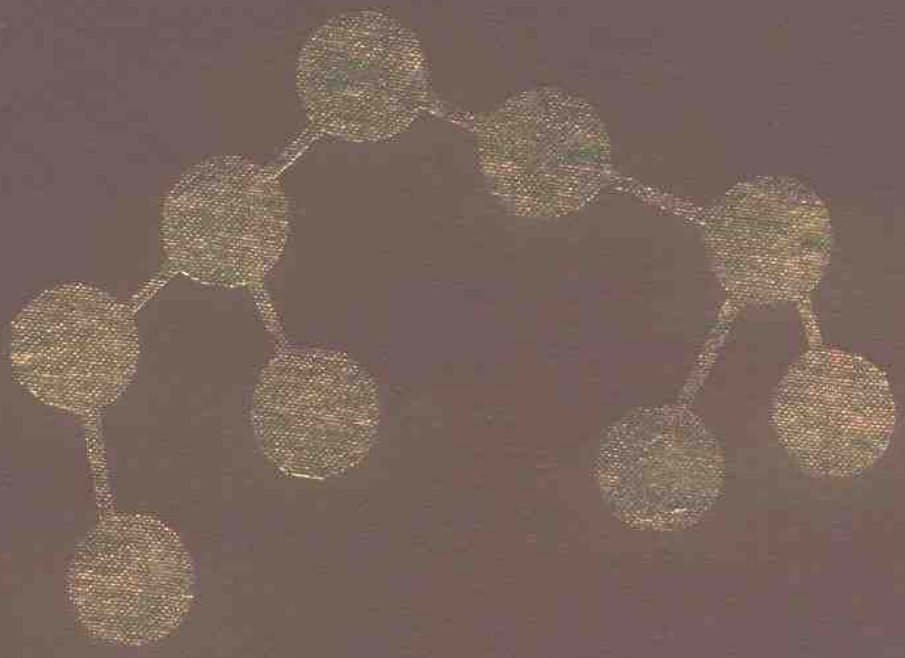
數據結構教程

嚴蔚敏 編著



DATA STRUCTURES

清華大學出版社

数 据 结 构 教 程

张少润　编著

厦 门 大 学 出 版 社

数据结构教程

张少润　编著

厦门大学出版社出版
福建省新华书店发行
厦门大学印刷厂印刷

开本787×1092　1/16　17.75印张　410千字
1988年2月第1版　1988年2月第1次印刷
印数：1—3000册
ISBN　7—5615—0055—0
O·3　定价：6.80元

前　言

数据结构是研究数据元素之间组织关系的学科，是计算机科学的核心课程之一。它包括数据的逻辑结构和数据的物理结构，虽然没有统一的意义，但本书把数据结构抽象定义为：数据结构是一个二元组 $B=(K,R)$，其中 K 是数据（结点）的集合，而 R 是 K 上关系的集合。由此定义出发，介绍抽象的基本概念，通过解问题阐述这些概念的用途，然后使用程序设计语言说明这些抽象概念如何变成具体的东西。抽象概念和具体东西的适当结合是为了使学生认识概念本身，它的实现和它的应用。

几年来的教学实践证明了，本书用流程图来描写多数算法，既清晰易懂，又容易改编成用其它高级语言书写的程序，对培养学生的自学能力很有好处。

书中使用扩充 PASCAL 语言，即在 PASCAL 语言的基础上吸取若干易于描述算法的 PL/1 语句。这是考虑到：它包含使程序容易阅读所必需的控制结构；允许使用多种方法执行诸如栈、链接表、树等数据结构；允许学生在现实情况中懂得选择和比较研究；有学过、使用 FORTRAN 或 PASCAL 进行程序设计的学生能较快看懂书中的程序，只要稍加修改就可变成能上机实现的 PASCAL 程序等。

本书共有九章。第一章是数据结构导论，指出本书从信息的观点来研究计算机科学，介绍什么是数据结构，并对算法作简单描述。第二章讨论串的基本概念及其运算，串的逻辑特性和物理表达。第三章讨论基本的且重要的线性数据结构——并列表。第四章和第五章含括有树和图等非线性数据结构。第六章讨论基本的存贮管理结构，包括废料收集等。第七章和第八章的内容是分类和查找。第九章是文件结构。

本书是根据1983年教育部颁发的部属高等学校（工科、综合大学）计算机软件专业《数据结构》教学大纲，结合本人在美国进修的收获和体会，总结几年来的教学实践经验全面介绍了各种数据结构及其实现方法。

本书按70学时编写，对某些不同层次的对象，可以免讲部分内容。书中各章都配有练习题，一并在书后给出。另外，书末列有一批参考书目供进一步阅读。

衷心地感谢国防科技大学陈火旺教授对本书编著工作的鼓舞和宝贵指导。同时也感谢我系不少教师、研究生和本科生提出的许多有益的意见、建议和为本书的出版所做的大量工作，其中有叶仰明、吴克西、吴辉荣、孙少斌、苏进安、欧阳幼文、林云、张莹等等。

由于编者水平有限和修改时间匆促，谬误之处难免，敬请读者正之。

编　者　1987.1

内 容 简 介

本书根据一九八三年教育部颁发的教学大纲编写而成。

书中以通俗易懂的语言全面介绍了各种数据结构以及实现方法，既注意了经典的数据结构，也反映了七十年代后期以来国内外的最新研究成果。全书共分九章，内容包括：数据结构导论、串的基本概念及其运算、线性数据结构——并列表、树和图等非线性数据结构、基本存贮管理结构、分类和查找、文件结构等。

本书结构清晰，循序渐进，易于自学，各章均配有习题，可作为高等（理、工）院校计算机科学及工程专业的教材，也可作为教师、研究生或软件工程技术人员的参考书。

目　　录

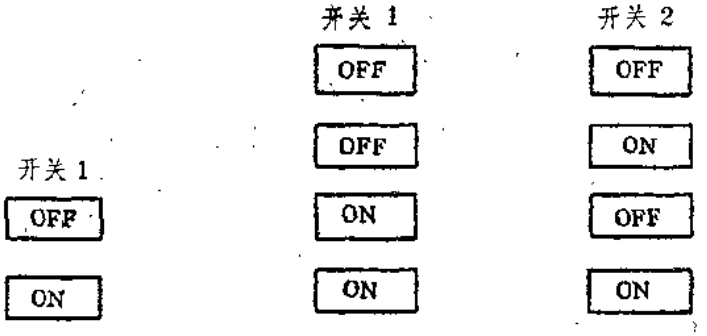
第一章　数据结构导论

什么是计算机科学？这是几十年来计算机科学家们感兴趣的问题之一。有的认为，这是研究数据（data）的科学，有的认为，这是研究算法（algorithm）的科学。目前，多数人主张应把计算机科学看成是研究信息（information）的科学。因此，计算机系的学生理解和掌握信息组织和处理的概念是特别重要的。

1.1 信　　息

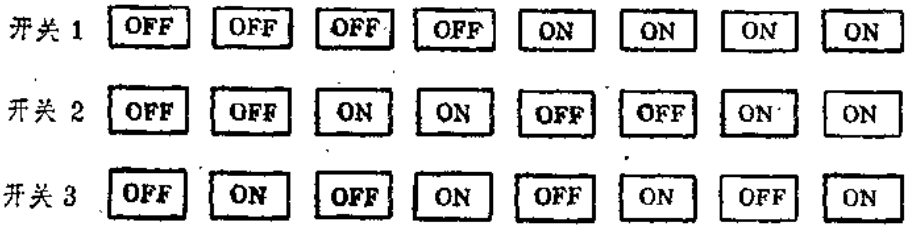
1.1.1 信息表示

我们可以把信息视为计算机科学领域所探索的基本事实，但却不便用句子或更基本的概念来定义它。然而，我们可以测量信息的数量。信息的基本单位是位（bit—"binary digit"的缩写），它表示相互排斥的两种概念之一。例如，电灯开关可以处于开或关两种状态之一，但不能两者同时具备。又如，若圆盘有8种可能位置，则它处于某种位置时，就排斥了其它7种可能的位置。现在，假设我们有二路开关且可以尽可能多地使用它们，那么表达具有8种状态的圆盘需要多少开关呢？很清楚，一个开关只表达两种情形（图1.1.1(a)），两个开关可以表达4种情形（图1.1.1(b)），三个开关可以表达8种情形（图1.1.1(c)）。一般地说，n个开关可以表达2^n种不同情形。



（a）一个开关（两种情形）

（b）二个开关（四种情形）



（c）三个开关（八种情形）

图1.1.1　开关状态

人们用二进制数字0和1来表示一个特定位（bit）上的画种情形，进而用由几个0和1组成的n位来表示各种情形。这样，三个位的各种组合（000、001、010、011、100、101、110和111）用来表示0到7这几个整数。而字符串11010110表示八个开关，从左边算起，其第1个和第2个开关处于“开”（1）状态，第3个开关处于“关”（0）状态，…，而第8个开关处于“关”状态。

值得提出的是，这些组合没有什么内在的特定要求，只要不存在两个不同整数对应于同一位组合即可，用这些组合来表示整数值的任何分配方案都是可取的。一旦这种分配方案确定后，特定组合唯一地表示一个特定的整数。

1.1.2 二进制和十进制整数

把各种位排列当为非负整数的最广泛使用的方法是二进制系统。在这个系统中，每一个位所在的位置表示一个2的幂，最右边位置的位表示2^0，即1；其左边隔壁位表示2^1，即2；再下一个左边表示2^2，即4；余此类推。因而可用2的各位幂的总和来表示一个整数。一个位串的所有位都是0，则表示数0。如果某一个特定位的位置出现1，其位所表达的2的幂包含于总和中；如果这特定位的位置出现0，则其2的幂是不含于总和的。例如在位串11010110中，位置1、2、4、6、7（计算从右边算起，最右位置记为第0位）是1，则11010110表示为整数$2^1+2^2+2^4+2^6+2^7=214$。任何一个位串表示一个唯一的非负整数，并且可用一个最小长度的位串表示一个非负整数。

然而，二进制数系统决不是用位串来表示非负整数的唯一方法。例如，可以用位串来表示十进制系统中的非负整数。在上述二进制记法中，可以用四位表示0至9之间的一个十进制数。可以把任一长度的位串分割成四个数字的连贯的集合，每个集合表示一个十进制整数。这样，这个字符串就表示了一个按通常十进制记号所写的十进制数字所组成的数，例如，在这个系统中，位串10010110分割成每个集合是四位的两串：1001和0110。第一串表示十进制数字9而第二串表示十进制数字6，因而整串表示整数96。这种表示称为二进制代码的十进制数。但是，在这种表示法中，并不是所有位串所表达的“十进制整数”都是合法的。那些比9大的值，诸如1010和1100在二进制代码的十进制数中都是非法的。

1.1.3 字符串

众所周知，名字、工作职称、地址等信息，是不能用数字来表示的。为了使得这样的非数字对象的表示有可能，就有必要用其它方法来解释位串，这样的信息通常用字符串的形式表示。如果用8位来表示字符，则可用串11000000表示字符‘A’，而11000001表示‘B’，则字符串‘AB’应当用1100000011000001表示。一般地说，字符串STR是由表达STR中的个别字符的位串连接起来表示的。

请注意，能用于表达一个特定字符的位串没有任何内在的关系。分配用来表示字符的位串是任意的，但应当是保持不变的。在分配位串表示字符时，应考虑某些方便的规则，例如分配两个位串表示两个字符的时候，可以让较小二进制值表示按字母顺序较早出现的字符。然而，这仅仅是为了方便，它是不受字符和位串之间的内在关系管辖的。某些计算机使用七位（因而允许可能有128字符），有的使用八位（因而有256字符），

有的使用十位（共有1024可能字符）。在一个特定的机器中，表达一个字符所需的位的数目称为字节大小，而这个数目的位组称为字节。事实上使用这么多不同字符（虽然它可以用来表示大小写字母、特殊字符、斜体字、黑体字或其它类型字符）的计算机是很少的，即许多种八位代码不被用来表示字符，因而出现信息的“浪费”。

总之，信息本身是没有意义的。我们可以指定一个意思给一个特定的位模式，只要在执行中始终保持不变即可。这就是位模式所具有的意义和解释。例如，位串01000000可可视为数 40（二进制代码的十制数），也可以视为数 64（二进制数）或者视为字符 串‘@’（ASCII码），解释位模式的方法通常称为数据类型。前面我们已经提出了三种数据表型：二进制非负整数，二进制代码的非负十进制数和字符串；在PASCAL 语言中，还有指示器类型等，关键的问题是如何确定数据类型所能解释的位模式以及哪种数据类型使用于解释一个具体的位模式。

1.1.4 硬件和软件

计算机存贮器可看作是位（开关）的集合。在任何计算机运算情况下，存贮器任一特定位或者是 0 或者是 1（关或开），一个位的情况称为它的值或者它的内容。

计算机存贮器中的位组合在一起成为更大的单位，如字节（在某些计算机中，几个字节的组合称为字的单元）。每个这样的单元（字节或者字）都分配给一个地址，它是一个在所有存贮器单元中识别一个特定单元的名字。这个地址通常是数字，以便我们可以说字节100或者字200。地址通常称为存贮单元（location ）。一个存贮单元的内容是在那个存贮单元里组成这个单元的位的内容。

每台计算机都有一套“自然”的数据类型，这意味着用一种方法来建造计算机，使得它在处理位模式时能使位模式与它们所表达的对象相一致。

例如，假设计算机包含一条把两个二进制数相加的指令，然后把其和放在存贮器的给定存贮单元供下一使用，则有一种方法能使计算机做到：

1、从两个给定存贮单元中截取位模式；

2、产生表达二进制数的第三个位模式，这个二进制数是由两个运算数所表达的两个二进制数的和；

3、在一个给定存贮单元里存贮结果位模式。

计算机软件按照设计要求执行那个特定指令，“懂得”在给定存贮单元把位模式解释为二进制整数。这与当开关处在特定位置时，电灯“知道”亮的情况是类似的。

如果这台计算机有一条把两个二进制代码的十进制数相加的指令，则它也有一个独立的内在方法把运算数视为十进制数。对于这两种运算，需要两条不同的指令，并且每条指令本身既执行了运算类型的不明显的标识符，也执行了它们明显的存贮单元的内容。

因此，知道在所使用的每个存贮单元里包括有哪种数据类型是程序设计者的职责。他也有责任来选择使用二进制或十进制加法指令以得到两数的和。高级语言能完成这个任务。例如，如果 PASCAL 程序设计者阐述。

```
VAR     x, y: integer;
        a, b: real;
```

则用四个存贮单元的空间来存贮四个不同的数。这四个存贮单元可以由标识符 x，y，a和b来访问。为了方便，使用这样的标识符来代替所指的特定存贮单元的数值地址。

x和y的存贮单元的内容将解释为整数，而a和b将视为实数。负责把 PASCAL 程序翻译成机器语言的 PASCAL编译程序将在句子

```
x: =x+y
```

中把加号“+”翻译成整数加法，而在句子

```
a: =a+b
```

中将把加号“+”翻译成实数加法。诸如加号“+”这样的运算符是一般的运算符，因为根据前后关系，它有若干种不同的意思，编译程序使得程序设计者用上下文关系和使用恰当方法有可能阐述所应执行的加法类型，象

```
y: =x;
```

这样的句子仅仅包括从存贮单元x中把位模式搬到存贮单元y中，象

```
y: =b;
```

这样的句子包含有把b 中的实数的位模式转换为 y 中的用整数表达的同一数的位模式，尽管编译程序能自动地注意到所有这些细节，但有时也会发生错误。

在 PASCAL 语言中，认识由 VAR 语句所扮演的角色是非常重要的，正是借助于 VAR 的“说明”，使得程序设计者能够阐述计算机存贮器的内容是如何由程序来解释的。在做这些工作的时候，“说明”阐述了对一个具体实体（例如，数值变量或字符串长度的精确性）需要多少存贮单元，如何解释那个存贮的内容（是作为整数或者实数或者作为字符串）以及其它具体的细节，VAR 语句也阐述了编译程序后来所使用的运算符号的确切定义。

1.1.5 实现概念

上面已把数据类型看作解释计算机存贮器内容的方法，一种计算机所支撑的自然数据类型集由计算机硬件所具备的功能来决定。然而，我们可以从完全不同的两个侧面来看“数据类型”这个概念；或者根据计算机所能做的，或者根据用户所要做的。例如，如果想要得到两个整数的和，他不必注意得到这个和的详细方法（处理硬件的位），而是考虑如何处理“整数”这个数学概念。

一旦“数据类型”从计算机硬件能力分离出来后，就可以考虑一大批数据类型了。什么是数据类型呢？数据类型是由一整套逻辑特性所定义的一个抽象概念。一旦定义了这样的抽象数据类型，并且定义了包含这种类型的合法运算后，我们可以实现这个数据类型（或者一个极其近似的值）。一个实现过程可能是硬件的实现，指设计并构造实现所需运算的必需线路；也可能是软件的实现，指写出已经存在的硬件指令组成的程序，按照所需的型式解释位串并执行所需的运算，这样，软件实现包括由先前存在的数据类型如何表达新数据类型的对象的说明，也包括如何处理这样的对象与此所定义运算相一致的说明。本书今后谈“实现”这一术语意指“软件实现”。

1.1.6 一个例子

假设某计算机硬件包含指令

move (from, to, length)

它把由 from 所指出地址的具有 length 长度的字符串复制到由 to 所指出的地址。这 length 用一个整数给出，from 和 to 由表达存贮单元的标识符说明。作为一个例子，move(a, b, 3)表示把从存贮单元 a 开始的三个字节复制到从存贮单元b开始的三个字节中去。

在这个运算中，move 指令的第一个操作数是标识符a 所表明的存贮单元的内容，而其第二个操作数不是存贮单元b的内容，它表明字符串的目的地。

现在假定计算机硬件包含通常的算术指令和分支指令，我们使用象 PASCAL 那样的符号来指明。例如指令

z: =x+y;

把在存贮单元 x 和 y 的字节的内容解释为二进制整数，相加后再把其和的二进制表达式送到存贮单元 z 这个字节（整数运算在长度上不超过一个字节并忽略上溢的可能性）。同时，在使用 z 访问存贮单元时使用 x和y 来访问存贮器内容。

有时，考虑要加个数量到某地址以求得另外一个地址。例如，a 是存贮单元内容，如果我们可能要访问超出 a 四个字节的存贮单元，我们不能把要访问的这个存贮单元看作 a+4，而应当使用记号 a(4)。

MOVE 指令处理了定长度字符串运算数（即字符串长度是已知的），定长度字符串和作大小排列的字节二进制整数可以考虑作为某特定机器的自然数据类型。

假设我们想要在机器不执行变长度字符串，即要使程序设计者不必阐述任何长度而能使用指令

movevar (from, to)

把字符串从存贮单元 from 搬到存贮单元 to。

为了执行这种新的数据类型，我们首先决定机器的存贮器是如何表示的，然后指出如何处理这种表示。很清楚，需要知道移动多少字节才能执行这条指令。因为movevar 运算不必阐明这个数目，这个数目应当包含于字符串本身的表达式。这样，长度为 l 的变长度字符串可由 l+1 字节（l<256）的连续集来表示。它的第一个字节包含长度 l 的二进制表达式，其余字节包含串中的字符的表达式。图 1.1.2 表示了三个这样的字符串的表达式。

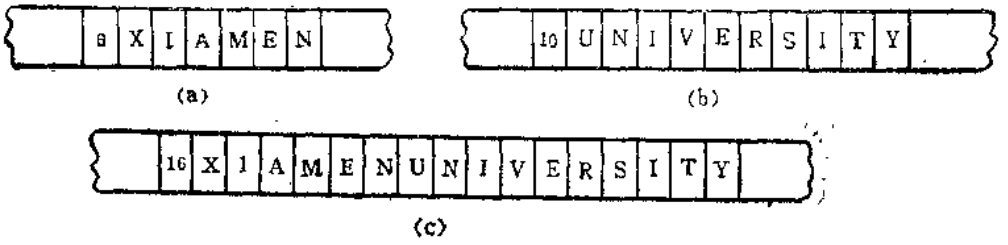


图1.1.2 变长度字符串

注意，图中 6、10、16不代表表达'6'、'10'和'16'的字符的位模式。如果一个字节用八位表示的话，表达这些数的位模式是00000110、00001010和00010000。

执行 movevar 运算的程序可以写为

```
move ( from, to, 1 );
FoR  i: =1 To  to  DO
move ( from (i), to (i), 1 );
```

其中 i 是辅助变量。

类似地，我们可以执行运算 concatuar (C1, C2, C3) 把存贮单元 C1和C2 的两个变长度字符串连结起来并把结果放到C3。在图1.1.2 中 (c)表示了(a)和(b)的这种连结。

```
(* MOVE  THE  LENGTH *)
   z: =C1+C2;
   move ( z, C3, 1 );
(* MOVE  THE  FIRST  STRING *)
   FOR i: =1  TO  C1  DO
   move ( C1 ( i )  C3 ( i ),1 );
(* MOVE  THE  SECOND  STRING *)
   FOR  i: =1  TO  C2  DO
   BEGIN  x: =CI+i;
          move ( C2(i), C3(x), 1 )
   END;
```

总之，一旦定义了运算movevar 后，concatuar 可以使用 movevar 来执行，形式如下：

```
(* MOVE  THE  SECOND  STRING *)
   movevar  ( C2, C3 ( CI ) );
(* MOVE  THE  FIRST  STRING *)
   movevar  ( C1, C3 );
(* UPDATE  THE  LENGTH  OF  THE  RESULT *)
   z: =C1+C2;
   move ( z, C3, 1 );
```

图1.1.3给出了图1.1.2字符串的运算状态，虽然后者显得短些，但不是更有效的。因为每次使用 movevar 时，使用于执行 movevar 的所有指令都得执行。

上述算法中的语句z: $=C_1+C_2$; 是有特别意义的。此加法指令的运算与操作无关（此时作为变长度字符串的一部分），它把操作数视为单字节整数而不管程序设计者是否作为它用。类似地，访问C3(C1)是指对这样的存贮单元，它的地址由存贮单元C1 的字节的内容和地址C3相加而得的。

注意，变长度字符串表达式仅仅允许字符串长度少于或等于一个单字节所能适应的最大二进制整数。如果一个字节是八位的，表示其最大字符串长度是255 (即2^8-1)，为符合这最大字符串长度，应当选择不同的表达式。

一旦给特定的数据类型对象选择了表示法并且写出了用这些表示法来运算的程序，

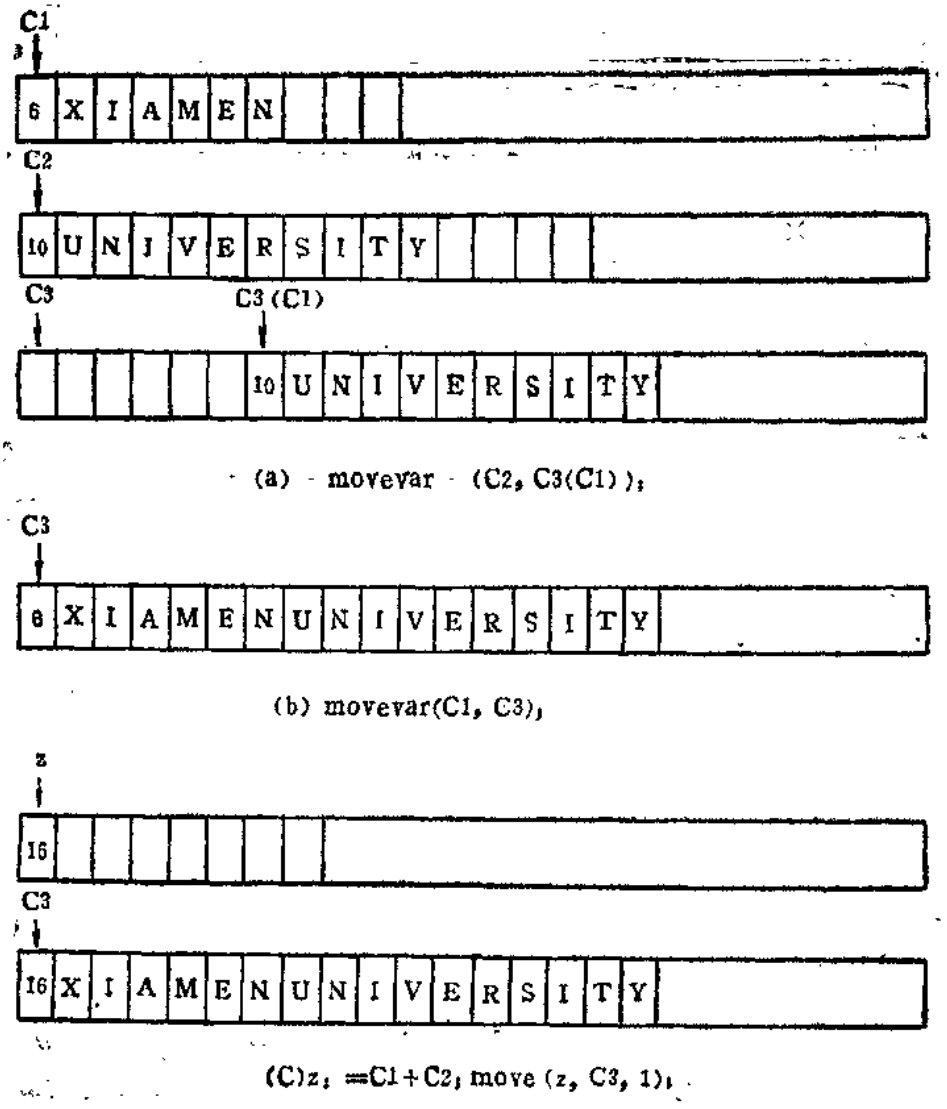


图1.1.2 concatvar 运算

程序设计者就能运用自如地使用各种数据类型来解问题。可以把机器原来的硬件加上执行比硬件提供的更复杂的数据类型的程序看作为比仅由硬件组成的机器“更好的”机器。原来机器的程序设计者不必考虑计算机是如何设计的以及使用什么线路来执行每条指令，他只要知道可以有什么样的指令以及如何执行这些指令。类似地，使用“新”机器（由硬件和软件所组成）的程序设计者不需要考虑如何执行各种数据类型的细节。所有程序设计者需要知道的是如何处理这些类型以及对于这些操作哪些运算是可能的。

1.2 什么是数据结构

上节我们从信息的角度来理解计算机科学的概念，这也有助于理解作为计算机科学的核心课程——数据结构。

本节从实例出发介绍数据结构研究的范围。了解其发展概况，抽象出其本质，然后再给数据结构下定义。

1.2.1 数据结构研究范围

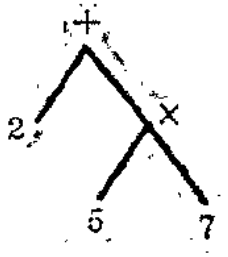


图1.2.1 表达式

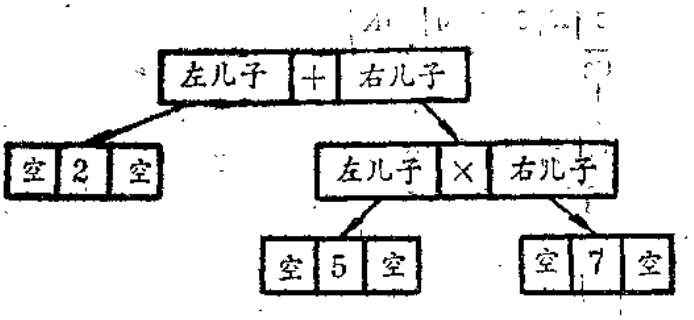


图1.2.2 图1.2.1树结构的存贮结构

“数据”是描述计算机处理的所有事物的通用表达形式，在硬件和机器代码上，所有数据均表示为二进制数字（位）的序列。

例如，某学生去商店购买一块2分钱的橡皮擦和五根7分钱的原子笔芯，其应付的人民币应由算术表达式 2 + 5 * 7 得来。算术表达 2 + 5 * 7 是从诸如 2，5，7 这样的整数所组成的数据成份，按照有规则的方法构成的，它进行了“+”和“*”运算。这个表达式结构可以看为一串字符或者一个树结构，如图 1.2.1 所示。图中每个运算符都是一个子树的根，子树的子孙是运算数。

当这个数据结构存贮在计算机时，应当满足的首要条件是容易存取。实现的办法可以把表达式 2 + 5 * 7 看成一字符串存贮起来，以便用元素 A[1] 或者 A(i)（视具体程序设计语言而定）来检索A中的第 i 个元素。还可以把字符串看为一个树结构存贮起来，这里顶端十有左儿子 2 和右儿子*，而*依次又有儿子 5 和 7，见图1.2.2。

值得指出的是，图1.2.1阐明了数据元素之间的逻辑关系，而图1.2.2指出了数字计算机如何实现这种关系的存贮结构。在计算机中，用五个三元素的存贮单元表达图1.2.2的存贮结构，这里每个存贮单元都包含一个运算符的一个元素和包含分别指出左，右儿子的两个指针元素，没有儿子的三个存贮单元，在其指针域中，用“空”字表示。这就表明了，数据结构就是研究数据的逻辑结构和物理结构以及它们之间的相互关系。概括地说，数据结构就是研究数据元素之间的组织关系，并对一种结构定义相应的运算，设计出相应的算法以保证经过这些运算后所得的新结构仍具有原来的结构类型。通常我们所提及的数组、记录、集合、线性表、树、栈和队，就是一些数据结构的例子。

1.2.2 数据结构简史

六十年代初，国内外尚未专门设立“数据结构”课程，一直到六十年代中期才开始设立有关课程，命名为“表处理语言”（List processing languages）。它主要研究若干个表处理系统。

六十年代后期，美国 Association for Computing Machinery 提出了大学计算机专业的课程设置，正式规定“数据结构”作为一门课程。著名计算机科学家 Knuth，