高等学校教材

软件开发技术

冯博琴　姜锦虎　陆丽娜　李友仁　侯　迪

高等教育出版社

高等学校教材

软件开发技术

冯博琴　姜锦虎　陆丽娜
李友仁　侯　迪

高等教育出版社

(京)112号

内容提要

本书是根据国家教委高等院校工科计算机基础课程教学指导委员会制订的“软件开发技术课程教学基本(非计算机专业适用)”要求编写的。

本书的目的是在学生初步学会编写小程序的基础上,用软件工程的思想向学生介绍软件开发技术和开发环境等基本知识,使学生能按照工程化的方法,结合专业需要 有效地开发具有实用价值的应用软件,并能编写相应的文档。全书共分十章,内容包括三部分:第一部分介绍数据结构与算法,第二部分介绍软件工程和软件开发技术,第三部分介绍软件开发环境和操作系统。

本书可作为高等学校非计算机专业软件开发技术课程的教材,也可作为各类计算机培训班的教材或参考书。

图书在版编目(CIP)数据

软件开发技术/冯博琴等编. —北京:高等教育出版社,1996

ISBN 7-04-005812-X

Ⅰ.软… Ⅱ.冯… Ⅲ.软件开发-高等学校-教材 Ⅳ.TP311.52

中国版本图书馆CIP数据核字(96)第19519号

*

高等教育出版社出版

北京沙滩后街55号

邮政编码:100009 传真:64014048 电话:64054588

新华书店总店北京发行所发行

国防工业出版社印刷厂印刷

*

开本787×1092 1/16 印张 18 字数440 000

1996年10月第1版 1996年12月第1次印刷

印数0001—1 710

定价 13.50 元

前　言

学生在学习高级语言程序设计课程之后，虽然对计算机程序设计有了一些了解，但离工科学生必须具有一定的计算机应用开发能力的要求尚有相当差距，因此还应再学一些计算机软、硬件方面的知识和软件开发技术，并进行一定数量的训练。软件开发技术是直接用于提高学生软件开发能力的一门计算机基础课，该课程的目的是在学生初步学会编写小程序的基础上，用软件工程的思想向学生介绍软件开发技术和开发环境等基本知识，使学生能按照工程化的方法，结合专业需要，有效地开发具有实用价值的应用程序，并能编写相应的文档。

本课程的基本要求是，了解数据结构和算法在程序设计中的重要性，掌握常用数据结构的简单应用；了解软件工程的基本思想，掌握结构化分析与设计的原理与方法，能运用这些技术对应用软件系统进行有效的分析、设计和编码；掌握软件测试技术，能独立设计程序的测试用例，编制测试软件；熟悉软件支持环境的基本要求，能根据要求正确地选择软件运行和开发环境。相应地，本教材包括三个部分：第一章介绍数据结构和算法，第九、十章讲述软件开发环境和操作系统，第二至八章介绍软件工程和开发技术。第三部分是本书的主题，当然要开发高质量的软件，其余两个部分内容也非常重要。

给没有开发过有一定规模的软件系统的学生讲授软件工程和开发技术，难以引起共鸣，但做过课题的人又往往由于自己不了解或没按软件开发规范进行开发给工作带来很大的后患和损失而深感遗憾。因此本课程除向学生讲清概念外，应尽可能运用实例讨论、课程设计和上机测试等实践环节，使学生熟悉工程化的软件开发技术和文档编制要求，提高学生开发应用软件的实际能力。

凡学过一门高级程序设计语言的学生均可学习本课程，其他计算机基础课程，例如软件技术基础、微机应用基础等课程，与本课程的内容可相互补充，并无先后关系。

本教材是根据国家教委高等学校工科计算机基础课程教学指导委员会制订的高等学校工科非计算机专业软件开发技术课程教学基本要求编写的，并由该课委会推荐出版。本书由西安交通大学冯博琴担任主编，参加编写的有：冯博琴（第一章）、姜锦虎（第二～六、八章）、李友仁（第七章）、侯迪（第九章）、陆丽娜（第十章）。

由于编者水平有限，书中难免还存在一些缺点和错误，而且计算机技术日新月异，观念也不断更新，因此本书的内容取舍和编写角度定会有不妥和疏漏之处，殷切期望同行和读者不吝指教。

冯博琴

1995.7

目　录

第一章　算法与数据结构

计算机在50多年前诞生时是作为一种高速的科学计算工具。凡是遇到复杂的、需要高速计算的问题，如天气预报、桥梁受力分析、原子裂变过程分析、卫星轨道测算、解阶数很高的微分方程或未知数个数很多的线代数方程组等非计算机莫属。但在计算机作为计算工具经历20～30年后，它在处理大宗数据方面也显示了无比的能力，比如在金融财政、旅游宾馆、工厂企业等有关经济数据的处理领域。特别在汉字、图象、声音等"进入"计算机之后，不仅更突出了计算机处理"信息"的能力，而且也大大扩展了计算机处理"数据"的范围。

§1.1　数 据 结 构

1.1.1　数 据 结 构

什么是数据？数据是指能够由计算机处理的数字、字母和符号。数据用来描述客观事物，它所包含的意义称为信息。计算机用于数值计算时，它的数据是数，而现在的多媒体计算机系统中，声音、图形、图、汉字也都是数据。随着计算机技术的发展，它的应用领域不断拓宽，计算机能够处理数据的形式也将日益增加。

项目名	资助者	奖励对象	金额	类别

图1.1　奖学金项目表

数据可以是一个集合，如某校设立的奖学金项目表如图1.1所示。所有的奖学金项目就构成了该校的奖学金数据，这个数据集合中有许多诸如以下具体的奖学金项目：

（钟兆琳奖学金　钟兆琳家属及弟子　本科生　0.15万　个人）

这称为数据元素，它是数据的基本单位。数据元素可以由若干个数据项组成，如上例中的"钟兆琳奖学金"就是其中一个数据项，它是数据的最小单位。

数据结构是指数据元素集合的内部结构或数据元素之间的关系。事实上，构成数据的数据元素并不是孤立的，它们之间存在着一定的关系以表达不同事物及其联系。数据结构作为一个学科分支主要是研究非数值性程序设计中计算机所操作的对象以及它们之间的关系和运算，概括地说是三个方面：

- 数据的逻辑结构
- 数据的存储结构
- 数据的运算

数据的逻辑结构是数据元素之间的逻辑关系。可以用一个二元组给出如下形式定义：

$$\text{Data-Structure}_c=(D,R)$$

其中，D是数据元素的集合，R是D上关系的集合。

按数据元素之间的逻辑关系，一般将数据结构分为两大类：线性数据结构和非线性数据结构。本章将要介绍的线性数据结构有线性表、栈、队列、串和数组，非线性数据结构有树和二叉树。

数据的逻辑结构便于描述数据元素间的逻辑关系，但要对数据进行处理，必须将数据结构“存入”计算机中。而数据的存储结构是指数据结构在计算机中的映象（或表示），要注意，这里不仅要把数据元素本身存储进来，而且要把数据元素之间的逻辑关系体现出来。

数据的存储结构一般可分为顺序存储结构和非顺序存储结构（亦称链式存储结构）。详细来看，可分为以下四种：

· 顺序存储结构。把数据元素按某种顺序放在一块连续的存储单元中。在这种结构中，数据元素之间的关系用数据元素在存储器中的相对位置来表示。

· 链式存储结构。顺序存储结构有局限性：首先它要求有一片连续单元足以存入给定数据，其二是要能预计所需空间大小，以后不得再增加，其三是如以后释放某部分空间，计算机也无法回收。实际问题往往要求不受以上约束，因此就出现了链式存储结构。

如把一个数据元素在计算机内的那部分存储称为一个结点，那末链式存储结构的特点是把每个结点分为两部分：一部分存放数据元素，称为数据域，另一部分存放指示下一个数据元素存储地址的指针，称为指针域。显然这种结构的特点是借指针来表示数据元素之间的逻辑关系。链式存储结构不要求所有结点占用一片连续空间，亦克服了顺序存储结构的三个局限性。

· 索引存储结构。在线性表（见 §1.2）中，数据元素可以排成一个序列 $R_1,R_2,\cdots,R_n$，每个数据元素 R_i 在序列中都有对应的位置数据 i，这就是数据元素的索引。索引存储结构就是通过 i 来确定 R_i 的存储位置，最明显的做法是建立一张索引表，索引表中第 i 项的值就是 R_i 的存储地址。

· 散列存储结构。这种方法是在数据元素（或其中一部分）与其存储地址之间建立一种函数关系 F，根据数据元素 R_i 就可以确定其存储位置 $F(R_i)$，在本章的 §1.4 将有叙述。

例 1.1　有一线性表 (R_1,R_2,R_3,R_4,R_5)，按以上四种存储结构可用图 1.2 表示。

数据的运算是指定义在某种数据逻辑结构上的操作，不同的逻辑结构上所定义的运算可能是不同的，正如我们不能把整数上的运算搬到复数上一样。由于数据结构关心的是非数值性程序设计中计算机操作的对象的运算，因此我们就不会对加减乘除之类的数值运算感兴趣，这里常见的有插入、删除、查找等运算。

程序设计语言中的数据类型是一个重要概念，它与数据结构有密切联系。我们可以把数据类型看作是程序设计语言中已经实现的数据结构，比如复数、数组等。类型说明是数据存储结构的形式描述。所以语言中提供的数据类型越多，语言的表现能力就越强，自然应用范围就会宽一些。

1.1.2 算　法

为处理一个实际问题进行程序设计，实质上就是要解决两个问题：设计算法和选择数据结构。

1. 何谓算法？

算法是有限命令的集合，如果遵循它，则可完成某一特定任务。算法有如下特性：

(1) 有输入：有零或多个输入数据；

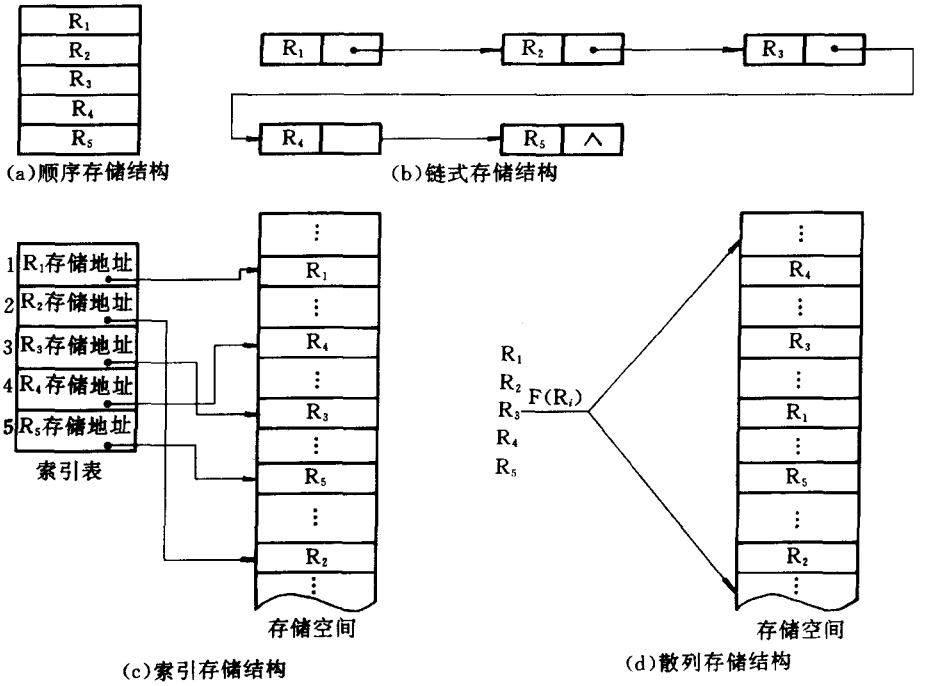

图 1.2 数据的四种存储结构

(2) 有输出:有一个或多个输出数据;

(3) 确定性:每一个步骤必须确定地定义,不能模棱两可;

(4) 有穷性:一个算法必须在执行有穷步之后终止,而且每一步都能在有限时间内完成;

(5) 能行性:算法中有待实现的每一步运算是基本的,人可以用纸和笔做有穷步完成之,因此运算的结果可以分析。

从上述特性可知,程序与算法不同,程序可以不具有有穷性,无限循环,如操作系统的任务调度是一个永远不能停止的运行程序。但在下面你会看到两者又有密切联系。

2. 数据结构+算法=程序

1976 年著名的计算机科学家 N. Wirth 写了一本书叫《ALGORITHMS+DATA STRUCTURES=PROGRAMS》,精辟地阐述了数据结构、算法和程序三者的关系:程序就是在数据的某些特定的表示方式和结构的基础上对抽象算法的具体表述;不了解施加在数据上的算法就无法决定如何构造数据,反之,算法的结构和选择却常常在很大程度上依赖于作为基础的数据结构。程序可看成是算法的一种表达形式,是算法在计算机上的实现,程序的质量固然与程序设计方法有关,但它更"先天地"取决于所采用的数据结构。简而言之,程序的构成与数据结构是两个不可分割地联系在一起的问题。

3. 算法的描述

一个算法可以采用多种方式来描述,常用的有:自然语言(英语/汉语)、流程图、类 PASCAL 语言。本章采用类 PASCAL 语言,这是由于现在所有的数据结构书几乎都是采用类 PASCAL。类 PASCAL 是在 PASCAL 语言基础上稍作变化而来,PASCAL 是一种最合适教

学的语言。有关类 PASCAL 知识可从表 1.1 中得到。

表 1.1　类 PASCAL 语言语句一览表

种类	名　称	表示形式	备　注
类型定义	指针类型 记录类型	TYPE LINK＝↑OBJECT; OBJECT＝RECORD DATA:DATATYPE; NEXT:LINK END	变量说明: P,Q,R:LINK; 表示 P,Q,R 是指向 OBJECT 类型数据的三个指针类型变量

算法形式	过　程	PROCEDURE 算法名(〈参数表〉);
		BEGIN
		〈语句组〉
		END;
			(a) 除参数表外,所有过程中出现的局部变量都不加类型说明
			(b) 假设类 Pascal 语言允许一切可能出现的变量类型

	函　数	FUNCTION 函数名(〈参数表〉):〈类型〉;
		BEGIN
		〈语句组〉
		END;	

简单语句	赋值语句	〈变量〉:＝〈表达式〉	复合语句,类 Pascal 语言中用"["和"]"替代 BEGIN 和 END
	复合语句	[〈语句组〉]	
	输入语句	READ(〈变量表〉)	
	输出语句	WRITE(〈表达式表〉)	
分支语句	条件语名	IF　〈条件〉 THEN　〈语句 1〉 ELSE　〈语句 2〉	
	分情形语句	CASE　〈变量名〉　OF 〈常量表 1〉:〈语句 1〉; 〈常量表 2〉:〈语句 2〉; … ENDCASE;	
重复语句	FOR 语句	FOR　I:＝〈初值〉　TO　〈终值〉　DO 〈语句〉; FOR　I:＝〈初值〉　DOWNTO　〈终值〉 DO 〈语名〉;	
	WHILE 语句	WHILE　〈条件〉　DO 〈语句〉;	
	REPEAT 语句	REPEAT 〈语句〉 UNTIL　〈条件〉;	

（续表）

种类	名　称	表示形式	备　注
其他	过程调用	CALL　过程名(参数表)	
	函数调用	变量名:=函数名(参数表)	
	函数值返回	RETURN(表达式)	
	出错处理	ERROR('文字')	
	基本函数	可以使用 Pascal 语言中的标准函数及一些数学符号来描述一些简单操作	

4. 算法的评价

算法的优劣是按时间复杂度和空间复杂度两个指标来评价的。时间复杂度是指算法在机器上运行消耗的时间长短，但实际去测试它却未必有意义，因为计算机系统不同，测试结果不能说明什么。所以现在常用先验分析估算方法：

算法所需要执行时间＝语句执行一次所用平均时间＊语句执行次数

其中语句执行一次所用平均时间亦会因软硬件不同而不可能给出一个确定的数值，因此通常就只考虑算法中语句执行次数。

例 1.2　估算计算 Fibonacci 数算法的语句执行次数。

```
1    procedure Fibonacci
2        read(n)
3        if n<0 then [print ('error'); stop]
4        if n=0 then [print ('0'); stop]
5        if n=1 then [print('1'); stop]
6        fnm2:=0; fnm1:=1
7        for i:=2 to n do
8            fn:=fnm1+fnm2
9            fnm2:=fnm1
10           fnm1:=fn
11       end {for}
12       print (fn)
13   end {procedure}
```

当 n＞1时，各语句的执行次数如下：

行号	执行次数	行号	执行次数	行号	执行次数
1	1	6	2	11	n−1
2	1	7	n	12	1
3	1	8	n−1	13	1
4	1	9	n−1		
5	1	10	n−1		

因此 Fibonacci 算法总共执行次数为5n+5。

有了算法执行次数，通常就可以用大 O 来计算算法执行时间：当且仅当存在两个常数 c 与 n_0，使所有 $n \geqslant n_0$ 时，都有 $|f(n)| \leqslant c|g(n)|$ 成立，则称 $O(g(n))=f(n)$ 为此算法所需的执行时间。上例中因 $5n+5 \leqslant 10n$，故 Fibonacci 算法执行时间为 O(n)。"O"是数量级的概念，常见的时间复杂度按递增序有

$$O(1),O(\log n),O(n),O(n \log n),O(n^2),O(n^3),\cdots$$

关于空间的复杂度是指运行算法时占有多少存储空间。算法实现时所需的存储空间包括两部分：第一部分是指令、常数、变量和输入数据所占空间，这是程序运行必须的；另一部分是算法执行时的辅助空间，这就与算法设计大有关系了。类似的记号有

$$O(1),O(n),O(n^2),O(n^3),\cdots$$

这同样是一个数量级的概念；n 与具体算法有关，表示问题的规模，如矩阵的阶，方程个数等。

以上的两种复杂度往往是互相矛盾的，追求较低的时间复杂度往往需要以增加空间复杂度为代价，反之亦然，读者要根据所追求的目标决定取舍。

§1.2　线性数据结构

本节讨论一种最简单、最常用的数据结构——线性表，以及两种特殊、重要的线性表——栈和队列。

1.2.1　线　性　表

1. 线性表的逻辑结构

定义1.1　线性表是 $n(n \geqslant 0)$ 个元素 $a_1,a_2,\cdots,a_n$ 的有限序列，记为 $(a_1,a_2,\cdots,a_n)$。其中数据元素的个数 n 称为表长度，$n=0$ 时称此线性表为空表。

从定义可以看出线性表有如下特性：

(1) 当 $n>0$ 时，除 a_1 无前趋，a_n 无后继外，其他 a_i 均有一个唯一的前趋和唯一的后继；a_i 在表中的位置可由序号 i 完全确定。

(2) 所有数据元素 a_i 在同一个线性表中必须是相同的数据类型，它可为一个数，一个符号或一组很复杂的结构。

例1.3　某学生的成绩如下：高等数学：90分，物理：85分，外语86分，C 语言：92分。我们可以按问题需要建立以下两种线性表：

表1：(90,85,86,92)

表2：((高数,90),(物理,85),(外语,86),(C 语言,92))

表2的数据元素是(课程名,成绩)，虽然不是单纯的一个数值，但所有元素均是用一种形式。按定义，允许 a_i 有十分复杂的形式，包括它还可以是一张线性表。

2. 线性表的顺序存储结构

要使线性表成为计算机可以处理的对象，那就必须将它的数据元素及其逻辑关系存储到计算机中。在这里我们先介绍一种最为直观的方式：顺序存储结构。这种结构的特点是将表的数据元素按其逻辑顺序依次存放到一组地址连续的存储单元中。这样逻辑上的相邻元素的存

储地址也是相邻的，所以线性表的逻辑关系隐含在存储单元的邻接关系中。再注意到数据元素的数据类型是相同的，因此每一个元素占用同样大小的存储单元。根据以上所说，线性表的顺序存储有以下特点：

设线性表 $A=(a_1,a_2,\cdots,a_n)$，a_i 占用 C 个存储单元，记 $Loc(a_i)$ 为元素 a_i 的存储地址，则有

$$Loc(a_i)=Loc(a_1)+C*(i-1),\qquad 1\leqslant i\leqslant n$$

可见 $Loc(a_i)$ 是元素 a_i 在 A 中的序号 i 的线性函数，只要知道 i，就可确定 a_i 的存储位置，因此线性表的顺序存储结构是一种随机存取结构。线性表的数据元素、序号及存储地址的关系可用图1.3表示。

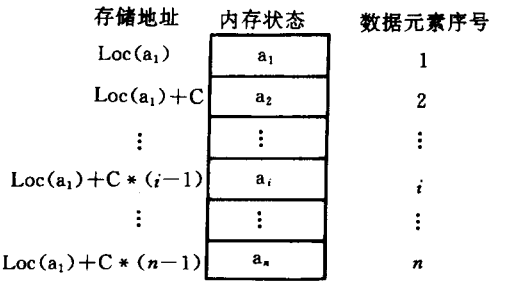

图1.3 线性表的顺序存储结构

线性表的顺序存储结构用类 PASCAL 语言描述如下：

```
CONST
  maxlength=线性表可能的最大长度;
TYPE
  sequenlist=RECORD
               data:Array [1.. maxlength] OF datatype
               last:0.. maxlength
             END
```

其中 data 描述存放线性表中 maxlengty 个元素的存储空间；last 指出线性表最后一个元素的序号。

设 L 为一线性表，在其上常见的运算有

Setnull(L)	置空表
Lenth(L)	求表长
Get(L,i)	取出表的第 i 个元素 a_i
Prior(L,i)	取 a_i 的前趋 a_{i-1}
Suce(L,i)	取 a_i 的后继 a_{i+1}
Locate(L,x)	定位函数，求元素 x 在 L 中位置
Insert(L,i,x)	在 L 的 a_i 前插入元素 x
Delete(L,i)	删去 a_i

在定义这些运算时，有时还应规定 i 的取值范围及运算之后 L 长度的变化。下面给出 In-

sert(L,i,x)和 Delete(L,i)两个运算的实现和算法的描述。

算法1.1　线性表的插入算法。

在线性表($a_1,a_2,\cdots,a_{i-1},a_i,\cdots,a_n$)的第 i 个位置插入元素 x,使之成为($a_1,a_2,\cdots,a_{i-1},x,a_i,\cdots,a_n$)。

```
PROCEDURE Insert(VAR L:sequenlist; i: integer; x:datatype);
  BEGIN
    IF L.last=maxlength
        THEN ERROR('overflow');{线性表已满不能插入}
    IF (i<1) OR(i>L.last+1)
        THEN ERROR ('without'){插入位置不合法}
        ELSE
              [ FOR k:=L.last DOWNTO i DO
                  L.data [k+1]:=L.data [k];{后移一个位置}
                  L.data[i]:=x; {将 x 插到 a(i-1)之后}
                  L.last:=L.last+1{线性表长加1}
               ]
  END;
```

算法1.2　线性表的删除算法。

在线性表($a_1,a_2,\cdots,a_{i-1},a_i,a_{i+1},\cdots,a_n$)中删除第 i 个元素 a_i,使之成为($a_1,a_2,\cdots,a_{i-1},a_{i+1},\cdots,a_n$)。

```
PROCEDURE Delete(VAR L:Sequenlist; i:integer);
    BEGIN
      IF (i<1) OR (i>L.last)
          THEN ERROR ('not exist') {不存在位置为 i 的元素}
      FOR k:=i TO L.last-1 DO
          L.data[k]:=L.data [k+1];{前移}
          L.last:=L.last-1;{表长减1}
    END;
```

我们曾经说过,线性表的顺序存储结构是一种随机存取,即存取表中元素所需的时间与元素在表中位置无关,而且速度相当快;但是在某一位置上插入或删去一元素时需要消耗一些时间,主要花在移动元素。显然移动元素的个数与插入或删除元素位置有关。读者自己可以证明算法1.1和1.2的时间复杂度均为 O(n),n 为表长。

3. 线性表的链式存储结构

线性表的顺序存储结构的优缺点非常明显:线性表的逻辑关系直接反映在物理结构上,使得存取每一元素可用一个线性函数直接算出地址来实现,这个时间复杂度是 O(1)。但正是由于这种结构苛求元素逻辑上的相邻必须保持物理位置上相邻,导致在插入元素时,必须移动大量元素以给新元素"腾位置",在删除元素时,又必须移动后继元素"填空档"。这就使得顺序存储结构只能适用于那些表中元素变动较少的线性表。对于要经常增删表中元素的情况,我们介绍另一种存储结构——链式存储结构。它在处理插入和删除时,不需移动元素。

(1) 单链表(单向链表)

链式存储在存储线性表时,对存储区没有连续的要求,数据元素之间的逻辑关系是由存储线性表元素的附加信息来实现。这种处理技巧很有一般性,常会用到。

单链表由结点组成,每个结点都有两个域:存储数据元素的数据域(data)和用于链接的指针域(link),指针域指出该元素的直接后继元素所在结点的存储位置。结点结构可表示为

用类 PASCAL 描述为

```
TYPE pionter=↑node;
     node=RECORD
            data:elementype
            link:pointer
          END;
```

结点是记录类型,它的 link 域是指向同类型结点的指针。

例1.4 有一批食品,它们的名字、价格、库存量和生产厂家等可以作为一些数据元素,所以这批食品就构成一张线性表:

((biscuit, 50元/1kg, 1000kg, 光华厂),(butter, 100元/1kg, 500kg,京华厂),…)

为书写简单,以名字代表数据元素,线性表就写成

(biscuit, butter, …)

它的单链表逻辑示意图如图1.4所示。

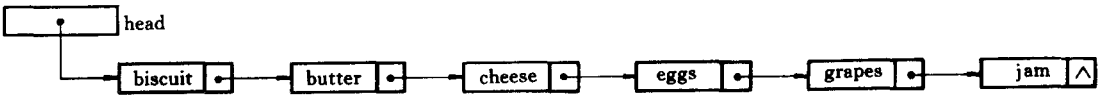

图1.4 单链表逻辑示意图

构造链表,有三个要素。一是链头,本例中链头是 head,它存放了链表第一个结点地址,链头代表整条链;二是链扣,即每一结点要包含下一结点的存储地址;三是链尾,它指出该链的终结,本例中即 jam 结点的 link 域存放一个特殊标记 NIL,图中习惯用 ∧ 表示。

链表的好处是:用链表可以把物理上不连续的"存储碎片"变成逻辑上连续的区域。或者说,用链表结构可以把一张很大的线性表分别安排在几个存储片中,只要这些片至少能放下一个结点就行。

下面讨论在单链表上的运算。常见的运算有找第 i 个结点,在第 i 个结点之前插入一个新结点,删除第 i 个结点,计算表长度,合并两个链表等。为了使空表与非空表统一处理,以简化算法,我们在单链表的第一个元素所在结点前加一个结点——头结点,它的指针域存储第一个元素所在结点的存储位置,数据域暂不利用。这样,空表的判定条件由 head=NIL 改为 head↑.link=NIL。如图1.5所示。

算法1.3 单链表的查找。

在单链表上找第 i 个元素所在结点存储位置,只得从头指针开始沿着指针域找,不能像顺序存储结构那样通过计算得到。

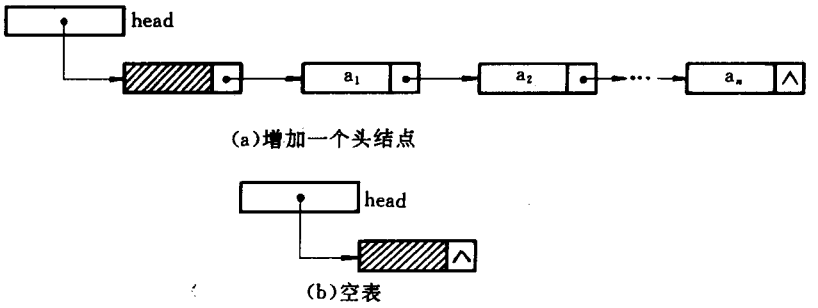

图1.5　带头结点的单链表

```
FUNCTION Get(head:pointer; i:integer):pointer;
    BEGIN
      p:=head;{临时指针置初值}
      counter:=0;{计数器置初值}
      WHILE (p≠NIL) AND (counter<i) DO
          [p:=p↑.link;
          conter:=counter+1];
          {沿链往后找,直到链尾或到达第 i 个结点}
      RETURN(p)
    END;
```

算法1.4　单链表的插入。

要求在存储数据元素 a_i 的结点之前插入一个数据元素为 b 的结点 s①。算法的关键是:"解开"存储数据元素 a_{i-1} 和 a_i 两结点的链,并将新结点"扣上"。见图1.6所示。

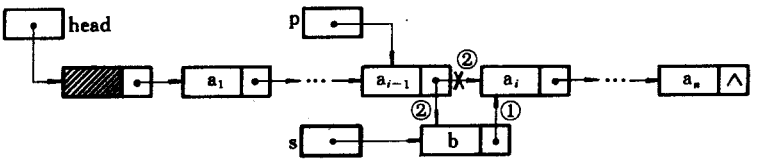

图1.6　在单链表中插入结点 s

```
PROCEDURE Insert(VAR head:linklist; i:integer;x:elementype);
    BEGIN
      p:=Get(head, i-1);{将 a_{i-1}所在结点位置置于 p}
      IF p=NIL
          THEN ERROR('without')
          ELSE [ New (s); {申请一个结点存储空间的过程调用}
```

① 结点 s 或 s 结点均指指针 s 所指的结点,即 s 中存放该结点的存储地址。

```
            s↑.data:=x;{新结点的数据域为x}
            s↑.link:=p↑.link;{见图1.6中①}
            p↑.link:=s ]{见图1.6中②}
            ]
    END;
```

算法1.5 单链表的删除。

从单链表中删除第i个结点，其关键是把第i个结点的直接后继存储地址存入第i个结点的直接前趋结点的link域中，即可实现把第i个结点从链中“摘下来”的动作。见图1.7。

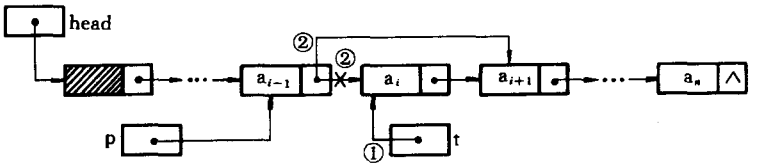

图1.7 在单链表中删去第i个结点

```
PROCEDURE Delete(VAR head:linklist;i:integer);
    REGIN
        p:=Get(head, i-1);
        IF p:=NIL
           THEN ERROR('without')
           ELSE [t:=p↑.link;{见图1.7中①}
                 p↑.link:=t↑.link;{见图1.7中②}
                 dispose(t)   {回收t结点的存储空间的过程调用}
                ]
    END;
```

(2) 循环单链表和双向链表

单链表是单向链表的简称，意为每个结点只有一个指针域，单向指出其后继。它是所有链表的最原始形式，下面对它作些扩充。

(a) 单链表的数据域可以有若干项，形为

data1	data2	data3	link

这样，结点就可存放十分复杂的数据。

(b) 单向循环链表。如果将单链表的链尾填上头指针的值，单链表就呈一个环形，称之为单向循环链表。如图1.8所示。

在有的场合，用循环链表可简化算法或提高效率。在单链表中一般要以头指针为入口点，而循环链表可以从表的任一结点开始访问表的其他结点。

(c) 双向链表。在单向(循环)链表的每个结点再增加一个指向该结点的前趋结点的指针域，就构成了双向(循环)链表。见图1.9和图1.10。

有的应用场合，很需要知道本结点的直接前趋结点，如在插入和删除时就是如此。单链表找后继结点容易，但找前趋结点要从头结点开始，在循环链表虽可从该结点开始，但要整整找