全国高等专科教育计算机类规划教材

数据结构
——C语言

李新燕　主编

机械工业出版社
CHINA MACHINE PRESS

全国高等专科教育计算机类规划教材

数据结构——C语言

主　编　李新燕
副主编　靳　敏

机械工业出版社

本书共分 10 章，第 1 章介绍了数据结构的基本概念，第 2～4 章分别介绍了线性表、栈和队列、串等常用的数据结构，第 5 章讲述了递归算法，第 6 章介绍了树形结构，第 7 章介绍了图形结构，第 8 章讲述了查找算法，第 9 章讲述了排序算法，第 10 章介绍文件的基本概念和结构。

本书内容安排合理、概念清晰、例题丰富、通俗易懂，可作为大、中专学生的教材，也可以作为计算机爱好者的参考书。

本书配有电子教案供教师使用，可发电子邮件至 wangyx@mail.machineinfo.gov.cn 邮箱索取。

图书在版编目（CIP）数据

数据结构——C 语言/李新燕主编. —北京：机械工业出版社，2006.8
全国高等专科教育计算机类规划教材
ISBN 7-111-19619-8

Ⅰ.数… Ⅱ.李… Ⅲ.①数据结构-高等学校-教材②C 语言-程序设计-高等学校-教材 Ⅳ.①TP311.12②TP312

中国版本图书馆 CIP 数据核字（2006）第 082056 号

机械工业出版社(北京市百万庄大街 22 号 邮政编码 100037)
策划编辑：王玉鑫 孔熹峻
责任编辑：李学锋 版式设计：张世琴 责任校对：魏俊云
封面设计：姚 毅 责任印制：洪汉军
北京京丰印刷厂印刷
2006 年 8 月第 1 版·第 1 次印刷
184mm×260mm·11.5 印张·279 千字
0 001—4 000 册
定价：18.00 元

凡购本书，如有缺页、倒页、脱页，由本社发行部调换
本社购书热线电话（010）68326294
编辑热线电话（010）68354423
封面无防伪标均为盗版

前　言

近年来，我国高等院校不仅在计算机专业开设数据结构课程，而且在越来越多的非计算机专业中也开设了数据结构课程。随着计算机技术的发展，计算机在各个领域的应用过程中，都会涉及到数据的组织与程序的编排等问题，都会用到各种各样的数据结构。选择最合适的数据结构和存储表示方法，以及编制相应的实现算法的方法是计算机工作者不可缺少的知识。

主要内容：本书全面、系统地介绍了常用的数据结构及其查找、排序的各种方法。对每一种数据结构除了阐述各种数据结构所涉及的逻辑关系外，还讨论它们在计算机中的存储表示方法以及在这些数据结构上的运算（操作）和实际的执行算法，并对算法的效率进行简要的分析和讨论。

本书共分10章，第1章介绍了数据结构的基本概念，第2~4章分别介绍了线性表、栈和队列、串等常用的数据结构，第5章讲述了递归算法，第6章介绍了树形结构，第7章介绍了图形结构，第8章讲述了查找算法，第9章讲述了排序算法，第10章介绍文件的基本概念和结构。

本书的编者均具有多年的计算机教学经历，根据其自身多年的教学经验和学生特点以及计算机技术的发展趋势，对数据结构的基本概念、基本理论的阐述由浅入深、通俗易懂；淡化算法的设计分析和复杂的时空分析，书中所有算法的描述均采用可在计算机上调试运行的C语言函数，降低了算法设计的难度，同时方便学生在计算机上验证相关算法。

本教材概念清楚、内容丰富、详略得当、实用性强，适合各高等院校、高职高专学校数据结构教学用书；也适合广大计算机应用技术人员和计算机爱好者用来学习参考。

本教材还配有相应的习题与实验指导书，有利于读者深入理解所学的内容。

本书由李新燕任主编、负责组稿和统稿，靳敏任副主编。其中第6、7、10章由靳敏编写，其余由李新燕编写。在本书的编写过程中，柳青副教授审阅了本书的编写大纲，提出了不少宝贵意见，在此表示衷心地感谢。

由于时间仓促，加上编者的水平有限，书中难免有不当之处，欢迎大家批评指正。

编　者

目　录

第1章 绪 论

随着计算机产业的飞速发展，计算机的应用范围迅速扩展，如今，它已深入到人类社会的各个领域。计算机的应用已不再局限于科学计算，而是更多地用于实时控制、信息管理及数据处理等方面。与此对应，计算机加工处理的对象由纯粹的数值数据发展到字符、表格和图像等数据，这就给程序设计带来一些新的问题。要编写出一个“好”的程序，必须分析程序待处理的数据的特性及数据之间存在的关系，这就是“数据结构”这门学科形成和发展的背景。

数据结构是学习计算机知识的基础课，是十分重要的核心课程。所有的计算机系统软件和应用软件都要用到各种类型的数据结构。因此，要想更好地运用计算机来解决实际问题，仅掌握几种计算机程序设计语言是难以应付众多复杂的课题的。要想有效地使用计算机、充分发挥计算机的性能，还必须学习和掌握好数据结构的有关知识。扎实的打好“数据结构”这门课程的基础，对于学习计算机专业的其他课程，如操作系统、编译原理、数据库管理系统、软件工程、人工智能等都是十分有益的。

1.1 数据结构的基本概念

例 1-1 学生的学籍管理。

表 1-1 是学生的学籍表。每个学生的信息占一行，所有学生的信息按学号顺序依次排列；对它的操作通常是插入某个学生的信息，删除某个学生的信息，更新某个学生的信息，按条件检索某个学生的信息等。表中的一行称为一个数据元素，因此表中的数据元素之间存在着最简单的一对一的线性关系。

表 1-1 学生学籍表

学号	姓名	性别	专 业	籍贯
980001	石宝国	男	计算机科学与技术	广东
980002	何文颖	女	信息与计算科学	天津
990301	刘 丽	女	数学与应用数学	上海
990302	张会友	男	信息与计算科学	广东
…	…	…	…	…
…	…	…	…	…

一个数据元素

一个数据项

数据元素（Data Element）是数据的基本单位，它可以由不可分割的数据项组成。数据元素又可称为元素、结点、顶点、记录等。

数据项（Data Item）指不可分割的、有独立意义的最小单位的数据，数据项有时也称为字段（Field）或域。

数据（Data）指所有能输入到计算机中并被计算机程序处理的符号的总称。计算机输入

和处理的数据除数值外，还有字符串、表格、图像甚至声音等，它们都是数字编码范畴。整个表 1-1 是学生学籍信息的数据，单个学生的信息是其中的一个数据元素。

数据、数据元素、数据项实际上反映了数据组织的三个层次：数据可由若干个数据元素构成，而数据元素又可以由一个或若干个数据项组成。

例 1-2 某大学的教师人事关系。

虽然可以用二维表格将全校教师的名单列出，但采用图 1-1 所示的结构可以清晰地描述教师所在院、系及教研室。若知道某个教师所属的系，便可以从树根沿着某系某教研室很快找到该教师。图 1-1 所示的结构像一棵根在上倒挂的树，故称为树形结构，结点与结点之间存在着严格的一对多关系，图 1-1 查找的过程就是从树根沿分支到某个叶子的过程。

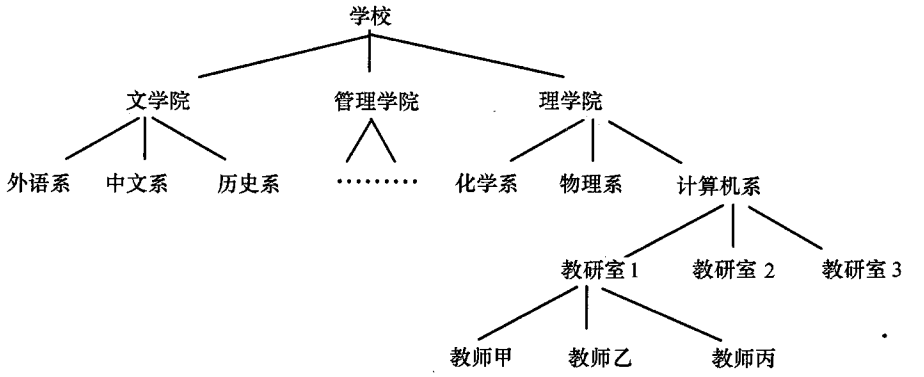

图 1-1 某校教师人事关系

例 1-3 公路交通网。

在 n 个城市之间建立交通网络，如图 1-2 所示。图中的小圆圈表示一个城市，两个圆圈之间的连线表示对应城市之间的高速公路，连线上的数值可以表示修造该路的造价，这一描述的结构为图状结构，元素间的关系为多对多。

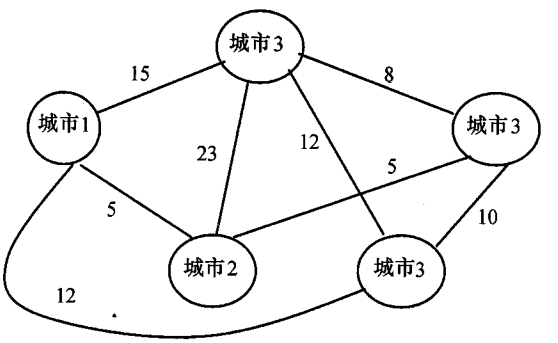

图 1-2 城市交通网络

通过以上三个例子可以看出：数据元素之间存在着逻辑关系，而线性表、树、图是三种基本的逻辑结构，其他各类的数据结构都是由这三种基本结构派生的。数据结构就是解决如何分析数据元素之间的关系、如何确立合适的逻辑结构、如何存储这些数据，并对未完成数据操作所设计的算法做出时间和空间的分析。

数据结构（Data Structure）的形式定义：

数据结构名称 =（D，S），其中 D 为数据元素的有限集，S 是 D 上关系的有限集。

“数据结构”定义中的“关系”指数据间的逻辑关系，也称为逻辑结构。数据的逻辑结构可以看作是从具体问题抽象出来的数学模型，它与数据的存储无关。通常有以下四种基本结构（见图 1-3）：

1）集合。数据元素之间除了“同属于一个集合”的关系外，没有其他的关系。
2）线性结构。数据元素之间存在着一对一的关系。
3）树形结构。数据元素之间存在着一对多的关系。
4）图状结构。数据元素之间存在着多对多的关系。

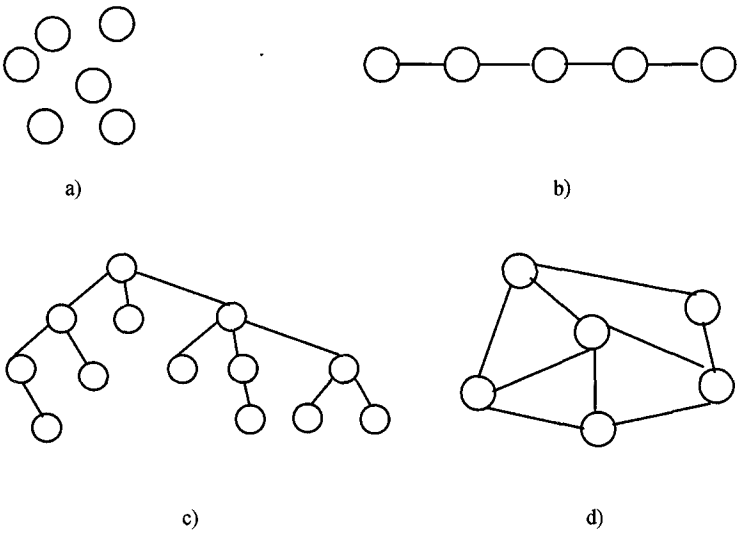

图 1-3　四种基本结构关系图
a）集合　b）线性结构　c）树形结构　d）图状结构

数据结构在计算机中的表示称为物理结构，又称存储结构。它所研究的是数据结构在计算机中的实现方法，包括数据结构中元素的表示及元素间关系的表示。数据元素在计算机中主要有两种不同的存储方式：顺序存储结构和链式存储结构。

1）顺序存储的特点是在内存中开辟一组连续的空间（高级语言中的数组）来存放数据，数据元素之间的逻辑关系通过元素在内存中存放的相对位置来确定，又称向量存储。

2）链式存储的特点是通过指针反映数据元素之间的逻辑关系，又称动态存储。

线性表（1，3，5，7，9）用顺序存储结构的方式依次存放在以3000为起地址的内存向量中，并且两个字长存放一个奇数，如图 1-4a 所示；采用链式存储，如图 1-4b 所

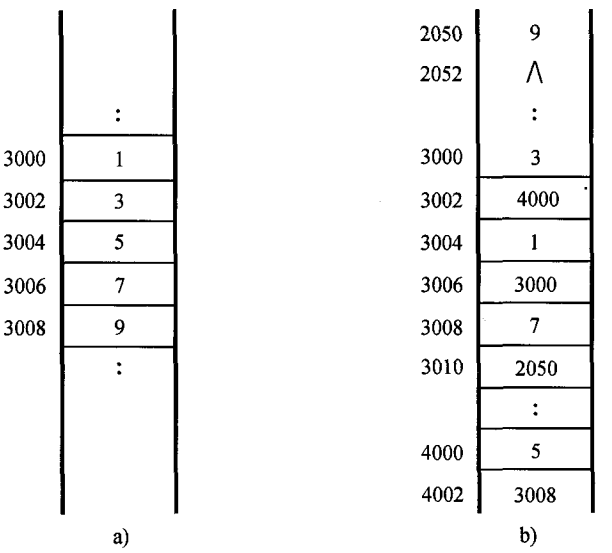

图 1-4　两种存储结构示意图
a）顺序存储　b）链式存储

示。线性表详见第2章。

数据类型是一个值的集合以及定义在这个值集上的一组操作的总称。例如，C语言中的整型，其内涵为一定范围（-32768~32767）的自然数集合，及定义在该集合上的加减乘除及取模、比较大小等操作。数据类型封装了数据存储与操作的具体细节。数据类型有两类：

1）原子类型。值在逻辑上不可分解，如int型、float型等。

2）结构类型。值由若干成分按某种结构组成，如struct stu等。

1.2 算法描述

在进行程序设计时，除需要某种程序设计语言外，还要掌握算法的设计、分析和数据结构的有关知识。瑞士的N·沃斯教授提出了以下著名公式：

程序=算法+数据结构

这个公式说明了算法和数据结构二者之间的密切关系，同时也说明了算法和数据结构对于程序设计的重要性。

1.2.1 算法的定义及特性

1. 定义

算法是对特定问题求解步骤的一种描述，它是指令的有限序列，其中每一条指令表示一个或多个操作。

2. 算法的五个特性

（1）有穷性　一个算法必须总是（对任何合法的输入值）在执行有穷步之后结束，且每一步都可在有穷时间内完成。以下例子不满足有穷性。

```
haha()
{     /* only a joke,do nothing. */
}
main()
{     printf("请稍等 ... 您将知道世界的末日 ...");
      while(1)
          haha();
}
```

（2）确定性　算法中每一条指令必须有确切的含义，读者理解时不会产生二义性。在相同的条件下，算法对于相同的输入只能得出相同的输出。以下例子不满足确定性。

```
float average(int *a,int num)        /*  num为数组a元素个数  */
{     int i;double sum=0;
      for(i=0;i<=num;i++)            /*  a[num]没有定义  */
            sum+=*(++a);
      return sum/num;
}
main()
```

```
{    int score[9] = {1,2,3,4,5,6,7,8,9};
     printf("%f",average(score,9));
}
```

(3) 可行性 一个算法是能行的，即算法中描述的操作都是可以通过已经实现的基本运算执行有限次来实现的。

(4) 输入 一个算法有零个或多个的输入，这些输入取自于某个特定的对象的集合。

(5) 输出 一个算法有一个或多个的输出，这些输出是同输入有着某些特定关系的量。

```
getsum(int num)
{    int i,sum = 0;
     for(i = 1;i <= num;i++)
                sum + = i;
}    /*  无输出的算法没有任何意义  */
```

1.2.2 算法设计的要求

(1) 正确性 算法正确性的四个层次：程序不含语法错误；程序对于几组输入数据能够得出满足规格说明要求的结果；程序对于精心选择的典型、苛刻而带有刁难性的几组输入数据能够得出满足规格说明要求的结果；程序对于一切合法的输入数据都能产生满足规格说明要求的结果。

(2) 可读性 算法应易于阅读和理解，以便于调试、修改和扩充。

(3) 健壮性 正确的输入能得到正确的输出这是算法必须具有的特性之一。但当遇到非法输入时，算法应能做出反应或处理(如提示信息等)，而不会产生不需要的或不正确的结果。

(4) 高效率与低存储量需求 效率指的是算法执行时间。对于解决同一问题的多个算法，执行时间短的算法效率高。存储量需求指算法执行过程中所需要的最大存储空间。对于解决同一问题的多个算法，应尽量选择少占存储空间的算法，两者还与问题的规模有关，表1-2是效率与存储量分析的一个示例。

表1-2 效率与存储量分析

	算法一	算法二

在3个整数中求最大者	max(int a,int b,int c)
	{if (a>b)
	  if(a>c) return a;
	  else return c;
	else
	  if(b>c) return b;
	  else return c;
	}/ *无需额外存储空间，只需2次比较 */
		max(int a[3])
		{int c,int i;
		c=a[0];
		for(i=1;i<3;i++)
		  if (a[i]>c) c=a[i];
		return c;
		}
		/* 需要2个额外的存储空间，2次比较，至少1次赋值 */
		/* 共需5个整型数空间 */

求100个整数中最大者	同上的算法难写，难读	max(int a[100])
		{int c,int i;
		c=a[0];
		for(i=1;i<100;i++)
		  if (a[i]>c) c=a[i];
		return c;
		}
		/* 共需102个整型数空间 */

1.2.3 算法评价

1. 算法效率的度量

一个用高级程序设计语言编写的程序在计算机上运行所需时间与下列因素有关：

1）机器执行指令的速度。

2）编译程序所产生的机器代码质量。

3）书写程序的语言。语言越高级，消耗时间越多。

4）问题的规模。规模越大，消耗时间越多。

度量一个算法的效率应当抛开具体的机器，仅考虑算法本身的效率高低。由此可见，算法效率只与问题的规模有关，即算法的效率是问题规模的函数。评价一个算法的优劣，可以在相同的规模下，考察算法执行时间的长短来进行判断。

从算法中选取一种对于所研究的问题来说是基本操作的原操作，以该基本操作重复执行的次数作为算法的时间量度。一般情况下，算法中的基本操作重复执行的次数是问题规模 n 的某个函数 f(n)，算法的时间量度记作：

$T(n) = O(f(n))$

上式表示随问题规模 n 的增大，算法执行时间的增长率和 f（n）的增长率相同，称作算法的渐近时间复杂度，简称时间复杂度。

原操作执行次数和包含它的语句的频度相同。语句的频度指的是该语句重复执行的次数。表 1-3 给出了 3 个程序段对应的频度及时间复杂度。

表 1-3 语句的频度分析

程序段	频度	时间复杂度
{++x;s=0;}	1	O(1)
for(i=1;i<=n;++i) {++x;s+=x;}	n	O(n)

for(j=1;j<=n;++j)
for(k=1;k<=n;++k)
{++x;s+=x;}
	n*n	O(n*n)

对于基本操作的执行次数不确定的时间复杂度，可以用平均时间复杂度和最坏情况下时间复杂度来分析。平均时间复杂度依基本操作执行次数概率计算平均；最坏情况下时间复杂度是指在最坏情况下基本操作执行次数。

例 1-4 求 4 * 4 矩阵元素和。

```
sum(int num[4][4])
{    int i,j,r=0;
     for(i=0;i<4;i++)
         for(j=0;j<4;j++)
             r+=num[i][j];              /*  原操作  */
return r;
}
```

以上算法段的时间复杂度为：T(4) = O(f(4))，其中 f(n) = n * n。

例 1-5 分析以下算法段的时间复杂度。

```
ispass(int num[4][4])
{    int i,j;
     for(i=0;i<4;i++)
         for(j=0;j<4;j++)
```

```
        if(num[i][j]! =i*4+j+1)          /*  原操作  */
            return 0;                    /*  原操作  */
    return 1;
}
```

最好情况：T(4) = O(1)

最坏情况：T(4) = O(n * n)

2. 算法的存储空间需求

类似于算法的时间复杂度，空间复杂度可以作为算法所需存储空间的量度。

记作：S(n) = O(f(n))

若额外空间相对于输入数据量来说是常数，则称此算法为原地工作。

如果所占空间量依赖于特定的输入，则除特别指明外，均按最坏情况来分析。

小 结

本章介绍了贯穿全书的基本概念和基本思想。介绍了数据、数据元素的概念，具有特定关系的数据元素集合称为数据结构；数据结构的逻辑表示与物理存储，逻辑结构与存储结构；数据的处理方法(算法)与处理结果(数据类型)以及算法特性、时间复杂性、空间复杂性。

复习思考题

1-1 什么是数据结构？

1-2 数据结构涉及哪几个方面？

1-3 两个数据结构的逻辑结构和存储结构都相同，但是它们的运算集合中有一个运算的定义不一样，它们是否可以认作是同一个数据结构？为什么？

1-4 线性结构的特点是什么？非线性结构的特点是什么？

1-5 数据结构的存储方式有哪几种？

1-6 算法有哪些特点？它和程序的主要区别是什么？

1-7 抽象数据类型的是什么？它有什么特点？

1-8 算法的时间复杂度指的是什么？如何表示？

1-9 算法的空间复杂度指的是什么？如何表示？

1-10 分析下列算法的时间复杂性：

(1)
```
sum = 0;
for (i = 1;i <= n;i++)
    sum = sum + i;
```

(2)
```
i = 1;
while(i <= n)
    i = i * 10;
```

(3)
```
for(i = 0;i < n;i++)
    if (a[i] < x) x = a[i];
```

(4)
```
for(i = 0;i < n;i++)
    for(j = 0;j < n;j++)
        printf("%d",i + j);
```

1-11 试写一算法，输入三个整数 X、Y 和 Z，然后按从大至小依次输出。

第2章 线 性 表

线性结构是一种最基本、最简单且应用十分广泛的数据结构，这种数据结构元素之间呈一对一的关系，即线性关系，通常称之为“线性表”。本章将讨论线性表的逻辑结构和存储结构（即物理结构），以及在各存储结构上实现相关操作的算法。

2.1 线性表的基本概念

2.1.1 线性表的定义及逻辑结构

线性表（Linear List）可以简单定义为：一个线性表是 n（$n \geqslant 0$）个数据元素的有限序列。$n=0$ 时，称为空表，$n>0$ 时，线性表的表示形式为（a_1，a_2，…，a_n）。

例如，26 个英文字母（A，B，C，D，…，X，Y，Z）是一个线性表，其中数据元素是英文大写字母。一个星期中的七天可放在一个线性表中：（星期一、星期二、……星期六、星期日）。又如，学生成绩见表 2-1，表中每一个学生的成绩情况为一个记录，它由学号、姓名、C 语言、计算机原理、英语五个数据项组成。

表 2-1 学生成绩表

学号	姓名	C 语言	计算机原理	英语
62004001	张小勇	80	83	77
62004002	李冠红	78	71	89
62004003	林月东	91	86	90
62004004	王莉平	85	80	78
…	…	…	…	…

综合以上例子，线性表具有以下特性：

(1) 同一表中的元素必定具有相同特性　线性表中的数据元素可以是各种各样的，但表中的所有数据元素其数据类型是一致的。表中的一个数据元素可以由若干个数据项组成，也可以只由一个数据项组成，通常把数据元素称为记录，有大量记录的线性表称为文件。

(2) 数据元素在线性表中的位置只取决于它的序号　线性表具有线性结构的特点，表中 a_i 元素的直接前驱元素是 a_{i-1}，a_i 元素的直接后继元素是 a_{i+1}。

线性表　（a_1，　a_2，　a_3，　$\cdots a_{n-1}$，　a_n）

序号　　1　　2　　3　　$\cdots n-1$　　n

(3) 对于一个非空的线性表，存在惟一的“第一个”数据元素；存在惟一的“最后一个”数据元素；除第一个数据元素之外，表中的每一个数据元素都只有一个前驱；除最后一个数据元素之外，表中的每一个数据元素都只有一个后继。

2.1.2 线性表的基本操作

对于线性表，可以进行的操作有以下几种：

1) InitList (L)。初始化操作函数，生成一个空的线性表 L。

2) ListLength (L)。求表长度的函数。函数值为线性表 L 中数据元素的个数。

3) GetElem (L, i)。取表中元素的函数。当 $1 \leqslant i \leqslant$ LENGTH(L) 时，函数值为线性表 L 中第 i 个数据元素，否则返回一特殊值。i 是该数据元素在线性表中的位置序号。

4) Locate (L, x)。定位函数。给定值 x，在线性表 L 中若存在和 x 相等的数据元素，则函数返回和第一个和 x 相等的数据元素的位置序号，否则返回 0。

5) Insert (L, x, i)。插入操作。在给定线性表 L 中第 i ($1 \leqslant i \leqslant$ LENGTH (L) +1) 个数据元素之前插入一个新的数据元素 x，使原来线性表的长度 n 变成 n+1。

6) Delete (L, i)。删除操作。删除在给定线性表 L 中第 i ($1 \leqslant i \leqslant$ LENGTH (L)) 个数据元素，使原来线性表的长度 n 变成 n-1。

7) Empty (L)。判空表函数。若 L 为空表，则返回布尔值“真”，否则返回布尔值“假”。

8) ClearList (L)。表置空操作。不管原来的线性表 L 是空表还是非空表，操作结果将 L 表置空。

以上基本操作中，1)、5)、6)、8) 是加工型操作，其他都是引用型操作。需要说明的是：

1) 某数据结构上的基本运算，不是它的全部运算，而是一些常用的基本的运算，而每一个基本运算在实现时也可能根据不同的存储结构派生出一系列相关的运算来。例如，线性表的查找在链式存储结构中还会有按序号查找；再如插入运算，也可能是将新元素 x 插入到适当位置上等，不可能也没有必要全部定义出它的运算集。掌握了某一数据结构上的基本运算后，其他的运算可以通过基本运算来实现，也可以直接去实现。

2) 在上面各操作中定义的线性表 L 仅仅是一个抽象在逻辑结构层次的线性表，尚未涉及到它的存储结构，因此每个操作在逻辑结构层次上尚不能用具体的某种程序语言写出具体的算法，而算法只有在存储结构确立之后才能实现。

2.2 线性表的顺序存储结构及操作

2.2.1 线性表的顺序存储结构

线性表的顺序存储是计算机中最简单、最常用的一种存储方式，即用一组地址连续的存储单元依次存放线性表的元素。由于同一线性表中元素的类型相同，可设定一个元素占用 b 个存储单元，$LOC(a_1)$ 为线性表的存储起始地址，即表中第一个元素存放的地址。则存在如下关系：

$$LOC(a_{i+1}) = LOC(a_i) + b \qquad (1 \leqslant i \leqslant n)$$

$$LOC(a_i) = LOC(a_1) + (i-1) * b \quad (1 \leqslant i \leqslant n)$$

线性表顺序存储的特点是：表中相邻的元素 a_i 和 a_{i+1}所对应的存储地址 LOC (a_i) 和地址 LOC (a_{i+1}) 也是相邻的。也就是说表中元素的物理关系和逻辑关系是一致的。只要知道线性表的起始地址 LOC (a_1) 和一个元素占用的存储单元 b，表中任意一个元素的存储起始地址可用公式得到：$LOC(a_i) = LOC(a_1) + (i-1) * b$，图 2-1 是顺序表的逻辑表示和对表中元素存储地址计算的分析示意图。

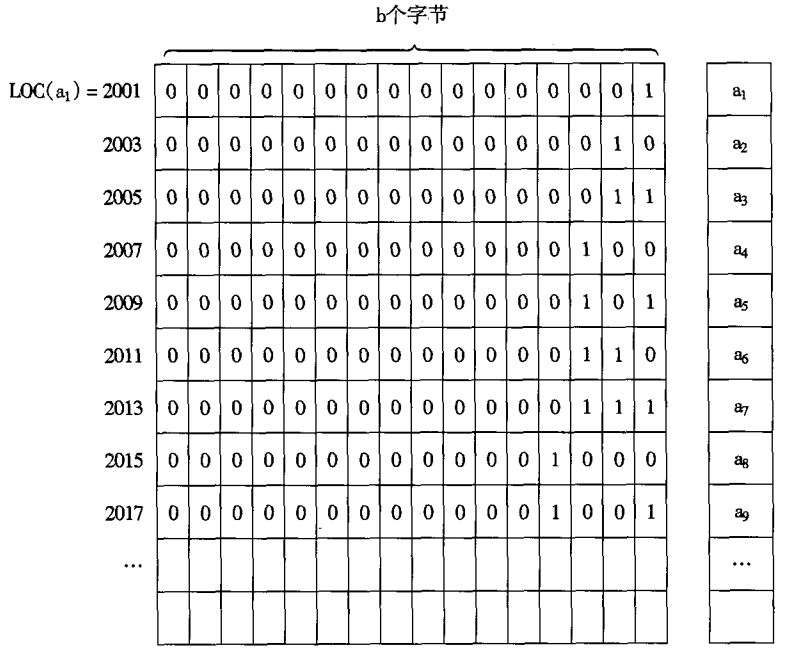

图 2-1 顺序表的逻辑表示及存储地址计算示意图

线性表的这种机内表示称做线性表的顺序存储结构或顺序映象。

顺序存储结构的线性表简称为顺序表。顺序表的特点是以元素在计算机内物理位置相邻来表示线性表中数据元素之间的逻辑关系。

由图 2-1 及计算公式可以看出，计算顺序表中每一个元素的存储起始地址的时间是相同的，读取表中元素所需的时间也是一样的。顺序表中任一元素都可以随机存取，所以线性表的顺序存储结构是一种随机存取的存储结构。在这种结构上很容易实现线性表的某些操作，如随机存取表中第 i 个元素等。

顺序表的数据类型描述如下：

```
#define   ELEMTYPE   int
#define   MAXSIZE    100
typedef   struct
{       ELEMTYPE   data[MAXSIZE];
        int   len;
}SEQLIST;
```

2.2.2 顺序表的基本操作

1. 初始化操作

对于一个线性表可以用以下函数进行初始化：

```
void   initseqlist (SEQLIST * a)
{      a.len = 0 ;
}
```