

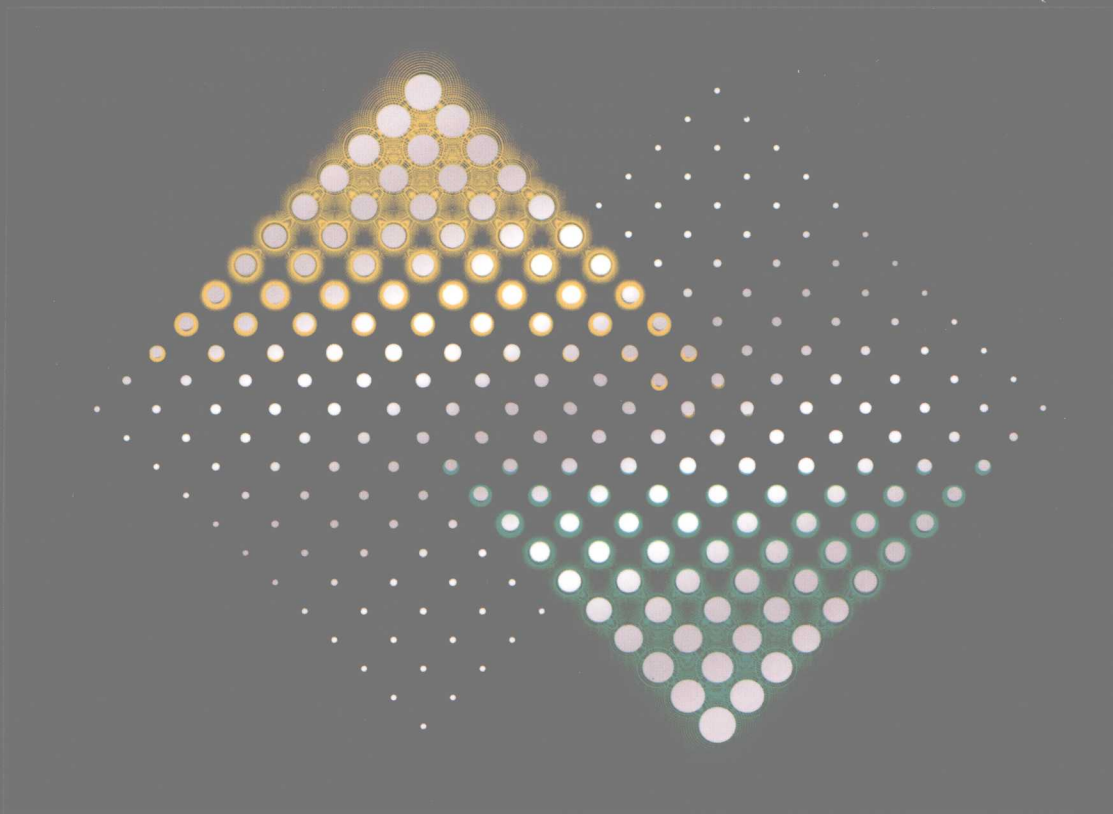


新编计算机类本科规划教材

# 数据结构

## ——使用C语言（第4版）

朱战立 编著



电子工业出版社

PUBLISHING HOUSE OF ELECTRONICS INDUSTRY

<http://www.phei.com.cn>

新编计算机类本科规划教材

# 数 据 结 构

## ——使用 C 语言

(第 4 版)

朱战立 编著

電 子 工 業 出 版 社

Publishing House of Electronics Industry

北京 · BEIJING

## 内 容 简 介

数据结构是计算机学科各专业的一门重要的专业基础课。本书包含了 2009 年研究生入学统考大纲的全部内容。本书系统地介绍了各种典型的数据结构,以及递归、查找和排序的方法。本书采用理论叙述简洁准确、实践应用举例丰富完整的方法编写,从而达到理论和实践密切结合的教学目的。本书采用 C 语言描述算法。

本书内容丰富,难度适中,文字简洁准确,图文并茂,应用实例多,教学参考资料丰富。

本书既可作为计算机本科、专科学生的教材,也可供从事计算机工程和应用工作的科技工作者参考。

未经许可,不得以任何方式复制或抄袭本书之部分或全部内容。  
版权所有,侵权必究。

## 图书在版编目(CIP)数据

数据结构:使用 C 语言/朱战立编著. —4 版. —北京:电子工业出版社,2009.1  
新编计算机类本科规划教材  
ISBN 978-7-121-07867-5

I. 数… II. 朱… 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. TP311.12 TP312

中国版本图书馆 CIP 数据核字(2008)第 183063 号

责任编辑:冉 哲

印 刷:北京市海淀区四季青印刷厂

装 订:涿州市桃园装订有限公司

出版发行:电子工业出版社

北京市海淀区万寿路 173 信箱 邮编 100036

开 本:787×1092 1/16 印张:23 字数:588 千字

印 次:2009 年 1 月第 1 次印刷

印 数:4 000 册 定价:34.00 元

凡所购买电子工业出版社图书有缺损问题,请向购买书店调换。若书店售缺,请与本社发行部联系,联系及邮购电话:(010) 88254888。

质量投诉请发邮件至 [zltts@phei.com.cn](mailto:zltts@phei.com.cn), 盗版侵权举报请发邮件至 [dbqq@phei.com.cn](mailto:dbqq@phei.com.cn)。

服务热线:(010) 88258888。

# 前 言

数据结构是计算机学科各专业一门重要的专业基础课，也是其他计算机相关专业的一门必修课或选修课。数据结构课程主要讨论现实世界中数据的各种逻辑结构、在计算机中的存储结构以及各种算法的设计问题。数据结构课程的目的，是使学生掌握组织数据、存储数据以及处理数据的基本概念和软件设计的基本方法，从而为进一步学习后续专业课程打下坚实的基础。

本书包含了 2009 年研究生入学统考大纲的全部内容。本书讨论的典型数据结构问题包括线性表、堆栈、队列、串、数组、递归、广义表、树、二叉树、图、排序、查找、文件等。对于线性表、堆栈、队列、串、数组、广义表、树、二叉树和图等基本数据结构问题，都详细讨论了各自的逻辑结构、存储结构以及各种算法的设计方法。排序和查找是两个应用广泛的算法设计问题，本书讨论了几种典型的排序算法，讨论了静态查找、动态查找和哈希查找的存储结构和查找方法。广义表、树、二叉树和图这些非线性结构的算法经常要设计成递归算法，本书专设一章讨论递归算法的设计方法等问题。

数据结构课程是一门理论和实践结合密切的课程。本书采用理论叙述简洁准确、实践应用举例丰富完整的方法编写。理论采用丰富、完整的设计实例予以说明，设计实例从侧面解释了概念和应用方法，从而达到理论和实践密切结合的教学目的。本书采用 C 语言描述算法。

本书具有如下特点。

(1) 内容丰富，难度适中，文字简洁准确，图文并茂。

(2) 本书的所有算法都经上机调试通过，包括各章的操作实现函数、各章的程序设计实例以及习题解答中给出的算法设计。

(3) 习题全面，覆盖面广，择要解答。每章最后设计了大量的习题，覆盖了各章的全部教学内容，并在附录 B 中给出了部分习题解答。

(4) 课内上机参考资料丰富。数据结构课程是一门理论结合实践的课程，通常要求包含 10 多课时的课内上机实习（或称项目设计）。本书各章的习题部分都专门设计了一定数量的上机实习题。另外，附录 A 还给出了上机实习报告书写规范和一个上机实习报告书写实例，可供学生参考。

本书作者 20 多年来一直从事数据结构课程的教学工作，曾编著过若干本采用不同算法描述语言的数据结构教材。本书是在经过长期使用的以前出版的教材基础上，参照新的研究生入学统考大纲，通过作者进一步修改、补充和完善而成的。

根据作者的经验，使用本教材授课约需 54~80 课时，其中包括约 10 课时的课内上机实习。使用本书的读者可登录华信教育资源网 (<http://www.huaxin.edu.cn>) 免费下载教学课件。

作 者  
2008.8

# 目 录

第 1 章 绪论	1
1.1 数据结构的基本概念	1
1.1.1 数据、数据元素、数据元素的数据类型	1
1.1.2 数据的逻辑结构	2
1.1.3 数据的存储结构	3
1.1.4 数据的操作	4
1.1.5 数据结构课程讨论的主要内容	4
1.2 抽象数据类型	4
1.3 算法和算法的时间复杂度	5
1.3.1 算法	5
1.3.2 算法的性质和设计目标	7
1.3.3 算法的时间效率分析	8
1.3.4 算法耗时的实际测试	10
1.3.5 数据元素个数和时间复杂度	13
1.4 算法书写规范	14
习题 1	14
第 2 章 线性表	17
2.1 线性表概述	17
2.1.1 线性表的定义	17
2.1.2 线性表的抽象数据类型	17
2.2 线性表的顺序表示和实现	18
2.2.1 顺序表的存储结构	18
2.2.2 顺序表操作的实现	19
2.2.3 顺序表操作的效率分析	22
2.2.4 顺序表应用举例	23
2.3 线性表的链式表示和实现	26
2.3.1 单链表的存储结构	26
2.3.2 单链表的操作实现	29
2.3.3 单链表操作的效率分析	34
2.3.4 单链表应用举例	35
2.3.5 循环单链表	36
2.3.6 双向链表	36
2.4 静态链表	40
2.5 算法设计举例	41
2.5.1 顺序表算法设计举例	41

2.5.2 单链表算法设计举例 .....	42
习题 2 .....	44
<b>第 3 章 堆栈和队列</b> .....	<b>48</b>
3.1 堆栈 .....	48
3.1.1 堆栈的基本概念 .....	48
3.1.2 堆栈的抽象数据类型 .....	49
3.1.3 堆栈的顺序表示和实现 .....	50
3.1.4 堆栈的链式表示和实现 .....	53
3.2 堆栈应用 .....	55
3.2.1 括号匹配问题 .....	55
3.2.2 算术表达式计算问题 .....	58
3.3 队列 .....	64
3.3.1 队列的基本概念 .....	64
3.3.2 队列的抽象数据类型 .....	64
3.3.3 顺序队列以及存在的问题 .....	65
3.3.4 顺序循环队列的表示和实现 .....	66
3.3.5 链式队列 .....	69
3.3.6 队列的应用 .....	72
3.4 优先级队列 .....	76
3.4.1 顺序优先级队列的设计和实现 .....	77
3.4.2 优先级队列的应用 .....	79
习题 3 .....	81
<b>第 4 章 串</b> .....	<b>86</b>
4.1 串概述 .....	86
4.1.1 串及其基本概念 .....	86
4.1.2 串的抽象数据类型 .....	87
4.1.3 C 语言的串函数 .....	88
4.2 串的存储结构 .....	90
4.3 串基本操作的实现算法 .....	92
4.4 串的模式匹配算法 .....	97
4.4.1 Brute-Force 算法 .....	98
4.4.2 KMP 算法 .....	101
4.4.3 Brute-Force 算法和 KMP 算法的比较 .....	106
习题 4 .....	109
<b>第 5 章 数组</b> .....	<b>112</b>
5.1 数组概述 .....	112
5.1.1 数组的定义 .....	112
5.1.2 数组的实现机制 .....	112
5.1.3 数组的抽象数据类型 .....	113
5.2 动态数组 .....	113

5.2.1	动态数组的设计方法 .....	113
5.2.2	动态数组和静态数组对比 .....	117
5.3	特殊矩阵的压缩存储 .....	118
5.4	稀疏矩阵的压缩存储 .....	120
5.4.1	稀疏矩阵的三元组顺序表 .....	121
5.4.2	稀疏矩阵的三元组链表 .....	123
习题 5	.....	124
<b>第 6 章</b>	<b>递归算法</b> .....	<b>127</b>
6.1	递归的概念 .....	127
6.2	递归算法的执行过程 .....	128
6.3	递归算法的设计方法 .....	131
6.4	递归过程和运行时栈 .....	133
6.5	递归算法的时间效率分析 .....	135
6.6	递归算法到非递归算法的转换 .....	137
6.7	设计举例 .....	141
6.7.1	一般递归算法设计举例 .....	141
6.7.2	回溯算法及设计举例 .....	143
习题 6	.....	147
<b>第 7 章</b>	<b>广义表</b> .....	<b>150</b>
7.1	广义表概述 .....	150
7.1.1	广义表的概念 .....	150
7.1.2	广义表的抽象数据类型 .....	151
7.2	广义表的存储结构 .....	152
7.2.1	头链和尾链存储结构 .....	152
7.2.2	原子和子表存储结构 .....	152
7.3	广义表的操作实现 .....	153
7.3.1	头链和尾链存储结构下的操作实现 .....	153
7.3.2	头链和尾链存储结构应用举例 .....	159
7.3.3	原子和子表存储结构下的操作实现 .....	161
7.3.4	原子和子表存储结构应用举例 .....	164
习题 7	.....	164
<b>第 8 章</b>	<b>树和二叉树</b> .....	<b>166</b>
8.1	树 .....	166
8.1.1	树的定义 .....	166
8.1.2	树的表示方法 .....	167
8.1.3	树的抽象数据类型 .....	168
8.1.4	树的存储结构 .....	169
8.2	二叉树 .....	171
8.2.1	二叉树的定义 .....	171
8.2.2	二叉树的抽象数据类型 .....	172

8.2.3	二叉树的性质 .....	173
8.3	二叉树的设计和实现 .....	175
8.3.1	二叉树的存储结构 .....	175
8.3.2	二叉树的操作实现 .....	177
8.4	二叉树遍历 .....	179
8.4.1	二叉树遍历的方法和结构 .....	179
8.4.2	二叉链存储结构下二叉树遍历的实现 .....	181
8.4.3	二叉树遍历的应用 .....	182
8.4.4	非递归的二叉树遍历算法 .....	185
8.5	线索二叉树 .....	186
8.5.1	线索二叉树及其用途 .....	186
8.5.2	中序线索二叉树的设计 .....	188
8.5.3	中序线索二叉树循环操作的设计 .....	190
8.5.4	设计举例 .....	191
8.6	哈夫曼树 .....	193
8.6.1	哈夫曼树的基本概念 .....	193
8.6.2	哈夫曼编码问题 .....	194
8.6.3	哈夫曼编码问题设计和实现 .....	195
8.7	等价问题 .....	200
8.8	树与二叉树的转换 .....	204
8.9	树的遍历 .....	205
习题 8	.....	206
<b>第 9 章</b>	<b>图 .....</b>	<b>209</b>
9.1	图概述 .....	209
9.1.1	图的基本概念 .....	209
9.1.2	图的抽象数据类型 .....	211
9.2	图的存储结构 .....	212
9.2.1	图的邻接矩阵存储结构 .....	212
9.2.2	图的邻接表存储结构 .....	213
9.3	图的实现 .....	214
9.3.1	邻接矩阵存储结构下图操作的实现 .....	214
9.3.2	邻接表存储结构下图操作的实现 .....	219
9.4	图的遍历 .....	224
9.4.1	图的深度和广度优先遍历算法 .....	224
9.4.2	图的深度和广度优先遍历算法实现 .....	225
9.5	最小生成树 .....	228
9.5.1	最小生成树的基本概念 .....	228
9.5.2	普里姆算法 .....	229
9.5.3	克鲁斯卡尔算法 .....	234
9.6	最短路径 .....	235



9.6.1	最短路径的基本概念 .....	235
9.6.3	每对顶点之间的最短路径 .....	240
9.7	拓扑排序 .....	242
9.8	关键路径 .....	245
习题 9	.....	249
<b>第 10 章</b>	<b>排序</b> .....	252
10.1	排序的基本概念 .....	252
10.2	插入排序 .....	254
10.2.1	直接插入排序 .....	254
10.2.2	希尔排序 .....	256
10.3	选择排序 .....	258
10.3.1	直接选择排序 .....	258
10.3.2	堆排序 .....	259
10.4	交换排序 .....	264
10.4.1	冒泡排序 .....	264
10.4.2	快速排序 .....	266
10.5	归并排序 .....	269
10.6	基数排序 .....	271
10.7	排序算法性能比较 .....	275
习题 10	.....	275
<b>第 11 章</b>	<b>查找</b> .....	279
11.1	查找的基本概念 .....	279
11.2	静态查找 .....	280
11.2.1	顺序表 .....	280
11.2.2	有序顺序表 .....	281
11.2.3	索引顺序表 .....	283
11.3	动态查找 .....	286
11.3.1	二叉排序树和平衡二叉树 .....	286
11.3.2	B_树和 B <sup>+</sup> 树 .....	293
11.4	哈希查找 .....	298
11.4.1	哈希表的基本概念 .....	299
11.4.2	哈希函数构造方法 .....	300
11.4.3	哈希冲突解决方法 .....	302
11.4.4	哈希表设计 .....	304
习题 11	.....	308
<b>第 12 章</b>	<b>文件</b> .....	312
12.1	文件概述 .....	312
12.1.1	文件的演变过程及基本概念 .....	312
12.1.2	文件的存储介质 .....	313
12.1.3	文件的基本操作 .....	314

12.2 顺序文件.....	315
12.3 索引文件.....	316
12.4 ISAM 文件 .....	317
12.5 VSAM 文件 .....	319
12.6 散列文件.....	321
习题 12 .....	322
<b>附录 A 上机实习内容规范和上机实习报告范例 .....</b>	<b>323</b>
附录 A.1 上机实习内容规范 .....	323
附录 A.2 上机实习报告范例 .....	323
<b>附录 B 部分习题解答 .....</b>	<b>329</b>
<b>参考文献 .....</b>	<b>358</b>

# 第1章 绪 论

计算机是对各种各样数据进行处理机器。要对数据进行处理，首先就要对数据进行有效的组织。因此，在计算机中如何有效地组织数据和高效地处理数据就是计算机科学的基本研究内容，也是继续深入学习后续课程的基础。本章主要对数据结构课程学习中将遇到的基本概念做概括性的叙述，这些内容将贯穿数据结构课程的整个学习过程。

本章内容主要包括：数据结构的基本概念、抽象数据类型的概念和意义、算法和算法的时间复杂度。

## 1.1 数据结构的基本概念

### 1.1.1 数据、数据元素、数据元素的数据类型

数据是人们利用文字符号、数字符号以及其他规定的符号对现实世界的事物及其活动所做的抽象描述。

例如，“今天天气情况是，最高温度为  $5^{\circ}\text{C}$ ，最低温度为  $-5^{\circ}\text{C}$ ”，就是关于今天天气情况的描述数据。又如，“班上甲同学姓名叫张三，乙同学姓名叫李四”，就是关于班上同学姓名的描述数据。

表示一个事物的一组数据称作一个数据元素。构成数据元素的数据称作该数据元素的数据项。

例如，要描述学生信息，可包括学生的学号、姓名、性别、年龄等数据。学生的学号、姓名、性别、年龄等数据就构成学生情况描述的数据项；包括学号、姓名、性别、年龄等数据项的一组数据就构成学生信息的一个数据元素。表 1-1 是一个包含 3 个数据元素的学生信息表。

表 1-1 学生信息表

学号	姓名	性别	年龄
2000001	张三	男	20
2000002	李四	男	21
2000003	王五	女	22

在讨论数据结构时，关于数据元素、数据项的描述都使用某种高级程序设计语言来描述，本书采用 C 语言描述。学生信息的数据元素是由多个数据项构成的，其数据元素的数据类型的 C 语言描述方法定义为如下结构体：

```
struct Student
{
    long number;
    char name[10];
    char sex[3];
}
```

```
    int age;
};
```

通过上述定义，用户自定义的结构体 `struct Student` 就可像 C 语言中基本数据类型 `char`、`int`、`float` 一样使用。

为使用简便起见，还可以把上述定义改写为：

```
typedef struct Student
{
    long number;
    char name[10];
    char sex[3];
    int age;
} StudentType;
```

经过上述定义后，`StudentType` 就被看作和 `struct Student` 含义相同的标识符。

上述学生情况数据元素的数据类型定义为 `StudentType`，这与把数据元素定义为 `int`、`float`、`long`、`char` 等数据类型一样，都是给出了具体数据元素的数据类型。

但是，像数学一样，数据结构课程讨论的大部分算法都可以适用于任何数据类型的数据元素。这时，为了考虑算法的通用性，算法要处理的数据元素也可以是不具有实际含义的数据元素。我们把没有实际含义的数据元素称作**抽象数据元素**。在本书中，抽象数据元素用  $a_0, a_1, \dots, a_{n-1}$  表示。

在设计算法时，任何数据元素都要指定数据类型，抽象数据元素也不例外。本书用符号 `DataType` 表示抽象数据元素的数据类型。当软件设计问题具体确定时，抽象数据元素的数据类型将被具体数据元素的数据类型取代。例如，若线性表的数据元素类型为 `DataType`，当设计的具体线性表的数据元素类型为 `int` 时，可通过预先定义 `DataType` 为 `int` 来完成程序的设计；当设计的具体线性表的数据元素集合为表 1-1 中的学生信息时，可通过定义 `DataType` 为 `StudentType` 来完成程序的设计。具体的 C 语言语句为：

```
typedef int DataType;
或 typedef StudentType DataType;
```

### 1.1.2 数据的逻辑结构

数据元素之间的相互联系方式称为**数据的逻辑结构**。

按照数据元素之间的相互联系方式，数据的逻辑结构主要可分为线性结构、树型结构和图型结构三种。

线性结构的定义是：除第一个和最后一个数据元素外，每个数据元素只有一个唯一的前驱数据元素和一个唯一的后继数据元素。线性结构可以表示为图 1-1 (a) 所示的形式，图中，A、B、C、D 为数据元素，A 是第一个数据元素，D 是最后一个数据元素，A 是 B 的前驱数据元素，C 是 B 的后继数据元素；依次类推。

树型结构的定义是：除根结点外，每个数据元素只有一个唯一的前驱数据元素，可有零个或若干个后继数据元素。图 1-1 (b) 是一个树型结构的例子。对于数据元素 A、B、C、D、E、F、G，数据元素 A 是根结点，A 没有前驱数据元素，有两个后继数据元素 B 和 C；数据元素 B 的前驱数据元素为 A，后继数据元素为 D 和 E；数据元素 C 的前驱数据元素为 A，没有后继数据元素；如此等等。

图型结构的定义是：每个数据元素可有零个或若干个前驱数据元素和零个或若干个后继数据元素。图 1-1 (c) 是一个图型结构的例子。对于数据元素 A、B、C、D、E、F、G，若以 A 为起始点，则数据元素 E 有两个前驱数据元素 B 和 C，有两个后继数据元素 F 和 G。

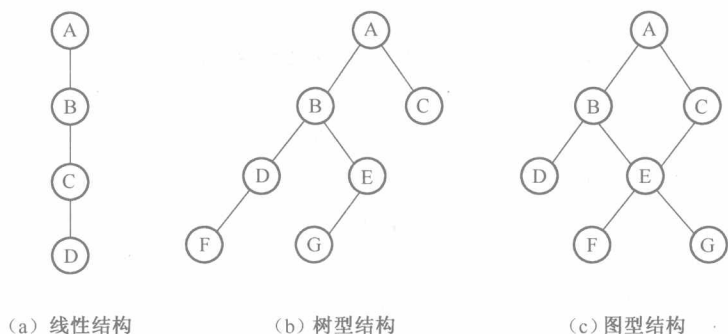


图 1-1 基本的数据逻辑结构

树型结构和图型结构也可以归为非线性结构。数据元素之间不存在如图 1-1 (a) 所示的一对一关系的结构都称作非线性结构。

### 1.1.3 数据的存储结构

任何需要计算机进行管理和处理的数据元素都必须首先按某种方式存储在计算机中。数据元素在计算机中的存储方式称为**数据的存储结构**。数据存储结构的基本形式有两种：一种是顺序存储结构，另一种是链式存储结构。

**顺序存储结构**是指把数据元素存储在一块连续地址空间的内存中，其特点是，逻辑上相邻的数据元素在物理上也相邻，数据间的逻辑关系表现在数据元素的存储位置关系上。当采用高级程序设计语言表示时，实现顺序存储结构的方法是使用数组。图 1-2 (a) 所示为线性结构数据元素  $a_0, a_1, \dots, a_{n-1}$  的顺序存储结构示意图。其中， $0, 1, 2, \dots, n-2, n-1$  既是数据元素的编号，也是存储数据元素  $a_0, a_1, \dots, a_{n-1}$  的数组下标。

指针是指向物理存储单元地址的变量。我们把由数据元素域和指针域组成的一个结构体称为一个**结点**。**链式存储结构**是使用指针把相互直接关联的结点（即直接前驱结点或直接后继结点）链接起来，其特点是，逻辑上相邻的数据元素在物理上（即内存存储位置上）不一定相邻，数据间的逻辑关系表现在结点的链接关系上。图 1-2 (b) 所示为线性结构数据元素  $a_0, a_1, \dots, a_{n-1}$  的链式存储结构示意图。其中，上一个结点到下一个结点的箭头表示上一个结点的指针域中保存的下一个结点在内存中的存储地址。**head** 是指向第一个结点的指针，通常称作**头指针**。

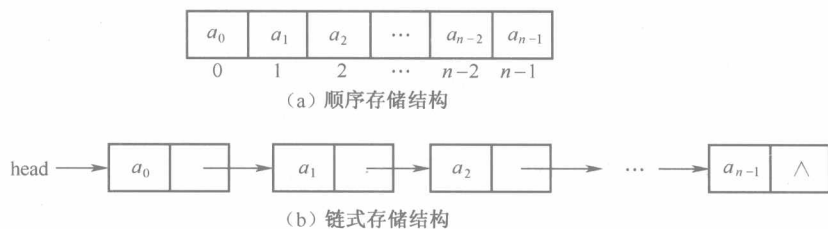


图 1-2 基本的存储结构

顺序存储结构和链式存储结构是两种最基本、最常用的存储结构。除此之外，利用顺序存储结构和链式存储结构进行组合，还可以有一些更复杂的存储结构。

### 1.1.4 数据的操作

一种数据类型数据允许进行的某种操作称作**数据的操作**，一种数据类型数据所有的操作称作**数据的操作集合**。

数据结构课程在讨论数据的操作时，一般从抽象和具体两个角度进行讨论。在抽象角度下，数据的操作主要讨论数据操作所完成的逻辑功能。在抽象角度下，数据的操作一般与数据的逻辑结构一起讨论；在具体角度下，数据的操作主要讨论数据操作的具体实现算法。例如，若某软件要对表 1-1 中的学生信息进行处理，对学生信息可能进行的操作有：插入一条数据元素，删除一条数据元素，列出所有数据元素的值等。所以，该问题数据的操作有：插入一条数据元素，删除一条数据元素，列出所有数据元素的值等。在其抽象角度下，这些操作的逻辑功能如其字面含义所述；在具体角度下，表 1-1 的学生信息既可采用图 1-2 (a) 的顺序存储结构存储数据元素，也可采用图 1-2 (b) 的链式存储结构存储数据元素，不同的存储结构操作实现的具体算法将不同。

### 1.1.5 数据结构课程讨论的主要内容

数据结构课程主要讨论表、堆栈、队列、串、数组、树、二叉树、图等典型的常用数据结构，在讨论这些典型的常用数据结构时，主要从它们的逻辑结构、存储结构和数据操作 3 个方面进行分析讨论。例如，我们在第 2 章中讨论线性表时，2.1 节将讨论线性表的抽象数据类型（即线性表的逻辑结构和逻辑结构意义下的操作功能），2.2 节将讨论线性表的顺序存储结构和顺序存储结构下各基本操作的具体实现算法，2.3 节将讨论线性表的链式存储结构和链式存储结构下各基本操作的具体实现算法。其他各章中对堆栈、队列、串、数组、树、二叉树、图等进行讨论的章节安排次序与第 2 章的类同。

## 1.2 抽象数据类型

**类型**是一组值的集合。

例如，`int` 类型就是具体计算机所能表示的 `int` 类型数值的集合，通常，`int` 类型的数值范围是  $-32768 \sim 32767$ 。又如，`float` 类型就是具体计算机所能表示的 `float` 类型数值的集合。

**数据类型**是指一个类型和定义在这个类型上的操作集合。

例如，当我们说计算机中的 `int` 数据类型时，我们不仅指 `int` 类型所能表示的  $-32768 \sim 32767$  的数值范围，还指 `int` 类型的数据允许进行的加 (+)、减 (-)、乘 (\*)、除 (/) 和求模 (%) 操作。

在数据结构课程中，通常把在已有的数据类型基础上设计新的数据类型的过程称作**数据结构设计**。

**抽象数据类型** (Abstract Data Type, ADT) 是指一个逻辑概念上的类型和这个类型上的操作集合。

从定义看，数据类型和抽象数据类型的定义基本相同。数据类型和抽象数据类型的不同之处仅仅在于：数据类型通常指的是高级程序设计语言支持的基本数据类型，而抽象数据类

型指的是在基本数据类型支持下用户新设计的数据类型。数据结构课程主要讨论表、堆栈、队列、串、数组、树、二叉树、图等典型的常用数据结构，这些典型的常用数据结构就是一个个不同的抽象数据类型。

盖楼时，如果用砖、水泥、沙子来盖，则不仅建造周期长，而且楼不可能盖得很高（否则将不安全）；如果用水泥预制板（即更大的模块）来盖，则不仅建造周期短（水泥预制板由专门的公司按规范的规格提供），楼能盖很高，而且所建造的高楼能保证安全。从数学的观点看，水泥预制板使高楼建造过程的接缝数量大大减少，从而大大降低了高楼建造的复杂度。

抽象数据类型使软件设计成为工业化流水线生产的一个中间环节。一方面，根据给出的抽象数据类型定义，负责设计这些抽象数据类型的专门公司（或专门设计人员）设计该抽象数据类型的具体存储结构以及在具体存储结构下各操作的具体实现算法；另一方面，利用已设计实现的抽象数据类型模块，负责设计应用软件的专门公司（或专门设计人员）可以安全、快速、方便地完成该应用软件系统的设计。这样的方法与使用水泥预制板建造高楼的方法类同。水泥预制板规格的设计是高楼建造的一个中间环节。一方面，根据给出的水泥预制板规格，负责提供水泥预制板的专门公司建造水泥预制板；另一方面，根据给出的水泥预制板规格，高楼的设计人员和建造人员可以安全、快速、方便地完成高楼的设计和建造。

软件的设计采用模块化方法，抽象数据类型（如表、堆栈、队列、串、数组、树、二叉树、图等）就是构造大型软件的最基本模块。用这些已有专门公司设计好的抽象数据类型，就可以安全、快速、方便地设计功能复杂的大型软件。

数据结构课程讨论表、堆栈、队列、串、数组、树、二叉树、图等基本数据结构的功能和设计方法。在大部分高级程序设计语言的类库包中，这些常用的数据结构都已设计完成。设计人员通常不需要重新设计和实现这些数据结构，只需要根据问题的需要，首先把相应的数据结构头文件包含进来，然后定义具体的变量，调用相应的函数，即可完成所要完成的功能。从这个角度看，有些人可能会觉得数据结构课程的学习没有必要。对于计算机学科的学生，以及那些希望深入学习掌握计算机软件设计方法的学生来说，本课程对典型数据结构的分析以及对典型算法设计方法的讨论和训练，既能帮助学生打好软件设计的扎实基础，也能帮助学生掌握软件模块化设计的基本方法。

## 1.3 算法和算法的时间复杂度

### 1.3.1 算法

算法是描述求解问题方法的操作步骤集合。

算法要用某种语言来描述。描述算法的语言主要有三种形式：文字形式、伪码形式和程序设计语言形式。文字形式是指用中文或英文这样的文字来描述算法。伪码形式是指用一种仿程序设计语言的语言（因这样的描述语言不是真正的程序设计语言，所以称作伪码）来描述算法。程序设计语言形式是指用某种高级程序设计语言来描述算法。用高级程序设计语言描述算法的优点是，这样的算法描述既简洁易读，又可以直接输入计算机调用或运行。本书采用 C 语言这种高级程序设计语言描述算法。

下边给出算法设计的两个例子。从这两个例子，读者可体会用高级程序设计语言描述算法的基本方法以及这种描述方法的优点。

**【例 1-1】** 设计一个把存储在数组中的有  $n$  个抽象数据元素  $a_0, a_1, \dots, a_{n-1}$  逆置的算法。

逆置就是把数据元素序列  $a_0, a_1, \dots, a_{n-1}$  变换为数据元素序列  $a_{n-1}, \dots, a_1, a_0$ ，并要求原数组中的数据元素值不被改变。

**【算法参数设计】** 这个算法的参数应该包括三个：表示原数组的输入参数  $a$ ，表示数据元素个数的输入参数  $n$ ，表示逆置后数组的输出参数  $b$ 。

算法设计如下：

```
void Reverse(DataType a[], int n, DataType b[] )
{
    int i;

    for(i = 0; i < n; i++)
        b[i] = a[n - 1 - i];
}
```

该算法共进行  $n$  次赋值，算法的实现过程如图 1-3 所示。

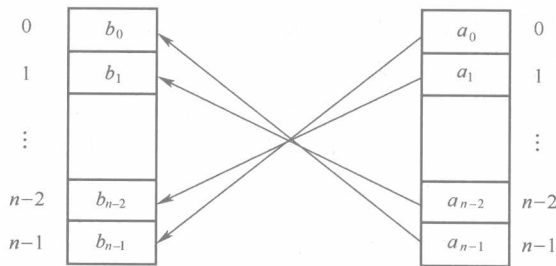


图 1-3 逆置算法实现方法

**【例 1-2】** 设计一个把存储在数组中的有  $n$  个抽象数据元素  $a_0, a_1, \dots, a_{n-1}$  就地逆置的算法。就地逆置就是把数据元素序列  $a_0, a_1, \dots, a_{n-1}$  变换为数据元素序列  $a_{n-1}, \dots, a_1, a_0$ ，并要求这种变换在原数组中进行。

**【算法参数设计】** 这个算法的参数应该包括两个：表示原数组和就地逆置后数组的参数  $a$ ，表示数据元素个数的参数  $n$ 。

算法设计如下：

```
void Reverse(DataType a[], int n)
{
    int i, m = n/2;
    DataType temp;

    for(i = 0; i < m; i++) //进行 m 次调换
    {
        temp = a[i];
        a[i] = a[n - 1 - i];
        a[n - 1 - i] = temp;
    }
}
```

**注意：**在 C 语言中，当除数和被除数都是整数时，运算符 “/” 表示整数相除，即商只取



整数部分。例如，当  $n=10$  时， $n/2$  的运算结果为 5，即变量  $m$  得到赋值 5；当  $n=11$  时， $n/2$  的运算结果仍为 5，即变量  $m$  仍得到赋值 5。

该算法共进行  $n/2$  次调换，第一次调换的过程如图 1-4 所示。

### 1.3.2 算法的性质和设计目标

任何算法设计都应满足以下性质。

- ① 输入性：具有零个或若干个输入量。
- ② 输出性：至少产生一个输出量或执行一个有意义操作。
- ③ 有限性：执行语句的序列是有限的。
- ④ 确定性：每一条语句的含义明确，无二义性。
- ⑤ 可执行性：每一条语句都应在有限的时间内完成。

用高级程序设计语言描述算法时，一个算法就是一个函数。算法的输入量就是函数的输入参数，算法的输出量就是函数的输出参数。由于高级程序设计语言规范了语句格式，不允许二义性语句，因此在用高级程序设计语言语句组合出的算法中，只要没有无限循环，则必然满足算法的有限性、确定性和可执行性的性质。

应用程序通常是通过调用函数（即算法）来完成的。程序和算法的唯一区别是，程序允许无限循环，而算法不允许无限循环。构成无限循环的一组语句如下：

```
while(1)                                //循环条件永真
{
    ...                                  //任意语句序列
}
```

算法设计应满足以下 5 个目标。

- ① 正确性。算法应确切地满足具体问题的需求。这是算法设计的基本目标。
- ② 可读性。算法的可读性有利于人们对算法的理解，这既有利于程序的调试和维护，也有利于算法的交流和移植。算法的可读性主要体现在两个方面：一是变量名、常量名、函数名等的命名要见名知意；二是要有足够多的注释。
- ③ 健壮性。当输入非法数据时，算法要能做出适当的处理，而不应产生不可预料的结果。
- ④ 高时间效率。算法的时间效率是指运行算法需要花费时间的多少。对于同一个问题，如果有多个算法可供选择，则应尽可能选择运行时间短的算法。运行时间短的算法称作高时间效率的算法。
- ⑤ 高空间效率。算法的空间效率是指运行算法需要占用的额外内存空间的多少。对于同一个问题，如果有多个算法可供选择，则应尽可能选择占用内存空间少的算法。占用内存空间少的算法称作高空间效率的算法。

算法的高时间效率和高空间效率通常是矛盾的。例如有些问题，若算法占用了较多的内存空间，则算法只需较少次循环就能实现，因而时间效率会提高；若算法占用了较少的内存空间，则算法需要较多次循环才能实现，因而时间效率会降低。在目前计算机内存价格快速下降的趋势下，当算法设计的时间效率目标和空间效率目标发生矛盾时，对于大多数情况来说，算法的时间效率目标应首先被考虑。

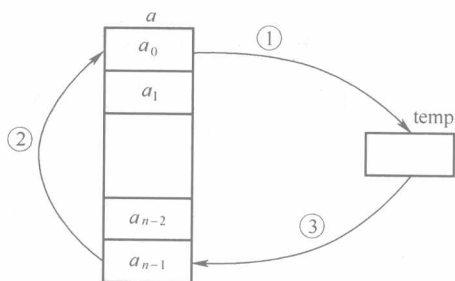


图 1-4 就地逆置算法实现方法