一篇主要为数据结构实训内容，包括线性表、栈、队列、树与二叉树、图、查找和排序中的不同例子。每个例子都有完整的设计过程，将问题分析、数据结构定义、功能函数设计、主要算法分析、完整代码、测试过程逐一给出，所有的代码均已调试通过(在Turbo C 环境下)。第二篇为针对配套教材《数据结构》中的习题的答案，答案中有分析有算法。

《数据结构》和《数据结构实训与案例分析》这两本配套教材将数据结构的理论、实验、实训或课程设计、习题与习题解答、案例分析等内容完整地结合在一起。有理论、有实验、有提出的问题，也有解答的方案，不论是作为教材或自学，都是理想的选择。

本书作为高职教材，如果用于数据结构实训或计算机软件实训，参考学时为 2~4 周。

本书是作者在多年从事 C 语言、数据结构以及计算机软件课程教学工作和计算机软件开发工作的基础上编写的，由沈阳理工大学教师李筠、姜学军主编，王艳梅、陈腾蛟副主编，宋洪波、田悦、徐志勇、苑擎飏、姚旭东、袁凤莲、马永轩、许亮、王海涛等参编。

由于作者水平有限，书中难免存在错误之处，欢迎读者提出宝贵意见。

编　者

目　录

第一篇　实训与案例分析

第二篇　配套教材习题解答

第一篇 实 训 篇

第1章 概 述

1.1 课程实训的作用

数据结构的教学环节分为三个方面：数据结构理论、数据结构实验、数据结构课程实训。针对以上几个方面，学习者都感到有一定的困难，原因有以下几点：①数据结构理论内容抽象；②动态存储结构难以理解；③使用多种技术(递归等掌握较为困难)；④算法描述和算法设计无从下手等。

数据结构课程实训过程可以将数据结构理论、数据结构实验和实际问题很好地结合在一起。通过对实际问题功能设计的编码实现，使学生能针对问题按数据结构的指导思想存储数据，按算法实现的步骤完成功能模块的设计，按规范的软件编码理论进行程序设计，最后完成编码调试。

数据结构的学习过程是进行复杂程序设计的训练过程。技能培养的重要程度不亚于知识传授，学习过程不仅要有理论内容，还应培养应用知识解答复杂问题的能力，形成良好的算法设计思想、方法技巧与风格，进行构造性思维，强化程序抽象能力和数据抽象能力。因此，学习数据结构，仅从书本上学习是不够的，必须经过大量的实践，在实践中体会构造性思维的方法，掌握数据组织与程序设计的技术。

《数据结构实训与案例分析》就是从问题出发，从实训的角度分析如何选择数据的逻辑结构和存储结构，分析功能实现的过程，并完成算法选择、编码实现及调试分析这几个过程。书中第一篇“实训篇”就是针对数据的不同逻辑结构，在每种逻辑结构上选择 1~2 个不同的实际例子进行设计的。实训篇共 5 章，除了本章内容外，还有其他 4 章，包括线性表应用、树与二叉树的应用、图的应用、查找和排序的应用，各章包含实例如下。

(1) 线性结构的应用共 6 个实例。

① 线性表顺序存储结构的实现。

② 线性表链式存储结构的实现。

③ 学生信息管理系统的设计。

④ 航班信息管理设计。

⑤ 表达式求值的设计。

⑥ 外设与 CPU 同步输出的设计。

(2) 树和二叉树的实例共 3 个。

① 二叉树顺序结构的实现。

② 二叉树链式结构的实现。

③ 哈夫曼编码设计与应用。

(3) 图的实例有两个。

① 最短路径的应用与实现。

② 高校排课系统的应用。

(4) 查找和排序部分设计了一个通信录的实例。

上述实例从不同的方面表现了数据结构的应用，通过每种数据结构的具体实例，循序渐进地启发完成数据结构的应用设计。每个实例都从提出问题、设计要求开始，到选择使用的数据结构、问题的分析与实现，最后给出完整可运行的源程序，同时给出了测试样例。学习者也可以在具体实例的基础上，自己去开发设计，举一反三，真正提高实践的能力。

课程实训是学生对课程所学知识的综合运用，它与课堂听讲、上机实验、课外练习、自学研究相辅相成，构成一个完整的课程教学体系。“数据结构”是一门实践性强的课程，其中对算法设计和程序编写的掌握尤为重要。学生虽然可以通过与课堂教学同步的上机实验完成相关内容的练习，但却往往局限于一些功能简单、彼此之间关系独立的算法和程序。课程实训是一种综合训练，致力于培养学生全面、灵活的算法设计思想和较高的编程能力，为今后从事计算机开发与应用打下基础。

编写本书的出发点不是要给学生几个课程实训实例，而是希望通过一些典型的课程实训实例训练，使学生掌握如何利用数据结构知识去解决实际问题。

1.2 课程实例的编写过程

“数据结构”课程实训中牵涉到主要章节的基本理论，包括线性结构(线性表、栈、队列)、图、树的特点、存储方式、运算原理和方法、典型应用和两种重要操作查找、排序的基本原理和方法。本书中的每个实例都是从以下几方面说明设计过程的。

(1) 任务的提出和分析问题：根据设计题目的要求，充分地分析和理解问题，明确问题要求做什么，限制条件是什么，明确所要实现的操作。

(2) 数据结构设计：对问题描述中涉及的操作对象确定逻辑结构和物理结构，并定义相应的数据类型。

(3) 函数设计：确定各个主要模块间的关系，写出主要函数的算法框架或画出流程。考虑系统功能，使得系统结构清晰、合理、简单和易于实现。

(4) 程序编码：用程序设计语言实现各个函数的设计。编写实现函数功能的代码，加入一些注解，使程序中的逻辑概念清晰。

(5) 程序运行结果分析：设计测试数据，说明运行结果。

1.3 课程实训的实施

整个课程实训一般分为四个阶段来完成。第一阶段为选题及准备阶段；第二阶段为设计、编程及调试阶段；第三阶段为测试及检查阶段；第四阶段为学生编写课程实训实验报告阶段。前面三个阶段本书内容中都有体现，下面说明第四阶段如何编写课程实训报告。

课程实训报告的参考内容如下。

1. 需求分析

这个部分说明程序设计的任务，说明程序要实现哪些功能。输入数据或给出的参数，输出数据或输出参数。

2. 概要设计

这个部分说明实现任务中用到的数据结构、主要功能模块、模块之间的层次(调用)关系。

3. 详细设计

这个部分给出定义的所有数据类型，写出算法或流程图。对主程序和其他模块也都需要写出伪码算法。

4. 调试分析与测试结果

这个部分的内容包括：

- 调试过程中遇到的问题是如何解决的。
- 算法的时间和空间分析及改进设想。
- 列出测试结果，包括输入和输出。

5. 课程设计总结

这个部分说明课程设计中的体会，有哪些收获。

6. 参考文献

在此列出参考的相关资料和书籍。

第2章 线性结构

线性结构是数据关系最简单的结构，元素之间是一维线性关系。它的特点是：除第一个元素和最后一个元素外，每个数据元素有唯一的前驱和唯一的后继。第一个元素(又称首元素)没有前驱，最后一个元素(又称尾元素)没有后继。数据结构的线性结构有线性表、栈、队列等，本章通过不同的实例说明线性结构的应用。

2.1 线性表应用

线性表是具有 n(n≥0)个元素的有限序列，当 n=0 时，是空表；当 n>0 时，除第一个元素和最后一个元素外，其他元素有且仅有唯一的前驱和唯一的后继，第一个元素无前驱，最后一个元素无后继。线性表上定义的基本操作有：建立线性表、销毁线性表、在线性表的第 i 个位置插入元素 e、删除线性表的第 i 个元素、按元素值查找其在表中的位置、取出线性表中第 i 个位置的元素、求线性表的长度等。

线性表的基本操作定义了解决线性表问题的基本(或较小)的操作集，当实现复杂问题时，将扩充基本操作集。比如解决线性表一类的问题时可以通过线性表上的基本操作实现复杂的运算。例如：学生信息管理问题实现的基本操作是对学生信息的查找、插入、删除、修改等。如果要实现两个表的合并，基本操作中没有定义，可以通过基本操作实现两线性表的合并操作，实现过程见《数据结构》第 2 章。还有一些操作如查找和排序操作也是基本操作没包含的，这些不包含在基本操作中的可以通过调用基本操作的运算来实现。当解决具体问题时，定义的操作一定包含基本操作，是否加入基本操作外的一些操作可以根据其调用频率来确定。

实现操作算法，首先是建立一种存储结构。线性表有两种存储结构，顺序存储结构和链式存储结构。顺序存储结构是用连续空间存储线性表，设线性表中有 n 个元素，每个数据元素需占用 k 个存储单元，第一个元素的存储地址作为存放元素的起始位置。如图 2.1 所示给出了线性表的顺序存储结构。

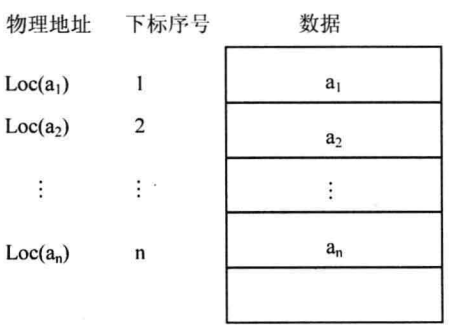

图 2.1 线性表的顺序存储结构

在顺序表中，元素 a_i 的存储地址是该节点在表中的逻辑位置 i 的线性函数。假设线性表中第一个元素 a_1 的存储位置为 $Loc(a_1)$，第 i 个元素 a_i 存储位置为 $Loc(a_i)$，k 为每个元素存储时所占的字节数。则两元素之间满足关系：

$$Loc(a_i) = Loc(a_1) + (i-1)\times k$$

顺序存储结构可以借助于高级程序设计语言中的一维数组来表示，一维数组的下标与元素在线性表中的序号相对应。因为有插入、删除等操作，所以线性表占用的空间是变化的，线性表中要有一个记载实际长度的整数，因此线性表用 C 语言可以描述成一个结构体，结构体包括一个数组向量和两个整型数。

线性表的链式存储结构是用一组不连续的存储单元来存放线性表中的数据，因此链表中节点的逻辑次序和物理次序不一定相同。为了正确地表示节点间的逻辑关系，在存储线性表时，存储每个数据元素值的同时，还要存储指示其后继节点的地址(或位置)信息，这两部分信息组成的存储映像称为节点。线性链表是用 n 个节点连接起来的，每个节点包含一个数据域和一个指针域，指针域指向表中的直接后继，如图 2.2 所示。

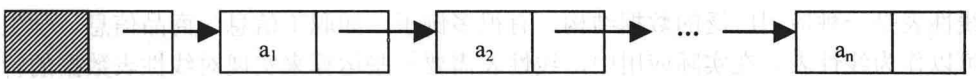

图 2.2 线性表的链式存储结构

不同的存储结构操作算法的实现效率(即时间复杂度和空间复杂度)不同，具体问题中选择哪种存储结构是根据实现问题需要的操作效率决定的。下面是线性表不同存储结构上算法实现的优点和缺点。

(1) 从空间的角度看。

① 线性表的顺序存储结构只需要存储数据元素，无需为存储前驱或后继关系占用空间，因为物理相邻的数据逻辑上也相邻；链式存储结构除了存储数据元素外，还要有一个指针存储数据关系。所以从空间的角度看，顺序存储结构比链式存储结构占用空间少。

② 顺序存储结构要求开辟一块连续的空间，如果空间大小不够，就不能分配空间。而链式存储结构既可以分配一块连续存储空间，也可以是不连续存储空间，只要有一块可存放元素的空间(包括值链域)就可以分配给链表利用，所以链表在空间的利用上是灵活有效的。

(2) 从操作的角度看。

① 顺序存储结构方便实现随机存取或修改第 i 个数据元素，链式存储结构上要存取或修改第 i 个元素，首先要从链表的头指针开始查找到第 i 个元素，然后才能实现修改操作，所以随机存取或修改某一个元素在顺序存储结构中实现更便利。

② 插入或删除数据元素时，在顺序存储结构上要将操作点后面的元素向前或向后移动，链式存储结构上，只需改变链指针。所以插入或删除操作链表的优势也明显高于线性表的顺序存储结构。

综合两种存储结构的特点，选择线性表的存储结构时，要充分考虑各种操作的效率，选择最合适的存储结构。

2.1.1 线性表在顺序结构上的基本操作的设计

1. 数据类型定义

线性表在顺序存储结构上的数据类型定义如下：

```
typedef struct                    /*线性表在顺序表存储时的类型定义*/
{
    ElemType *elem;               /*存放线性表的一维数组向量*/
    int  Length, Listsize;        /*线性表的长度和线性表的最大空间*/
} SeqList;
```

ElemType 数据元素类型为整型，定义如下：

```
typedef int ElemType;             /*线性表数据类型为整型*/
```

2. 线性表顺序结构功能函数设计

线性表是一种应用广泛的数据结构，有很多例子，如职工信息、商品信息、学生信息等都可以作为线性表。在实际应用中，线性表需要一些运算来实现对线性表数据的管理。主要函数如下。

(1) 创建顺序表：

```
InitList(SeqList  *L, int n)
```

建立长度为 n 的线性表 L，存放数据元素，线性表的长度 n 和数据元素值由键盘输入。

(2) 插入元素：

```
InsElemList(SeqList *L, int i, ElemType x)
```

在线性表 L 中，第 i 个位置前插入元素 x。用于实现在表中按位置添加一个数据元素，操作的结果是将增加的元素插入到第 i 个位置前。插入时首先判断 i 是否合法？还要判断空间是否够用，如果不够，则增加空间。最后移动元素，实现插入并将线性表长度加 1。

(3) 删除元素：

```
DelElemList(SeqList *L, int i, ElemType *x)
```

在线性表 L 中，删除第 i 个数据元素，删除前先判断 i 位置是否正确或者是否是空表，条件成立则显示删除失败，否则将 i 后面的元素向前移动，长度减 1。若完成元素删除的操作，将删除值保存在 x 中。

(4) 按位置查找元素(即取某个元素)：

```
GetElemList(SeqList  *L, int i, ElemType *x)
```

用于查看线性表 L 中的第 i 个元素，并将线性表中的第 i 个元素存入 x 中。

(5) 按值查找元素位置：

```
LocateList(SeqList *L, ElemType x)
```

在线性表 L 中查找值为 x 的数据元素的位置，如果查找成功，返回 x 元素的位置，如果没查找到，返回 0(这个返回值应该是数组向量中不存在的下标，如果数组中下标为 0 的

单元存放数据元素，此处返回值可以为-1)。在顺序存储结构中，元素的位置与此元素的下标位置相关，因此可以通过元素的位置实现检索或修改元素的值。

(6) 显示顺序表：

```
OutputList(SeqList *L)
```

将线性表 L 的所有数据元素显示输出，用于观察线性表的变化。当插入、删除元素后，线性表发生变化，通过显示线性表中的数据将这种变化表现出来。

(7) 显示菜单。Menu()函数将实现的功能通过菜单显示。显示菜单中各个选项与各功能对应，如图 2.3 所示。当选择菜单 2 后，执行代码为上面第 2 条对应的函数 InsElemList()，完成插入元素的功能。线性表的操作是通过菜单来调用函数，实现插入、删除、查询、修改等功能。为了便于观察程序的运行状况，设置一个显示函数。当线性表中的数据有变化时，通过显示函数观察 L 表中数据元素的变化。

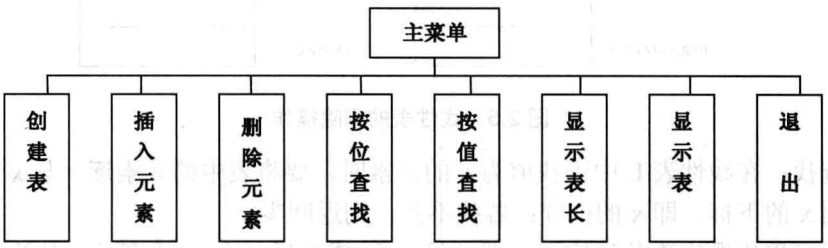

图 2.3　主菜单功能模块

3. 一些算法的设计

(1) 插入：如果线性表中有 n 个元素，在线性表的第 i 个位置前插入元素 x。实现过程如下。判断空间大小是否够，若已满，则开辟新空间，若开辟空间失败或 i 的位置是不合理，结束。否则将线性表第 n 个元素(元素下标为 L.len)到第 i 个数据元素，每个元素向下移动一个单元。移动后第 i 和 i+1 个位置都存放 a_i 元素。将插入元素 b 放入第 i 个单元，线性表长度加 1，插入结束，如图 2.4 所示。

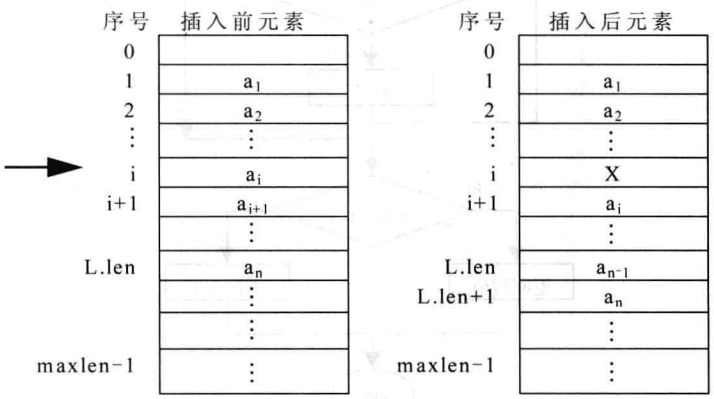

图 2.4　线性表的插入操作

(2) 删除：删除线性表中第 i 个元素时，先判断线性表中是否有元素、要删除元素的

位置是否正确。如果线性表中无数据元素或位置出错，就结束程序并返回 0，否则就做删除工作。将第 i+1 ~ L.len 个位置的数据元素全部上移一个单位，最后线性表长度减 1，如图 2.5 所示。

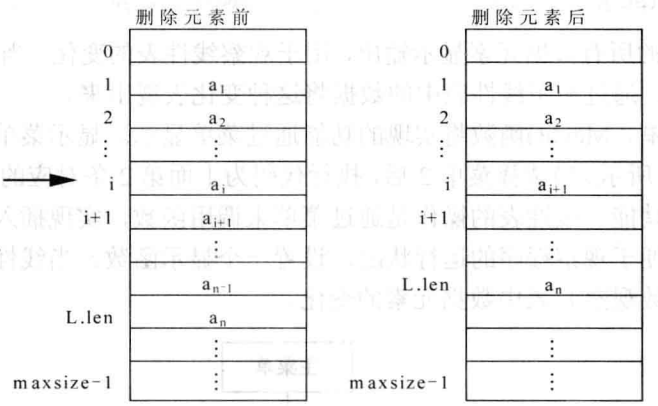

图 2.5　线性表的删除操作

(3) 查找：在线性表 L 中查找值为 x 的元素时，要将表中的元素逐一与 x 比较，若相等，则返回 x 的下标，即 x 的位置；若找不到，则返回零。

查找时可采用顺序查找法实现，即从第一个(或最后一个)元素开始，依次将表中的元素与 x 比较，若相等，则查找成功，返回该元素在数组中的下标序号；否则查找失败，返回 0，算法流程如图 2.6 所示。

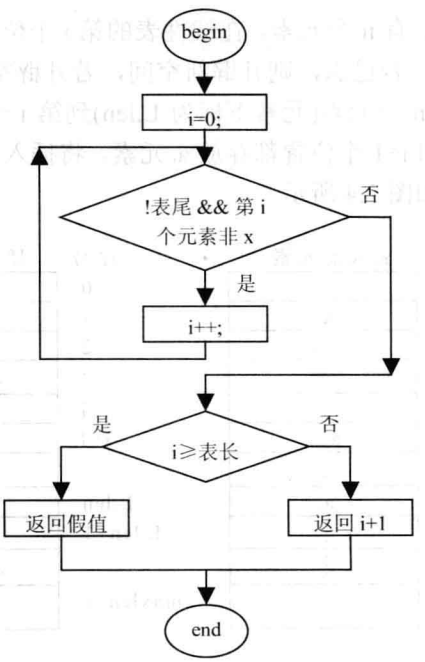

图 2.6　查找元素 x 的位置的流程

4. 程序代码实现

代码 2-1：

```
#include "stdio.h"
#define MAXLEN 20              /*定义常量 MAXLEN 为 20，表示存储空间总量*/
#define INCREMENT 10           /*定义常量 INCREMENT 为 10，表示存储空间增量*/
#define OK 1                           /*定义常量*/
#define OVERFLOW 0                     /*定义常量*/

typedef int ElemType;                      /*定义 ElemType 为 int 类型*/
typedef struct                             /*顺序表存储类型*/
{
   ElemType *elem;                 /*存放线性表的数组*/
   int  Length, Listsize;          /*顺序表的长度，顺序表的最大空间*/
} SeqList;

int InitList(SeqList *L, int n) /*初始化顺序表 L，空间大小为 MAXLEN，表长为 n */
{
   int i;
   L->elem = (ElemType*)malloc(MAXLEN * sizeof(ElemType));
   if (!L->elem)
   {
      printf("线性表初始化失败");
      exit(OVERFLOW);                  /*线性表初始化失败*/
   }
   L->Listsize = MAXLEN;
   printf("请输入%d 个整型数据: ", n);
   for(i=0; i<n; i++)
      scanf("%d", &L->elem[i]);
   L->Length = i;                      /*线性表的长度为 n */
   return OK;                          /*线性表初始化成功*/
}

/*取第 i 个数据元素，放到 x 中，如成功返回 1，否则返回 0 */
int GetElemList(SeqList *L, int i, ElemType *x)
{
   if (i<1 || i>L->Length)
      return  0;
   else
   {
      *x = L->elem[i-1];
      return 1;
   }
}

int LocateList(SeqList *L, ElemType x)      /*在表 L 中定位元素 x */
{
   int i = 0;
   while(i<L->Length && L->elem[i]!=x)
      i++;
```

```
    if (i >= L->Length)
        return 0;
    else
        return i+1;                                  /*返回的是元素位置*/
}

int InsElemList(SeqList *L, int i,
  ElemType x)    /*在表 L 中在第 i 位中插入新元素 x */
{
    int j;
    ElemType *newbase;
    if (L->Length >= MAXLEN)                         /*空间不够另外开辟空间存放*/
    {
        newbase = (ElemType*)realloc(L->elem,
          (L->Listsize + INCREMENT) * sizeof(ElemType));
        if (!newbase) exit(OVERFLOW);                /*空间开辟失败*/
        L->elem = newbase;                           /*新空间基址*/
        L->Listsize += INCREMENT;                    /*增加存储容量*/
    }
    if (i<1 || i>L->Length+1)                        /*检查给定的插入位置的正确性*/
    {
        printf("插入位置出错! ");
        return 0;
    }
    for (j=L->Length-1; j>=i-1; j--)                 /*节点移动*/
        L->elem[j+1] = L->elem[j];
    L->elem[i-1] = x;                                /*新元素插入*/
    L->Length++;                                     /*顺序表长度增 1 */
    return 1;                                        /*插入成功，返回*/
}

int DelElemList(SeqList *L, int i,
  ElemType *x)       /*在表 L 中删除第 i 个位置的元素*/
{
    int  j;
    if (L->Length == 0)
    {
        printf("顺序表为空! ");
        return 0;                                    /*表空，不能删除*/
    }
    if (i<1 || i>L->Length)                          /*检查是否空表及删除位置的合法性*/
    {
        printf("不存在第 i 个元素");
        return 0;
    }
    *x = L->elem[i-1];                               /*用指针变量*x 返回删除的元素值*/
    for(j=i; j<L->Length; j++)                       /*节点移动*/
        L->elem[j-1] = L->elem[j];
    L->Length--;                                     /*顺序表长度减 1 */
    return 1;                                        /*删除成功，返回*/
}
```

```
void OutputList(SeqList *L)                    /*显示输出顺序表L的每个元素函数*/
{
    int i;
    for(i=0; i<L->Length; i++)
       printf("%5d ", L->elem[i]);
}

int Menu()                                     /*操作菜单*/
{
    int c;
    clrscr( );                                 /*清屏*/
    printf("\n       线性表的顺序存储结构操作菜单");
    printf("\n\t ***************************************************");
    printf("\n\t |              1——建立顺序表                    |");
    printf("\n\t |              2——插入元素                      |");
    printf("\n\t |              3——删除元素                      |");
    printf("\n\t |              4——按位置查找元素                |");
    printf("\n\t |              5——按元素值查找其在表中位置      |");
    printf("\n\t |              6——求顺序表的长度                |");
    printf("\n\t |              7——显示线性表中的数据元素        |");
    printf("\n\t |              0——结束!                         |");
    printf("\n\t ***************************************************");
    printf("\n\t 请选择菜单(0-7): ");
    scanf("%d", &c);
    return c;
}

main()
{
    SeqList L;
    ElemType x;
    int t=1, i, n, loc, c;
    while(1)
    {
       c = Menu();
       switch(c)
       {
       case 1:                                 /* 1——建立顺序表 */
          printf("请输入线性表的元素个数: ");
          scanf("%d", &n);
          InitList(&L, n);
          printf("建立的线性表如下: ");
          OutputList(&L);
          break;
       case 2:                                 /* 2——按位置插入元素 */
          printf("请输入要插入元素的位置: ");
          scanf("%d", &i);
          printf("请输入要插入的元素值: ");
          scanf("%d", &x);
          if(InsElemList(&L, i, x))
```