# 公共基础知识和 Visual Basic 语言程序设计

主　编　王永全
副主编　金惠芳　胡耀芳　王学光

北京大学出版社
PEKING UNIVERSITY PRESS

**图书在版编目（CIP）数据**

**公共基础知识和** Visual Basic **语言程序设计/王永全主编. —北京：北京大学出版社，**2006.2
（21 **世纪大学计算机基础教育系列丛书**）
ISBN 7-301-10495-2

Ⅰ. **公…** Ⅱ. **王…** Ⅲ. ①**电子计算机-水平考试-自学参考资料** ②BASIC **语言-程序设计-水平考试-自学参考资料** Ⅳ. TP3

**中国版本图书馆** CIP **数据核字（**2006**）第** 002596 **号**

**书　　名：公共基础知识和** Visual Basic **语言程序设计**
**著作责任者：王永全　主编**
**责任编辑：杨丽明　邵传勇　杨福生　张　晗　王业龙**
**标准书号：**ISBN 7-301-10495-2/TP·0875
**出版发行：北京大学出版社**
**地　　址：北京市海淀区成府路** 205 **号**　100871
**网　　址：**http：//cbs.pku.edu.cn
**电　　话：邮购部** 62752015　**发行部** 62750672　**编辑部** 62752027
**电子信箱：**pl@pup.pku.edu.cn
**排　版　者：北京高新特打字服务社**　82350640
**印　刷　者：**
**经　销　者：新华书店**
787 **毫米**×1092 **毫米**　16 **开本**　26.5 **印张**　678 **千字**
2006 **年** 2 **月第** 1 **版**　2006 **年** 2 **月第** 1 **次印刷**
**定　　价：**52.00 **元**

# 编写者名单


**主　编**　王永全

**副主编**　金惠芳　胡耀芳　王学光

**编写者**　(按撰写章节先后顺序)

王学光　徐玉麟　王永全　金惠芳　胡耀芳

陈海燕　刘　琴　王　弈　赵祖荫　张　权

# 前　言

1994 年，国家教育部面向社会推出了全国计算机等级考试（NCRE）。十多年来，这一考试对促进计算机知识的普及和计算机应用技术的大力推广起到了重要作用。为了便于社会各阶层特别是普通高等学校和高职高专的学生衡量自己的计算机应用能力，以及用人单位考核工作人员应用计算机的水平，全国计算机等级考试在级别选择与调整、科目设置与内容更新等方面不断地进行完善和修订。2004 年，教育部考试中心对 NCRE 的考试科目、考核内容和考试形式再次进行了一定程度的调整，并推出了新的考试大纲（2004 年版）。新考试大纲特别在二级 Visual Basic 语言程序设计的笔试中新增了有关公共基础知识的考试内容（占 30%）。

为满足普通高校和高职高专各专业学生学习或选修 Visual Basic 语言程序设计知识，以及社会各阶层考生的需求，我们严格按照 2004 年新考试大纲的要求，组织了既熟悉等级考试又精通专业技术的强大编写队伍，专门针对二级考试科目中的 Visual Basic 语言程序设计编写了本书。本书将对学习者或准备参加全国计算机等级考试二级 Visual Basic 语言程序设计的考生在较短的时间内掌握公共基础知识和 Visual Basic 语言程序设计方法，从而提高学习效率和效果，最终轻松通过考试起到积极作用。

公共基础知识部分对学习者或考生的基本要求是：掌握算法基本概念；掌握基本数据结构及其操作；掌握基本排序和查找算法；掌握结构化程序设计方法；掌握软件工程的基本方法，具有初步应用相关技术进行软件开发的能力；掌握数据库的基本知识，了解关系数据库的设计。

Visual Basic 语言程序设计部分对学习者或考生的基本要求是：熟悉 Visual Basic 集成开发环境；了解 Visual Basic 中对象的概念和事件驱动程序的基本特性；了解简单的数据结构和算法；能够编写和调试简单的 Visual Basic 程序。

本书特色（包括光盘配套资料）主要体现在：

1. 专业的编写队伍

本书由从事全国计算机等级考试授课、辅导并具有丰富教学经验的专家和教师共同编写。本书内容紧扣 2004 年考试大纲，并将二级公共基础知识和 Visual Basic 语言程序设计的内容有机、协调地整合在一起。书中的例题和练习部分与 Visual Basic 语言程序设计的学习内容及与上机考试相关的操作要求密切相关；书中的选择题与填空题笔试指导部分紧扣公共基础知识和 Visual Basic 语言程序设计的笔试知识点，以及相应的重点和难点知识。这样可以使读者少走弯路，从而极大地提高学习效率。

2. 简洁的体系结构

本书循序渐进地介绍二级公共基础知识（第一篇）和 Visual Basic 语言程序设计的技术（第二篇）。第 1 至 14 章各章均以最新大纲要求的考核内容开始，针对考核知识点以及重点和难点知识，分“内容概述”、“同步训练”及“笔试指导”三部分进行讲解与分析（第 1、2、3、4、6 章不含“同步训练”部分）；第 15 章通过“上机考核典型题目分类训练”及“上机考核模拟精选

题及分析”更有针对性地对操作内容进行了分类指导。

3. 丰富的例题练习

本书第 5 章至第 14 章各章中的“内容概述”部分，根据学习内容和考核要求，编排了有代表性的相关操作例题或编程例题，该部分的例题统一编号为“例题章-X（X 为该章的例题序号）”，如第 11 章的“内容概述”部分的第三个例子“例题 11-3”，有关文件名取为“LT-11-3. frm”、“LT-11-3. vbp”、“LT-11-3. exe”等，存放于配套光盘文件夹名为“第 11 章”的子文件夹“LT-11”中（第 Y 章“内容概述”中的例题操作结果文件存放于配套光盘中文件夹名为“第 Y 章”的子文件夹“LT-Y”中），供教师讲授之用或学习者自学参考（注：第一篇中的公共基础知识部分没有操作和编程方面的例题）。

本书第 5 章至第 14 章各章中的“同步训练”部分，选取了有代表性的操作题或编程题，以便让读者巩固并扩展所学知识以及操作和编程方面的应用技能。该部分的同步训练题统一编号为“练习章-Z（Z 为该章的同步训练题序号）”，如第 11 章的“同步训练”部分的第三个练习“练习 11-3”，有关文件名取为“LX-11-3. frm”、“LX-11-3. vbp”、“LX-11-3. exe”等，存放于配套光盘文件夹名为“第 11 章”的子文件夹“LX-11”中（第 Y 章“同步训练”中的练习操作结果文件存放于配套光盘文件夹名为“第 Y 章”的子文件夹“LX-Y”中），供教师讲授之用或学习者自学参考。（注：第一篇中的公共基础知识部分没有操作和编程方面的练习）

4. 准确的讲解分析

根据学习内容或操作与编程方面的知识和技能，本书第 1 章至第 14 章各章中的“笔试指导”部分，选择了有助于进行笔试考试的题目，在对其进行解答的同时，还对所选题目涉及的知识点加以说明，从而使读者达到对所学知识触类旁通、牢固掌握的目的。

5. 完备的参考资料

本书附带的配套光盘包含了第 5 章至第 15 章的例题、练习和典型题的素材及操作结果文件（或代码），可供教师讲授本书内容和学生学习（或自学）时参考。另外，配套光盘中还收集了自 2002 年 9 月全国计算机等级考试第一次进行二级 Visual Basic 语言程序设计科目考试以来的历次考核笔试题及其参考答案共 8 套（包括样题）。以上资料为教师教学、本书使用者或考生综合复习提供了极大的便利。

全书由王永全任主编，金惠芳、胡耀芳和王学光任副主编。第 1 章、第 2 章、第 3 章由王学光编写；第 4 章由徐玉麟编写；第 5 章、第 6 章、第 9 章、第 13 章由王永全编写；第 14 章由王永全、赵祖荫编写；第 7 章由金惠芳编写；第 8 章由胡耀芳编写；第 10 章由陈海燕编写；第 11 章由刘琴编写；第 12 章由王弈编写；第 15 章由张权编写。主编在审阅过程中还对一些章节的内容作了合理的修改与整理。全书由王永全拟定编写大纲并统稿。

本书的编写得到了华东政法学院各级领导及华东政法学院信息科学技术学院领导的关心和支持。在审阅过程中还得到了顾勋梅博士的热情协助。在此一并表示诚挚的感谢！

由于时间紧迫以及作者水平有限，书中存在缺点和错误，在所难免，恳请专家和广大读者不吝指正。

编 者

E-mail：quanywang@163. com

2005 年 11 月

# 目　　录

## 第一篇　公共基础知识



## 第二篇　Visual Basic 语言程序设计

# 第一篇　公共基础知识

## 第 1 章　数据结构与算法基础知识

### 考 核 内 容

- 算法的基本概念；算法复杂度的概念和意义（时间复杂度与空间复杂度）
- 数据结构的定义；数据的逻辑结构与存储结构；数据结构的图形表示；线性结构与非线性结构的概念
- 线性表的定义；线性表的顺序存储结构及其插入与删除运算
- 栈和队列的定义；栈和队列的顺序存储结构及其基本运算
- 线性单链表、双向链表与循环链表的结构及其基本运算
- 树的基本概念；二叉树的定义及其存储结构；二叉树的前序、中序和后序遍历
- 顺序查找与二分法查找算法；基本排序算法（交换类排序、选择类排序、插入类排序）

掌握线性表、栈和队列的基本概念及基本运算，了解数据结构的概念以及算法的特性。掌握二叉树的性质、存储结构及三种遍历，了解树的基本性质。掌握各种排序的基本思想、稳定性和时间复杂度。掌握二分查找和顺序查找算法。

### 1.1　内 容 概 述

#### 1.1.1　算法

1. 算法的基本概念

任何解题过程都是由一定的步骤组成的。通常把解题过程的准确而完整的描述称为解该问题的算法。算法是一组严谨地定义运算顺序的规则，并且每一个规则都是有效且明确的，此顺序将在有限的次数下终止。

对于一个问题，如果可以通过一个计算机程序在有限的存储空间内运行有限长的时间而得到正确的结果，则称这个问题是算法可解的。但算法不等于程序，也不等于计算方法。当然，程序也可以作为算法的一种描述，但程序通常还需考虑很多与方法和分析无关的细节问题，这是因为在编写程序时要受到计算机系统运行环境的限制。通常，程序的编制不可能优于算法的设计。通俗地讲，一个算法就是一种解题方法。

⑴ 算法的基本要素

一个算法通常由两种基本要素组成：一是对数据对象的运算和操作，二是算法的控制结构。

① 算法中对数据对象的运算和操作

每个算法实际上是按解题要求从环境能进行的所有操作中选择合适的操作所组成的一组指令序列。因此，计算机算法就是计算机能处理的操作所组成的指令序列。

通常，计算机可以执行的基本操作是以指令的形式描述的。一个计算机系统能执行的所有指令的集合，称为该计算机系统的指令系统。计算机程序就是按解题要求从计算机指令系统中选择合适的指令所组成的指令序列。在一般的计算机系统中，基本的运算和操作包括以下四种：

一是算术运算：主要包括加、减、乘、除等运算。

二是逻辑运算：主要包括“与”、“或”、“非”等运算。

三是关系运算：主要包括“大于”、“小于”、“等于”、“不等于”等运算。

四是数据传输：主要包括赋值、输入、输出等操作。

算法的主要特征是着重于算法的动态执行，它区别于传统的着重于静态描述或按演绎方式求解问题的过程。传统的演绎数学以公理系统为基础，问题的求解过程通过有限次推演来完成，每次推演都将对问题作进一步的描述，如此不断推演，直到直接将解描述出来为止。而计算机算法则是使用一些最基本的操作，通过对已知条件一步步地加工和变换，从而实现解题目标。所以，这两种方法的解题思路是不同的。

② 算法的控制结构

一个算法的功能不仅取决于所选用的操作，而且还与各操作之间的执行顺序有关。算法中各操作之间的执行顺序称为算法的控制结构。

算法的控制结构给出了算法的基本框架，它不仅决定了算法中各操作的执行顺序，而且也直接反映了算法的设计是否符合结构化原则。描述算法的工具通常包括传统流程图、N-S 结构化流程图、算法描述语言等。一个算法一般都可以由顺序、选择、循环三种基本控制结构组合而成。

⑵ 算法的基本特征

作为一个算法，一般应具有以下几个基本特征：

① 可行性 (effectiveness)

算法的可行性是指算法中每个步骤的执行时间是有限的。

② 确定性 (definiteness)

算法的确定性，是指算法中的每一个步骤都必须是有明确定义的，不允许有模棱两可的解释，也不允许有多义性。这一性质也反映了算法与数学公式的明显差别。

③ 有穷性 (finiteness)

算法的有穷性，是指算法必须能在有限的时间内做完，即算法必须能在执行有限个步骤后终止。算法的有穷性还应包括合理的执行时间的含义。因为，如果一个算法需要执行千万年，则显然失去了实用价值。

④ 拥有足够的情报

通常，算法中的各种运算总是要施加到各个运算对象上，而这些运算对象又可能具有某种

初始状态,这是算法执行的起点或依据。因此,一个算法执行的结果总是与输入的初始数据有关,对于不同的输入,将会输出不同的结果。一般来说,当算法拥有足够的情报时,此算法才是有效的;而当提供的情报不够时,算法可能无效。

⑶ 算法设计基本方法

计算机解题的过程实际上是在实施某种算法,这种算法称为计算机算法。计算机算法不同于人工处理的方法。

本节介绍工程上常用的几种算法设计方法,在实际应用时,各种设计方法之间往往存在着一定的联系。

① 列举法

列举法的基本思想是:根据提出的问题,列举所有可能的情况,并根据问题中给定的条件检验哪些是需要的,哪些是不需要的。因此,列举法常用于解决“是否存在”或“有多少种可能”等类型的问题,例如,求解不定方程。在实际问题中(如寻找路径、查找、搜索等),局部使用列举法是很有效的。

列举算法是计算机算法中的一个基础算法。列举法的特点是算法比较简单。但当列举的可能情况较多时,执行列举算法的工作量将会很大。因此,在用列举法设计算法时,使方案优化,尽量减少运算工作量,是应该重点注意的。

② 归纳法

归纳法的基本思想是:通过列举少量的特殊情况,经过分析,最后找出一般的关系。显然,归纳法要比列举法更能反映问题的本质,并且可以解决列举量为无限的问题。但是,从一个实际问题中总结归纳出一般的关系,并不是一件容易的事情,尤其是要归纳出一个数学模型,则更为困难。从本质上讲,归纳就是通过观察一些简单而特殊的情况,最后总结出一般性的结论。

归纳是一种抽象,即从特殊现象中找出一般的关系。但由于在归纳过程中不可能对所有的情况进行列举,因此,最后由归纳得出的结论只是一种猜测,还需要对这种猜测加以必要的证明。实际上,通过精心观察而得到的猜测得不到证实,或者最后证明猜测是错的,这样的情况经常发生。

③ 递推

所谓递推,是指从已知的初始条件出发,逐次推出所要求的各中间结果和最后结果。其中,初始条件或是问题本身已经给定,或是通过对问题的分析与化简而确定。递推本质上也属于归纳法,工程上的许多递推关系式实际上都是通过对实际问题的分析与归纳得到的,因此,递推关系式往往是归纳的结果。递推算法在数值计算中是极为常见的。但是,对于数值型的递推算法必须要注意数值计算的稳定性问题。

④ 递归

人们在解决一些复杂问题时,为了降低问题的复杂程度(如问题的规模等),一般总是将问题逐层分解,最终将其归结为最简单的问题。这种将问题逐层分解的过程,实际上并没有对问题进行求解,而只是在解决了最终那些最简单的问题后,再沿着原来分解的逆过程逐步进行综合,这就是递归的基本思想。由此可以看出,递归的基础是归纳。

递归分为直接递归与间接递归两种。如果一个算法 P 显式地调用自己,则称为直接递归;如果算法 P 调用另一个算法 Q,而算法 Q 又调用算法 P,则称为间接递归。

有些实际问题,既可以归纳为递推算法,又可以归纳为递归算法。但递推与递归的实现方法不同:递推是从初始条件出发,逐次推出所需求的结果;而递归则是从算法本身到达递归边界的。通常,递归算法比递推算法更加简单易懂,其结构比较简练。特别是在许多比较复杂的问题中,很难找到从初始条件推出所需结果的全过程,此时,设计递归算法要比递推算法容易得多。但递归算法的执行效率比较低。

⑤ 减半递推技术

实际问题的复杂程度往往与问题的规模有密切的联系。因此,利用分治法解决这类实际问题是非常有效的。所谓分治法,就是对问题分而治之。工程上常用的分治法是减半递推技术。

所谓"减半",是指将问题的规模减半,但问题的性质不变;所谓"递推",是指重复"减半"的过程。

⑥ 回溯法

递推和递归算法本质上是对实际问题进行归纳,而减半递推技术是归纳法的一个分支。在工程上,对于一些实际问题,很难归纳出一组简单的递推公式或直观的求解步骤,而且也不能进行无限的列举。解决这类问题,一种有效的方法是"试"。通过对问题的分析,找出一个解决问题的线索,然后沿着这个线索逐步试探,对于每一步的试探,若成功,就得到问题的解;若失败,就逐步回退,换别的线索再进行试探。这种方法称为回溯法。回溯法在处理复杂数据结构方面有着广泛的应用。

2. 算法复杂度

算法复杂度主要包括时间复杂度和空间复杂度。

(1) 算法的时间复杂度

所谓算法的时间复杂度,是指执行算法所需要的计算工作量。

为了能够比较客观地反映出一个算法的效率,在度量一个算法的工作量时,不仅应该与所使用的计算机、程序设计语言以及程序编制者无关,而且还应该与算法实现过程中的许多细节无关。为此,可以用算法在执行过程中所需基本运算的执行次数来度量算法的工作量。

算法所执行的基本运算次数与问题的规模有关。一般地,我们将算法求解问题的输入量称为问题规模。例如,矩阵乘积问题的规模就是矩阵的阶数。

因而可以归纳出,算法的工作量用算法所执行的基本运算次数来度量,而算法所执行的基本运算次数是问题规模的函数,即

$$\text{算法的工作量} = f(n)$$

其中,n 是问题的规模。例如,两个 n 阶矩阵相乘所需要的基本运算次数为 $n^3$,即计算工作量为 $n^3$,也就是时间复杂度为 $n^3$。

在具体分析一个算法的工作量时,存在这样的问题:对于一个固定的规模,算法所执行的基本运算次数还可能与特定的输入有关,但实际上又不可能将所有可能情况下算法所执行的基本运算次数都列举出来。例如,在长度为 n 的一维数组中查找值为 x 的元素,若采用顺序搜索法,即从数组的第一个元素开始,逐个与被查值 x 进行比较。显然,如果第一个元素恰为 x,则只需要比较一次。但如果 x 为数组的最后一个元素,或者 x 不在数组中,则需要比较 n 次才能得到结果。因此,在这个问题的算法中,其基本运算(即比较)次数与具体的被查值 x 有关。

在同一个问题规模下,如果算法执行所需的基本运算次数取决于某一特定输入时,可以用

两种方法来分析算法的工作量：一种是平均性态分析（Average Behavior），用“A（n）”表示，是指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，即考虑它对所有可能的输入数据集的期望值，此时相应的时间复杂度为算法的平均时间复杂度。然而在很多情况下，各种输入数据集出现的概率难以确定，因此算法的平均时间复杂度也难以确定。另一种更可行也更常用的方法是讨论算法在最坏情况下的时间复杂度，用“W（n）”表示，是指在规模为 n 时，算法所执行的基本运算的最大次数，即分析最坏情况以估算算法执行时间的一个上界。

设 x 是所有可能输入中的某个特定输入，t（x）是算法在输入为 x 时所执行的基本运算次数，则

$$W(n) = \max_{x \in D_n} \{t(x)\}$$

其中，$D_n$ 表示当规模为 n，算法执行时所有可能输入的集合。显然，W（n）的计算要比 A（n）方便得多。由于 W（n）实际上给出了算法工作量的一个上界，因此，它比 A（n）更具有实用价值。

例如，采用顺序搜索法，在长度为 n 的一维数组中查找值为 x 的元素，则从数组的第一个元素开始，逐个与被查值 x 进行比较。基本运算为 x 与数组元素的比较。在这个例子中，最坏情况发生在需要查找的 x 是数组中的最后一个元素或 x 不在数组中的时候，此时显然有：

$$W(n) = \max\{t_i \mid 1 \leqslant i \leqslant n+1\} = n$$

因而，这种算法最坏情况的时间复杂度为 n。

⑵ 算法的空间复杂度

一个算法的空间复杂度，一般是指执行这个算法所需要的内存空间。

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，以及算法执行过程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及某种数据结构所需要的附加存储空间（例如，在链式结构中，除了要存储数据本身外，还需要存储链接信息）。如果额外空间量相对于问题规模来说是常数，则称该算法是原地（in place）工作的。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。

### 1.1.2　数据结构的基本概念

数据结构作为计算机的一门学科，主要研究以下三个方面的问题：

⑴ 数据集合中各数据元素之间所固有的逻辑关系，即数据的逻辑结构。

⑵ 在对数据进行处理时，各数据元素在计算机中的存储关系，即数据的存储结构。

⑶ 数据的运算，即对数据施加的操作。

研究以上问题的主要目的是提高数据处理的效率。所谓提高数据处理的效率，主要包括两个方面：一是提高数据处理的速度，二是尽量节省在数据处理过程中所占用的计算机存储空间。

1. 数据结构定义

简单地说，数据结构是指相互有关联的数据元素的集合。例如，向量和矩阵就是数据结构，在这两个数据结构中，数据元素之间有着位置上的关系；又如，图书馆中的图书卡片目录，也是一个较为复杂的数据结构，对于列在各卡片上的各种书之间，可能在主题、作者等问题上相互关联，甚至一本书本身也有不同的相关成分。

数据元素具有广泛的含义。一般来说,现实世界中客观存在的一切个体都可以是数据元素。例如,描述一年四季的季节名:春、夏、秋、冬,可以作为季节的数据元素;表示数值的各个数:18、11、35、23、16……可以作为数值的数据元素;表示家庭成员的各成员名:父亲、儿子、女儿,可以作为家庭成员的数据元素。甚至每一个客观存在的事件,如一次演出、一次借书、一次比赛等,也可以作为数据元素。总之,在数据处理领域中,每一个需要处理的对象都可以抽象成数据元素。数据元素一般简称为元素。

在实际应用中,被处理的数据元素一般有很多,而且作为某种处理,其中的数据元素通常具有某种共同特征。例如,{春,夏,秋,冬} 中的四个数据元素有一个共同特征,即它们都是季节名,分别表示了一年中的某个季节,从而这四个数据元素构成了季节名的集合。又如,{父亲,儿子,女儿} 中的三个数据元素也有一个共同特征,即它们都是家庭的成员名,从而构成了家庭成员名的集合。一般来说,人们不会同时处理特征完全不同且互相之间没有任何关系的各类数据元素,对于具有不同特征的数据元素总是分别进行处理。

一般情况下,在具有相同特征的数据元素集合中,各个数据元素之间存在某种关系(即联系),这种关系反映了该集合中的数据元素所固有的一种结构。在数据处理领域中,通常把数据元素之间这种固有的关系简单地用前后件关系(或直接前驱与直接后继关系)来描述。

例如,在考虑一年四个季节的顺序关系时,“春”是“夏”的前件(即直接前驱,下同),而“夏”是“春”的后件(即直接后继,下同)。同样,“夏”是“秋”的前件,“秋”是“夏”的后件;“秋”是“冬”的前件,“冬”是“秋”的后件。又如,在考虑家庭成员间的辈分关系时,“父亲”是“儿子”和“女儿”的前件,而“儿子”与“女儿”都是“父亲”的后件。

前后件关系是数据元素之间的一个基本关系,但前后件关系所表示的实际意义随具体对象的不同而不同。一般来说,数据元素之间的任何关系都可以用前后件关系来描述。

(1) 数据的逻辑结构

数据结构是指反映数据元素之间关系的数据元素集合的表示。更通俗地说,数据结构是指带有结构的数据元素的集合。而所谓结构,实际上就是指数据元素之间的前后件关系。

由上所述,一个数据结构应包含以下两方面的信息:

① 表示数据元素的信息。

② 表示各数据元素之间的前后件关系。

数据元素之间的前后件关系是指它们的逻辑关系,与它们在计算机中的存储位置无关。因此,上面所述的数据结构实际上是数据的逻辑结构。

所谓数据的逻辑结构,是指反映数据元素之间逻辑关系的数据结构。

由以上论述可知,数据的逻辑结构有两个要素:一是数据元素的集合,通常记为“D”;二是D上的关系,它反映了D中各数据元素之间的前后件关系,通常记为“R”。即一个数据结构可以表示成

$$B = (D,R)$$

其中,B表示数据结构。为了反映D中各数据元素之间的前后件关系,一般用二元组来表示。例如,假设a与b是D中的两个数据,则二元组(a,b)表示a是b的前件,b是a的后件。这样,在D中的每两个元素之间的关系都可以用这种二元组来表示。

例 1-1　一年四季的数据结构可以表示成：

$$B = (D,R)$$
$$D = \{春,夏,秋,冬\}$$
$$R = \{(春,夏),(夏,秋),(秋,冬)\}$$

例 1-2　家庭成员的数据结构可以表示成：

$$B = (D,R)$$
$$D = \{父亲,儿子,女儿\}$$
$$R = \{(父亲,儿子),(父亲,女儿)\}$$

例 1-3　n 维向量

$$X = (x_1,x_2,\cdots,x_n)$$

也是一种数据结构，即 $X = (D,R)$。其中，数据元素的集合为：

$$D = \{x_1,x_2,\cdots,x_n\}$$

关系为：

$$R = \{(x_1,x_2),(x_2,x_3),\cdots,(x_{n-1},x_n)\}$$

(2) 数据的存储结构

数据的逻辑结构在计算机存储空间中的存放形式称为数据的存储结构（也称数据的物理结构）。

数据处理是计算机应用的一个重要领域，在进行数据处理时，被处理的各数据元素总是被存放在计算机的存储空间中，而且各数据元素在计算机存储空间中的位置关系与它们的逻辑关系可能不同。例如，在前面提到的一年四个季节的数据结构中，“春”是“夏”的前件，“夏”是“春”的后件，但在对它们进行处理时，在计算机存储空间中，“春”这个数据元素的信息不一定被存储在“夏”这个数据元素信息的前面，而可能在后面，也可能不是紧邻在前面，而是中间被其他的信息隔开。

因此为了表示存放在计算机存储空间中的各数据元素之间的逻辑关系（即前后件关系），在数据的存储结构中，不仅要存放各数据元素的信息，而且还要存放各数据元素之间的前后件关系的信息。

一般来说，一种数据的逻辑结构根据需要可以表示成多种存储结构，常用的存储结构包括顺序、链接、索引等。而采用不同的存储结构，其数据处理的效率是不同的。所以，在进行数据处理时，选择合适的存储结构是很重要的。

2. 数据结构的图形表示

一个数据结构除了可以用二元关系表示外，还可以直观地用图形表示。在数据结构的图形表示中，对于数据集合 D 中的每一个数据元素，用中间标有元素值的方框表示，一般称之为数据结点，简称为结点。为了进一步表示各数据元素之间的前后件关系，对于关系 R 中的每一个二元组，用一条有向线段从前件结点指向后件结点。

例如，一年四季的数据结构可以用如图 1-1 所示的图形来表示；反映家庭成员间辈分关系的数据结构可以用如图 1-2 所示的图形表示。

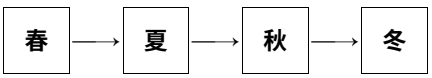


图 1-1 一年四季数据结构的图形表示

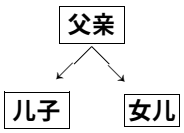


图 1-2 家庭成员间辈分关系数据结构的图形表示

显然,用图形方式表示一个数据结构是很方便的,而且也比较直观。有时在不会引起误会的情况下,前件结点到后件结点连线上的箭头可以省略。例如,在图 1-2 中,即使将“父亲”结点与“儿子”结点连线上的箭头以及“父亲”结点与“女儿”结点连线上的箭头都去掉,也同样能表示“父亲”是“儿子”与“女儿”的前件,“儿子”与“女儿”均是“父亲”的后件,而不会引起误会。

例 1-4 用图形表示数据结构 B = (D,R),其中:

$D = \{d_i \mid 1 \leqslant i \leqslant 7\} = \{d_1, d_2, d_3, d_4, d_5, d_6, d_7\}$

$R = \{(d_1, d_3), (d_1, d_5), (d_2, d_4), (d_3, d_6), (d_5, d_7)\}$

这个数据结构的图形表示如图 1-3 所示。

在数据结构中,没有前件的结点称为根结点,没有后件的结点称为终端结点(也称为叶子结点)。例如,在图 1-1 所示的数据结构中,元素“春”所在的结点(简称为结点“春”,下同)为根结点,结点“冬”为终端结点;在图 1-2 所示的数据结构中,结点“父亲”为根结点,结点“儿子”与“女儿”均为终端结点;在图 1-3 所示的数据结构中,有两个根结点 $d_1$、$d_2$,有三个终端结点 $d_6$、$d_7$、$d_4$。数据结构中,除了根结点与终端结点外的其他结点一般均称为内部结点。

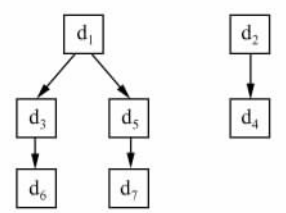


图 1-3 例 1-4 数据结构的图形表示

通常,一个数据结构中的元素结点可能是动态变化的,根据需要或在处理过程中,可以在一个数据结构中增加一个新结点(称为插入运算),也可以删除数据结构中的某个结点(称为删除运算)。插入与删除是对数据结构的两种基本运算。除此之外,对数据结构的运算还包括查找、分类、合并、分解、复制和修改等。在对数据结构进行处理的过程中,不仅数据结构中的结点(即数据元素)个数在动态地变化,而且各数据元素之间的关系也有可能在动态地变化。例如,一个无序表可以通过排序处理而变成有序表;一个数据结构中的根结点被删除后,它的某一个后件可能就变成了根结点;在一个数据结构中的终端结点后插入一个新结点,则原来的那个终端结点就不再是终端结点,而成为内部结点了。

3. 线性结构与非线性结构

如果在一个数据结构中没有数据元素,则称该数据结构为空的数据结构。在一个空的数据结构中插入一个新的元素后,该数据结构就变为非空;在只有一个数据元素的数据结构中,若将该元素删除,该数据结构就变为空的数据结构。

根据数据结构中各数据元素之间前后件关系的复杂程度，一般将数据结构分为两大类型：线性结构与非线性结构。

如果一个非空的数据结构满足下列两个条件：第一，有且只有一个根结点；第二，每一个结点最多有一个前件，也最多有一个后件，则称该数据结构为线性结构。线性结构又称线性表。

但特别需要说明的是，在一个线性结构中插入或删除任何一个结点后还应是线性结构。根据这一点，如果一个数据结构满足上述两个条件，但当在此数据结构中插入或删除任何一个结点后就不满足这两个条件了，则不能称该数据结构为线性结构。

如果一个数据结构不是线性结构，则称之为非线性结构。如例 1-2 中反映家庭成员间辈分关系的数据结构，以及例 1-4 中的数据结构，它们都属于非线性结构。显然，在非线性结构中，各数据元素之间的前后件关系要比线性结构复杂，因此，对非线性结构的存储与处理比线性结构复杂得多。线性结构与非线性结构都可以是空的数据结构。一个空的数据结构究竟是属于线性结构还是属于非线性结构，要根据具体情况来确定。如果对该数据结构的运算是按线性结构的规则来处理的，则该数据结构属于线性结构，否则属于非线性结构。

### 1.1.3　线性表及其顺序存储结构

1. 线性表的基本概念

线性表 (Linear List) 是最简单、最常用的一种数据结构。

线性表由一组数据元素构成。数据元素的含义很广泛，在不同的情况下，它可以有不同的含义。例如，一个 n 维向量 $(x_1, x_2, \cdots, x_n)$ 是一个长度为 n 的线性表，其中的每一个分量都是一个数据元素；又如，英文小写字母表 (a,b,c,…,z) 是一个长度为 26 的线性表，其中的每一个小写字母都是一个数据元素；再如，一年中的四个季节 (春，夏，秋，冬) 是一个长度为 4 的线性表，其中的每一个季节名都是一个数据元素。

矩阵也是一个线性表，只不过它是一个比较复杂的线性表。在矩阵中，既可以把每一行看成是一个数据元素 (即一个行向量为一个数据元素)，也可以把每一列看成是一个数据元素 (即一个列向量为一个数据元素)。而其中的每一个数据元素 (一个行向量或一个列向量) 实际上又都是一个简单的线性表。

数据元素可以是简单项 (如上述例子中的数、字母、季节名等)，也可以由若干个数据项组成。例如，某班的学生情况登记表是一个复杂的线性表，表中每一个学生的情况组成了线性表中的每一个元素，每一个数据元素包括姓名、学号、性别、年龄和健康状况五个数据项，如表 1-1 所示。在这种复杂的线性表中，由若干数据项组成的数据元素称为记录 (record)，而由多个记录构成的线性表又称为文件 (file)。因此，这个学生情况登记表就是一个文件，其中，每一个学生的情况都是一个记录。

表 1-1　学生情况登记表

| 姓　名 | 学　号 | 性　别 | 年　龄 | 健康情况 |
|---|---|---|---|---|
| 赵　军 | 800356 | 男 | 20 | 良好 |
| 刘建萍 | 800361 | 女 | 19 | 一般 |
| 王　亮 | 800362 | 男 | 21 | 良好 |
| … | … | … | … | … |

综上所述,线性表是由 n (n≥0) 个数据元素 $a_1$,$a_2$,…,$a_n$ 组成的一个有限序列,表中的每一个数据元素,除了第一个外,有且只有一个前件;除了最后一个外,有且只有一个后件。一个线性表或是一个空表,或可以表示为:

$$(a_1,a_2,\cdots,a_i,\cdots,a_n)$$

其中,$a_i$ (i = 1,2,…,n) 是属于数据对象的元素,通常称为线性表中的一个结点。

显然,线性表是一种线性结构。数据元素在线性表中的位置只取决于它们自己的序号,即数据元素之间的相对位置是线性的。

非空线性表有如下一些结构特征:

(1) 有且只有一个根结点 $a_1$,它无前件。

(2) 有且只有一个终端结点 $a_n$,它无后件。

(3) 除根结点与终端结点外,其他所有结点有且只有一个前件,也有且只有一个后件。线性表中结点的个数 n 称为线性表的长度。当 n = 0 时,称为空表。

2. 线性表的顺序存储结构

在计算机中存放线性表,一种最简单的方法是顺序存储,也称为顺序分配。

线性表的顺序存储结构具有以下两个基本特点:

(1) 线性表中所有元素所占的存储空间是连续的。

(2) 线性表中各数据元素在存储空间中是按逻辑顺序依次存放的。

由此可以看出,在线性表的顺序存储结构中,其前后件两个元素在存储空间中是紧邻的,且前件元素一定存储在后件元素的前面。

在线性表的顺序存储结构中,如果线性表中各数据元素所占的存储空间 (字节数) 相等,则要在该线性表中查找某一个元素是很方便的。

假设线性表中的第一个数据元素的存储地址 (指第一个字节的地址,即首地址) 为 ADR ($a_1$),每一个数据元素占 k 个字节,则线性表中第 i 个元素 $a_i$ 在计算机存储空间中的存储地址为:

$$ADR(a_i) = ADR(a_1) + (i - 1)k$$

即在顺序存储结构中,线性表中每一个数据元素在计算机存储空间中的存储地址由该元素在线性表中的位置序号唯一确定。一般来说,长度为 n 的线性表:

$$(a_1,a_2,\cdots,a_i,\cdots,a_n)$$

在计算机中的顺序存储结构如图 1-4 所示。

| 存储地址 | … | 所占空间大小 |
|---|---|---|
| ADR ($a_1$) | $a_1$ | 占 k 个字节 |
| ADR ($a_1$) + k | $a_2$ | 占 k 个字节 |
| … | … | … |
| ADR ($a_1$) + (i - 1) k | $a_i$ | 占 k 个字节 |
| … | … | … |
| ADR ($a_1$) + (n - 1) k | $a_n$ | 占 k 个字节 |
| | … | |

图 1-4 线性表的顺序存储结构