计算机技术及应用基础

（修订版）

王同胜　　主编
张连芳　王　平　编著
赵　政　赵文钦

天津大学出版社

内容提要

全书共分4章，包括数据结构、数据库、计算机网络和软件工程等。各章相对独立，又相互联系。书中以计算机技术基础内容为主，同时介绍了最新技术。

本书除作为非计算机类理工科硕士研究生教材外，还可作为计算机专业的学生、各类科技人员、企业管理人员和教师的参考书，也可作为非计算机专业本科生高年级的选修课教材。

图书在版编目（CIP）数据

计算机技术及应用基础/王同胜主编；张连芳编著. —天津：天津大学出版社. 1999. 8

ISBN 7-5618-1216-7

Ⅰ.计… Ⅱ.①王… ②张… Ⅲ.电子计算机-高等学校-教材 Ⅳ.TP3

中国版本图书馆CIP数据核字（1999）第34960号

出版发行	天津大学出版社
出 版 人	杨风和
地　　址	天津市卫津路92号天津大学内(邮编:300072)
电　　话	发行部:022-27403647　邮购部:022-27402742
印　　刷	天津市宝坻县第二印刷厂
经　　销	全国各地新华书店
开　　本	787mm×1092mm　1/16
印　　张	17.75
字　　数	444千
版　　次	1999年8月第1版
印　　次	2001年9月第2次
印　　数	5 001—9 000
定　　价	19.00元

前　言

当前，以计算机和信息技术为核心的知识经济时代正在崛起，一场世界范围的信息革命方兴未艾。为适应社会信息化建设的需要，特编写本书作为非计算机专业类理工科硕士研究生的学位课程教材。期望研究生通过本书的学习能进一步扩大知识领域、优化知识结构、强化高层次人才的逻辑思维和工程设计能力，并在计算机和计算机网络日益成为人们工作平台、计算机应用日益普及的时代，为提高计算机应用能力打下初步基础。

1992～1998年，我校曾使用了《计算机技术与工程应用基础》（以下简称原书）一书作为非计算机专业类硕士研究生学位课教材，并起到了很好的作用。但时过多年，情况发生了很大变化。第一，90年代以来，计算机技术发展日新月异，特别是Internet /Intranet的飞速发展，以数据库为核心的信息系统的蓬勃发展和建设，对人们的计算机基础知识和应用能力提出了更高要求。第二，随着各校本科生计算机基础的加强，研究生入学时的计算机基础和能力已经有了很大提高。他们多数已具备了必要的计算机基础知识和一定的编程能力。我们在总结原书经验的基础上，重写一本新书，以满足当前研究生人才培养的需要。

本书共分4章。各章内容既相互独立，又相互联系。考虑到现在入学时的研究生已具有了一定的计算机基础，故本书除介绍了算法与数据结构作为软件基础外，主要写了面向应用的数据库、计算机网络和软件工程等内容，仍以基础为主，同时介绍了数据库、计算机网络的最新技术和面向对象的编程、分析和设计的最新思想。文字上力求深入浅出，全书图文并茂，每章后面附有习题，便于自学。

本书编写具体分工如下：第1章由张连芳副教授和王平老师编写；第2章由赵政副教授编写；第3章由王同胜教授编写；第4章由赵文钦教授和张钢副教授编写。全书由王同胜教授统稿和定稿。在本书编写过程中，得到了研究生院的资助。研究生院的孙学珠教授、滕建辅教授，计算机系的张新荣教授、冯尧锴老师等都给予了许多指导和帮助。在此一并表示感谢。

由于时间仓促和作者水平所限，书中难免有许多不足和不妥之处，敬请各位读者和专家批评指正。

作者

1999.3

目　录

第1章 数 据 结 构

1.1 导言

数据结构和算法是计算机软件的基础。计算机科学各领域及应用都离不开数据结构和算法。

以 Wegnor 为代表的一些人，认为计算机科学是“包含一种关于计算图式的新的思想方法”。他认为在计算机科学中具有核心作用的概念是“信息结构的转换”。计算机科学就是“一种关于信息结构转换的科学”。而以 Knuth 为代表的另一种观点认为“计算机科学是算法的学问。算法是精确定义的一系列规则，指出怎样从给定的输入信息经过有限步骤产生所求的输出信息。‘算法’的特殊表示称为‘程序’，如同用‘数据’作为‘信息’的特殊表示一样。”到底应该强调“信息结构”（或“数据结构”）还是应该强调“算法”（或“程序”）不是我们研究的目的，但无疑“信息结构”和“算法”应该是计算机科学研究的基本课题。而且，应该强调指出，数据结构和算法之间存在本质的联系，失去一方，另一方就不再有意义。当论及一种类型的数据结构时，总是离不开对这种类型数据结构需要施加的各种运算（又称操作、转换），例如查找、插入、删除和修改等，而且只有通过对这些运算算法的研究，才能更深刻地理解这种数据结构的意义与作用。反之，当论及一种算法时，也总是自然地联系到作为该算法处理的对象和结果的数据结构。

1.1.1 什么是数据结构

数据结构首先是指数据间的关系，其次包括数据在存储介质上的存储方式和在这类数据结构上的运算。所以数据结构概念一般包括数据间的逻辑关系（常称逻辑结构）、数据存储方式（常称存储结构或物理结构）和数据运算三个方面。

1.1.2 数据的逻辑结构

描述数据关系用数据的逻辑结构。它可以用二元组

$$B=(K,R)$$

表示。其中 K 是结点的有穷集合；R 是 K 上的关系的有穷集合。在下文的叙述中，凡不致引起混淆时，常简称数据的逻辑结构为数据结构。

一、结点的类型

现在从结点的值的组成方式及数据类型的角度讨论结点的分类。设 K 是结点的有穷集合，且每个结点 $k(k\in K)$ 的值（即数据）可以由 n 个字段组成，第 i 个字段记为 W_{ik}，所以 k 可以表示为

$$k=(W_{1k},W_{2k},\cdots,W_{nk}) \quad (n\geqslant 1)$$

其中，每个 W_{ik} 是一个组合项。它可以由一个或多个初等项和组合项组成，也可以就是一个初等项。

结点可分为两大类，即初等类型和组合类型。只包含一个初等项的结点属于初等类型，否则是组合类型。有以下五种基本的初等类型。

1）整数类型（integer） 取值为计算机可以表示的整数。

2）实数类型（real） 取值为计算机可以表示的实数。

3）布尔类型（boolean） 取值为真（true）或假（false）。

4）字符类型（char） 取值为计算机可以表示的字符集元素。

5）指针类型（pointer） 取值为计算机存储介质上的一个地址或相对地址。指向结点的指针是指向该结点的第一个单元的地址或者指向该结点在结构中的相对位置。当指针值存入计算机时，形式上与整数相似，但由于指针的含义是代表地址，通常不能用来作算术运算，也不能取负值。

组合类型是由初等类型的某种方式组合而成的。不同的组合方式形成不同的类型，例如数组类型、记录类型等。

二、结构的分类

设$B=(K,R)$是一个逻辑结构，r是一个K到K的关系，$r\in R$。若k、$k'\in K$，且$\langle k, k'\rangle \in r$，则称$k'$是$k$的后继，$k$是$k'$的前驱。这时$k$和$k'$是相邻的结点（都是对$r$而言）。如果不存在一个$k'$，使$\langle k, k'\rangle \in r$，则称$k$为关于$r$的终端结点（即无后继的结点）。如果不存在$k'$，使$\langle k', k\rangle \in r$，则称$k$为关于$r$的开始结点（即无前驱的结点）。如果$k$既不是终端结点又不是开始结点，则称$k$是内部结点。

本章一般只讨论包含一个关系r的关系集合R，即$R=\{r\}$。对于R中包含多个关系的情形，可以用类似的方法讨论。

数据结构分为线性结构和非线性结构。在线性结构里（如果结点集合不空）有且仅有一个终端结点和一个开始结点，并且所有的结点最多有一个前驱和一个后继。线性表就是典型的线性结构。在线性表的逻辑结构$B=(K,R)$中，若K是包含n个结点$\{k_1,k_2,\cdots,k_n\}$（即表中有n个表目）的集合，$R=\{r\}$，则关系r是由有序对$\langle k_{i-1}, k_i\rangle$构成的集合，即

$$r=\{\langle k_{i-1},k_i\rangle \mid k_i\in K, 1<i\leqslant n\}$$

也即K里所有的结点都可以按r排成一个序列k_1、k_2、……、k_n。其中k_1为开始结点，k_n为终端结点。非线性结构中最重要的是树结构，在树的逻辑结构$B=(K,R)$中，K是包含n个结点的集合，$R=\{r\}$，而r满足下列条件：①有且仅有一个称为根的结点无前驱；②其它结点有且仅有一个前驱；③每个非根结点都存在着一条从根到该结点的路径（若根为w，该结点为k，则存在一个结点序列k_0、k_1、……、k_s，其中$k_0=w$，$k_s=k$，使$\langle k_{i-1}, k_i\rangle \in r$，$1\leqslant i\leqslant s$）。最一般的情形是图结构，在图的逻辑结构$B=(K,R)$中，$K$是包含$n$个结点的集合，对于$K$中结点相对于关系$r(r\in R)$的前驱和后继的个数都不作限制。

三、结点和结构

结点和结构是相对而言的，这有些类似于元素和集合的关系。不严格地说，结构是由若干结点及其相互之间的关系所组成。在某种场合，当需要研究某结点的内部组成情况时，就把该结点看作一个结构，而把组成该结点的数据项看作结点，把数据项组成该结点的组成方式看作结点间的关系。所以，实际上一个结点常常是由若干个初等项按某种规则组合起来

的。这种组合方式可以是线性的，也可以是非线性的。例如，当关心数组和数组的关系时，可以把数组定义为结点；而当关心数组内各元素之间的关系时，又可以把数组元素看成结点，而数组就成了所讨论的数据结构。在另外一些情况下，可以把一个二维数组看成是向量的向量，从而简化关系，这时向量就成了结构中的结点。

由于数据结构讨论的基本单位是结点，所以很多情况下并不重视结点的构成，而是把结点抽象地看作一个整体，重点讨论结点之间的关系。

在本章以后的叙述中，当需要描述结点的组成时，将采用类似 PASCAL 语言中的类型说明方法。

例 1　某系学生成绩登记表中每个学生的数据是一个结点。它由学号、姓名、专业、考试成绩四个域（亦称字段）组成。其中考试成绩是一个组合项，它又包括数学、物理、程序设计、政治和英语五个成绩的初等项。采用 PASCAL 的类型说明表示如下：

```
TYPE score = 0..100;
     exam = ARRAY [1..5] OF score;
     student = RECORD
                 no: integer;
                 name: string;
                 specialty: string;
                 c: exam
                 END
```

其中 score、exam、student 是类型标识符。score 称为子界类型，取值是从 0 到 100 的整数，用 0..100 来说明。exam 称为数组类型，由五个类型为 score 的元素组成，数组元素的下标是 1 到 5 之间的整数。student 称为记录类型（在本例中它对应一个结点），由四个域组成。no、name、specialty 和 c 是域标识符，分别表示学号、姓名、专业和考试成绩。类型说明不仅描述了结点的组成方式，同时还给出了组成结点的各个域的名称（域标识符）。域的类型也在类型说明中描述。integer 表示整数，string 表示字符串。务须注意，类型说明仅给出了一类变量的取值范围，并没有指出哪些变量属于该类型，更没有给出变量的具体值。只有当用变量说明把某变量的标识符与该类型标识符联系在一起时，才确定变量的类型，而通过给变量赋值才能使变量定值。例如，在例 1 中加入变量说明

```
VAR A1, A2, A3: Student;
```

就定义了三个变量 A_1、A_2、A_3。它们都具有类型说明中 student 的类型。当执行赋值语句

```
A1·name←'ZHANG MIN'
A2·c [3] ←95
```

以后，就给变量 A_1 的 name 字段赋值为 ZHANG MIN，给 A_2 的 c 字段的下标为 3 的初等项赋值为 95。

为了直观，有时亦用图示法表示一个逻辑结构。表示方法是每一个结点 k 用一个长方形框表示，里面写上 k。当关心 k 的值时，可以按字段顺序标出字段值。长方形框用有向线段联系，用来表示结点间的前驱；后继关系，对同一结点集合上的不同关系，在图上用不同

的线型区分。

例 2 一批数据的逻辑结构为 $B=(K,R)$，其中

$K=\{k_1,k_2,\cdots,k_9\}$

$R=\{r\}$

$r=\{\langle k_1,k_2\rangle,\langle k_1,k_3\rangle,\langle k_1,k_4\rangle,\langle k_1,k_7\rangle,\langle k_1,k_8\rangle,\langle k_4,k_5\rangle,\langle k_4,k_6\rangle,\langle k_8,k_9\rangle\}$

可用图 1－1－1（a）表示。

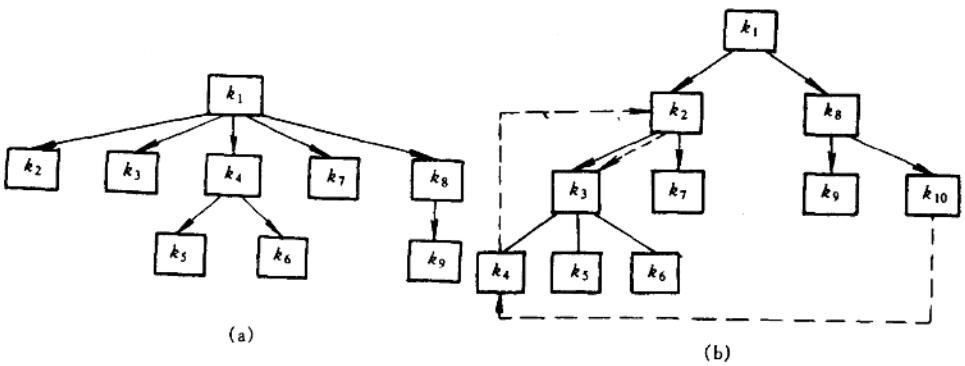

图 1－1－1 逻辑结构表示法

（a）含单个关系的逻辑结构图；（b）含多个关系的逻辑结构图

例 3 一批数据的逻辑结构为$B=(K,R)$，其中

$K=\{k_1,k_2,\cdots,k_{10}\}$

$R=\{r_1,r_2\}$

$r_1=\{\langle k_1,k_2\rangle,\langle k_1,k_8\rangle,\langle k_2,k_3\rangle,\langle k_2,k_7\rangle,\langle k_3,k_4\rangle,\langle k_3,k_5\rangle,\langle k_3,k_6\rangle,\langle k_8,k_9\rangle$
$\langle k_8,k_{10}\rangle\}$

$r_2=\{\langle k_{10},k_4\rangle,\langle k_4,k_2\rangle,\langle k_2,k_3\rangle\}$

可用图 1－1－1（b）表示。其中实线有向线段表示 r_1，虚线有向线段表示 r_2。

1.1.3 数据的存储结构

数据的逻辑结构是从逻辑关系上观察数据，它与数据的存储无关，是独立于计算机的。数据的存储结构是逻辑结构在计算机存储器里的实现，它依赖于计算机。

计算机的存储器（主存）由有限多个存储单元组成，每个存储单元有唯一的地址，各存储单元的地址是连续编码的。每个存储单元 Z 都有唯一的后继单元 $Z'=\mathrm{suc}(Z)$。Z 和 Z'常称为相邻单元。一片相邻的存储单元叫做存储区域，记作 M。在不产生混淆的情况下，Z 可以用来表示一个存储单元，也可以用来表示该存储单元的地址。地址 $Z'=\mathrm{suc}(Z)=Z+1$。

设有逻辑结构 $B=(K,R)$。要把 B 存储在计算机中，首先必须建立一个从 K 的结点到 M 区域的映象 $S:K\to M$，即对于每一个 $k\in K$ 都有唯一的 $Z\in M$ 使 $S(k)=Z$。Z 为结点 k 所占存储空间的起始单元。同时这个映象应具有明显或隐含体现关系 R 的能力。

由于计算机的存储空间是有限的，所以如何合理地使用它，使有限的存储空间发挥最大的作用，是存储管理问题，亦可用数据结构所论及的方法解决。

通常用 LOC(k)表示结点 k 对应的存储单元的地址。有两种基本的存储映象方法，下

面分别介绍。

一、顺序方法

这种方法主要用于线性的数据结构。它把逻辑上相邻的结点存储在物理上相邻的存储单元里，结点之间的关系由存储单元的邻接关系体现。如果结点 k 所占存储空间的第一个单元为 Z，即 LOC(k)=Z，而最后一个单元为 Z_1，那么 Z_1 的后继单元 $Z'=\mathrm{suc}(Z_1)$就是 k 的后继k'的第一个存储单元。

例 4　$B=(K,R)$, $K=\{k_1,k_2,k_3,k_4,k_5\}$, $R=\{r\}$,

$r=\{\langle k_1,k_2\rangle,\langle k_2,k_3\rangle,\langle k_3,k_4\rangle,\langle k_4,k_5\rangle\}$

假定每个结点占一个存储单元，结点 k_1 存放在 200 号单元中，则顺序存储实现如图 1－1－2 所示。

二、链接方法

这种方法是给结点附加上指针字段，即将结点所占的存储单元分作两部分，一部分存放结点本身的信息，称数据项；另一部分存放此结点的后继结点对应的存储单元的地址，称指针项。指针项可以包括一个或多个指针，以指向结点的一个或多个后继，或记录其它信息。

当用 info 表示结点的数据项、用 link 表示结点的指针项时，如果 k'是 k 的后继结点，就有

$$\mathrm{LOC}(k')=k\cdot\mathrm{link}$$

前面例 4 的逻辑结构可以用链接的方法存储，如图 1－1－3 所示。

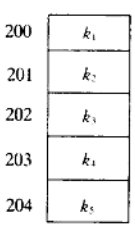

图 1－1－2　顺序存储的线性数据结构

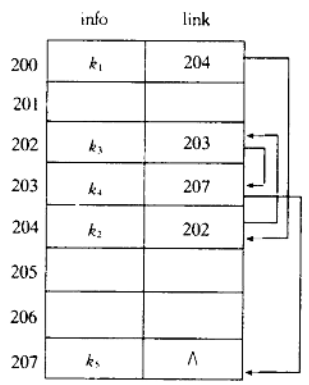

图 1－1－3　链接存储的线性数据结构

其中记号 Λ 表示空指针，即该指针不具有有意义的值（不表示任何具体结点的单元地址）。

例 5　逻辑结构为 $B=(K,R)$，其中

$K=\{k_1,k_2,k_3,k_4,k_5,k_6\}$, $R=\{r\}$,

$r=\{\langle k_1,k_2\rangle,\langle k_1,k_3\rangle,\langle k_2,k_4\rangle,\langle k_2,k_5\rangle,\langle k_3,k_6\rangle\}$

用链接的方式实现这个逻辑结构。因为有些结点有两个后继，所以指针项需包括两个指针，分别指向两个后继。本例的逻辑结构图示与链接存储表示在图 1－1－4 中。

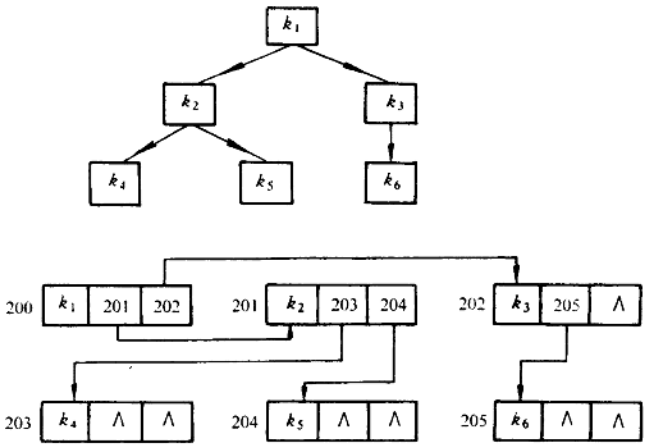

图 1-1-4 一个非线性结构的图示和存储表示

1.2 线性结构

本节讨论顺序表、链表、内排序和线性表上的检索。线性结构是数据组织中最常用最基本的结构形式，是其它结构的基础。

1.2.1 顺序表

线性表的逻辑结构 $B=(k,R)$ 中 $K=\{k_1,k_2,\cdots,k_n\}(n\geqslant 0)$；$R=\{r\}$，$r=\{\langle k_i,k_{i+1}\rangle \mid k_i\in K,k_{i+1}\in K,1\leqslant i<n\}$。这样的集合，将所有结点按先后顺序排成一个线性序列：

$$k_1,\ k_2,\ k_3,\ \cdots,\ k_n$$

其中，k_1 是开始结点，k_n 是终端结点，k_i 是 k_{i+1} 的前驱，而 k_{i+1} 是 k_i 的后继。对所有的 $i=1$、2、……、$n-1$，上述关系均成立。常将线性表记作：

$$(k_1,\ k_2,\ \cdots,\ k_i,\ \cdots,\ k_n)\qquad (n\geqslant 0)$$

其中 k_i（$i=1,\ \cdots,\ n$）称为表目，i 称为该表目的索引。

线性表在计算机内有许多存储方法，其中简单又自然的是顺序法：把表目按其索引值从小到大一个接一个地存放在一片相邻的单元里。用顺序方法存储的线性表简称“顺序表”。

在本小节内将讨论几种最简单又常用的顺序表——向量、栈和队列。

1.2.1.1 向量

具有同一类型结点（表目）的线性表，谓之向量。向量中的表目又称“元素”。元素的索引又称为“下标”。

一、向量的运算

向量运算有如下几种：

① 取出向量中第 i 个元素；

② 修改向量中第 i 个元素；

③ 在向量第 i 个元素后面插入一个新元素；

④ 删除第 i 个元素；

⑤ 按某种要求重排向量中各元素的顺序；

⑥ 在向量中查找满足某条件的元素。

最常用的存储向量的方法是顺序存储。由于向量元素具有同一类型，故每个元素占用的存储空间相同，令其为 1。假设向量的第一个元素(k_1)的位置为 $LOC(k_1)$，则元素 k_i 的存放位置为：

$$LOC(k_i) = LOC(k_1) + (i-1) * 1$$

为了讨论简单，假设向量中元素个数 n 不超过某个整数 n_0，并假定向量中元素的类型是某种已定义的类型 datatype。于是，向量 k 可说明如下：

```
TYPE vtype = ARRAY [1..n0] OF datatype;
VAR k: vtype;
```

在向量 k 中，k [1]、k [2]、……、k [n] 存放向量中各元素，而 k [$n+1$]、……、k [n_0] 作为备用的结点空间，以备插入新的元素时使用。

拟在由 n 个元素组成的向量中，在第 i 个 ($0<i\leqslant n$) 元素后面插入一个值为 x 的元素（图 1-2-1）。当 $i=0$ 时表示将 x 作为向量的第一个元素。其中 n 和 x 等变量的说明如下：

```
VAR x: datatype;
    n: 1..n0;
    j: 0..n;
```

插入过程用算法 1-1 实现。

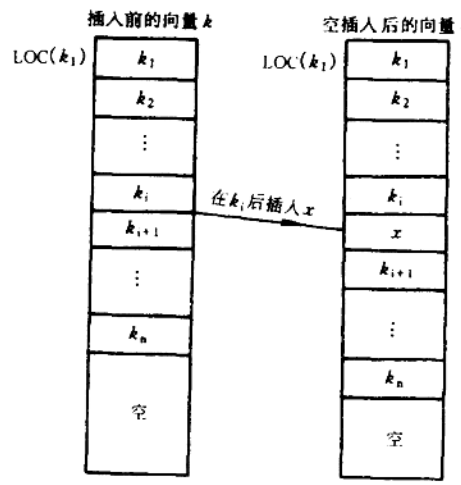

图 1-2-1　在向量 k 中插入 x 示意图

算法 1-1　向量插入

(1) [检查参数]

① 若 i>n，则 print ('error')；算法结束

② 若 n=n0，则 print ('overflow')；算法结束

(2) [后移]

循环，j 以 -1 为步长，从 n 到 i+1，执行

k [j+1] ←k [j]

(3) [插入]

k [i+1] ←x；n←n+1

(4) [算法结束]

二、目录表

如果线性表的表目不是同一种类型，所占空间可能不等，则每个表目的地址难以计算。为了提高运算速度，可采用索引方法，即建立目录表。目录表的每一表目为一个指针，指向对应线性表表目的存储空间地址。这样，目录表是一个指针向量，可以通过它快速地找到任

一线性表表目的地址。有了目录表后，线性表本身如何存储已经无关紧要，只要求目录表是顺序存放即可。插入或删除时只要移动目录表中的表目，而不必再移动线性表本身。

更一般的目录表可以包含两个字段，除指针外，另一个字段放置线性表表目的关键字(Key)。于是，可以对目录表按关键字值进行排序和检索运算。这种改进实际上并不增加额外开销，因为此时线性表中的关键字字段已被删去。被删去关键字字段的线性表称为“属性表”。

基于以上的讨论，均可假定线性表所有表目具有同一种类型。图 1－2－2 给出一个线性表建立目录表后的存储方式，其中目录表和线性表都是顺序存储的；图 1－2－3 给出一般的目录表和属性表的存储方式，其中目录表采用顺序存储，而属性表的存储是任意的。

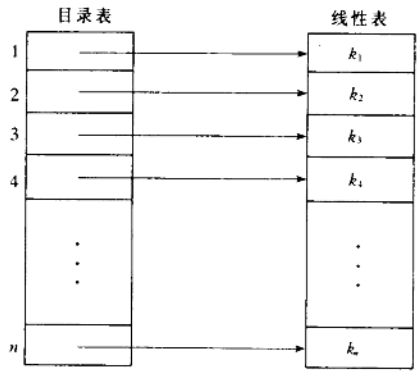

图 1－2－2 简单的目录表与线性表

1.2.1.2 栈

只准许在同一端进行插入与删除的线性表叫栈。允许插入、删除的一端叫做“栈顶”，另一端叫做“栈底”。

按动态插入、删除的规律，常把“栈”称为“后进先出表”（LIFO 表）或“下推表”。

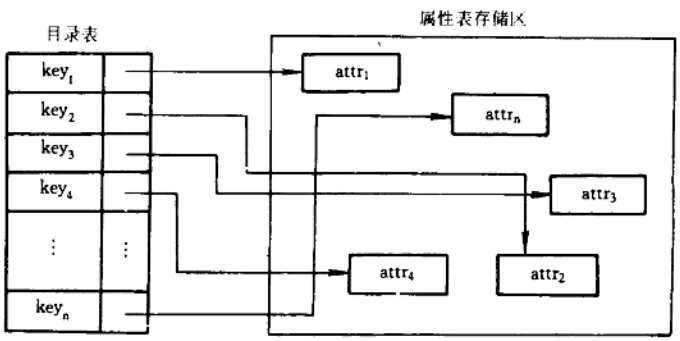

图 1－2－3 目录表和属性表

常用顺序方式存储栈。分配一块连续的存储区放置表目，并用一个变量指向当前的栈顶(图 1－2－4)。假设栈中表目数最大不超过整数 m_0，并假设所有表目具有同一类型 datatype，则可用下列方式定义栈类型 stack 和栈变量 ST：

```
TYPE stack RECORD
        s: ARRAY [1..m0] OF datatype;
        t: 0..m0
        END;
VAR ST: stack;
```

这里，把存放栈中表目的数组 s 和指向栈顶元素位置的变量都作为 ST 的分量定义。这样定义会使得今后对栈的引用只需涉及记录变量ST，而无需指出ST的细节。这符合结构

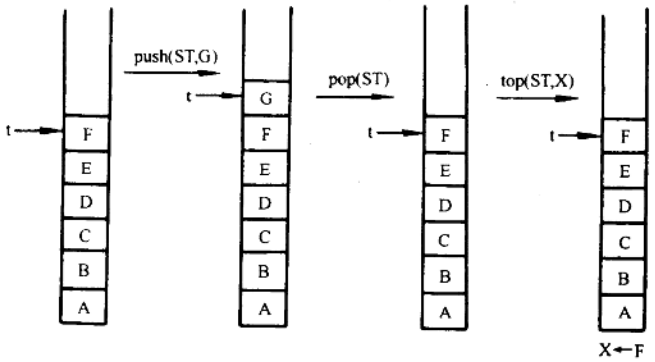

图 1－2－4　栈的顺序存储及主要运算

程序设计的思想。然而，在书写时需注意，用 ST·t 表示指向栈（ST）顶表目的变量（t），用 ST·s 表示存放于栈（ST）中的数组（s）。只当不会引起混淆时，方可简化写作 t 或 s。在 PASCAL 语言中，为了简化书写，须在引用前冠以 WITH 语句：

WITH ST DO

〈语句〉

语句中，所有 ST 记录变量的域名 s 和 t 都不必再加前缀 ST·限定符。

栈的基本运算有以下四种。

1）push（ST，X）　往栈 ST 中插入（推入）一个值为 X 的表目。

2）pop（ST）　从栈 ST 中删除（弹出）一个表目。

3）top（ST，X）　把栈顶表目的值读到变量 X 中，栈保持不变。

4）sempty（ST）　判定 ST 栈是否为空栈的函数。当 ST 中无表目（即 t＝0）时，该函数取真值（true），否则取假值（false）。

这四种运算的具体算法如下。

算法 1－2　栈的推入

push（ST，X）

（1）若 t＝m_0，则 print（'overflow'），否则 t←t＋1；s（t）←X

（2）［算法结束］

算法 1－3　栈的弹出

pop（ST）

（1）若 t＝0，则 Print（'underflow'），否则 t←t－1

（2）［算法结束］

算法 1－4　读栈顶元素

top（ST，X）

（1）若 t＝0，则 print（'error'），否则 X←s［t］

（2）［算法结束］

算法 1－5　判定栈是否为空的布尔函数

Semptg（ST）

（1）若 t＝0，则 sempty←true，否则 sempty←false

(2)[算法结束]

算法 1-6 取出栈顶元素的函数

ptop (ST)

(1) top (ST, X)

(2) pop (ST)

(3) ptop←X

(4)[算法结束]

1.2.1.3 栈的应用——表达式求值

在计算机使用的各种数据结构中，栈是应用最广的一种。这里只介绍一种在编译程序中经常使用的技术，即把栈用于表达式的转换与求值。

一、表达式的两种形式

这里讨论的只限于算术表达式，而且做了简化。下列文法给出了表达式的定义。

基本符号集：

{0,1,…,9,+,-,*,/,(,),.,@}

表达式文法：

〈表达式〉::=〈项〉|〈表达式〉〈加法运算符〉〈项〉

〈加法运算符〉::=+|-

〈项〉::=〈因式〉|〈项〉〈乘法运算符〉〈因子〉

〈乘法运算符〉::=*|/

〈因子〉::=〈常数〉|(〈表达式〉)

〈常数〉::=〈数字〉.|〈数字〉〈常数〉

〈数字〉::=0|1|2|3|4|5|6|7|8|9

用上述文法定义的表达式与通常的表达式一样。把运算符放在两个运算对象中间，所以这种表达式又称为中缀表达式。中缀表达式的计算必须按照如下规则进行：

① 先执行括号内的计算，后执行括号外的计算，在具有多层括号时，按层次反复使用本条规则；

② 在同层内先计算乘（*）和除（/），后计算加（+）和减（-）；

③ 在多个*、/（或+、-）运算可以选择时，按从左至右的顺序执行。

为处理方便，编译程序常常先将中缀表达式转换成等价且必须严格从左至右计算的形式——后缀表达式。在后缀表达式中，不再引入括号，运算符放在两个运算对象的后面，所有计算按运算符出现的顺序从左至右进行。显然，计算后缀表达式比计算中缀表达式简单。

例如：

① 7.

② 372.

③ (2.+14.)

④ 3.*(5.-2.)+7.

等等，都是符合上述文法定义的中缀表达式。与之等价的后缀表达式为：

① 7.

② 372.