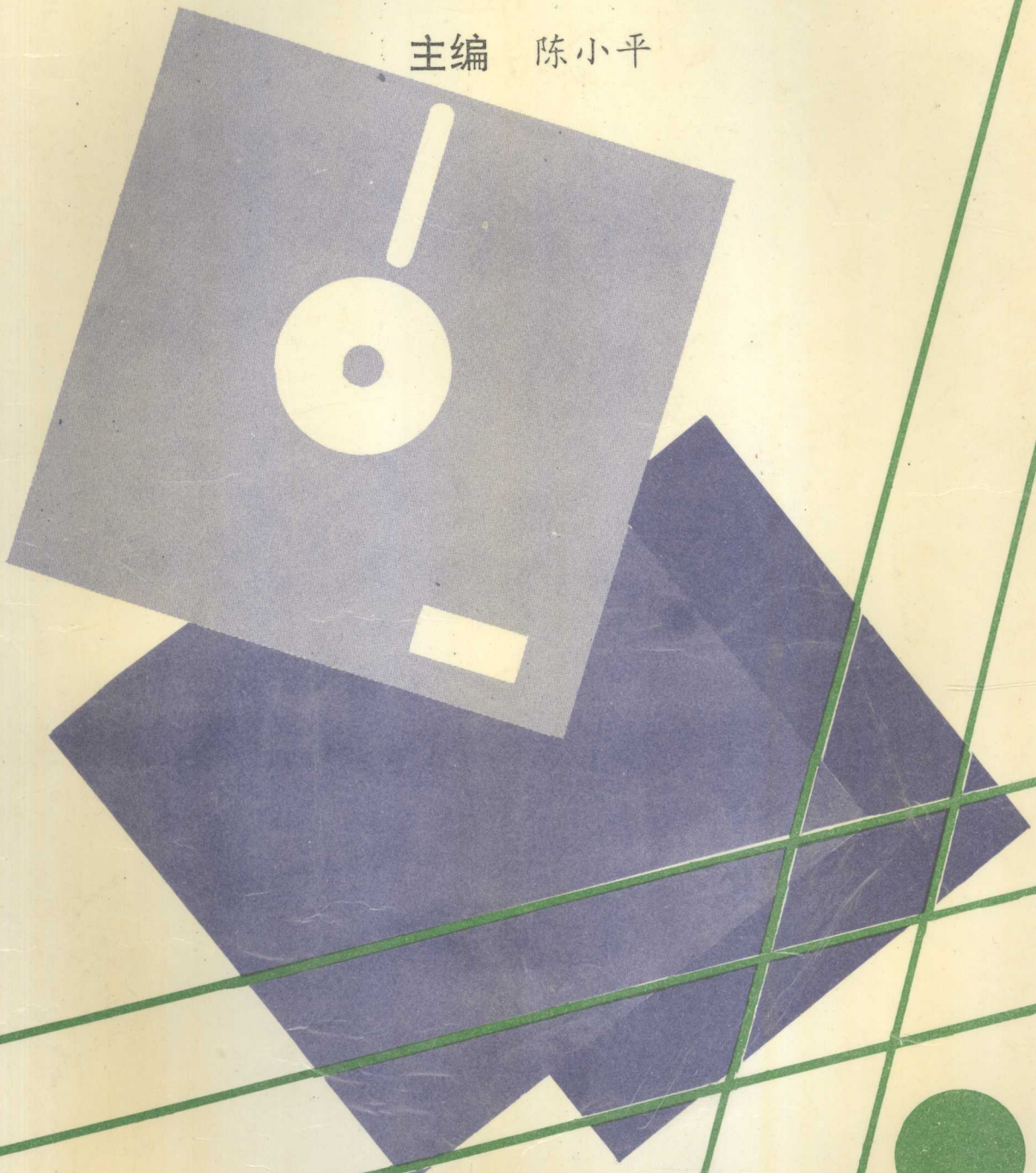
全国高等教育自学考试教材
计算机及其应用专业

数据结构

主编　陈小平



南京大学出版社

全国高等教育自学考试教材
计算机及其应用专业

数 据 结 构

主 编 陈小平
主 审 王世明

南京大学出版社

内 容 简 介

本书系统地介绍了各种常见的数据结构，对数据结构的基本概念、基本原理和基本方法做了深入浅出的介绍，并对有关背景做了较多的交待，对算法设计做了详细、通俗的讲解，每章附带小结和适量的习题。上述特点使本书特别适合于自学。

本书可供计算机及其应用专业自学考试学生及专业人员使用，亦可作为高等院校数据结构课程的教科书或参考书。

(苏)新登字第 011 号

数 据 结 构
SHUJU JIEGOU

陈小平 主 编

南京大学出版社出版发行
（南京大学校内 邮政编码：210008）
安徽省地方志印制中心印刷

开本：787×1092/16 印张：10.25 字数：246 千
1994 年 2 月第 1 版 1995 年 7 月第 2 次印刷
印数 33001—43000
ISBN 7-305-02614-X/TP·96
定价：12.50 元

出 版 前 言

高等教育自学考试教材是高等教育自学考试工作的一项基本建设。经国家教育委员会同意，我们拟有计划、有步骤地组织编写一些高等教育自学考试教材，以满足社会自学和适应考试的需要。《数据结构》是为高等教育自学考试计算机及其应用专业组编的一套教材中的一种。这本教材根据专业考试计划，从造就和选拔人才的需要出发，按照全国颁布的《高等教育自学考试计算机及其应用专业（专科）数据结构自学考试大纲》的要求，结合自学考试的特点，组织高等院校一些专家学者集体编写而成的。

计算机及其应用专业《数据结构》自学考试教材，是供个人自学、社会助学和国家考试使用的。现经组织专家审定同意予以出版发行。我们相信，随着高教自学考试教材的陆续出版，必将对我国高等教育事业的发展，保证自学考试的质量起到积极的促进作用。

编写高等教育自学考试教材是一种新的尝试，希望得到社会各方面的关怀和支持，使它在使用中不断提高和日臻完善。

全国高等教育自学考试指导委员会

一九九四年元月

编者的话

数据结构是计算机及其应用专业的一门重要的专业基础课，它不仅为后继软件课程提供了必需的知识基础，也为专业人员提供了必要的技能训练。

按照全国高等教育自学考试指导委员会电类专业委员会93年6月9日组织召开的计算机及其应用专业教材工作会议拟定的计划，我们编写了这本教材。在符合国家教育委员会自学考试办公室批准的数据结构自学考试大纲要求的前提下，本书对有关内容做了精选。为了满足自学的需要，本书对有关的背景做了较多的交待，对难以理解的内容、特别是算法设计做了详细、通俗的解释。本书不仅适合计算机及其应用专业自学考试学生使用，也可作为高等院校数据结构课程的教材或教学参考书。

本书第五章、第七章及第三章的大部分由郑诚编写，第八章由倪志伟编写，陈小平编写了第一、二、四、六章及第三章的第二节，并对全书进行了统稿。

解放军电子工程学院王世明教授、中国科技大学龚育昌副教授和安徽大学谢荣传副教授反复仔细审阅了书稿，并提出宝贵的意见与建议。中国计算机函授学院牛允鹏教授、张宁副教授和胡学联副教授对本书的编写工作给予了大力支持和帮助。此外，作者还得到中国科技大学黄刘生同志的无私帮助。中国计算机函授学院激光照排中心的同志为本书的录入、排版付出了辛勤的劳动。谨向以上各位同志表示衷心的感谢。

编写高等教育自学考试教材是一项新的尝试，加之编者水平有限、时间仓促，难免顾此失彼、以致书中存在各种疏漏。恳请读者批评指正。

主　编

一九九四年三月十五日

目　次

第一章 概 论

用数字式计算机解决任何问题都离不开程序设计。程序设计的实质是数据表示和数据处理，而这种表示和处理应通过一个渐进的过程逐步完成。数据结构课程主要讨论这个过程中的一些基本问题。本章将概括地介绍有关的基本概念、基本思想、基本原理及实际背景。

§1.1 引 言

今日世界，能被计算机解决的问题种类繁多，确实达到了令人眼花缭乱的地步。然而，说到底，用数字式计算机解决问题的实质是对数据的加工处理。“数据”是计算机加工处理的对象；没有数据，计算机解题就变成“无米之炊”。但是，数据要能被计算机加工处理，首先必须能够存储在机器中，成为能被机器直接操作的对象。数据在计算机存储器中的这种存在形式称为机内表示。显然，数据的机内表示与数据在现实生活和实际问题中的表现形式（姑且称为“机外表示”）是不同的。因此，为了让计算机去加工处理数据，必须首先将数据从机外表示转化为机内表示。这项任务称为数据表示。另外，一个实际问题通常不仅包括数据，还包括处理要求。因此仅仅把数据转化为机内表示并不能完全解决问题，还要用适当的可执行语句编制程序，以便让计算机去执行对数据机内表示的各种操作，从而实现处理要求，即得到所需的结果。这项工作称为数据处理。因此，对计算机专业人员来说，无论面对的具体问题是什么，必须完成的两项基本任务是：数据表示和数据处理。下面用一个简单的例子来说明这两项任务。

例 1—1 考虑某单位职工档案管理问题。为了简单起见，假定每个职工的档案只包括以下五个项目：工作证号码、姓名、性别、出生日期和职称。一般地说，档案管理人员很可能将这些档案组织成表格形式，如图 1—1 所示。表中每一行反映了一个职工五个方面的情况，在本例的假设下构成一个职工的档案。由所有职工的档案组成的这张表格就是本问题中的数据。

0001	刘建国	男	19491001	工程师
0002	黄 红	女	19650506	助 工
0005	张 华	女	19461118	高 工
0007	王 军	男	19600110	技术员
⋮	⋮	⋮	⋮	⋮

图 1—1 数据示例

本问题中的处理要求包括该单位所需的所有能用计算机完成的档案管理工作，其中至少应包含以下功能。①查找：需要时在表格中找出某人的档案；②读取：阅读通过查找找到的档案；③插入：调入新职工时，将该职工的档案加到表格中；④删除：某职工调离时，从表格中撤销其档案；⑤更新：某职工情况变化（如提职称）时，修改其档案的有关内容。上述

每一项功能又可称为一个“运算”(详见以下几节)。

假如现在要用计算机解决上述档案管理问题,即实现档案自动(计算机)管理,设计人员必须完成两项基本任务。第一,将如图 1—1 所示的表格形式的档案转化为某种适当的机内表示,使之成为能被计算机直接处理的对象。第二,用计算机程序代替档案管理人员完成档案管理工作;也就是说,要编写出实现全部处理要求的程序。这里,第一项任务就是数据表示,第二项任务就是数据处理。如果这两项任务都落实了,整个问题也就解决了。

明确了基本任务之后,下一个关键问题是:计算机专业人员应怎样完成他们的基本任务?也就是说,应该用什么样的方法、步骤去完成上述基本任务?根据程序设计理论和软件工程实践,为求解实际问题而进行的程序设计应是一个渐进的过程。从大范围来看,软件工程学认为,软件系统的生存期可分为软件计划、需求分析、软件设计、软件编码、软件测试和软件维护等六个阶段(参见图 1—2),一般认为,程序设计应包括前五个阶段。因此,程序设计(包括数据表示和数据处理两个方面)是一个复杂的过程,需要经过一些不同的阶段,每个阶段完成不同的工作,编写程序只是其中的一个阶段。

阶段	任务	工作结果
计划	理解工作范围	计划任务书
需求分析	定义用户要求	需求规格说明书
设计	建立软件结构	设计说明书
编码	编写程序	程序
测试	发现和排除错误	可运行的程序系统
维护	运行和管理	改进的程序系统

图 1—2 软件开发阶段

数据结构课程并不研究由上述六个阶段构成的程序开发全过程,而是集中讨论以其中的设计阶段为核心、同时涉及编码阶段和分析阶段的一个小范围内的若干基本问题。本书对这些问题的讨论将以这个小范围内的下述设计过程为背景:首先,建立实际问题的一个恰当的数学模型,并设计出一个求解该问题的初步的“非形式算法”。然后,对这个非形式算法(及数学模型)进行“逐步求精”,直至得到一个“实现”即求解该问题的程序。注意,这个过程的每一步骤都包括数据表示与数据处理两个方面,参见图 1—3。

我们将在以下几节对这个设计过程及有关问题做进一步讨论。在进入讨论之前,首先必须明确以下几点。

①数据表示任务是逐步完成的。在上述程序设计过程中,数据的表示形式经历了如下变化:机外表示——→数学模型——→机内表示。

②数据处理任务也是逐步完成的。在上述程序设计过程中,数据处理方面的变化过程是:处理要求——→非形式算法——→程序的可执行部分。

③数据表示与数据处理是密切相关的,数据的某种表示形式总是与某种相应的数据处理形式相联系,比如数学模型与非形式算法相联系。

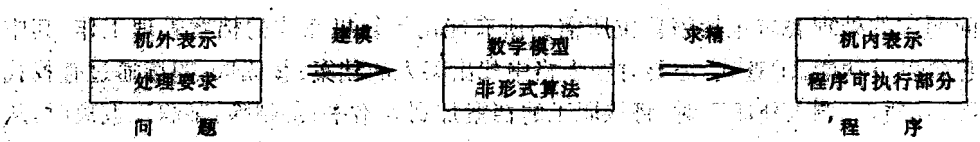


图 1—3 逐步求精的设计过程

§1.2 数据及其逻辑结构

从图1—3可以看出，数据表示方面的第一个大步骤是机外表示转化为数学模型，本节讨论这一步骤的一些基本概念和基本方法。

一、数据元素和数据项

随着计算机科学的发展和计算机应用的普及，计算机加工处理的对象已从早期的数值、布尔值等扩展到字符串、表格、图像甚至语音等等。凡能被计算机存储、加工的对象通称为数据。在数据结构这门课程中，我们经常遇到的另外两个基本概念是数据元素和数据项。

数据元素是数据的基本单位，在程序中作为一个整体而加以考虑和处理。换句话说，数据元素被当作运算的基本单位，并且通常具有完整确定的实际意义。

在很多情况下，数据元素又是由数据项组成的，但数据项通常不具有完整确定的实际意义、或不被当作一个整体对待。

例1—1中的表格是一个数据，它是由各个职工的档案构成的。每一个档案及档案中的每一项目也都是数据。容易看出，档案是本问题中的数据元素，这是因为：每个档案在此问题中被当作运算的基本单位。例如，删除运算不是同时作用于整个表格或多个档案，也不是作用于一个档案中的个别项目，而是作用于整个一个档案；这就是说，档案是删除运算的基本单位。因此，每个职工的档案是本问题中的一个数据元素。另一方面，档案中的各个项目是本问题中的数据项。显然，每个项目是作为档案的一个成份出现的，但单独的项目没有完整确定的实际意义。比如，将表格中第一条档案的各个项目拆开分别地看，则"0001"、"刘建国"、"男"、"19491001"和"工程师"等数据分别表示一个编号、一个人名、一种性别、一个日期和一个职称，它们各自在档案管理这个实际问题中并无完整的实际意义（但它们合在一起却构成职工刘建国的档案，因而具有完整的实际意义）。更重要的是，各个项目在本问题中不能当作运算的基本单位（否则，删除运算可以只删除一个档案中的一个项目比如人名，那将造成严重的后果）。所以，档案中的项目是数据项而不是数据元素。

有些数据元素不能再分解为数据项的组合，我们认为这种数据元素是由一个数据项组成的，这个数据项就是该数据元素自身。

从某种意义上说，数据、数据元素和数据项实际上反映了数据组织的三个层次。因此，尽管广义地说，数据元素和数据项也可以看成是数据，本书以下不再把数据元素和数据项称为数据，以便明确区别数据的不同层次。

二、数据的逻辑结构

在科学研究中，为了把握一个难以直接把握的复杂对象，往往采取以下两个步骤：①将这个复杂对象分解成一些较简单的成份或要素；②考察这些成份或要素之间的关联方式。对于计算机工作者来说，数据是一种复杂对象。为了把握这种复杂对象，以便完成数据表示任务，我们接下来要继续研究数据元素之间、以及同一个数据元素中的数据项之间的关联方式。为了研究的方便，我们假定数据项之间的关联方式是简单的（即认为数据元

素是数据项的集合)和次要的,集中精力研究数据元素之间的关联方式(注意,可将这种研究的结果应用于数据项之间的关系,如果数据项之间的关联方式也比较复杂的话)。

在一个问题中,数据元素之间可以存在多种关系。例如,对于图 1—1 所示的表格,若两个职工档案中"性别"项的值相同,则可以说这两条档案有"同性"关系。但这种关系对我们现在的问题(即如何组织数据)来说并不重要。从数据结构的观点看,重要的是数据元素之间的逻辑关系。所谓逻辑关系是指数据元素之间的关联方式或称"邻接关系"。例如,在图 1—1 所示的表格中,第一条档案与第二条档案是邻接的即相互关联的,因此这两个数据元素之间有逻辑关系。但是,第一条档案与第三条档案不相邻接即相互不关联,因此它们之间没有逻辑关系。数据元素之间逻辑关系的整体称为逻辑结构。数据的逻辑结构就是

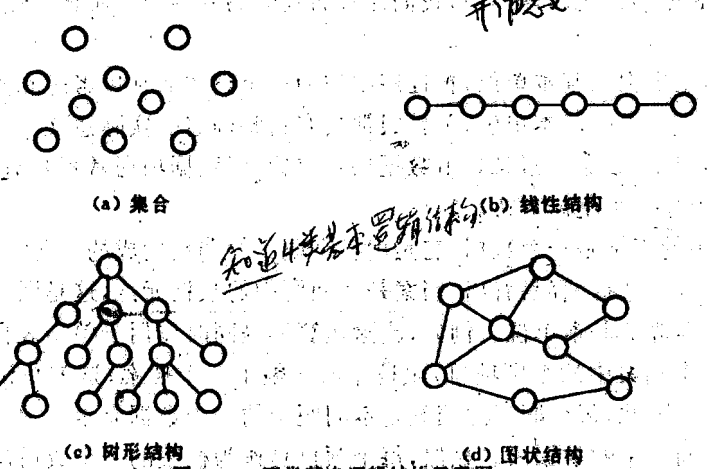


图 1—4 四类基本逻辑结构示意图

数据的组织形式。图 1—1 中数据的逻辑结构如图 1—4(b)所示。图 1—4 中的小圆圈称为结点。一个结点代表一个数据元素(有时也把结点和数据元素当作同义词)。结点之间的连线代表逻辑关系,即相应数据元素之间的邻接关系。图 1—4(b)中的逻辑结构反映了例 1—1 中表格作为一个数据的组织形式,即数据元素(档案)"一个接一个地排列"。

图 1—4 所示为四类基本逻辑结构,它们反映了四类基本的数据组织形式。集合中任何两个结点之间都没有逻辑关系,组织形式松散。线性结构中结点按逻辑关系依次排列形成一条"锁链"。树形结构具有分支、层次特性,其形态有点象自然界中的树。图状结构最复杂,其中的各个结点按逻辑关系互相缠绕,任何两个结点都可以邻接。

关于逻辑结构,有以下几点需特别注意:

1)逻辑结构与数据元素本身的形式、内容无关。例如,在图 1—1 所示表格中的每个档案里增加一个项目"工资额",就得到另一个数据。但由于所有档案仍是"一个接一个排列"的,新数据的逻辑结构与原来的相同。

2)逻辑结构与数据元素的相对位置无关。例如,将图 1—1 所示表格中的所有档案按照工作证号码由大到小的顺序重新排列,得到另一个表格。由于新表格中的所有档案仍是"一个接一个排列"的,其逻辑结构与原表格的逻辑结构相同,还是线性结构。

3)逻辑结构与所含结点个数无关。比如在图 1—1 所示的表格中增加或撤销若干档案,所得表格仍为线性结构。

由此可见，一些表面上很不相同的数据可以有相同的逻辑结构。因此，逻辑结构是数据组织的某种"本质性"的东西。事实上，逻辑结构是数据组织的主要方面。抓住这种本质，不仅有利于分析数据机外表示(如图1—1所示的表格)的组织形式，更重要的，是使程序设计人员可以根据解题需要重新组织数据，即把数据中的所有数据元素按所需的逻辑结构重新加以安排。例如，对例1—1中的数据——表格，既可以照原样组织成线性结构，也可以重新组织成集合或树形构造，具体的选择应根据解题的需要而定。

各种逻辑结构就是本书将要详细介绍的数学模型。这些数学模型主要有以下三个方面的作用：

①利用它们可以重新组织数据以适应解题需要；

②在这些模型上可以进行非形式算法的设计(见下节)；

③为构造数据的机内表示提供某种"依据"(见§1.4)。

§1.3 算法和运算

本节讨论图1—3所示设计过程在数据处理方面的第一个大步骤(建立初步的非形式算法)中的一些基本概念。为了便于理解，先看一个示意性的例子。

例1—1(续) 现采用图1—3所示的程序设计过程求解上述职工档案管理问题。假设已决定采用线性结构(本例中记为S)做为此问题的数学模型，下面继续考虑数据处理方面的第一个大步骤。为简单起见，这里只考虑下述一个特殊的处理要求：当该单位进行了一次职称评定之后，修改有关人员的档案(这里，假定"修改"就是简单地将某职工档案中的"职称"项修改为该职工的新职称名称)。

我们可以按下述基本思想来解决这个问题：对每一位职称发生变动的职工，输入其工作证号码a和提升后的职称b，在S中找出工作证号码为a的结点，然后将该结点"职称"项的值修改为b。假定职称发生变动的职工共有n人，根据上述思想可设计出求解此问题的初步的非形式算法如下：

```
1.  for i:=1 to n do
2.      [ 输入(a,b);
3.        在S中找出"工作证号码"项的值等于a的结点X;
4.        将结点X"职称"项的值修改为b ];
```

显然，上述非形式算法不是程序，但与程序有共同之处，规定了解题的方法和步骤。这种规定又称为算法。一般地，一个算法规定了求解给定类型问题所需的所有"处理"和这些"处理"的执行顺序，使得给定类型的任何问题能在有限时间内被机械地求解。上述非形式算法是算法的一个例子。其中，第2、3、4行各为一个"处理"，整个非形式算法规定这三个"处理"依次反复执行n次。根据这个非形式算法，例1—1(续)中考虑的处理要求可在有限时间内机械地实现。

算法是计算机科学的一个基本概念，也是程序设计的一个核心概念。任何算法总是用某种语言描述的，即用该语言表达算法中的所有"处理"及其执行顺序。根据描述算法的语言的不同，可将算法分为以下三类：

①运行终止的程序可执行部分。运行终止的程序可执行部分是用程序设计语言描述

的算法，这种算法可直接在计算机上运行，从而使给定问题在有限时间内被机械地求解。为了叙述方便，有时又将这种算法称为程序。

②伪语言算法。采用某种"伪程序设计语言"描述的算法称为伪语言算法。伪语言算法不可直接在计算机上运行（可在某种"抽象机"上运行），但容易编写和阅读，故适于教学。下节将介绍本书采用的一种伪程序设计语言——类 PASCAL 语言。

③非形式算法。用自然语言（如汉语）、同时可能还使用了程序设计语言或伪程序设计语言描述的算法称为非形式算法。在图 1—3 所示的程序设计方法中，非形式算法起着不可替代的作用。本书将只涉及形式算法和伪语言算法。

一个算法通常包含两种基本成份：

1)"处理"成份，其作用是表达对数据的加工。例如，例 1—1(续)中的第 2、3、4 行各为一个"处理"，每个"处理"对数据进行一种加工；

2)控制成份，其作用是规定"处理"成份的执行次序。例如，例 1—1(续)非形式算法中的第 1 行规定循环体中的三个"处理"连续执行 n 次。

控制成份主要在程序设计课程中研究，这里只讨论非形式算法中的"处理"成份。容易看出，非形式算法中的有些"处理"直接对应于程序设计语言中的一条语句，如上述非形式算法的第 2 行；另一些"处理"不直接对应于程序设计语言中的任何(单独的)语句，如上述非形式算法的第 3、4 行。一般来说，后一类"处理"是数学模型上的操作，我们将这种处理称为"运算"。由于本书只考虑以逻辑结构为数学模型的情况，因此，运算是逻辑结构上的操作。例如，上述非形式算法第 3 行中的运算是在线性结构 S 上的一种操作。

所谓"逻辑结构上的操作"是指对逻辑结构的加工，这种加工以一个或多个(本书以下只考虑一个)逻辑结构及其他有关参数为对象，以经过修改的逻辑结构或从原逻辑结构中提取的有关信息为结果。根据操作的效果可将运算分成以下两种基本类型：

①加工型运算，其操作改变了原逻辑结构的"值"，如结点个数、某些结点的内容等(本书一般不考虑改变逻辑结构种类，如线性结构变成树形结构的运算)；

②引用型运算，其操作不改变原逻辑结构，只从中提取某些信息作为运算的结果。

根据上面的规定，例 1—1 中的五个运算可分别定义为某种逻辑结构(比如线性结构) S 上的操作如下：

查找 引用型运算，功能是找出满足某种条件的结点在 S 中的位置；

读取 引用型运算，功能是读出 S 中某指定位置上结点的内容；

插入 加工型运算，功能是在 S 的某指定位置上增加一个新结点；

删除 加工型运算，功能是撤销 S 某指定位置上的结点；

更新 加工型运算，功能是修改 S 中某指定结点的内容。

上述五种运算是在各种实际问题中经常出现的常见运算。

显然，上面给出的这五个运算的定义只涉及对逻辑结构 S 的操作，与 S 所含结点的具体内容无关，也不涉及如何实现给定功能的问题。由于运算的定义不能不涉及作为操作对象的逻辑结构(如上面的五个定义中都提到逻辑结构 S)，因此我们有时将以某种逻辑结构 S 为操作对象的运算称为"定义在 S 上的运算"。

关于运算，需要强调以下几点：

第一，在图 1—3 所示的程序设计方法中，"逐步求精"的目的是将非形式算法(及相应

的数学模型)转换成程序。由于非形式算法中的控制成份和那些直接对应于语句的"处理"较容易转换,运算的求精即对运算的转换(详见§2.4及下节)是逐步求精的主要内容之一。相应地,本书将把重点放在运算而不是非形式算法的求精上。

第二,在很多场合,虽然需要解决的实际问题不同,设计出的初步的非形式算法也不同,但非形式算法中出现的运算却大体相同。因此有必要着重研究这些常见运算。

第三,运算决定对逻辑结构的操作方式,从而影响其适用范围。例如,本书第三章介绍的"栈"与"队列",虽然它们的逻辑结构都是线性结构,但由于定义了不同的运算,它们的操作方式极不相同,因此适用场合也极不相同。

一般地,本书将要详细介绍的"线性表"、"栈"、"队列"、"二叉树"、"图"、"查找表"等等都不仅是一种逻辑结构,而且包含了该逻辑结构的某类典型的操作方式。为了叙述方便,本书有时也将这种附带了操作方式约束的逻辑结构称为"数据结构"(注意,这并不是"数据结构"这个术语的定义;事实上,这个术语尚无一致公认的定义)。

由于以上原因,本书在介绍每种数据结构(如"线性表"、"栈"、"队列"等等)时,不仅介绍其逻辑结构,而且介绍一组反映其典型操作方式的运算,并将这些运算称为该数据结构的基本运算。

§1.4 存储实现与运算实现

上文已指出,"逐步求精"的最终目标是得到一个"实现"(即程序)。根据上面的讨论,实现又包括两个主要方面:存储实现(数据表示方面)和运算实现(数据处理方面)。

一、存储实现

存储实现的基本目标是建立数据的机内表示。由于已经建立的逻辑结构是设计人员根据解题需要选定的数据组织形式,因此存储实现建立的机内表示应遵循选定的逻辑结构。另一方面,由于逻辑结构不包括结点内容即数据元素本身的表示,因此存储实现的另一主要内容是建立数据元素的机内表示。按上述思路所建立的数据的机内表示称为数据的存储结构。一般地,一个存储结构包括以下三个主要部分:

①存储结点(在不致混淆时简称为结点),每个存储结点存放一个数据元素;
②数据元素之间关联方式的表示,也就是逻辑结构的机内表示;
③附加设施,如为便于运算实现而设置的"哑结点"等等。

其中,前两个部分是一切存储结构都必须具备的,第三个部分则根据需要而决定是否设置。由于我们假定数据元素内部的组织形式比较简单,存储结点本身的组织也比较简单。因此,存储结构的主要部分是数据元素之间关联方式的表示。由于每个存储结点存放一个数据元素,数据元素之间的关联方式便由存储结点之间的关联方式间接地表达。通常,存储结点之间可以有四种关联方式,称为四种基本存储方式。

1)**顺序存储方式** 每个存储结点只含一个数据元素。所有存储结点相继存放在一个连续的存储区里,用存储结点间的位置关系表示数据元素之间的逻辑关系。按这种方式表示逻辑关系的存储结构称为顺序存储结构。

2)**链式存储方式** 每个存储结点不仅含有一个数据元素,还包含一组指针。每个指针

指向一个与本结点有逻辑关系的结点，即用附加的指针表示逻辑关系。按这种方式组织起来的存储结构称为链式存储结构。

3)索引存储方式　每个存储结点只含一个数据元素，所有存储结点连续存放。此外增设一个索引表，索引表中的索引指示各存储结点的存储位置或位置区间端点。按这种方式组织起来的存储结构称为索引存储结构。

4)散列存储方式　每个结点含一个数据元素，各个结点均匀分布在存储区里，用散列函数指示各结点的存储位置或位置区间端点。相应的存储结构称为散列存储结构。

每一种存储结构都可以在两个级别上讨论。其一是机器级，即直接讨论存储结构在计算机存储器里的表示形式。其二是语言级，即用程序设计语言中的类型说明、变量说明等手段来描述存储结构。语言级描述可经编译自动转换成机器级，因此也可以看成是一种机内表示。本书将着重在语言级上讨论存储结构，这种讨论比较简单，也比较实用。

二、运算实现

上节已指出，运算的求精是逐步求精的一项主要工作，也是本书的重点内容之一。运算求精的目标是得到运算的实现。一个运算的实现是指一个完成该运算功能的程序。由于运算只描述处理功能，不包括处理步骤和方法，因此运算实现的核心是处理步骤的规定，即算法设计。一般来说，对一个复杂的运算仍需按照逐步求精的方法构造它的实现(参见§2.4)。为简单起见，本书在大多数场合将简化求精过程，即先介绍有关的基本思想与技巧，然后直接写出运算的实现。另外，本书将使用一种伪语言——类PASCAL语言来描述实现。这种语言比较简洁，使算法易于书写和阅读，而且也易于改写成用程序设计语言书写的程序。本书以下约定，用类PASCAL语言书写的算法简称为算法。

类PASCAL语言来自对标准PASCAL语言的修改。类PASCAL语言与标准PASCAL语言的主要差别如下。

1)局部量的说明可以省略(但形参表中及函数值类型的说明须保留)。重要的变量需在注解中用文字说明其类型和作用。

2)除作为过程体和函数体标志之外的begin,end可用方括号代替。

3)分情形语句可以采用下述形式

```
case
    <条件1>:<语句组1>;
    <条件2>:<语句组2>;
    ⋮
    <条件n>:<语句组n>;
    else    :<语句组n+1>
endcase;
```

其中，else:<语句组n+1>可以不出现；endcase可以简写为end。

4)不含goto语句，增加以下二语句：

①出错处理语句error(字符串)，其功能是终止它所在算法的执行并回送表示出错信息的字符串。

②返回语句return和return(表达式)。在过程体中，return的作用是终止过程的执

行；在函数体中，return(表达式)的作用是终止函数的执行并将表达式的值回传给函数名。

值得指出的是，本书着重介绍的存储实现与运算实现是实现的主要任务，但并不是全部任务。另外，在实际工作中，即使得到了一个实现(程序)，还需要对其进行测试和修改。

§1.5 算法分析

一般地，对同一个问题可以设计出求解它的不同算法；因此，一个运算可以有不同的实现，即可以设计出实现该运算功能的不同算法。这就产生了如何评价这些算法的问题；通过这种评价，应能使设计人员正确的区分出不同实现的相对优劣，从而为算法设计和选择提供依据。通常从以下几个方面评价算法(包括程序)的质量：

①**正确性** 算法应能正确地实现预定的功能(即处理要求)。

②**易读性** 算法应易于阅读和理解，以便于调试、修改和扩充。

③**健壮性** 当环境发生变化(如遇到非法输入)时，算法能适当地做出反应或进行处理，不会产生不需要的运行结果。

④**高效率** 即达到所需要的时空性能。

这些指标往往是互相冲突的，因此在实际评价中应根据需要有所侧重。数据结构课程着重讨论算法的时空性能，这并不意味着这一指标比别的指标更重要(实际情况往往正好相反)，而仅仅是由于学习阶段所限。下面介绍如何确定一个算法的时空性能，这项工作称为"算法分析"。

一个算法的时空性能是指该算法的时间性能(或时间效率)和空间性能(或空间效率)；前者是算法包含的计算量，后者是算法需要的存储量。我们用下例给出计算量的直观说明。

例1—2 设变量a、b、c、d中各含一个整数。求a、b、c中的最大值与d的乘积。

对于这个问题，可以设计出两个不同算法如图1—5所示。这两个算法的主要区别是：max1先将d乘到a、b、c上然后再求最大值；而max2则先求a、b、c的最大值然后再与d相乘。显然，对于任意给定的输入a、b、c、d，算法max1的计算量比max2的大，因为max1多做两次乘法或者说多做两条赋值语句。因此，如果按这两个算法分别写成程序，则在相同的硬、软件环境下，前者花费的运行时间一定比后者多；也就是说，后者的时间性能比前者高。这就是计算量的直观意义。

```
procedure max1(a,b,c,d:integer);
  begin a:=a*d;b:=b*d;c:=c*d;
    if a>b then x:=a else x:=b;
    if c>x then x:=c;
    write(x)
  end.
```

```
procedure max2(a,b,c,d:integer);
  begin
    if a>b then x:=a else x:=b;
    if c>x then x:=c;
    x:=x*d; write(x)
  end.
```

图1—5 两个算法

由上例可进一步看出，为了评价算法的时间性能，并不需要知道计算量的精确值，只要能反映出求解同一问题的不同算法的计算量之间的差异即可。通常采用下述办法来估算求解某类问题的各个算法在给定输入下的计算量：①根据该类问题的特点合理地选择

①合理选择一种或几种操作作为"标准操作"；②确定每个算法在给定输入下共执行了多少次标准操作，并将此次数规定为该算法在给定输入下的计算量。对例 1—2 中的问题，若分别取乘法、赋值语句和条件判断为标准操作，则两算法在输入 a=1,b=2,c=3,d=4 下的计算量如图 1—6 所示。显然，无论是以乘法还是赋值语句为标准操作，都能正确反映这两个算法时间性能上的差异，因而是合理的。但在此问题中以条件判断为标准操作则不能正确反映这种差异，因而是不合理的。本书在以下各章的算法分析中将采用公认合理的标准操作。若无特殊说明，将默认以赋值语句为标准操作。

算法 \ 标准操作	乘法	赋值语句	条件判断
max1	3	5	2
max2	1	3	2

图 1—6　算法 max1 和 max2 在输入(1,2,3,4)下的计算量

以上讨论的是算法在给定输入下的计算量。通常，一个算法在不同输入下的计算量是不同的。例如，若以赋值语句为标准操作，算法 max1 和 max2 在输入(3,2,1,4)下的计算量分别为 4 和 2，与在输入(1,2,3,4)下的计算量不同。考虑到这种情况，通常用以下两种方式来确定一个算法的计算量：

①以算法在所有输入下的计算量的最大值作为算法的计算量，这种计算量称为算法的最坏情况时间复杂性或最坏情况时间复杂度。

②以算法在所有输入下的计算量的加权平均值作为算法的计算量，这种计算量称为算法的平均时间复杂性或平均时间复杂度。

例如，以赋值语句为标准操作，算法 max1 和 max2 的最坏情况时间复杂性分别为 5 和 3。平均时间复杂性的分析技术参见 §2.2。

最坏情况时间复杂性和平均时间复杂性通称为时间复杂性(或时间复杂度)。算法 max1 和 max2 的时间复杂性是一个定值。但在一般情况下，一个算法的时间复杂性是算法输入规模的函数。一个算法的输入规模或问题的规模是指作为该算法输入的数据所含数据元素的数目、或与此数目有关的其他参数。例如，对于例 1—1 中实现查找运算的算法来说，其输入规模是表格所含档案的条数 n。查找算法的时间复杂性随 n 而变；即为 n 的函数。又如，对图 1—7 所示求两个 n 阶方阵乘积的算法来说，习惯上以方阵阶数 n 作为输入规模。易知第 3 行的赋值语句共执行 n^2 次，第 4 行循环体中的赋值语句共执行 n^3 次，故此算法的最坏情况时间复杂性和平均时间复杂性均为 n 的函数：$T(n)=n^3+n^2$。

```
procedure  matrixmlt(var A,B,C:array [1..n, 1..n] of  integer);
  //求 n 阶方阵 A、B 的乘积 C//
  begin
1.    for  i:=1  to  n  do
2.      for  j:=1  to  n  do
3.        [  C[i,j]:=0;
4.          for  k:=1  to  n  do  C[i,j]:=C[i,j]+A[i,k]*B[k,j] ]
  end;
```

图 1—7　求方阵乘积的算法

显然，上述 T(n)与 n^3 的数量级相同(即为同阶无穷大量)。我们称算法 matrixmlt 的时间复杂性量级为 $O(n^3)$，记为 $t(n)=O(n^3)$。在很多情况下，我们更关心算法的时间复杂