个人计算机数据结构

Y. 朗赛姆　M. J. 奥京斯汀　A. M. 特宁鲍姆 著

计算机应用与教学系列

Data Structures for Personal Computers

科学出版社

计算机应用与教学系列

个人计算机数据结构

Y. 朗赛姆　M. J. 奥京斯汀　A. M. 特宁鲍姆 著
刘永道　陈桂敏　周海旗　杨志娟 译
刘德贵 校

科学出版社
1992

（京）新登字 092 号

内 容 简 介

本书是一本应用 BASIC 高级语言实现数据结构方法的教科书. 作者以其丰富的教学经验，深入浅出的语言，全面地介绍了数据结构的基本内容，阐明了用程序语言实现常用的数据结构的方法.本书将数据结构的基本概念与高级程序的设计技巧结合起来，是一本有实用价值的教学用书.

书中所提出的大多数概念问题都用实例进行说明，其中某些实例所采用的方法是研究中的重要课题.

书中各章均有大量例题和习题，以供读者选做和参考. 这些习题的种类广泛，具有一定难度，这将有助于读者进一步掌握书中所述内容.

本书可作为计算机科学与应用专业及有关专业的大学生和研究生的教材，也可供从事计算机软件工作的科技人员参考.

Yedidyah Langsam　Moshe J. Augenstein　Aaron M. Tenenbaum
DATA STRUCTURES FOR PERSONAL COMPUTERS
Prentice-Hall, Inc., 1985

计算机应用与教学系列
个人计算机数据结构
Y. 朗赛姆　M. J. 奥京斯汀　A. M. 特宁鲍姆 著
刘永道　陈桂敏　周海旗　杨志娟 译
刘德贵 校
责任编辑 王春晖
科学出版社出版
北京东黄城根北街 16 号
邮政编码：100707
中国科学院印刷厂印刷
新华书店北京发行所发行　各地新华书店经售
*
1992 年 10 月第 一 版　开本：787×1092　1/16
1992 年 10 月第一次印刷　印张：27 1/4
印数：1—4 000　字数：631 000
ISBN 7-03-002964-X/TP · 219
定价：17.30 元

译 校 者 的 话

目前在高校计算机专业教学中，数据结构和程序设计方法均作为专业基础课程先后独立开设，前者是后者应用的基础．为了把这两门课程有效地结合起来，使学生既能掌握抽象的数据结构概念，又能应用程序设计语言具体实现抽象概念，这里向读者介绍一本应用 BASIC 高级语言实现数据结构方法的教科书．

由于种种原因本书翻译出版较晚，但使用 BASIC 语言实现数据结构的方法，至今仍不过时．原因是：1）BASIC 是深受初学者欢迎并广泛流行的计算机高级语言，对普及使用计算机起着重要作用；2）BASIC 是与个人计算机相匹配的语言，既易于学习，又有其实用价值；3）在非计算机领域的学习和应用中，BASIC 语言对解决相对简单的问题仍有重要作用．因此，仍有必要向读者介绍使用 BASIC 语言设计和实现数据结构的图书，本书作为一本教学用书，在教学方法上起到了推陈出新的作用．

随着数据结构概念的不断深化和 BASIC 语言的不断发展（结构化 BASIC 语言），用更高级的 BASIC 语言或其它计算机语言（如 Pascal）来研究数据结构的教科书也将会相继出现．

本书由刘永道翻译第一至第四章，陈桂敏翻译第五章，周海旗翻译第六、七章，杨志娟翻译第八、九章．最后由刘德贵统一校订了全书．在翻译过程中，译者曾对原著中一些印刷错误作了纠正．限于译者水平，译文中一定有许多不妥之处，敬请读者批评指正．

译校者

序　　言*

本书面向两部分读者，一部分读者是对程序设计，尤其是对 BASIC 程序设计已经达到基本熟练水平的程序人员．为此本书对个人计算机中普遍使用的 BASIC 程序设计作了有关编程技巧方面的介绍．这对于编程技巧水平一般的程序人员可能会有深刻的影响，因为在实践中程序人员可能已经体会到了在解决包罗万象，甚至更复杂的问题时，往往需要掌握更高水平的程序设计技术．因此对这种能提高程序设计技巧的数据结构课题的研究，将是那些高水平的程序人员进一步追求的共同目标．

另一部分读者是在大专院校学习计算机科学技术专业的学生．个人计算机和计算机科学教育在今天日益迅速扩大，使程序设计这门技术变得很普遍，甚至在中学都增设了一到两门程序设计的入门课程．虽然这些课程一般只适合于两年制中专或高中，但许多四年制学院的计算专业也都开设了这类课程．在这类学校中，BASIC 语言是常用的程序设计语言．

编写本书的基本目的是在增强高级程序设计技巧的同时，向读者介绍数据结构的基本概念．

我们已经给学生开过几年的数据结构课程，学校中设有一学期的高级程序设计课程和一学期的汇编语言设计课程．我们发现尽管在程序设计方法教学中花去了相当可观的时间，但是学生们仍没有充分掌握程序设计的方法，而且他们自己也不能实现抽象的程序结构．学习较好的学生还能抓住些要领，而学习较差的学生基本没有达到要求．根据这些情况，我们得到一个确定的结论——第一门数据结构课必须与第二门程序设计课紧密结合．本书就是这种信念的产物．

书中从介绍抽象的概念出发，说明如何运用这些概念来解决问题．同时还讲到了如何应用程序设计语言具体实现抽象出的概念．我们同时强调概念的抽象形式和具体形式，所以同学们在学习有关概念本身的同时，还要掌握它的实现及应用．

本书所用语言为 BASIC，虽然有许多语言都能很好地支持程序设计技术，而且在实现抽象数据结构方面还优于 BASIC，但我们选用了 BASIC，其原因是：首先，个人计算机普遍采用了 BASIC 语言，使 BASIC 成为今天使用最广泛的高级语言，用术语来说，这叫作可接近性 (accessibility)；其次，非计算机专业的学校对使用计算机也表现出浓厚的兴趣，其中许多人对数据结构有兴趣，但缺乏运用高级语言编程的技巧，而且一般也只有少数课程才有这些内容；最后，许多计算机专业学校对 BASIC 能普遍接受的程度还相差很远(大概永远是如此)，但都正在深入认识其在计算机科学程序应用中的作用，特别是前面提到过的那些中小规模专业学校更是如此．虽然许多人批评 BASIC 存在不足，但仍可以正确地使用它．在第二章我们将介绍一种对 BASIC 能实现兼容的方法，而且本书以后各章都将继续强调使用这种方法．以本书作为教材，在用 BASIC 进行程序设计

---

* 译者略有删节．——译者注

的这个前提条件下，相当于开设一学期课程的内容．那些不太熟悉 BASIC 语言的读者，应参阅语言一节中所介绍的某些文献中的内容．

本书第一章讲述数据结构，其中 1.1 节讲抽象数据结构的概念和实现的方法． 1.2 节讲数组、数组的实现方法及其应用． 1.3 节讲数据集合与使用 BASIC 实现的方法．

第三章讲使用 BASIC 进行结构化程序设计的技术及算法，这些技术也代表书中其他各部分在讲程序设计时采用的一种风格．

第三章讲堆栈及其用 BASIC 实现的方法． 由于这是第一次讲解这种新的数据结构，因此还会讲到这种数据结构在实现时存在的缺点和问题．3，4 节讲语句后缀、前缀和无缀表示方法．

第四章讲队列、链表和适用结点方式的数组的概念原理，以及实现它们的方法．

第五章讲递归及其应用． 由于递归方法在大多数 BASIC 语言版本中没有实现，所以只说明采用仿真递归的方法．

第六章讲树的构造原理，第七章讲图的概念，第八章讲排序原理，第九章讲检索方法．

在本书最后，我们给出了一系列有关 BASIC 程序设计和数据结构方面的文献资料，读者可进一步参考阅读．

当按一学期课程讲授时，第七章以及第一、二、六、八和九章中的一部分内容可以省略．

第二章所讲到的算法是采用介于英语和 BASIC 程序语言的中间形式来描述的． 它们是用英语描述的一种高级结构方式．学习这些算法时，读者应着重关心求解问题的方法，而不必过多考虑有关变量说明和实际语言的特性．在算法转换为程序的内容中，我们将介绍这些方面的问题以及它们本身的缺欠．

BASIC 程序和算法描述所用的缩进书写方式是第二章所讲的一种形式． 为改进程序的易读性，我们采用了一种有效的工具：算法和程序之间的区别，分别用小写斜体字和大写罗马字母的形式来表示．

书中提出的大多数概念问题都将用实例进行说明．某些实例所采用的方法都是研究中的重要课题(如后缀表示法，多字运算等)，而且实际上也可以照此来处理．其它一些实例也说明了各种不同的实现方法(如树的顺序存储问题)．在用一学期教完的课程安排中，教师们可根据内容任意选用书中所列实例，有些也可以留给学生们自己阅读．仅有一、二学期时间授课的学校，教师们很难做到详尽讲述所有实例，因而授课时一定要预先做好安排．我们认为教材应适合教学发展阶段，用详细实例说明问题比草率讲解多方面的题目更重要．

书中习题的种类很广，并有一定难度．有些习题是训练性的，以保证进一步掌握教材中提出的问题．另外一些习题涉及到对书中所列程序的修改，或算法的改变．还有一些习题是用来介绍新概念的，这些习题也都是很有趣的．往往有这样的情况，一组成功的习题不仅是对一种新课题的完整研究，而且还可能作为一项方案或一篇论文的基础．我们建议教师在给学生们留作业时，仔细挑选适合学生水平的习题．每一学期给学生们留一些程序设计项目的作业(通常 5 到 12 题，视难度而定)，本书中习题中有几个这种项目的题目．

在写本书过程中遇到的最难决择的问题之一就是对各种 BASIC 版本的使用问题. 为使所编程序能在各种个人计算机中运行，希望采用最普通版本中最低水平的 BASIC. 由于采用了最小 BASIC 的子集，因此我们所编的程序可能不完全具备普通个人计算机中所用的标准 BASIC 本身性能上的优点. 但我们确保书中所有程序均可在 Radio Shack BASIC Level II, Microsoft BASIC-80 和 IBM PC BASIC 等支持下运行. 其中 Radio shack BASIC Level II 是相当接近其它两个 BASIC 功能的子集，它给我们提供了完全可以相信的所有性能，其所受的唯一限制是它区别变量时只用其名称中的前两个字符，禁止使用嵌入保留字的内容. Applesoft 的 BASIC 也有同样的限制. 我们在书中尽量使用有意义的变量名，但仍遵守上述那些限制. 当然在某些 BASIC 版本中没有这些限制，程序员可以随意使用变量名，很少会出现有毛病的变量名. 我们没有使用 Microsoft BASIC-80 和 IBM PC 中 BASIC 的那些高级性能(如 WHILEWEND 结构，MOD 内置功能等)，而且当前大量个人计算机中用的 BASIC 也都没有这些高级性能. 但我们将在第二章中介绍这些结构，而且在讲解算法时也用它们来实现.

我们感到不能忽视的一种性能就是 IF-THEN 结构中的 ELSE 子句. 在不用 IF-THEN-ELSE 的结构中，程序可能变得无法掌握，而且还会大大降低学习价值. 但遗憾的是 Applesoft BASIC 语言不支持 ELSE 子句. 使用 Applesoft 的程序人员可以用第二章提出的方法来仿真 ELSE 子句.我们还可以用 DEF 语句来定义变量类型，而且可以不依靠专门类型的符号. 这一点在 Applesoft BASIC 中无效，但可用类型符号插入的方法来补救. 本书采用的其它所有性能，在 Applesoft BASIC 中都还是很有效的. 书中每个程序（或子程序）都已经在使用 BASIC Level II 的 Radio Shack Mode III, 或使用 Microsoft BASIC 80 的 Apple II Plus 以及使用 BASIC 的 IBM PC 等三种机型上进行过测试.

Y. 朗赛姆

M. J. 奥京斯汀

A. M. 特宁鲍姆

目　　录

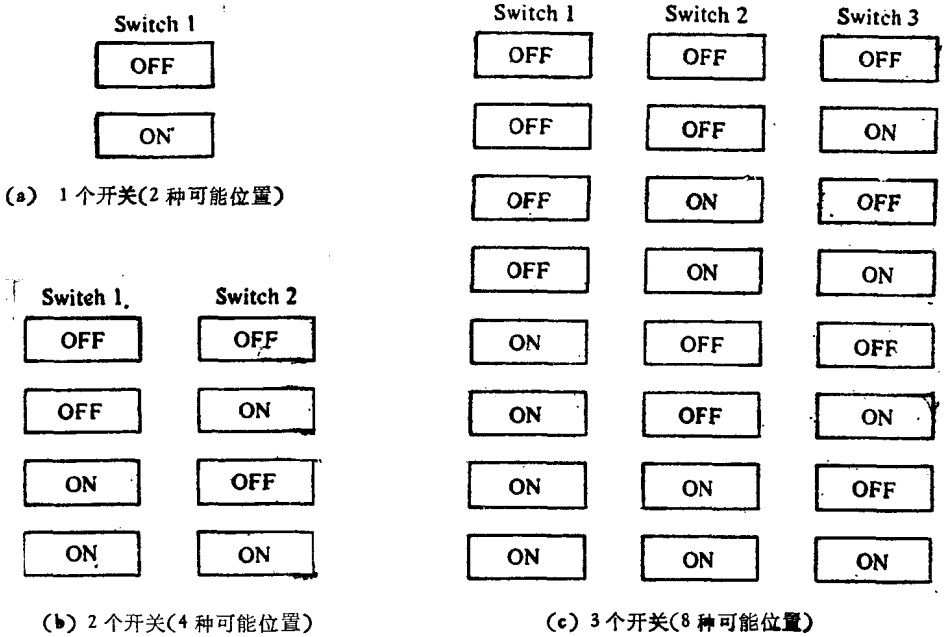
第一章　数据结构简介

计算机是一种对信息进行加工处理的机器。研究计算机，主要是研究计算机中信息的组织、加工和使用等几个方面。对于一个计算机学科的学生来讲，为了继续该领域的学习，理解信息的组织和加工处理的概念尤为重要。

1.1　信息及其含义

如果从根本上讲，计算机科学是研究信息的科学，那么，首先就会提出这样一个问题：什么是信息？遗憾的是，尽管信息的概念是整个领域的基础，但却难以作出准确的回答。在这个意义上，计算机科学里的信息概念类似于几何中的点、线、面的概念——它们都是未定义的术语。对这些术语我们能够作一些说明，但是却不能用更基本的概念来解释它们。

在几何中，尽管线这一概念本身并未定义，我们却能讨论线的长度。线的长度是一个测量值。同样，在计算机科学里，我们也能测量信息的量。信息的基本单位是比特(bit)*，其值可以取为两个相反的可能值中的一个，例如，一个电灯开关，同一时刻只可能处于两个位置中的一个，而不能同时处于两个位置。事实上，一个电灯开关，要么位于"接通"位置，要么位于"断开"位置，这种情况就是信息的一个比特（1bit）。如果一个器件可取的状



(a)　1个开关(2种可能位置)

(b) 2个开关(4种可能位置)

(c) 3个开关(8种可能位置)

图　1.1.1

* bit：本书译为比特，也可称为"数字位"，或简称"位"。——译者注

态多于两个，那么，这个器件所处的特定状态就超过一个比特的信息。例如，一个表盘有8个可能的位置，它若位于位置4，就不可能位于其它7个位置。而一个电灯开关位于"接通"就不能位于"断开"。

另一方面，假如我们只有一种双向开关，那么使用多少才能满足我们的需要，例如表示8个位置的表盘需要多少个这种开关呢？很清楚，一个开关只能表示2个位置，见图1.1.1(a)。两个开关可以表示4个不同位置，见图1.1.1(b)。这样，表示8个不同位置就需要3个开关。一般来说，n个开关可以表示2^n个不同的可能状态。

用二进制数0和1可以表示一个特定位的两种可能状态。实际上，"位"(bit)是二进制数字位(binary digit)的缩写。已知n比特，就是说用n个"1"和"0"的字符串来表示它们的组态。例如，101011代表6个开关，其中，第1个开关(从左开始数)是"接通"(1)，第2个开关是"断开"(0)，第3个"接通"，第4个"断开"，第5和第6个"接通"。

我们已经知道，3个比特就足以表示8个可能的状态，这3个比特的8种可能组态(000,001,010,011,100,101,110,111)可以用来表示整数0—7。然而，这些位的组合并非一成不变，并非特定的组合表示特定的整数。只要不把两个整数同时用同一个数位组合来表示，那么，分配给所有这些位组的整数值就都是等效的。一旦完成了这种分配方式，则一个特定的位组就可以单值地转换为某个特定的整数。这些位组怎样转换为整数？下面我们就来介绍几种广泛使用的方法。微机中用BASIC解释程序表示整数，可以表示更为复杂的形式。但是，所用的这种表示法的精确细节并不特别重要，重要的是，一旦把数位串处理成整数的方法确定下来后，规定技巧的细节就与用户无关。

1.1.1 二进制整数和十进制整数

把位组转换成非负整数时，用得最广的方法是二进制数制。在这个数制里，每一个二进制位用一个以2为底的幂来表示，最右一个二进制位为2^0(等于1)；下一位为2^1(等于2)；再下一位为2^2(等于4)，等等。一个整数可用2的幂的和来表示。一个全零数串代表0，如果在某一特定位的位置上有一个1，那么，由该位所在位置表示的2的幂即包含在和数里；但如果是0，那么2的幂就不包含在和数里。例如，在位组00100110里，位置1，2和5均为1(计数从右到左进行，其中最右的位置为位置0)。于是00100110所表示的整数为$2^1+2^2+2^5=2+4+32=38$。按照这种表示方法，长度为n的任何一个二进制数串，就表示为0到2^n-1间的一个唯一的非负整数；而0到2^n-1之间的任何一个非负整数，也可唯一地表示成一个n位长的二进制数串。

表示二进制负数有两种广泛采用的基本方法。第一种方法是反码记数法(1的补数)，用这种方法表示负数，是把该数串的各位按绝对值取反。例如，00100110表示38，而11011001则表示−38。这意味着，该数的第一位不再用来表示2的幂，而是专门用来表示这个数的符号。一个数串首位为0，表示正数；一个数串首位为1则表示负数。一个n位的二进制数，其代表的数的范围是$-2^{n-1}+1$(1后面跟$n-1$个0)到$2^{n-1}-1$(0后面跟$n-1$个1)。注意，按这种表示法，数字0有两种表示方法：每一位都为0的表示"正0"和每一位都是1的表示"负0"。

表示二进制负数的第二种方法叫补数(补码)记数法。这种方法是在一个负数的反码表示上加1。例如，11011001在反码记数法里表示−38，而在补码记数法中，表示−38则

要用 11011010. 给出一个 n 位的二进制数，表示的范围为 -2^{n-1}(1 后面跟 $n-1$ 个 0)到 $2^{n-1}-1$ (0 后面跟 $n-1$ 个 1). 注意，-2^{n-1} 在补码记数法里能表示，而在反码记数法中就不能表示，而且 2^{n-1} 的绝对值，两种记数法都不能用 n 位数字来表示，另外，在补码记数中，用 n 位二进制表示的整数“0”，只有一种表示法. 下面我们就来进行分析. 考虑用 8 位二进制“0”(00000000)来表示整数“0”的情况，其反码为 11111111，这是反码记数法中的“负0”. 加 1 形成的补码形式为 100000000，它是一个 9 位长的数，由于只允许有 8 位，最左边的位(或上溢)被舍去，余下的 00000000 为“负 0”.

二进制数制绝不是用二进制位表示整数的唯一方法，例如，一个二进制数串也可按十进制数制来表示一个整数. 方法如下：按上面描述的二进制记数法，可用四位二进制位来表示 0—9 中的一位十进制数. 一个任意长的二进制位串，可以分成若干连续的四位组，其中每一组表示一位十进制数. 这个数串此时就表示由这些十进制数按传统的十进制记数法构成的整数. 例如，在该数制里，位串 00100110 分成为 4 位一组的两个位串，它们分别是 0010 和 0110，第一个表示十进制数 2，第二个表示十进制数 6，因而，整个位串就表示了整数 26，这种表示法称为十进制数的二进制编码表示法.

十进制数的二进制编码表示法表示的非负整数有一个重要特点：并不是所有的二进制位串表示十进制整数都有效. 4 个二进制位可用来表示 16 种不同可能状态中的一种，因而一个 4 位二进制组有 16 种可能的状态. 显然，在十进制整数的二进制编码中，只用了这 16 种可能状态中的 10 种. 也就是说二进制为 10 或大于 10 的编码，如 1010，1100 等，在十进制整数的二进制编码中都将是无效的.

1.1.2 实数

计算机表示实数，通常使用浮点记数法. 浮点记数法有许多类型，每一种类型都各具特色. 其基本原理是：把一个实数表示成一个尾数乘以一个底数的整数次幂(指数). 底数通常固定不变，而且表示的实数不同，尾数和指数也不同. 一个实数可用不同的尾数和指数来表示. 例如，底固定为 10，则数 387.53 可以表示成 38753×10^{-2}(10^{-2} 为 0.01)，尾数是 38753，指数为 -2. 另外，也可表示成 0.38753×10^{3} 或 387.53×10^{0}. 我们选择的这种表示法是尾数为一个后面不带零的整数.

上面描述的浮点记数法不一定在任何机器上都能实现. 一个用 32 位二进制串表示的实数，前 8 位为指数，后 24 位为尾数，底固定为 10. 尾数和指数皆为二进制整数的补码，例如，整数 38753 的 24 位二进制表示是 000000001001011101100001，-2 用 8 位二进制补码表示是 11111110. 于是，387.53 表示成00000000100101111011000011111111 0.

其他实数的浮点表示分别是

0	00000000000000000000000000000000
100	00000000000000000000000100000010
5	00000000000000000000000101111111
.000005	00000000000000000000000101111010
12000	00000000000000000000010000000011
−387.53	11111111011010001001111111111110
−12000	11111111111111111111010000000011

浮点记数法的优点是，既可用来表示绝对值很大的数，也可用来表示绝对值很小的数．例如，上面的记数表示，能够表示的最大数为 $(2^{23}-1)\times 10^{127}$．这个数确实很大；最小数为 10^{-128}，它是非常小的数，在一台特定机器上能够表示的数的精度受到限制，其限制因素是该机的尾数的有效二进制数的位数．并不是说，最大数和最小数之间的每一个数都能表示．我们的表示法只允许 23 个有效位．这样，像 1 千万零 1 这个数，尾数需要 24 个有效二进制数，我们不得不近似表示成 1 千万 (1×10^7)，它只需要一位有效数字．

1.1.3 字符串

众所周知，信息并不都总为数值型的，像姓名、职务名称、地址等一些内容，在计算机里就必须按一定方式表示．为了能够表示这类非数值对象，还需要有解释二进制位串的其他方法．一般来说，这类信息是用字符串的形式表示的．例如，在某些计算机里，用 8 个二进制位 00100110 表示字符 "&"．用 8 个二进制位的不同组合来分别表示 "A"，"B"，"C"，以及表示一台特定计算机所具有的每一个字符． 苏联的计算机，用二进制位的不同组合格式表示俄文字符，而以色列的计算机却用二进制位的不同组合格式表示希伯来(Hebrew) 文字符(事实上，所用字符对机器都是透明的，字符集随着字符发生器或打印机的不同而改变)．如果用 8 个二进制位来表示一个字符，由于 8 个二进制位的不同组合格式只有 256 个，故可表示的字符最多 256 个． 如果用二进制位串 01000001 表示"A"，用 01000010 表示"B"，二进制位串 0100000101000010 就表示字符串"AB"．一般来说，如果把表示 STR 单个字符的二进制位串连起来，就可表示字符串 STR．

与整数情况一样，一个特定的二进制位串表示一个特定的字符，并非一成不变，二进制位串与字符间的分配关系完全是任意的．但是分配关系一旦被确定，就不能再变．在二进制位串与字符的分配上，可以使用一些简单的规则．例如，把两个二进制位串分配给两个字母，使二进制值小的一个分配给字母中靠前的一个字母．然而，这样的规则只是为了方便，它对字符和二进制位串的固有关系没有任何约束力．事实上，计算机不同，用来表示字符的二进制位的位数也不同．有些计算机用 7 位 (所以，至多允许表示 128 个字符)，有的用 8 位(最多可表示 256 个字符)，有些计算机则用 10 位(最多表示 1024 个可能字符)．一台特定计算机表示一个字符所需的二进制位数称为字节宽度 (byte size)，而这一组二进制位组成的那个数称为一个字节．大部分微机中的字节宽度为 8 位．

注意，用 8 个二进制位表示一个字符时，即意味着可以表示 256 个字符．我们很难找到一台使用这么多不同字符的计算机 (尽管可以想象计算机中含有大小写字母、特殊字符、斜体、黑体和其他类型的字符，并且许多个人计算机还把 256 个编码中的一部分用于图形字符)，因而，8个二进制位编码的大部分没有用来表示字符．大部分编码不用来表示可显示或可打印的字符，而是作为通讯或 I/O 设备的控制码．

大多数微机用 ASCII 码表示内部字符． ASCII (美国信息交换标准码)是一个标准体制，计算机制造商用它来表示各种字符和符号，以便使一个公司生产的计算机能与另一个公司生产的打印机(或另一台计算机)通讯．

这样，我们看到信息本身并没有意义．我们可以把任何一个意义赋于一个特定的二进制位的组合 (bit pattern)，只要这种做法保持不变，赋于二进制位组合的意义就是对

二进制位组合的解释．例如，二进制位串 00100110 可以解释成数 38（二进制数）、数目字 26（二进制编码的十进制数）或字符“&”．对二进制位组合的解释方法通常称为数据类型（data type）．我们已经给出了几种数据类型：二进制整数、二进制编码的十进制非负整数、实数、字符串．关键的问题是确定用什么数据类型解释二进制位的组合是合适的；确定在解释一个特定的二进制位的组合时，使用哪一种数据类型．

1.1.4 硬件和软件

计算机的存储器（memory，storage，或 core）是一组简单的二进制位（开关）．在计算机操作的任何瞬间，存储器里的任何一个特定位不是 0 就是 1（开关断开或接通）．一个二进制位的设置情况，称为该位的数值（或该位的内容）．

在计算机存储器里，把位组合成较大的单位，例如字节．在某些计算机里，又把几个字节组合成更大的单位，叫做字．每一个这种单位（字节或字随机器而定）都分配有一个地址（address），所谓地址就是在存储器的所有单元中识别某一特定单元的名字．地址一般都是数值型的，因此，可以称为 746 字节或 937 字．地址常常称为单元（location），单元的内容就是组成该单元的二进制位的数值．

每一台计算机都有一组“固有的”数据类型．也就是说，计算机本身是由用某种方式处理二进制位组合的机制构成的，其处理的方法与位组合所表示的对象一致．如假定计算机有一条指令，该指令将两个二进制整数相加，并把两数之和放入存储器内一给定的单元，以备今后使用，那么计算机里就应具有下面这样一种机制：

1）从两个给定单元里取出两个操作数的位组合；

2）产生第三个位组合，它代表一个二进制整数，该数表示了两个操作数的二进制整数之和；

3）把所得结果的位组合放入指定单元．

计算机之所以“知道”把给定单元里的位组合解释成二进制整数，是因为执行该特定指令的硬件已被设计成这样之故．这与电灯的情况类似，当开关处于一特定位置时，电灯也“知道”它处于接通位置．

如果在同一计算机里还有一条把两个实数相加的指令，那么就应有一个单独的机内机制，把两个操作数解释成实数．这两条不同的指令，都需要两个操作数，而且每一个指令除了带有它们的明确的单元地址之外，本身还带有一个隐含操作数的类型识别．所以，程序员的责任就是了解每一个所用单元里所含的数据是哪种类型，并选择适当的指令（即要获得两数之和，是选择整数加法还是选择浮点数加法）．

高级编程语言有助于这项工作的进行．为使程序员方便，访问特殊的存储器单元，使用标识符或变量名，而不用数字地址．在 BASIC 里，我们把标识符写成字母打头的字母数字序列（注：尽管许多 BASIC 版本中允许变量名任意长，但在某些 BASIC 版本中只有前两个字符才有效，因此，变量 SUB，SUM 和 SU 将被当成同一个变量处理．此外，绝大多数 BASIC 版本，变量名的选择都有严格的限制，在此限制下，变量名不能含有嵌入的“保留字”，例如，变量名 BEFORE 就是不允许的，因为它含有保留字 FOR．其他版本的 BASIC，把变量名全部限制为仅有两个字符，我们将在 2.1 节中作进一步的讨论）．

如果一个 BASIC 程序员写出如下语句:

```
10    DEFINT  X,Y
10    DEFDBL  A,B
```

那么,以字母 X,Y 开始的任何一个变量都被看为一个整数,而以字母 A,B 开始的任何一个变量则看为双精度实数(即,双倍长尾数的浮点数).于是,我们把放在 XVAR 和 YVAR 单元里的内容看作是整数,而把放在 AVAR 和 BVAR 里的内容看作实数,负责把 BASIC 语句转换成机器语言的解释程序,将把语句

```
100    X = X + Y
```

里的"+"号翻译成整数加法,而语句

```
200    A = A + B
```

里的"+"号则翻译成实数加法.像"+"号这样的算符,根据它的上下文内容有几种不同的含义,所以是一个一般的运算符.解释程序使程序员得到了解脱,程序员不必再检查上下文和采用的相应版本,来规定所必须执行的加法类型.注意,在某些 BASIC 方言里(即 Applesoft 语言)*,只能用把"类型说明"字符加到变量名上的方法来规定变量的"类型",因此,X$ 即表示一个字符串变量,而 X% 则表示整型变量.许多 BASIC 语言(TRS 80-II),除了允许用"类型说明"字符作类型说明外,还允许用语句 DEF 作类型说明.进一步的讨论,见 2.1 节.读者必须弄清楚在所用的 BASIC 语言设计里,类型说明使用的是什么方法.

高级语言中判明类型规定这一关键问题是十分重要的.程序员将借助这些说明,规定程序如何解释计算机存储器里的内容.在进行数据类型说明的过程中,还要规定具体实体需要多大的存储空间,该存储器内容怎么解释,等等其他细节.同时该说明也要把后面所用操作符的含义给解释程序作出准确规定.

1.1.5 实现方法的概念

迄今为止,我们一直把数据类型看成解释计算机存储器中内容的一种方法.一台特定计算机到底支持哪些类固定数据类型,取决于这台计算机的硬件功能.然而,我们可以不从计算机能做什么,而从用户想做什么这一截然不同的观点来观察分析"数据类型"的概念.例如,一个人想得到两个整数相加之和,而他们对于用来得到这个和的机制细节并不很关心,相反感兴趣的是处理一个整数的数学概念,而并不是处理硬件的数字位.计算机硬件可以用来表示一个整数,而且只有在表示方法成立时才有用.

一旦"数据类型"这一概念脱离了计算机硬件的能力,可考虑的数据类型数目就会有无穷多种,数据类型是一个由一组逻辑特性定义的抽象概念.一旦定义了这种数据类型,并规定了涉及此数据类型的合法操作,我们就可以"实现"这种数据类型(或与之相近的数据类型).实现方式可以是硬件的实现,也可以是软件的实现.在硬件实现里,要设计执行所需操作的电路,并将此电路作为计算机的一部分装入计算机;而在软件实现里,要求写出一个由现有机器硬件指令组成的程序,该程序能按所要求的方式对数字串予以解释,并执行所需操作.因此,一个软件实现,将包括一个如何用原有的数据类型对象来表示新的数据类型的规范,以及一个如何根据为它定义的操作来控制这一对象的规范.本书后面

* BASIC 方言指非标准的 BASIC 版本.——译者注

所到的术语“实现”都是指“软件实现”。

1.1.6 实例

下面我们用一个例子来说明这些概念。假如一台计算机硬件有如下一条指令：

MOVE [*SOURCE*（源），*DEST*（目的），*length*（字长）]。该指令把 *length* 个字节的定长字符串从 *SOURCE* 规定的源地址放入由 *DEST* 规定的目的地址里。我们用大写斜体字母表示硬件指令和单元。长度必须规定为一整型常数，为此，我们用小写字母来代表，而用表示存储单元的标识符来规定 *SOURCE* 和 *DEST*。这种指令的一个例子就是 *MOVE* (*A*,*B*,3)，意思是把单元 *A* 开始的三个字节拷贝到单元 *B* 开始的三个字节之中。

注意，标识符 *A* 和 *B* 在这种操作中所起的作用不同。由标识符 *A* 指定的单元的内容是 *MOVE* 指令的第一操作数。但是，第二个操作数却不是单元 *B* 的内容，因为这些内容与该指令的执行无关。相反，该单元本身就是操作数，因为该单元规定了字符串的目的地址。尽管一个标识符总代表一个单元，但通常用标识符方式来访问那个单元的内容。无论标识符访问的是单元还是单元的内容，只要从上下文看，总会看出来的。作为指令 *MOVE* 第一个操作数代表的标识符是表示访问存储器的内容，而作为第二个操作数代表的标识符访问的是一个单元。

我们还假定计算机硬件含有常用的运算指令和转移指令，它们可用与 BASIC 形式类似的表达式来表示。例如，指令

$$Z = X + Y$$

把在单元 *X* 和 *Y* 里的字节内容解释成二进制整数，计算它们的和，并将此和的二进制表示送入 *Z* 单元的字节之中（我们不能对长度大于 1 个字节的整数进行运算，并忽略可能出现的上溢）。这里，我们再把 *X* 和 *Y* 用来作为访问存储器的内容，而用 *Z* 来表示访问存储器的单元，而从上下文看，原来的解释是正确的。

有时，我们希望在一个地址上加一个量，以便得到另一个地址。例如，如果 *A* 是存储器中的一个单元，我们想要访问 *A* 单元后 4 个字节处的一个单元。我们不能把这个单元当作 *A* + 4 来访问，因为 *A* + 4 这个字符专门用来表示单元 *A* 的整数值和整数 4 的和。所以，我们为了访问这个单元，引入了用符号 *A*(4) 表示的方法。同样，对把位于 *X* 字节的二进制整数值加到地址 *A* 所得到一个新地址，为了访问这个新地址，我们引入符号 *A*(*X*)。

同前面定义一样，*MOVE* 指令也要求程序员规定所传送字符串的长度。因此，这条指令所涉及的操作数是一个定长的字符串（即字符串的长度必须是已知的）。我们可以把定长字符串和字节型的二进制整数看成是一台特定机器的固有数据类型。

假如我们希望在这台机器上实现变长字符串操作，即我们要求程序员应使用如下指令：

MOVEVAR (*SOURCE*, *DEST*)

把一个字符串从单元 *SOURRCE* 移至单元 *DEST*，而不需要规定任何长度。

为了实现这种新的数据类型，我们必须首先确定这种数据类型在计算机存储器中如何表示，然后，再说明该表示是如何处理的。很清楚，要执行这条指令，必须知道要传送的

字节有多少. 由于 *MOVEVAR* 操作并没规定此数,因此,字符串表示法本身就必须含有这个数.一个长度为 *l* 的变长字符串,可以用 *l*+1 个相邻的字节表示 ($l < 256$). 其中,第一个字节存放长度 *l* 的二进制表示,余下各字节用来存放该字符串中各字符的表示.图 1.1.2 中说明了这种字符串的三种表示法. 注意, 这些图中的数字 5 和 9 并不是代表"5"和"9"字符的位组合, 而是表示整数 5 和 9 的位组合 00000101 和 00001001. 同样, 图 1.1.2(c)中的 14 代表位组合 00001110.

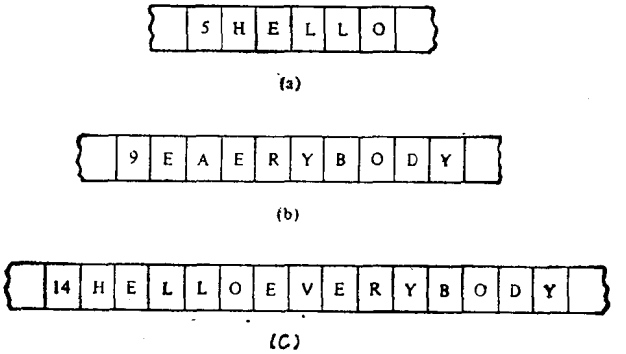


图 1.1.2

实现 *MOVEVAR* 操作的程序可以写成(*I* 是一个辅助存储单元):

```
forI = 1 to DEST
    MOVE(SOURCE(I),DEST(I), 1)
nextI
```

同样, 我们可以实现操作 *CONCATVAR* (*C*1, *C*2,*C*3), 该操作把存放在 *C*1 和 *C*2 两个单元里的变长字符串连接成一个字符串, 并将结果放在单元 *C*3 中. 图 1.1.2(c) 即为图 1.1.2(a) 和(b) 中的两个字符串的连接结果.

```
'move the length
Z = C1 + C2
MOVE(Z, C3,1)
'move the first string
forI = 1 to C1
    MOVE(C1(I), C3(I),1)
next I
forI = 1 to C2
    X=C1 + I
    MOVE(C2(I), C3(X), 1)
next I
```

然而,一旦操作 *MOVEVAR* 被定义, 我们就能用它来实现 *CONCATVRR* 操作:

```
MOVEVAR(C2, C3(C1)):    'move the second string
MOVEVAR(C1, C3):        'move the first string
Z=C1 + C2:              'update the length of the result
MOVE(Z, C3,1)
```

图 1.1.3 示出了对图 1.1.2 字符串进行操作的各个阶段. 尽管后面这个方法的程序短一