Access 实践教程

吴宏瑜　主编

四川大学出版社

Access 实践教程

主　编　吴宏瑜
副主编　吴拾音
编　委　戴　蓉　孙亚飞　马义玲
　　曾　新　王　凡　戴丽娟
　　夏　欣

江苏工业学院图书馆
藏书章

四川大学出版社

责任编辑:毕　潜
责任校对:傅　奕　段悟吾
封面设计:罗　光
责任印制:李　平

图书在版编目(CIP)数据

Access 实践教程 / 吴宏瑜主编. —成都：四川大学出版社，2009.1
ISBN 978-7-5614-4217-3

Ⅰ.A… Ⅱ.吴… Ⅲ.关系数据库-数据库管理系统，Access-教材 Ⅳ.TP311.138

中国版本图书馆 CIP 数据核字（2008）第 203402 号

书名　Access 实践教程

主　　编　吴宏瑜
出　　版　四川大学出版社
地　　址　成都市一环路南一段 24 号 (610065)
发　　行　四川大学出版社
书　　号　ISBN 978-7-5614-4217-3
印　　刷　郫县犀浦印刷厂
成品尺寸　185 mm×260 mm
印　　张　20
字　　数　480 千字
版　　次　2009 年 1 月第 1 版
印　　次　2009 年 1 月第 1 次印刷
印　　数　0 001～3 500 册
定　　价　33.00 元

◆读者邮购本书，请与本社发行科联系。电 话:85408408/85401670/85408023　邮政编码:610065
◆本社图书如有印装质量问题，请寄回出版社调换。
◆网址：www. scupress. com. cn

版权所有◆侵权必究

前　言

Microsoft Access 2003 是 Microsoft 公司开发的小型桌面数据库开发软件，一直占据着不可替代的领先地位，它在应用软件领域独领风骚。Microsoft Access 2003 具有广泛的实用性，在使用它进行数据库设计时，用户仅需要告诉它要干什么，而不需要了解其中具体的细节，这样便减轻了我们的负担。所以可以说，Microsoft Access 2003 是我们在进行数据库设计时最得力的“助手”之一。

本书共分 4 章。第 1 章详细讲述了计算机公共基础知识，提供了习题分析和本章练习及答案；第 2 章以考点形式归纳和总结了 Access 基本数据库理论、数据表、查询、窗体、报表、数据访问页、模块 VBA 等实用的数据库重点知识，提供了习题分析和本章练习及答案；第 3 章以 8 个实验为主线，操作步骤为辅导，重点练习 Access 中的数据表、查询、窗体、报表、数据访问页、模块 VBA 等知识点，让学习者能学以致用；第 4 章组织了 8 套综合练习，以达到理解和巩固所学知识的目的。全书内容详细，重点和技巧突出，可帮助学习者顺利地通过计算机的二级考试。

本书由四川大学计算机基础教学中心老师编写，吴宏瑜任主编，吴拾音任副主编，戴蓉、孙亚飞、马义玲、曾新、王凡、戴丽娟、夏欣任编委。本书从著书到出版的过程中，得到了很多老师无私的帮助和全力的支持，在此表示衷心的感谢。

本书图例和示例中，含有通信地址、电子邮件地址、网页地址、图片和表格数据等，内容如有雷同，纯属巧合。

由于编者水平有限，时间仓促，书中的缺点和错误在所难免，欢迎广大读者和用户批评指正。

编　者

2008 年 12 月

目　录

此为试读，需要完整PDF请访问：www.ertongbook.com

第 1 章　软件技术基础

本章要求掌握的主要内容包括算法、基本数据结构及其运算、查找与排序技术、资源管理技术、数据库技术、应用软件设计与开发技术。

1.1　数据结构与算法

数据结构是计算机软件和计算机应用专业的核心课程之一，在众多的计算机系统软件和应用软件中都要用到各种数据结构。因此，仅掌握几种计算机语言是难以应付众多复杂的实际需求的。要想有效地使用计算机，就必须学习数据结构的有关知识。

1.1.1　算法

1. 算法的基本概念

算法是指解题方案的准确而完整的描述。算法不等于程序，也不等于计算机方法，程序的编制不可能优于算法的设计。

算法的基本特征是一组严谨地定义运算顺序的规则，每一个规则都是有效的、明确的，此顺序将在有限的次数下终止。算法的基本特征包括以下四个方面：

①可行性。

②确定性。算法中每一步骤都必须有明确的定义，不允许有模棱两可的解释，不允许有多义性。

③有穷性。算法必须能在有限的时间内做完，即能在执行有限个步骤后终止，执行时间合理。

④拥有足够的情报。

算法的基本要素包含两点：一是对数据对象的运算和操作，二是算法的控制结构。

求解同一计算问题可能有许多不同的算法，到底如何来评价这些算法的好坏，以便从中选出较好的算法呢？评价一个好的算法有以下三个标准：

①正确性（Correctness）。算法应满足具体问题的需求。

②可读性（Readability）。算法应该易读，以便于阅读者对程序的理解。

③健壮性（Robustness）。算法应具有容错处理，当输入非法数据时，算法应对其作出反应，而不是产生莫名其妙的输出结果。

2. 算法性能选择

要找到一个占用存储空间小、运行时间短、其他性能也好的算法是非常困难的，因为上述要求有时相互抵触。通常，要节约算法的执行时间，则要以牺牲更多的空间为代价；而为了节省空间，就可能要耗费更多的计算时间。因此，只能根据具体情况有所侧重。

例如，若该程序使用次数较少，则力求算法简明易懂；对于反复多次使用的程序，应尽可能选用快速的算法；若待解决的问题数据量极大，机器的存储空间较小，则相应算法主要考虑如何节省空间。

3. 算法描述语言

算法描述语言是一种具体描述算法细节的工具，算法工具只面向读者，不能在计算机中直接使用。算法描述语言在形式上非常简单，它类似于程序语言，因此非常适合那些以算法或逻辑处理为主的模块功能描述。

（1）语法形式

算法描述语言的语法不是十分严格，它主要由符号与表达式、赋值语句、控制转移语句、循环语句和其他语句构成。符号命名、数学及逻辑表达式一般与程序书写一致，赋值用箭头表示。语句可有标识，标识可以是数字，也可以是具有实际意义的单词。例如，循环累加可表示为：loop：i←i+1。

（2）控制转移语句

无条件转移语句用“GOTO 语句标识”表示，条件转移语句用“IF C THEN S1 ELSE S2”。其中，C、S1 和 S2 可以是一个逻辑表达式，也可以是用“{}”括起来的语句组。如果 C 为“真”，则 S1 被执行；如果 C 为“假”，则执行 S2。

（3）循环语句

循环语句有以下两种形式：

①WHILE 语句的形式为“WHILE C DO S”，其中 C 和 S 同上，如果 C 为“真”，则执行 S，且在每次执行 S 之后都要重新检查 C；如果 C 为“假”，控制就转到紧跟在 WHILE 后面的语句。

②FOR 语句的形式为“FOR i=init TO limit BY step DO S”，其中 i 是循环控制变量，init、limit 和 step 都是算术表达式，而 S 同上，每当 S 被执行一次时，i 从初值加步长一次，直到 i>limit 为止。

（4）其他语句

在算法描述中，还可能要用到其他一些语句，因为它们都是用最简明的形式给出的，故很容易知道它们的含义。例如，EXIT 语句，RETURN 语句，READ（或 INPUT）和 OUTPUT（或 PRINT，WRITE）语句等。

算法的描述方法可以归纳为以下四种：

①自然语言。

②图形。如 N-S 图、流程图，图的描述与算法语言的描述对应。

③算法语言。即计算机语言、程序设计语言、伪代码。

④形式语言。用数学的方法，可以避免自然语言的二义性。

用各种算法描述方法所描述的同一算法，该算法的功用是一样的，但允许在算法的描述和实现方法上有所不同。

人们的生产活动和日常生活离不开算法，都在自觉或不自觉地使用算法，例如人们到商店购买物品，会首先确定购买哪些物品，准备所需的费用，然后再决定到哪个商场选购、怎样去商场、行走的路线，对购买物品不满意又如何处理，购买物品后做什么等。以上购物的算法是用自然语言描述的，也可以用其他描述方法描述该算法。

4. 算法效率的度量

算法执行的时间是算法优劣和问题规模的函数。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。考察一个程序的执行时间通常有以下两种方法：

①事后统计的方法。该方法由于是异地、异时、异境，不利于较大范围内的算法比较。

②事前分析估算的方法。见表 1.1。

表 1.1　程序在计算机上运行所需时间的影响因素

算法本身选用的策略	时　　间
问题的规模	规模越大，消耗时间越多
书写程序的语言	语言越高级，消耗时间越多

综上所述，为便于比较算法本身的优劣，应排除其他影响算法效率的因素。

5. 算法的复杂度

算法复杂度包括算法时间复杂度和算法空间复杂度。

算法时间复杂度是指执行算法所需要的计算工作量。算法的工作量可用算法的基本运算次数来量度，而算法所执行的基本运算次数是问题规模的函数，即：算法的工作量＝$f(n)$，其中 n 是问题的规模。人们常需要描述特定算法相对于 n（输入元素的个数）需要做的工作量。算法中基本操作重复执行的次数是问题规模 n 的某个函数，用 $T(n)$ 表示，若有某个辅助函数 $f(n)$，使得当 n 趋近于无穷大时，$T(n)/f(n)$ 的极限值为不等于零的常数，则称 $f(n)$ 是 $T(n)$ 的同数量级函数，记作 $T(n)=O(f(n))$，称 $O(f(n))$ 为算法的渐进时间复杂度，简称时间复杂度。

人们希望能够比较算法的运行时间和空间要求，并使这种比较能与程序设计语言、编译系统、机器结构、处理器的速度及系统的负载等复杂因素无关。

例 1　交换 i 和 j 的内容。

```
Temp=i;
i=j;
j=temp;
```

这三条单个语句的频度均为 1，该程序段的执行时间是一个与问题规模 n 无关的常数。算法的时间复杂度为常数阶，记作 $T(n)=O(1)$。如果算法的执行时间不随着问题规模 n 的增加而增长，即使算法中有上千条语句，其执行时间也不过是一个较大的常数。

此类算法的时间复杂度是 $O(1)$。

例 2

```
sum=0;      (1 次)
for (i=1; i<=n; i++)      (n 次)
for (j=1; j<=n; j++)      (n² 次)
sum++;      (n² 次)
```

解：$T(n)=2n^2+n+1=O(n^2)$

例 3

```
for (i=1; i<n; i++)
{
y=y+1;    ①
for (j=0; j<=(2*n); j++)
x++;    ②
}
```

解：语句①的频度是 $n-1$，语句②的频度是 $(n-1)\times(2n+1)=2n^2-n-1$。$T(n)=2n^2-n-1+(n-1)=2n^2-2$，该程序的时间复杂度 $T(n)=O(n^2)$。

例 4

```
a=0; b=1;    ①
for (i=1; i<=n; i++) ②
{
s=a+b;    ③
b=a;    ④
a=s;    ⑤
}
```

解：语句①的频度是 2，语句②的频度是 n，语句③的频度是 $n-1$，语句④的频度是 $n-1$，语句⑤的频度是 $n-1$，则 $T(n)=2+n+3(n-1)=4n-1=O(n)$。

例 5

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

解：语句①的频度是 1，设语句②的频度是 $f(n)$，则有 $2^{f(n)}\leqslant n$，即 $f(n)\leqslant\log_2 n$，取最大值 $f(n)=\log_2 n$，则该程序的时间复杂度 $T(n)=O(\log_2 n)$。

例 6

```
for (i=0; i<n; i++)
{for (j=0; j<i; j++)
{for (k=0; k<j; k++)
x=x+2;}}
```

解：当 $i=m$，$j=k$ 时，内层循环的次数为 k，当 $i=m$ 时，j 可以取 0，1，…，$m-1$，所以这里最内层循环共进行了 $0+1+\cdots+m-1=(m-1)m/2$ 次。i 从 0 取到 n，

则循环共进行了 $0+(1-1)\times1/2+\cdots+(n-1)n/2=n(n+1)(n-1)/6$次。所以，时间复杂度为 $O(n^3)$。

人们还应该区分算法的最坏情况的行为和期望行为，如快速排序的最坏情况运行时间是 $O(n^2)$，但期望时间是 $O(n\log_2 n)$。通过每次都仔细地选择基准值，人们就可能把平方情况（即 $O(n^2)$ 情况）的概率减小到几乎等于 0。在实际中，精心实现的快速排序一般都能以 $O(n\log_2 n)$ 时间运行。

下面是一些常用的时间复杂度的记法。

访问数组中的元素是常数时间操作，或称 $O(1)$ 操作。一个算法如果能在每个步骤去掉一半数据元素，如二分检索，通常它就取 $O(\log_2 n)$ 时间。用 strcmp 比较两个具有 n 个字符的串需要 $O(n)$ 时间。常规的矩阵乘算法是 $O(n^3)$，因为算出每个元素都需要将 n 对元素相乘并加到一起，所有元素的个数是 n^2。常见的时间复杂度按数量级递增排列依次为：常数 $O(1)$，对数阶 $O(\log_2 n)$，线性阶 $O(n)$，线性对数阶 $O(n\log_2 n)$，平方阶 $O(n^2)$，立方阶 $O(n^3)$，…，k 次方阶 $O(n^k)$，指数阶 $O(2^n)$。显然，时间复杂度为指数阶 $O(2^n)$ 的算法效率极低，当 n 值稍大时就无法应用。

在有些情况下，算法中的基本操作重复执行的次数还随问题的输入数据集的不同而不同。例如在冒泡排序的算法中，当要排序的数组 a 初始序列为自小至大有序时，基本操作的执行次数为 $n-1$；当初始序列为自大至小有序时，基本操作的执行次数为 $n(n-1)/2$。对这类算法的分析，可以采用以下两种方法来分析：

- 平均性态（average behavior）。所谓平均性态，是指利用各种特定输入下的基本运算次数的加权平均值来度量算法的工作量。设 x 是所有可能输入中的某个特定输入，$p(x)$ 是 x 出现的概率（即输入为 x 的概率），$t(x)$ 是算法在输入为 x 时所执行的基本运算次数，则算法的平均性态定义为 $A(n)=\sum_{x\in D_n}p(x)t(x)$，其中 D_n 表示当规模为 n 时，算法执行时所有可能输入的集合。
- 最坏情况复杂性（worst-case complexity）。所谓最坏情况复杂性，是指在规模为 n 时，算法所执行的基本运算的最大次数。

算法空间复杂度是指执行这个算法所需要的内存空间。一个算法的空间复杂度具体是指算法运行从开始到结束所需的存储空间量，算法执行所需的存储空间包括以下两部分：

- 固定部分：主要包括程序代码、常量、简单变量、定长成分的结构变量所占空间。
- 可变部分：算法执行过程中所需的额外空间，与处理数据的大小和规模有关。一般采用压缩存储技术来尽量减少额外空间。

类似于算法的时间复杂度，空间复杂度也可以作为算法所需存储空间的量度。一个算法所占用的存储空间包括算法程序所占的空间、输入的初始数据所占的存储空间以及算法执行中所需要的额外空间，其中额外空间包括算法程序执行过程中的工作单元以及某种数据结构所需要的附加存储空间，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。

在许多实际问题中，为了减少算法所占的存储空间，通常采用压缩存储技术，以便尽量减少不必要的额外空间。

此为试读，需要完整PDF请访问：www.ertongbook.com

1.1.2 数据结构

1. 选择合适的数据结构解决应用问题

(1) 计算机处理问题的分类

①数值计算问题。

在计算机发展初期，人们使用计算机主要是处理数值计算问题。例如线性方程的求解，该类问题涉及的运算对象是简单的整型、实型或布尔型数据。程序设计者的主要精力集中于程序设计的技巧，无需重视数据结构。

②非数值性问题。

随着计算机应用领域的扩大和软、硬件的发展，“非数值性问题”显得越来越重要。据统计，当今处理非数值性问题占用了90%以上的机器时间，这类问题涉及到的数据结构更为复杂，数据元素之间的相互关系一般无法直接用数学方程式加以描述。因此，解决此类问题的关键已不再是分析数学和计算方法，而是要设计出合适的数据结构，才能有效地解决问题。

(2) 非数值问题求解

著名的瑞士计算机科学家沃思（N Wirth）教授曾提出：算法+数据结构=程序。数据结构是指数据的逻辑结构和存储结构。

例如，电话号码查询问题。编一个查询某个城市或单位的私人电话号码的程序，要求对任意给出的一个姓名，若该人有电话号码，则迅速找到其电话号码，否则指出该人没有电话号码。要解此问题，首先构造一张电话号码登记表，表中每个结点存放两个数据项：姓名和电话号码。

要写出好的查找算法，取决于这张表的结构及存储方式。最简单的方式是将表中结点顺序地存储在计算机中，查找时从头开始依次查对姓名，直到找出正确的姓名或是找遍整个表均没有找到为止。这种查找算法对于一个不大的单位或许是可行的，但对于一个有成千上万私人电话的城市就不实用了。若这张表是按姓氏排列的，则可另造一张姓氏索引表，那么查找过程是先在索引表中查对姓氏，然后根据索引表中的地址到电话号码登记表中核查姓名，这样查找登记表时就无需查找其他姓氏的名字了。因此，在这种新的结构上产生的查找算法就更为有效。

2. 基本概念

数据（Data）是对信息的一种符号表示。在计算机科学中，数据是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。

数据元素（Data Element）是数据的基本单位，在计算机程序中通常作为一个整体进行考虑和处理。一个数据元素可由若干个数据项组成，数据项是数据不可分割的最小单位。

数据对象（Data Object）是性质相同的数据元素的集合，是数据的一个子集。

数据结构（Data Structure）是相互之间存在一种或多种特定关系的数据元素的集合。数据结构的形式定义为数据结构是一个二元组，即 Data-Structure=(D, S)，其中，D 是

数据元素的有限集，S 是 D 上关系的有限集。

例如，复数的数据结构定义为 Complex=(C，R)，其中，C 是含两个实数的集合{C1，C2}，分别表示复数的实部和虚部。R={P}，P 是定义在集合上的一种关系{〈C1，C2〉}。

数据之间的相互关系称为逻辑结构，如图 1.1 所示，通常分为以下四类基本结构：

- 集合结构中的数据元素除了同属于一种类型外，别无其他关系。
- 线性结构中的数据元素之间存在一对一的关系。
- 树型结构中的数据元素之间存在一对多的关系。
- 图形结构或网状结构中的数据元素之间存在多对多的关系。

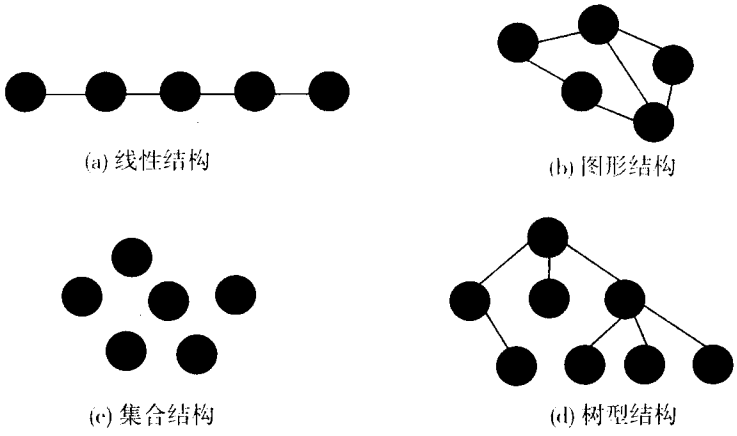

图 1.1　数据的逻辑结构

数据结构在计算机中的表示称为数据的物理结构，又称为存储结构，即数据的物理结构是数据的逻辑结构在存储器里的实现，分为顺序存储、链接存储、索引存储、散列存储四种方式。其中，顺序存储结构是用数据元素在存储器中的相对位置来表示数据元素之间的逻辑关系；链接存储结构是在每一个数据元素中增加一个存放地址的指针，用此指针来表示数据元素之间的逻辑关系。

数据结构就是研究数据的逻辑结构和物理结构以及它们之间的相互关系，并对这种结构定义相应的运算，而且确保经过这些运算后所得到的新结构仍然是原来的结构类型。

3. 数据的逻辑结构

(1) 线性结构

其特征是该结构有且只有一个开始元素和终点元素，所有元素只有一个直接前驱和一个直接后继。例如线性表，它是由 $n(n\geqslant 0)$ 个相同类型的数据元素 a_1，a_2，…，a_n 所构成的有限序列，其中数据元素在不同的情况下可以有完全不同的含义。常见的线性结构包括线性表、堆栈、队列、数组、串。

(2) 非线性结构

其特征是该结构中一个数据元素可能有多个直接前驱和直接后继。

4. 线性表及其顺序存储结构

(1) 线性表 (Linear List) 的定义

由 $n(n\geqslant 0)$ 个数据元素 (结点) a_1, a_2, …, a_n 组成的有限序列，其中数据元素的个数 n 定义为表的长度。当 $n=0$ 时称为空表，常常将非空的线性表 ($n>0$) 记作：(a_1, a_2, …, a_n)，这里的数据元素 a_i ($1\leqslant i\leqslant n$) 只是一个抽象的符号，其具体含义在不同的情况下可以不同。

例如，26 个英文字母组成的字母表 (A, B, C, …, Z)。又如，某校从 2000 年到 2005 年各种型号的计算机拥有数量的变化情况：(160, 270, 480, 800, 1200, 2200)。

(2) 线性表的逻辑特征

- 在非空的线性表，有且仅有一个开始结点 a_1，它没有直接前驱，而仅有一个直接后继 a_2。
- 有且仅有一个终端结点 a_n，它没有直接后继，而仅有一个直接前驱 a_{n-1}。
- 其余的内部结点 a_i ($2\leqslant i\leqslant n-1$) 都有且仅有一个直接前驱 a_{i-1} 和一个直接后继 a_{i+1}。
- 线性表是一种典型的线性结构。

(3) 线性表的顺序存储结构

把线性表的结点按逻辑顺序依次存放在一组地址连续的存储单元里，用这种方法存储的线性表简称顺序表。假设线性表的每个元素需占用 l 个存储单元，并以所占的第一个单元的存储地址作为数据元素的存储位置，则线性表中第 $i+1$ 个数据元素的存储位置 $LOC(a_{i+1})$ 和第 i 个数据元素的存储位置 $LOC(a_i)$ 之间满足下列关系：

$$LOC(a_{i+1}) = LOC(a_i) + l$$

线性表的第 i 个数据元素 a_i 的存储位置为：$LOC(a_i)=LOC(a_1)+(i-1)*l$。

(4) 插入运算

线性表的插入运算是指在表的第 i ($1\leqslant i\leqslant n+1$) 个位置上，插入一个新结点 x，使长度为 n 的线性表 (a_1, …, a_{i-1}, a_i, …, a_n) 变成长度为 $n+1$ 的线性表 (a_1, …, a_{i-1}, x, a_i, …, a_n)。在顺序表上做插入运算，平均要移动表上一半结点。这里的问题规模是表的长度，设它的值为 n。该算法的时间主要花费在循环的结点后移语句上，该语句的执行次数 (即移动结点的次数) 是 $n-i+1$。由此可看出，所需移动结点的次数不仅依赖于表的长度，而且还与插入位置有关。当 $i=n+1$ 时，由于循环变量的终值大于初值，结点后移语句将不进行，这是最好情况，其时间复杂度 $O(1)$；当 $i=1$ 时，结点后移语句将循环执行 n 次，需移动表中所有结点，这是最坏情况，其时间复杂度为 $O(n)$。当表长 n 较大时，算法的效率相当低。

由于插入可能在表中任何位置上进行，因此需分析算法的平均复杂度。在长度为 n 的线性表中第 i 个位置上插入一个结点，令 $E_{is}(n)$ 表示移动结点的期望值 (即移动的平均次数)，则在第 i 个位置上插入一个结点的移动次数为 $n-i+1$。故 $E_{is}(n) = \sum_{i=1}^{n+1} p_i(n-i+1)$，不失一般性，假设在表中任何位置 ($1\leqslant i\leqslant n+1$) 上插入结点的机会是均等

的，则 $p_1=p_2=p_3=\cdots=p_{n+1}=\frac{1}{(n+1)}$，因此，在等概率插入的情况下，$E_{is}(n)=\sum_{i=1}^{n+1}\frac{1}{n+1}(n-i+1)=\frac{n}{2}$。虽然 $E_{is}(n)$ 中 n 的系数较小，但就数量级而言，它仍然是线性阶的。因此，算法的平均时间复杂度为 $O(n)$。

(5) 删除运算

线性表的删除运算是指将表的第 i ($1\leqslant i\leqslant n$) 个结点删除，使长度为 n 的线性表 (a_1，…，a_{i-1}，a_i，a_{i+1}，…，a_n) 变成长度为 $n-1$ 的线性表 (a_1，…，a_{i-1}，a_{i+1}，…，a_n)。该算法的时间分析与插入算法相似，结点的移动次数也是由表长 n 和位置 i 决定。若 $i=n$，则由于循环变量的初值大于终值，前移语句将不执行，无需移动结点；若 $i=1$，则前移语句将循环执行 $n-1$ 次，需移动表中除开始结点外的所有结点。这两种情况下算法的时间复杂度分别为 $O(1)$ 和 $O(n)$。

删除算法的平均性能分析与插入算法相似。在长度为 n 的线性表中删除一个结点，令 $E_{de}(n)$ 表示所需移动结点的平均次数，删除表中第 i 个结点的移动次数为 $n-i$，故 $E_{de}(n)=\sum_{i=1}^{n}p_i(n-i)$，式中，$p_i$ 表示删除表中第 i 个结点的概率。在等概率的假设下，$p_1=p_2=p_3=\cdots=p_n=\frac{1}{n}$。由此可得 $E_{de}(n)=\sum_{i=1}^{n}\frac{1}{n}(n-i)=\frac{n-1}{2}$。即在顺序表上做删除运算，平均要移动表中约一半的结点，平均时间复杂度也是 $O(n)$。

5. 栈和队列

(1) 栈

①栈的定义。

栈 (Stack) 是限制在表的一端进行插入和删除运算的线性表，通常称插入、删除的这一端为栈顶 (Top)，另一端为栈底 (Bottom)。当表中没有元素时，称为空栈。

假设栈 S=(a_1，a_2，a_3，…，a_n)，则 a_1 称为栈底元素，a_n 称为栈顶元素。栈中元素按 a_1，a_2，a_3，…，a_n 的次序进栈，退栈的第一个元素应为栈顶元素。换句话说，栈的修改是按后进先出的原则进行的，因此，栈称为后进先出表 (Last In First Out，LIFO) 或先进后出表 (First In Last Out，FILO)。栈的结构如图 1.2 所示。

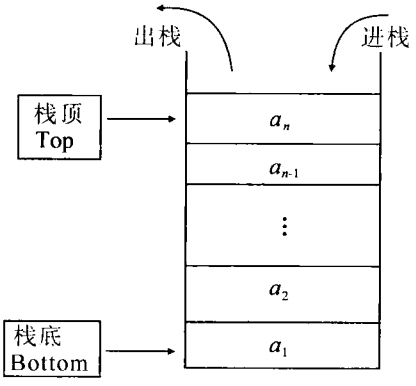

图 1.2　栈结构

②栈顺序存储。

由于栈是运算受限的线性表，因此线性表的存储结构对栈也适应。

栈的顺序存储结构简称顺序栈，它是运算受限的线性表，因此，可用数组来实现顺序栈。因为栈底位置是固定不变的，所以可以将栈底位置设置在数组的两端的任何一个端点；栈顶位置是随着进栈和退栈操作而变化的，故需用一个整型变量 top 来指示当前栈顶

的位置，通常称 top 为栈顶指针。因此，顺序栈的类型定义只需将顺序表的类型定义中的长度属性改为 top 即可。在程序设计语言中，用一维数组 S(1：m) 作为顺序栈的存储空间，其中 m 为最大容量。

顺序栈的类型定义如下：

```
#define StackSize 100
typedef char datatype;
typedef struct {
  datatype data [stacksize];
  int top;
} seqstack;
```

设 S 是 SeqStack 类型的指针变量。若栈底位置在向量的低端，即 S->data [0] 是栈底元素，那么栈顶指针 S->top 是正向增加的，即进栈时需将 S->top+1，退栈时需将 S->top-1。因此，S->top<0 表示空栈，S->top=stacksize-1 表示栈满。当栈满时再做进栈运算必定产生空间溢出，简称“上溢”；当栈空时再做退栈运算也将产生溢出，简称“下溢”。上溢是一种出错状态，应该设法避免；下溢则可能是正常现象，因为栈在程序中使用时，其初态或终态都是空栈，所以下溢常常用来作为程序控制转移的条件。

③栈的运算。

- 进栈运算：是指在栈顶位置插入一个新元素。该运算的基本操作：先判栈满（栈指针指向栈空间的最后一个位置），若栈没有上溢，再将栈顶指针（top）加 1，新元素插入到栈顶指针指向的新位置。
- 出栈运算：是指在栈顶位置取出一个元素。该运算的基本操作：先判栈空（栈指针为 0），若栈没有下溢，从当前栈顶指针位置取出元素赋给变量，再将栈指针（top）减 1。
- 读栈顶元素：是指在栈顶位置将一个元素赋给变量，不删除栈内元素。该运算的基本操作：先判栈空（栈指针为 0），若栈没有下溢，从当前栈顶指针位置取出元素赋给变量，栈指针不变。

（2）队列

①队列的定义。

队列（Queue）也是一种运算受限的线性表。它只允许在表的一端进行插入，而在另一端进行删除。允许删除的一端称为队头（front），允许插入的一端称为队尾（rear）。

例如，排队购物，操作系统中的作业排队。先进入队列的成员总是先离开队列，因此队列亦称为先进先出（First In First Out）的线性表，简称 FIFO 表。队列的结构如图 1.3 所示。

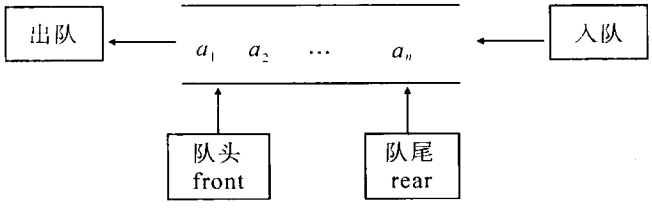

图 1.3 队列结构

当队列中没有元素时称为空队列。在空队列中，依次加入元素 a_1，a_2，…，a_n 之后，a_1 是队头元素，a_n 是队尾元素。显然退出队列的次序也只能是 a_1，a_2，…，a_n，也就是说，队列的修改是依先进先出的原则进行的。

②循环队列——队列的顺序表示和实现。

队列的顺序存储结构称为顺序队列，顺序队列实际上是运算受限的顺序表，和顺序表一样，顺序队列也是必须用一个向量空间来存放当前队列中的元素。由于队列的队头和队尾的位置是变化的，因而要设两个指针分别指示队头和队尾元素在队列中的位置，它们的初始值在队列初始化时均应置为 0。

入队时将新元素插入所指的位置，然后队尾加 1。出队时，删去所指的元素，然后队头加 1 并返回被删元素。由此可见，当头尾指针相等时，队列为空。在非空队列里，头指针始终指向队头元素，而尾指针始终指向队尾元素的下一位置。与栈类似，队列中亦有上溢和下溢现象，此外，顺序队列中还存在“假上溢”现象。因为在入队和出队的操作中，头尾指针只增加不减小，致使被删除元素的空间永远无法重新利用。因此，尽管队列中实际的元素个数远远小于向量空间的规模，但也可能由于尾指针已超出向量空间的上界而不能做入队操作。该现象称为假上溢。

为充分利用向量空间，克服上述的假上溢现象，可将向量空间想象为一个首尾相接的圆环，并称这种向量为循环向量，存储在其中的队列称为循环队列（Circular Queue）。循环队列的结构如图 1.4 所示。在循环队列中进行出队、入队操作时，头尾指针仍要加 1，朝前移动。只不过当头尾指针指向向量上界（QueueSize－1）时，其加 1 操作的结果是指向向量的下界 0。

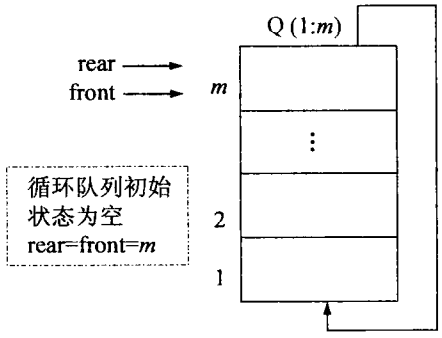

图 1.4　循环队列结构

显然，因为循环队列元素的空间可以被利用，除非向量空间真的被队列元素全部占用，否则不会上溢。因此，除一些简单的应用外，真正实用的顺序队列是循环队列。

由于入队时尾指针向前追赶头指针，出队时头指针向前追赶尾指针，故队空和队满时头尾指针均相等。因此，我们无法通过 front＝rear 来判断队列是“空”还是“满”。

解决此问题的方法有三种：一是另设一个布尔变量，以判别队列的空和满；二是少用一个元素的空间，约定入队前，测试尾指针在循环意义下加 1 后是否等于头指针，若相等，则认为队满（注意：rear 所指的单元始终为空）；三是使用一个计数器记录队列中元素的总数（实际上是队列长度）。

③队列运算。

为了算法设计方便，我们约定：空队列时，front＝rear＝0；插入新的数据元素时，rear＝rear＋1；队头元素出队列时，front＝front＋1；在非空队列中，头指示器 front 总是指向队列中实际队头元素的前面位置，而尾指示器 rear 总是指向队尾元素。

循环队列是队列的存储空间首尾相连，这样，当队列的最后一个单元已占用时（rear＝MAXSIZE），只要第 1 个单元是空，就可以加入入队结点元素。

- 入队运算：在循环队列的队尾加入一个新元素。入队运算的基本操作：先将队尾指

此为试读，需要完整PDF请访问：www.ertongbook.com