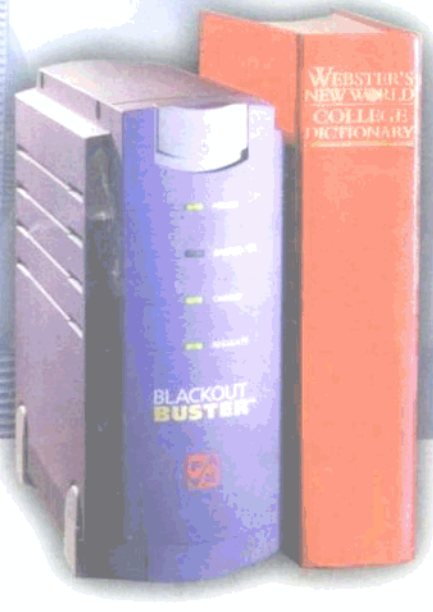
张绍民 李淑华 主编



数据结构教程

（C语言版）

中国电力出版社

前　言

随着电子计算机的普及，计算机的应用领域也不断扩展，数值计算是计算机应用的一个重要方面，而非数值数据的处理则是计算机应用更广泛、更重要的领域。数据结构研究的对象正是非数值数据的逻辑结构和存贮结构，以及能进行的各种基本运算。数据结构为非数值数据的研究提供了基础理论和方法。因此数据结构是计算机科学中一门十分重要的专业基础课程。

从逻辑角度看，数据可归结为三种基本结构，即线性结构、树结构和图结构；从存贮角度看，数据可归结为四种基本结构，即顺序结构、链接结构、索引结构和散列结构。每一种逻辑结构可根据不同需要采用不同的存贮结构，或者数种不同存贮结构的组合。数据的逻辑结构和存贮结构确定后，再结合指定的算法，就容易利用某一种设计语言编写出程序。数据结构这门课程，对计算机软件的开发有重要的意义。

目前绝大多数数据结构方面的教材和参考书均是以 Pascal 语言或类 Pascal 语言做为算法描述语言的。随着 C 语言的应用范围越来越广泛，上述这类教材已难以满足要求，而对用 C 语言讲述数据结构的这类教材的需要已越来越强烈。本书就是针对这一需要而组织编写的。

本书以 C 语言做为算法描述语言来讲述数据结构，内容包括线性表、栈、队列、矩阵的压缩存贮、树与二叉树、图、查找、排序与文件等。全书以结构化程序设计的思想对各种数据结构的算法进行了讨论。所有算法均给出了 C 语言的源程序，相关内容的算法组织成可编译执行的程序。这些程序都以源代码方式给出，所有程序均在 Turbo C 2.0 环境下调试通过。

本书是编者在多年从事 C 语言、数据结构的教学工作和计算机软件开发工作的基础上编写的。其中第 1、3、4、5 章由张绍民编写，第 2 章由肖志东编写，第 6 章由王堃编写，第 7 章由李淑华编写。由张绍民和李淑华两位同志任主编并负责全书的整体结构设计和统稿工作。

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

1998 年 7 月

目　　录

第1章 概 述

世界上第一台电子计算机出现于1946年，是用来计算弹道导弹的飞行轨道的。计算机在发展的初期，其应用范围是数值计算，所处理的数据都是整型、实型、布尔型等简单数据，以此为加工对象的程序设计称为数值型程序设计。后来，随着电子技术的发展，计算机逐渐进入到商业、制造业等其他领域，从而广泛地应用于数据处理和过程控制。与此相应，计算机能处理的数据也不再是简单的数值，而是字符串、图形、图像、语音等复杂的数据。这些复杂的数据不仅量大，而且具有一定的结构。例如一幅图像是一个由简单数值组成的矩阵，一个图形中的几何坐标可以组成表，此外语言编译过程中所使用的栈、符号表和语法树，操作系统用到的队列、磁盘目录树等，都是有结构的数据。数据结构所研究的就是这些有结构的数据，因此，数据结构的知识不论对研制系统软件还是开发应用软件都是非常重要的。它是学习软件知识和提高软件设计水平的重要基础。

本书将针对计算机专业的实际需要，介绍必须具备的数据结构基础知识，为今后从事应用软件开发打下坚实基础。

1.1 什么是数据结构

电子计算机可以完成很多工作。但是无论计算机做什么工作，都必须先由软件人员根据需要编好程序输入到计算机中，计算机才能按程序完成相应的处理工作。软件人员根据需要编制程序时一般须经过下面几个阶段：1）对实际问题进行分析，从而抽象出一个可描述这个问题的数学模型；2）针对这个数学模型找出解决该问题的方法；3）根据解决问题的方法编写程序；4）对程序进行调试、改错、测试，直至得到问题的解。上述四个步骤中，得出描述问题的数学模型是求解实际问题的至关重要的一步。例如，描述电路分析问题的数学模型可能是一组微分方程，而描述某些管理问题的数学模型可能是一组具有约束条件的代数方程。但是有很多问题无法用一组数学公式来描述，下面我们来考察两个实际问题。

【例1-1】 计算机管理工资问题。

要利用计算机管理工资，首先需将每个职工的有关工资信息存入计算机中。工资信息应怎样存放才能使得处理比较方便呢？一个简单的方法是建立一张如图1-1所示的表，表中每一行可以存放一个职工的有关工资的信息，每个职工在表中占有一行。

编 号	姓 名	性 别	基本工资	补 贴	洗理费	水电费	房 租	假 扣
0001	王 强	男	146	16	8	15	5.6	0
0002	李 红	女	73	7	8	0	0	1
⋮								

图1-1 工资表的存贮

工资信息以表的形式存入计算机后，对工资的处理工作就转化为对表的处理。如调入新职工时，可在表中增加一行用于存放新职工的工资信息，职工调出后可将表中相应的行删除，工资的修改也转化为对表中相应行的修改。因此由计算机管理工资问题抽象出的描述模型，就是包含每个职工工资信息的一个表及对该表所进行的插入、删除及修改等操作（也称运算）。

【例 1-2】 屏幕菜单管理问题。

菜单技术是目前人机界面中最常用的技术，所谓菜单技术就形式来说，就是在计算机显示屏幕上列出可选功能供用户选择，并根据用户的选择执行相应的功能。要实现菜单的自动管理应怎样存放菜单呢？由于系统中只有一个主菜单，主菜单中可能有多个选择项，每个选择项可以是一个可执行的功能，也可能是又一个菜单，因此，菜单的这种组织结构可以用一棵树来描述。主菜单即是树根，功能性程序是树叶，而其他菜单项是树的分枝结点。一个具体的菜单可以描述成如图 1-2 所示的树。

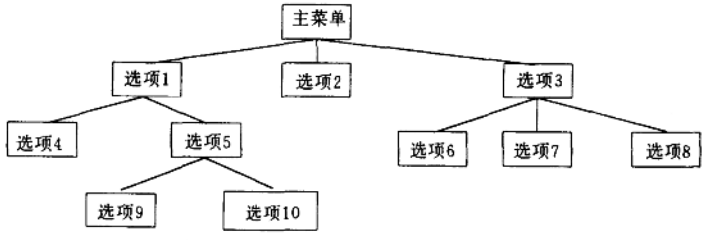


图 1-2 菜单树

菜单按树的形式存贮之后，所有的菜单操作都可以转化为对菜单树的操作。如显示菜单就是搜索出属于这个菜单的全部树结点并显示，菜单选项增加就相当于增加树的结点，等等。因此，菜单管理问题就可以由树及树上的操作来描述。

从上面两个例子我们可以看出，有很大一类问题其描述模型不再是数学公式，而是具有表、树等结构的数据集合及其运算（即操作）。这类问题也不是求解数学方程等数值计算所能解决的，而必须根据其非数值描述模型，对非数值数据进行运算才能解决。

数据结构这门学科的研究对象就是非数值程序设计中计算机操作的对象以及对象之间的关系和运算。

1.2 基本概念及术语

每一门学科都有其特定的概念和术语。这一节，我们介绍数据结构中常用的基本概念和术语。

数据是描述客观事物的数、字符和所有能输入到计算机中并能被计算机程序加工处理的符号的集合。如求解数学方程组的程序所加工的数据是整数和实数，编译程序处理的对象是字符串（即源程序），而图像处理程序处理的对象是一幅幅图像，即整数或实数组成的矩阵。

数据元素是数据的基本单位，是数据集合中的个体。数据元素不一定是简单的数或字符，有时一个数据元素是由若干个数据项组成的，这时称数据元素为记录。数据项是数据的最小单位，是不能再分割的有意义的最小单位。由记录组成的表称为文件。例 1-1 中，每个人的工资数据就是一个数据元素，也称为工资记录，它由九个数据项组成，将每个数据项再分割就

没有实际意义了。所有工资记录组成了工资文件。

数据对象是具有相同特性的数据元素的集合，是数据的一个子集。例如自然数的数据对象是集合{1，2，3…}。

数据结构是指数据元素的集合以及定义在该集合上的关系。例如工资表中数据元素集合是工资记录集，而数据元素集合上的关系就是它们在工资表中的先后顺序关系。例2中的数据元素集合是菜单选项集合，而数据元素之间的关系就是菜单项之间的父子关系，即上层菜单与下层菜单之间的关系。因此，数据结构指的是数据元素之间的结构，而不是数据元素本身的结构。另外，我们这里所说的数据元素之间的关系是指数据元素之间的逻辑关系，在这种关系下定义的数据结构称为数据的逻辑结构。与逻辑结构相对应的是数据的物理结构。数据的物理结构又称为数据的存贮结构，是指数据的逻辑结构在计算机中的映象（又称表示），即数据结构在计算机中的存贮方法。由于数据结构包括数据元素集合及数据元素之间的关系，所以数据的存贮结构也应该包含这两部分内容，即包括数据元素的映象及元素间关系的映象两部分。存贮结构中，数据元素被映象成内存中的一串二进制位，称这串二进制位为结点。一个结点可能占用一个机器字，也可能占用多个机器字。若结点占用一个机器字，则这个机器字的地址就称为结点的地址，若结点占用多个机器字，则第一个机器字的地址做为结点的地址。当数据元素由多个数据项组成时，则对应数据项的二进制位串称为数据域。数据元素之间的关系在计算机中有两种表示方法，从而可得到两种存贮结构，一种称为顺序存贮结构，另一种称为非顺序存贮结构，也称为链式存贮结构。

任何一个程序都涉及到数据的存贮结构。因为对于逻辑结构上的同一种运算，其具体的实现方法依存贮结构的不同而变化。所以，当描述问题的模型确定之后，首要问题就是确定其存贮结构。

数据结构上的运算是指为数据处理的需要而在数据结构上进行的操作。数据的运算是定义在逻辑结构上的，但运算则是在具体的存贮结构上完成的。各种不同的数据结构有各自的运算集合，下面是数据结构中经常用到的运算：

（1）检索　又称为查找，指在数据结构中寻找满足一定条件的数据元素。如在工资表中查找姓名为“王强”的人的工资情况。

（2）插入　在数据结构中增加新的数据元素。如单位中调入新职工，则应在工资表中增加新职工的工资信息。

（3）删除　从数据结构中去掉指定的数据元素，如某职工调出本单位，则应删除其工资记录。

（4）更新　修改指定数据元素的一个或多个数据项的值。如某职工调资了，则应修改其工资记录的基本工资数据项。

（5）排序　在线性数据结构中，在保持数据元素的内容及个数不变的条件下，按指定顺序将数据元素重新排列。如按基本工资从大到小重排工资表。

数据的运算是数据结构的一个重要方面。对任何一种数据结构的讨论都离不开对该结构上定义的运算及其实现方法的讨论。

数据类型是程序设计语言中所允许的变量种类，是变量所能取的值和所能做的运算的集合。一般的程序设计语言中都有一组它所允许的基本数据类型。如FORTRAN语言中提供了整型、实型、双精度型和布尔型。它不仅规定了每一种变量的取值范围，而且规定了对每种

类型量所能执行的运算，如整型可作加、减、乘、除四种运算。C 语言中规定了整型、字符型、浮点型、指针等基本数据类型，而且可以由基本数据类型构造出新的类型。当然，在这些新的构造类型上所能做的运算要由程序员自己定义，在这些构造类型上的运算在 C 语言中由函数来实现。总之，我们可以把数据类型看成是在程序设计语言中实现的数据结构。因此，数据类型实际上是数据结构（包括逻辑结构和存贮结构）及其运算的总称。

算法是解决某一特定类型问题的有限指令序列。算法的一个特点是它必须在执行有限步之后终止执行。算法和程序是有区别的：程序一般是指用计算机语言书写的在具体计算机上执行的过程或函数，但算法不一定是一个由计算机语言书写的程序，也不必一定在计算机上执行；算法必定在执行有限步后终止，而一个程序可能永远不终止执行。但是，多数算法还是为了能在计算机上执行而设计的。本篇的所有程序都是根据算法编写的，所以，我们有时交替使用算法和程序这两个名词，而不加严格区分。

1.3 数据的存贮结构

从上一节我们已经知道，数据的存贮结构有两类，一类是顺序存贮结构，一类是链式存贮结构。数据结构的存贮映象可采取其中的一种方式，也可采用二者相结合的方式。本节我们具体介绍这两种存贮结构。

1.3.1 顺序存贮结构

顺序存贮结构是指逻辑上相邻的数据元素，其结点的物理位置也相邻，数据元素之间的关系由结点的邻接关系来体现。由于计算机的内存单元是一维结构，所以，这种存贮结构主要用于实现线性的数据结构。

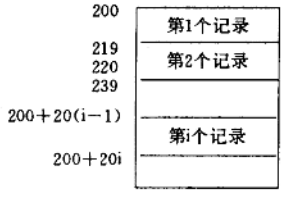


图 1-3 工资表的顺序存贮结构

例 1-1 中给出了工资的逻辑结构，每个职工记录的后面紧跟另一个职工的记录。利用顺序存贮结构实现该表时，可以这样来进行：为存贮这个表而分配一块连续的内存单元，设地址从 200 开始，表的第 1 个数据元素就可从 200 单元开始存放，若一个工资记录需占用 20 个单元，则第 1 个元素就占用了 200～219 这连续的 20 个单元，从 220 开始存放第 2 个元素，从 240 开始存放第 3 个元素，依次类推，直至将表中的元素存放完，如图 1-3 所示。

这个存放原则，保证了逻辑上相邻的两个记录存放在相邻的内存单元中，因此它是顺序存贮结构。

顺序存贮结构有下列特点：

(1) 结点中只存放数据元素本身的信息，无附加内容。

(2) 只要知道第 1 个结点的地址，就可以通过计算直接确定结构中第 i 个结点的地址，从而直接存取第 i 个数据元素。计算公式为

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

其中 $Loc(a_i)$是第 i 个结点的地址。$Loc(a_1)$是第 1 个结点的地址，L 表示一个结点所占的内存单元数。

(3) 由于可根据公式直接确定第 i 个结点的地址而直接存取第 i 个元素，所以数据元素的

存取操作速度较快。

（4）插入、删除数据元素时，由于需要保持数据元素之间的逻辑关系，必须移动大量元素，因此实现起来较慢。关于这一点，将在线性表一节中详细说明。

（5）顺序存贮是一种静态结构，连续的存贮空间一旦分配完毕，其大小就难以改变。因此，当表中元素个数难以估计时，分配的空间大小也就难以确定。预分配的空间太大会造成浪费，空间太小，又可能发生存放不下的溢出现象。

1.3.2 链式存贮结构

在链式存贮结构中，逻辑上相邻的数据元素，其结点的物理位置不一定相邻，因此结点之间是否邻接并不能反映数据元素的逻辑顺序。在这种情况下，存贮结构要反映数据元素在逻辑结构中的关系，只有在结点中增加信息来指明与其他结点之间的这种关系，这个增加的信息就是指针。前一个结点的指针指向后一个结点，多个结点的指针一起形成一个链，因此称这种存贮结构为链式存贮结构。链式存贮结构既可用于实现线性数据结构，也可用于实现非线性数据结构。对于非线性数据结构来说，由于每个数据元素在逻辑上可能与多个数据元素相邻，而计算机内存的一维地址结构限制了每一个结点只能与前、后各一个结点相邻。因此，结点的相邻反映不出一个元素与多个元素的相邻关系，所以非线性逻辑结构只能用链式存贮结构来实现。树、图等就是这种非线性数据结构。在链式存贮结构的每个结点中，数据域可分为两类：一类用于存放数据元素本身的信息，称做信息域；另一类用于存放指针，称做指针域。例 1-1 的工资表用链式存贮结构实现时，如图 1-4 所示。由于我们所关心的是结点所表示的元素之间的邻接关系而不是结点的具体内存地址和指针的值，所以，图 1-4 将指针画成箭头线且箭头指向指针所在结点的下一个结点，从而表示这两个结点所表示的数据元素在逻辑结构中是相邻的且发出箭头的元素在前。

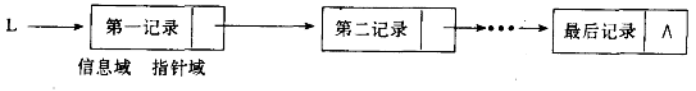


图 1-4　工资表的链式存贮结构

链式存贮结构的特点与顺序存贮结构的特点相反。

（1）结点中除存放数据元素本身的信息外，还需存放附加的指针。

（2）不存在直接确定第 i 个结点地址的公式。要存取第 i 个结点的信息，必须从第 1 个结点开始，沿指针顺序取出前 i－1 个结点，然后根据第 i－1 个结点的指针域确定第 i 个结点的地址，这时才能取出第 i 个结点的信息，所以存取速度较慢。

（3）链式存贮结构的一个主要优点是插入、删除元素时不必移动其他元素，速度较快。因此当数据元素个数变动较大、插入删除操作频繁时，线性数据结构可用链式存贮结构实现。

（4）链式存贮是一种动态存贮结构，当元素数目增加时可随时申请所需的空间，删除时无用的空间可归还给系统另作它用，故空间利用率较高，也不存在预分配空间的问题。

一般来说，线性数据结构既可用顺序存贮结构实现，也可用链式存贮结构实现。究竟用哪种存贮结构，应根据具体情况来选择。选择时的主要依据有两个：一个是数据结构上要执行的主要操作，另一个是对数据元素数目的估计。若在线性数据结构上执行的主要操作是插入、删除，则最好用链式存贮结构，否则应该用顺序存贮结构；若预先可以估计出元素的个数，则可以采用顺序存贮结构，否则宜用链式存贮结构。

1.4 关于算法的描述及算法分析的说明

研究数据结构的目的在于更好地进行程序设计，所以，本章在讨论各种数据结构上的运算时都将给出算法。鉴于本书的读者对象是非计算机专业的学生或计算机应用人员，所以，本章采用直接可上机运算的程序来描述算法。这样做虽然比较繁琐，但对于非计算机专业的读者来说是比较合适的。

本书用C语言来描述程序，原因在于C语言的数据类型丰富，语句精练、灵活且执行效率较高，表达能力强，可移植性好。另外，C语言的使用范围越来越广，越来越多的应用软件将会使用C语言。因此，读者通过阅读本章的内容，不但可以掌握数据结构的知识，也可以为从事C语言应用软件开发作好准备。

对于算法分析，本书将不做复杂的理论分析和推导，只给出直观的说明和分析结果以及适用的情况，以便应用时选择。

小　　结

1. 数据结构这门学科是为适应非数值程序设计的需要而产生的，其研究对象是非数值程序设计中计算机操作的对象和它们之间的关系及运算。

2. 数据是描述客观事物的数、字符以及所有能输入到计算机中并能被计算机处理的符号的集合。数据元素是数据的基本单位，是数据集合中的个体。数据元素的组成部分称为数据项，它是使数据有意义的最小单位。数据对象是具有相同特征的数据元素的集合，是数据的子集。

3. 数据结构是一个数据元素的集合以及定义在该集合上的关系的总称。单独的数据元素集合不是数据结构，单独的关系也不构成数据结构。数据结构在不考虑其在计算机内的具体存贮形式时称为逻辑结构，逻辑结构在计算机内的映象称为数据的存贮结构。数据的存贮结构包括数据元素的映象和数据元素之间关系的映象两部分。其中数据元素的映象称为结点。数据类型是程序设计语言中所允许的变量的种类，是变量所能取的值和所能做的运算的集合。数据类型可以看成是程序设计语言中已实现的数据结构。

4. 数据结构上的运算是为数据处理的需要而在数据结构的数据元素集合上所进行的操作。运算是数据结构的一个重要方面。运算是定义在数据的逻辑结构上的，但运算的实现是依赖于数据的存贮结构的。各种数据结构中常用的运算有检索、插入、删除、更新和排序等。

5. 算法是解决某一特定类型问题的有限指令序列，它的一个特点是在执行有限步之后终止，算法与程序既有区别又有联系。

6. 数据的存贮结构分成顺序和链式两类。顺序存贮结构中由结点之间的邻接关系来体现数据元素之间的逻辑关系。顺序存贮结构主要用于实现线性数据结构，其优点是对元素的存取速度较快，但插入或删除元素时需要移动元素，故速度较慢。链式存贮结构通过结点中的指针来实现数据元素之间的逻辑关系。链式存贮结构的主要优点是插入或删除数据元素时速度较快，但存取元素的速度较慢。链式存贮结构既可用于实现线性数据结构，也可用于实现非线性数据结构。非线性数据结构只能用链式存贮结构来实现。

练 习 题

1. 简述下列术语：数据、数据元素、数据对象、数据结构、存贮结构、算法和数据类型。

2. 说明数据结构的概念与程序设计语言中数据类型概念的区别和联系。

3. 讨论顺序存贮结构和链式存贮结构各自的特点、适用范围，并说明在实际应用中应如何选取数据存贮结构。

4. 编写一个程序，从键盘读入三个数，然后依从小到大的顺序输出。

5. 编写一个程序计算一元多项式 $P_n(x)=\sum_{i=0}^{n}a_iX_i$ 的值 $P_n(X_0)$，设 n、X_0 和 $a_i(0\leqslant i\leqslant n)$均为已知量，从键盘读入。你的程序中 $a_i(0\leqslant i\leqslant n)$的值是采用什么结构存贮的？

6. 写出一个从一组已知数中找出最大值和最小值的程序。

第2章　线性数据结构

线性数据结构是常用的数据结构，包括线性表、栈、队列等。线性结构的主要特点是：结构中只有一个开始点，它没有前趋；只有一个结束点，它没有后继；除开始点和结束点之外的数据元素都有且仅有一个直接前趋和一个直接后继，即结构中的每个数据元素最多与另外两个数据元素之间有联系。这一节我们介绍线性数据结构及其存贮结构和运算的实现。

2.1　线性表

2.1.1　线性表的定义

现实生活中，线性表的例子很多。如10个阿拉伯数字组成一个线性表为

(0，1，2，3，4，5，6，7，8，9)

其中数据元素即是数字。又如26个英文字母也可以看成是一个线性表，即

(a，b，c，…，x，y，z)

其每个数据元素都是英文字母。另一个复杂的线性表的例子是第1章例1-1中的工资表，该表中的每个数据元素都是一个工资记录。综上所述，线性表可描述如下：

线性表是n (n≥0) 个数据元素的有限序列，表中每个数据元素都属于同一数据对象。线性表通常表示为

$$(a_1, a_2, \cdots, a_n)$$

线性表中元素个数n定义为线性表的长度。当n=0时称线性表为空表。线性表中数据元素之间的关系定义为元素之间的先后邻接关系，所以线性表是有序结构。如a_1领先于a_2，a_2领先于a_3，…，a_{i-1}领先于a_i，a_i领先于a_{i+1}。我们称a_{i-1}是a_i的直接前趋，a_{i+1}是a_i的直接后继。又称a_1是表头，称a_n是表尾。因此，线性表中除表头外，每个数据元素都有一个直接前趋；除表尾外，每个数据元素都有一个直接后继。

2.1.2　线性表的运算

线性表是一种灵活的数据结构，对它即可进行简单的基本运算，如访问某个数据元素、插入元素、删除元素，也可以进行复杂的运算，如线性表的合并、拆分和拷贝等。一般来说，复杂的运算可以分解成简单的基本运算。因此，我们只介绍常用的基本运算如下。

(1) 初始化init(L)：将线性表L置成一个空表。

(2) 求表长length(L)：求线性表L的长度。该运算返回一个整数值，即表中元素的个数。当表为空时，返回0。

(3) 存取第i个元素get(L,i)：当1≤i≤n时，取出线性表的第i个元素a_i，否则无意义。

(4) 按特定元素查找表locate(L,x)：若表L中存在值为x的数据元素a_i，则返回a_i的位置i；若有多个数据元素的值为x，则返回最小的位置；若不存在值为x的数据元素，则返回0。

(5) 插入元素insert(L,i,x)：在第i-1和i个元素之间插入一个值为x的数据元素，使原

来长度为 n 的表：

$$(a_1, a_2, \cdots, a_{i-1}, a_i, \cdots, a_n)$$

变成一个长为 n+1 的表：

$$(a_1, a_2, \cdots, a_{i-1}, x, a_i, \cdots, a_n)$$

插入运算只有当 $1 \leqslant i \leqslant n+1$ 时才有意义。

(6) 删除元素 delete(L,i)：在表 L 中删去第 i 个元素，使原来长为 n 的表：

$$(a_1, a_2, \cdots, a_{i-1}, a_i, a_{i+1}, \cdots, a_n)$$

变成长为 n−1 的表：

$$(a_1, a_2, \cdots, a_{i-1}, a_{i+1}, \cdots, a_n)$$

删除运算只有当 $1 \leqslant i \leqslant n$ 时才有意义。

2.1.3 线性表的顺序存贮结构

线性表的顺序存贮结构是以结点的物理地址的相邻来表示数据元素之间的关系的。由于线性表中的元素只有直接前趋和直接后继关系，故可以将线性表的元素存放在一组连续的存贮单元中。若该表中每个数据元素的映象需要 L 个单元，连续存贮单元的起始地址为 add，则可将 a_1 存放在 add～add+L−1 中，a_2 存放在 add+L～add+2L−1 中，…。这样存放的结果就保证了逻辑上相邻的 a_1 和 a_2，在物理位置上也是相邻的，a_i 和 a_{i-1} 在逻辑上相邻，按上述存放原则，它们的物理位置也必是相邻的。这就是线性表的顺序存贮结构。由于规定用结点第 1 个单元的地址表示结点的地址[Loc(a_1)表示元素 a_1 的地址]，故有

$$\mathrm{Loc}(a_1)=\mathrm{add}$$
$$\mathrm{Loc}(a_2)=\mathrm{add}+L=\mathrm{Loc}(a_1)+L$$
$$\mathrm{Loc}(a_3)=\mathrm{add}+2L=\mathrm{Loc}(a_1)+2L$$
$$\cdots\cdots$$

一般地有

$$\mathrm{Loc}(a_i)=\mathrm{Loc}(a_1)+(i-1)*L$$

所以，只要知道线性表的起始地址，线性表中任一元素都可以在相同时间内存取。线性表顺序存贮结构如图 2-1 所示。

由于程序设计语言中的一维数组在计算机内存中也是用一块连续的存贮空间来存放的，所以，可以借助一维数组来描述线性表的顺序存贮结构。但应注意线性表与一维数组并不完全等价。线性表是一种逻辑结构，既可用顺序存贮方式实现，也可以用链式存贮方式实现，而且线性表的元素是有序的；一维数组不具备上述特性，它只是线性表实现顺序存贮的一种手段。

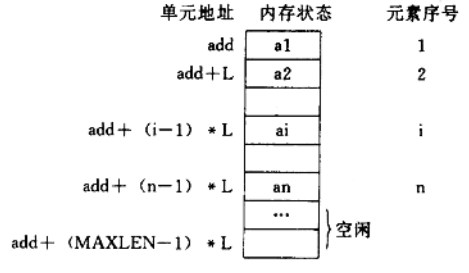


图 2-1 线性表的顺序存贮结构

利用 C 语言，线性表的顺序存贮结构可定义如下：

```
#define MAXLENGTH 100
typedef int ELEMENTYPE;
```

```
typedef struct
{
ELEMENTYPE elements[MAXLENGTH+1];
int last;
}SEQUENLIST;
```

在这个结构定义中，线性表的顺序存贮结构是一个SEQUENLIST型的结构。结构中有两个域，第一个域elements是一个数组，它定义了线性表中数据元素所占用的连续存贮空间。数组的每个元素都是一个ELEMENTYPE型的量，用于存放线性表的一个数据元素，ELEMENTYPE就是数据元素在内存的映象方法的描述。第二个域last用于指示线性表最后一个元素在数组中的位置，这是因为需要为线性表预分配一个较大的空间，以便能够适应存放线性表达到最长的极限情况，定义中的MAXLENGTH就是用于说明预分配空间大小的。然而在某一时刻，线性表中元素可能没有达到极值情况，因此线性表的元素就只占数组空间的一部分，数组的其余部分不是线性表的元素，因此，必须有一个量来记录这一情况，这个量就是last，它表示数组中从1到last的空间被线性表占用，其余空间不是线性表的元素，但在增加元素时可用。

为了在说明算法时简单一些，上述结构定义中将ELEMENTYPE定义成一个整型量，即将线性表的元素定义为整型，在实际应用中，数据类型还可根据具体情况修改，这对我们下面要介绍的各种算法及程序仍然适用。例如为了实现第1章中例1-1的工资表，只需将上述定义中的：

```
typedef int ELEMENTYPE;
```

修改成

```
typedef struct
{
  char number[4],name[20];
  int sex,basic _ wage,allowance,bath _ haircut _ fees;
  int water _ elec _ fees,rent,rest;
}ELEMENTYPE;
```

在顺序存贮结构中，线性表的某些基本运算很容易实现。如线性表的初始化只需将last的值赋成0即可，如下面的C语言函数init _ seqlist所示。

```
void init _ seqlist(L)
  SEQUENLIST *L;
{
L->last=0;
}
```

又如求表长只需返回last的值即可；取线性表的第i个元素只需返回数组的第i个元素即可。

```
int length _ seqlist(L)
  SEQUENLIST L;
{
```

```
return L.last;
}
ELEMENTYPE get _ seqlist(L,i)
SEQUENLIST L;
int i;
{
return L.elements[i];
}
```

函数 length _ seqlist 和 get _ seqlist 的功能分别是求表长和取第 i 个元素。根据上面三个函数可以知道，虽然 C 语言的数组下标从 0 开始，但我们这里没有用到数组的第 0 个元素，线性表的元素是从数组的第一个元素开始存放的，即数组中下标为 i 的元素存放线性表的第 i 个元素。

在线性表的顺序存贮结构中，插入与删除运算如何实现呢？要在线性表的第 i－1 个和第 i 个元素之间插入一个元素 x，就使一个长度为 n 的线性表：

$$(a_1, a_2, \cdots, a_{i-1}, a_i, \cdots, a_n)$$

变成了一个长度为 n+1 的线性表：

$$(a_1, a_2, \cdots, a_{i-1}, x, a_i, \cdots, a_n)$$

插入之后，不但表长发生了变化，a_{i-1}和 a_i 之间的逻辑关系也发生了变化。a_{i-1}原来的直接后继为 a_i，插入后变成了 x，a_i 成为 x 的直接后继。a_i 的直接前趋也由 a_{i-1}变成 x。原来表中的第 j 个元素（i≤j≤n）插入后成为表的第 j+1 个元素，x 成为第 i 个元素，线性表逻辑关系上的这些变化应该反映到存贮结构中去。由于线性表的顺序存贮结构是通过结点的物理地址反映逻辑关系的，所以要在存贮结构中反映上述变化，只有移动元素，即将 a_i～a_n 向后移动一个位置，将 x 放到第 i 个位置上，这样处理之后，所有因插入而发生的逻辑关系上的变化，就都反映在存贮结构中了。

图 2-2 展示了一个线性表在插入一个元素前后数据元素在数组中位置变化的情况。为了在第 2、3 个元素之间插入数据元素 14，将第 3～5 个元素在数组中依次向后移动一个位置。表长由原来的 5 增加至 6，元素 14 放到了第 3 个位置上，它成了元素 10 的直接后继和元素 17 的直接前趋，原来的第 4 个元素 25 现在成为第 5 个元素。

根据上述分析，在线性表顺序存贮结构中，插入运算的算法可描述为

```
void insert _ seqlist(L,i,x)
SEQUENLIST *L;
ELEMENTYPE x;
int i;
{
int j;
if (L->last>=MAXLENGTH)
  {
  printf("Overflow\n");
  return;
```

```
    }
  else
  if ((i<1)||(i>L->last+1))
    {
    printf("has no the place %d in list for insert\n",i);
    return;
    }
  else
    {
    for (j=L->last;j>=i;j--)
      L->elements[j+1]=L->elements[j];
      L->elements[i]=x;
      L->last++;
      }
}
```

算法 insert _ seqlist 在执行实际的插入操作之前，首先检验是否还有空间可供插入。若有可供插入的空间且插入的位置参数也正确，才执行实际的插入操作；若空间已满或不在规定的合法位置上插入，则给出错误信息后直接返回。

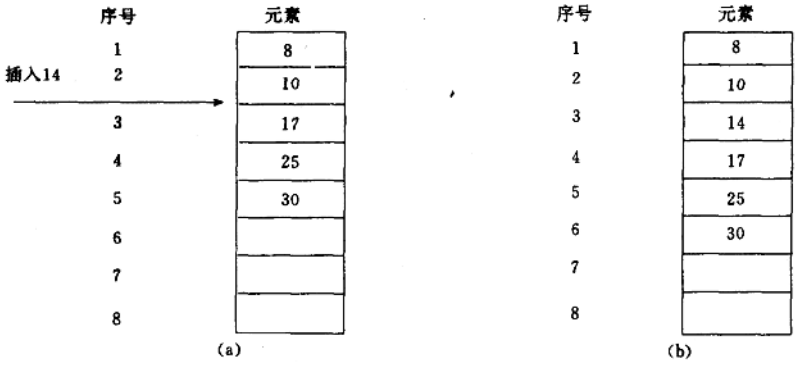


图 2-2　线性表插入元素示意图

(a) 插入前；(b) 插入后

线性表的删除运算使原来长为 n 的线性表：

$$(a_1, a_2, \cdots, a_{i-1}, a_i, a_{i+1}, \cdots, a_n)$$

变成长为 n−1 的线性表：

$$(a_1, a_2, \cdots, a_{i-1}, a_{i+1}, \cdots, a_n)$$

数据元素 a_{i-1}，a_i，a_{i+1}之间的逻辑关系发生了变化，原来的第 j 个元素 $(i+1 \leqslant j \leqslant n)$ 在删除 a_i 后成为表中的第 j−1 个元素。为了在存贮结构中反映这一变化，同样需要移动元素。一般情况下，删除第 i 个元素，需将第 i+1 至第 n 个元素依次向前移动一个位置。图 2-3 表示在线性表中删除元素 17 前后数据元素在数组中的位置变化情况。