数据结构教程

窦延平　霍义兴
汤宝骥　杭诚方　编著

上海交通大学

数据结构教程

窦延平　霍义兴
汤骥宝　杭诚方　编著

上海交通大学出版社

内 容 提 要

本书是按照IEEE推荐的"数据结构"教学大纲编写的，参考了国外出版的数据结构方面的最新教科书及文献，并从ADT的角度来描述数据结构。全书引入了许多新的概念和内容，如抽象数据结构、软硬件对数据结构的支持、数据抽象、Ada对数据结构的支持、类属数据结构等。在材料组织上，按照数据抽象和ADT、专门的数据结构、算法与数据结构的关系等分门别类地进行阐述，具有一定的先进性、科学性和系统性。本书不仅可作大专院校计算机系学生的教学用书，也适合电类、数学、信息处理及自动控制专业作教材，对从事计算机工作的科技人员也有一定参考价值。

数据结构教程

出 版：上海交通大学出版社
（淮海中路1984弄19号）
发 行：新华书店上海发行所
印 刷：常熟市印刷二厂
开 本：787×1092（毫米）1/16
印 张：20.25
字 数：498000
版 次：1990年 3 月 第一版
印 次：1990年 4 月 第一次
印 数：1—2800
科 目：218—304
ISBN7—313—00636—5/TP·39
定 价：4.00元

前　言

"数据结构"是计算机科学技术专业的一门基础理论课程，是计算机系学生的核心课程之一。在计算机科学技术的各个领域，选择合适的数据结构是一个重要的问题。通过"数据结构"的学习，可以进一步提高软件设计与程序编写能力，增强选择合理的数据结构与有效的算法的技能。近年来，计算机科学技术已经强有力地渗透到科学技术的其他领域。因此，计算机科学技术的基础理论与基本知识，尤其是"数据结构"研究的解决非数值问题的理论和方法，就变得越来越重要，所以科技工作者和工程技术人员都应予以掌握。

本书是按照IEEE computer society educational activies board推荐的"数据结构"教学大纲编写的，参考了国外出版的数据结构方面的最新的教科书及文献。在选材上，力求做到先进性、科学性、系统性。在材料的组织上，按照由简到繁及循序渐进的原则，便于读者掌握理解"数据结构"的基本概念、基本理论与基本方法。近几年来，从ADT(抽象数据类型)的角度来描述数据结构是一股新的趋势。因此，从抽象数据结构的观点来看，最好采用能直接支持ADT的语言(如：Ada、Modula-2等)作为描述语言。由于国内缺乏Ada、Modula-2的实践手段，因此还是采用了Pascal语言。本书不仅适合于大专院校计算机系的学生作为教学用书，也适合于电类、数学、信息处理及自动控制专业采用。本书对于从事计算机工作的科技人员也有一定的参考价值。

本书初稿第一章、第六章至第十章由窦延平副教授编写，第二章至第五章由汤宝骥老师编写，第十一章由霍义兴教授编写。全书各章节在征求了各方面意见之后，由窦延平副教授进行了全面的整理、修改与定稿。杭诚方老师对某些章节进行了详细的整理与修改。由于水平有限，错误在所难免，敬请读者指正。

衷心感谢孙永强教授对本书出版的关怀和支持。上海交通大学计算机系的其他教师也提供了许多宝贵的意见和建议，在此表示感谢。

作　者

一九八八年十二月

目　录

第一章　数据类型和数据结构导论

1.1　引言

在本章中，我们将介绍有关数据类型和数据结构的基本概念、定义及常用的表示方法。在1．2节，我们讨论数据类型(data type)同数据结构(data structure)之间的关系，并回答什么叫数据结构这个问题。1.3节中所介绍的抽象数据类型(abstract data type，简称ADT)是一个重要的概念.从ADT的概念出发，研究数据结构是近几年来计算机科学界出现的一股新倾向。利用ADT研究数据结构的优点在于信息隐蔽、模块化，说明的精确性、简单性、完整性以及实现的独立性等。

1.4节讨论数据结构的定义及结构的含义，并特别注重数据结构在高级语言(如Pascal、Ada)中的表示和实现。1．5节从机器一级的角度来讨论数据结构的表示和实现问题，同时还介绍了"原子"数据类型(atomic data type)，如：integer，real，boolean等在机器中的物理存储器中的二进制位的模式。

为了便于理解，在本节中我们先讨论一个非常简单的例子。这个例子和以后我们要讨论的数据类型有某些相似之处。

取一打蛋，把每一个蛋看作不可分解的物体，即不考虑组成蛋的各种成份，如蛋壳、蛋黄，蛋白等等。这样，每个蛋都可以看成"原子"元素。我们这里所讲的原子和物理学中所讲的"原子"不同，我们仅用它表示不再分解的物体。当然，任何物体都是可以分解的。如一座桥梁可以分解为桥桩、大梁、螺栓等。但是，在驾驶员眼中，他所关心的只是这座桥梁能否通过，而不去关心该桥梁由什么组成。即在他眼中，这座桥梁是一个"原子"。而对于桥梁工程师而言，他要关心的是桥梁的各个组成部分，在他看来，大梁、桥桩才是真正的"原子"。因此，我们可以把物体或物体的集合看作为"原子"。在不同的人看来，原子具有不同的含义。

让我们仍然考虑这一打蛋。为了区别这些蛋，可以在每一个蛋上写一个号码，其范围从1到12。我们可以不去管这些蛋是如何存放在具体的容器之中，而想象它们按照号码的大小整齐地排列在空间。这实际上已经给这些蛋某种结构(即这些蛋相互之间有确定的关系)：即按照号码的小大排成一行。对于这些蛋，可以进行一系列操作，如把某个号码的蛋拿走；把这些蛋之中重量最大的蛋找出来，这就需要把每个蛋去称一下；或者把一打蛋全部带上船。我们可以定义原子物体的集合为抽象的物体结构，但该原子物体的集合必须满足：这些原子物体具有某种结构且可以进行指定的操作。这样，以上讨论的这些蛋由于具有按照号码排成一行的线性结构且可以进行上述操作，因此它们是一种抽象的蛋结构。

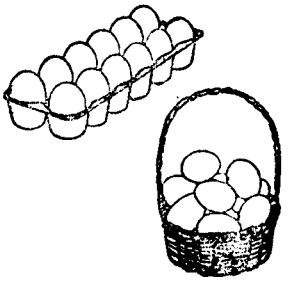
由于抽象的蛋结构仅仅存在于我们的想象之中，所以它是不真实的。虽然，它已经告诉我们可以在这些蛋上进行什么操作，但是并没有说明如何精确地进行。这样，为了精确地完成所指明的操作，必须使用实际的蛋和实际的设备(或工具)，所以，必须具有和抽象的蛋结构相应的物理的蛋结构，

首先，处理蛋的物理存储问题。我们可以用图 1.1 所示的纸板盒及篮子作为实际的存储媒介。选择图 1.1 下部所示的有三排位置的纸板盒存放这些蛋。这样，我们就可以通过简单的计算找到任意一个号码的蛋在纸板盒中的实际存放位置.如编号为k的蛋的存放位置是：

$$行 = (k-1)\ \mathbf{div}\ 4$$

$$列 = (k-1)\ \mathbf{mod}\ 4$$

这样，找到编号为 k 的蛋将是轻而易举的。另外，如果要把这一打蛋带上船，这种纸板盒也是很好的。可以在上面加一个盖子并放一些填充物，就可以安全地带走。



我们也可以使用篮子存放这些蛋。但是蛋存放在篮子中，总是乱七八糟而无一定的规律。所以要找到给定号码的蛋，必须把每个蛋的号码相比较，直到所给号码的蛋找到为止。另外，对于装船运输来讲，这种存储媒介也并不理想。

本书并不是研究如何存放蛋及对蛋进行操作的。我们感兴趣的是数据对象(data object)。与蛋的存储媒介是纸板盒、篮子等相对应，有关蛋的数据信息是存放在计算机的存储器之中的（如 RAM）取回或运送蛋是靠人和设备进行的，而对存储器中的蛋的数据信息的处理却是靠算法或程序完成的。算法完成这些操作的技术和人们处理蛋时的技术类似。我们可以想象数据原子具有某种结构而不管如何在具体的存储器中进行表示，同样可以定义在假想的结构.之中完成的某些操作.这实际上定义了一种抽象的数据结构.至于它的具体实现，可以留给另外一些人去完成。具体实现的某些细节完全可以被隐蔽在程序包或模块之中。在实际使用这些程序包或模块时，并没有必要了解这些细节。这样，仍然可以认为完成这些操作是按一种抽象的方式进行的。

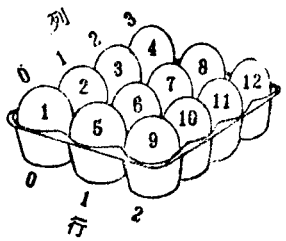


图 1.1 蛋的存储媒介

1.2 数据类型和数据结构

在定义数据类型和数据结构之前，我们首先给出数据值(data value)的概念。所谓数据值是作为一项来考虑的一组数据。当然，这一组数据也可能只有一个，如整数 593 就是一个数据值。

集合{3299，−1.04，0.395}整个地都可以作为一个数据值进行考虑。它可以分解为三个部分，每一部分是另一种类型的值(实数值)，它们之间的关系是同一个集合之中的数。如果一个数据值可以分解为几部分，那么每一部分则称之为组成元素或元素。32.99，−1.04，0.395 都是集合{32.99，−1.04，0.395}中的元素。

原子数据值也是数据值，不过该数据值不可以再被分解，它仅仅作为一个整体被考虑。如整数62953，就可以认为是一个原子数据值。当然，可以进一步分解该整数为6、2、9、5、3五个数字的集合。而每个数字，又可以分解为点的有规则的集合，不过作为一个数据值来讲，62953 是一个原子数据值。

1.2.1 数据类型

在回答什么是数据类型之前，首先看一下什么叫类型。类型的本质是识别一组个体共同

的特性，这些特性把该组个体辨认为同一类别。如果我们提供了可能的数据值以及作用在这些数据值上的操作的集合，那么这二者结合在一起就称之为数据类型。

数据类型通常分为两种。第一种是原子数据类型，即所提供的数据值是原子数据值。比如，在大多数程序设计语言中，integer(整数)就是一种原子数据类型。它的值是一系列连续的整数，即…，－2，－1，0，1，2，3，…，其操作包括：加法、减法、乘法、除法、求绝对值等等。

另一种数据类型是结构数据类型(structured data type)，或者称为数据结构.结构数据类型和原子数据类型不同，其数据值可以进一步分解为组成元素(Component element)，即数据元素。这些数据元素之间具有某种结构，换句话说，它们之间具有某些关系。由于数据结构是一种数据类型，所以它具有数据值及在这些值上操作的集合。另外，由于数据值可以分解为数据元素的集合，而数据元素也有可能进一步分解为原子数据值或者数据元素(这些数据元素之间也具有某种关系)。因此，也可以说，数据结构是一种数据类型，它的数据值可以进一步分解为数据元素的集合，该数据元素可以是原子数据值，也可以是另外一种数据结构。数据元素之间有一个关系(或结构)的集合。

数据结构的操作不仅可以作用在数据值上，它们同样可以作用在数据结构的数据元素或组成元素上。让我们看一个例子。

假定，有一种称之为 sample 的数据类型：

```
type  sample = array[1..3] of real
```

它的每一个值由排成一行的三个实数所组成。由三个实数组成的集合是 samp le 类型的一个数据值.由于这些实数是原子数据值，通过指定哪一个实数是第一个、第二个、第三个数据元素，而给它们确定了一种结构。因此，类型sample是一种数据结构。

数据结构 sample 的某些值在图 1．2 中给出。数据元素实数的下标指明了谁是第一个、第二个、第三个元素。

必须明白数据结构的每个数据值都有一个相应的结构.因此，即使它们有相同的组成元素，这二个数据值也未必相同。如：

值 A		值 B		value1	value2	value3
[1] 0.0		[1] 1.9		[1] 0.0	[1] 6.33	[1] 2.16
[2] 1.9	和	[2] 0.0		[2] 1.9	[2] 2.2	[2] 0.14
[3] 3.4		[3] 3.4		[3] 3.4	[3] 4.2	[3] 6.30

图 1.2 结构数据类型的某些值

并不相等。

让我们考虑 sample 类型的操作.设有三个类型为 sample 的变量：var a,b,c:sample，操作"＋"的含义是将把相应位置的实数加在一起。如：a:＝b＋c，所表示的相应操作为

```
[1] 7.4    [1] 5.3    [1] 2.1
[2] 2.3 ← [2] 2.2  +  [2] 0.1
[3]10.5    [3] 4.2    [3] 6.3
```

该操作产生一个相同类型的值并被保存在 a 之中。

另一种类型的操作是作用在数据元素上而不是作用在整个数据值上.如操作 x:＝a[2]，意义为把数据值 a 中的第二个数据元素(即实数)搜索出来并赋给左边的变量 x。

这样，我们就知道了数据结构可以有定义在它的数据值之上的操作，也有定义在这些值

的数据元素之上的操作.数据类型分为两种，一种是原子数据类型,另一种是数据结构。两者的主要区别在于，前者的数据值不可分解，而后者是可以进一步分解的。

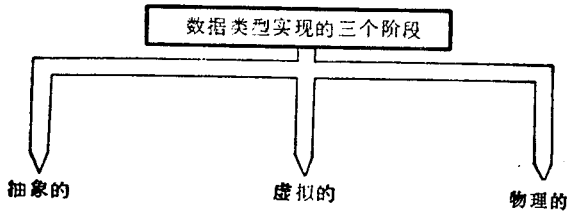
1.2.2 数据类型的实现

数据类型的实现分为三步，即 ADT 阶段、虚拟(virtual)数据类型阶段、物理(physical)数据类型阶段。在 ADT 阶段，仅仅认为数据类型存在于我们的想象之中，并把注意力只集中在感兴趣的性质上而把其他都排除在外。比如，我们有一张名字表，我们可以想象每个具有一个顺序号的名字都浮在空中。同样，可以想象在该名字表中的操作如何进行，如打印整个名字表、找到第三个名字等等。我们同样可以想象在程序中使用 ADT。事实上，我们可以写出利用这些数据及其定义在这些数据上的操作的程序。在这一阶段，我们既不关心数据在计算机之中的表示形式，也不关心实现这些操作的代码等具体细节。

要实现数据类型的话，通常都要利用程序设计语言写出程序，用以实际地保存数据及相应的操作.假定我们选择 Pascal 语言.我们就可用该语言提供的固有数据类型和数据结构去构造没有提供的数据类型和数据结构。例如：可以定义 a 为记录的三维数组.这样，我们所定义的数据类型和数据结构好象存在于一个"虚拟"的机器上，该虚拟机器执行 Pascal 的语句及实现 Pascal 的数据类型和结构。Pascal 虚拟机器可以看作为软件(编译或解释Pascal 代码的系统)同操作系统及宿主计算机硬件的结合体。我们将称这一种数据类型为虚拟的数据类型。

最后，任何一种数据类型及数据结构都必须被存放在物理存储器中才能进而通过计算机进行操作.从数据结构的观点看来，随机存储器(RAM)可以认为是字节 (byte) 的一维数组。无论是抽象数据类型还是虚拟的表示形式，最后都必须化为字节的一维数组才能执行。我们称在这一级上的数据类型为物理数据类型。

图 1.3 给出了这三种数据类型的总结。图 1.4 给出了在各种不同范畴中的数据值或数据元素的例子。



抽象数据类型：数据类型仅仅作为一种想象的结果并且只把注意力集中在数据类型的本质特征上而忽略实现的限制及细节。

虚拟数据类型：数据类型存在于虚拟的处理器上，如程序设计语言。

物理数据类型：数据类型存在于物理处理器上，如某一型号的计算机。

图 1.3 三种数据类型的总结

数据类型		(抽象的)	(虚拟的)	(物理的)
	(结构的)	List of temperatures	Pascal array of real	Implementation of an array on an DEC VAX
	(原子的)	Number of people	Pascal integer	DEC VAX integer

图 1.4 不同数据类型的数据值的例子

1.2.3 固有数据类型

在数字计算中，所有数据值都可以被表示为仅仅由0和1作为数据元素的一种数据结构。大多数处理器(我们这里指的是广义的处理器:包括高级语言、数据库系统、操作系统、计算机硬件系统等)都以 ADT 的形式提供了某些数据类型，这些数据类型对该处理器来讲是固有的。用户并不需要知道数据类型的具体表示及实现的任何细节。对具体的计算机硬件来说，这些数据类型直接借助于被称之为机器语言的指令集合加以实现。由于机器指令由宿主计算机的电路系统来完成，所以这些操作是很有效的。计算机固有的数据类型都是在它的基础上构造完成的.例如:Intel 8080 处理器固有的数据类型是：byte(字节),address(地址), integer(整数), binary-coded decimal(二-十进制编码), stack(栈)等。由于，直接使用机器语言有诸多不便，所以通常使用汇编语言编写程序。而把由汇编语言转换为机器语言的任务交由汇编程序去完成。

高级语言在更高的层次上向用户提供了某些固有的数据类型。不过，利用高级语言构造的数据类型必须被转化为宿主计算机硬件系统的固有类型，才能实现相应的操作或运算。这个工作通常由编译程序或解释程序完成。例如，Pascal 语言固有的数据类型是：boolean(布尔量)、Char(字符)，integer(整数)，real(实数)，pointer(指针)，array (数组)，record (记录)，file(文件)等。

所有计算机的硬件系统都有一种原子数据类型位 (bit)。但是只有非常少的高级语言能提供位作为一种数据类型。如果计算机的固有数据类型没有嵌入到高级语言中去，那么程序员在编写程序时要使用它们，必须用软件的方法解决，或者采取某些办法把高级语言同机器语言连接在一起。

1.3 ADT(抽象数据类型)

在本节中，我们将讨论 ADT 的使用，以及使用ADT 的优点。1.3.1 节中给出 ADT 的说明和设计，1.3.2节中则说明如何在程序设计语言中加以实现。

1.3.1 ADT 的说明

原子类型(atomic type)

我们已经知道原子数据类型包括所有可能值的集合(即它的值域 —— Domain) 及在这些值上的操作(operations)的集合。假定,我们定义了一种 ADT,如 color,其值是颜色.color的值域包括原色红(red)、黄(yellow)、蓝(blue)以及次色,即由两种原色混合而成的颜色。在 specification 1.1 中给出了该 ADT color 的值域及操作的说明。

Specification 1.1 Abstract Atomic Type *co'or*

Domain:Thes et of possible values is {*red*,*yellow*,*blue*,*green*,*orange*,*violet*}

Operations:We define four operations:

mix(c1,c2:color):color　　{*Make a color from c1 and c2*}

pre——c1 and c2 are primary colors.

post——Mix is the color formed by mixing colors c1 and c2 in equal amounts.

```
primary(c:color):boolean                    {Is c a primary color?}
  pre  —— None.
  post —— If c is a primary color,
          then primay is true
          else primary is false.

form(c:color; var c1,c2:color)              {Which colors form color c?}
  pre  —— c is not a primary color.
  post —— c1 and c2 are the two primary colors that form the color c.

assign(var c1:color; c2:color)              {Assign value c2 to c1.}
  pre  —— None.
  post —— c1 has the value of c2.
```

在 specification 1.1 中，所用到的"pre"及"post"是什么意思呢？"pre"是"precondition"的缩写，其意义为前提，它是一个断言。为了正确地执行诸如 mix 及 form 的操作，该断言必须为真。如果，前提 pre 为假，则操作的结果将不确定。例如，函数 mix 要求 c1 及c2 是原色，若 c1 和 c2 不是原色，则该操作执行后的结果无法确定。可能返回一个错误的信号或者执行程序停止工作。

在 specification 1.1 中用到的"post"是 postcondition 的缩写，它的意思是结论。"post"也是一个断言，说明操作的完成情况或作用结果。必须明确地对它加以说明。该断言并不告诉我们结果如何完成，仅仅告诉我们结果是什么，当然它要求前提 pre 为真。

前提和结论都可以用谓词演算的数学方法进行精确描述。我们现在使用语言进行描述，这种方法比谓词演算冗长、啰嗦，且不准确。

现在，看一下过程 form 的前提，它要求 c 的值不是原色。由于过程 primary 已被说明，它允许用户测试 c 是否原色。这样，在将 c 的值传送给 form 之前，可以使用 primary 去测试变量之值是否满足要求。所以正确使用过程 form 的语句应如下所示：

```
        :
  if primary(c) then begin c1:=c; c2:=c end else form(c,c1,c2);
```

这样，我们可以定义 form 返回 c1 = c2 = c，如果 c 的值是原色的话。因此，form 的说明可改变为

```
form(c:color; var c1,c2:color)
  pre  —— None
  post —— if c is primary   {c 若是原色}
          then c1 = c2 = c
          else c1 and c2 are the two primary colors that form color c,
```

{否则c1和c2是形成色c的原色}

如何书写form的说明，和所使用的程序包有很大关系。ADT设计者的典型任务之一是设计简单、明了、准确的说明。

结构抽象类型

假定要定义一个名为letterstring的ADT。它的值是一串英文大小写字母及空格。可以设想这些字符串是线性的，每个字符串的字符总数不超过80。由于该数据类型是结构型的，必须对其结构作形式上的描述：这些字符串除最左面(第一个)字符之外，每个字符都有一个唯一的前件(即其左方字符)。除最后一个字符之外，每个字符都有一个唯一的后件(即其右方字符)。这样，不大于80个字符的字符串的集合都是类型letterstring的一个数据值。这些可能值的集合形成了letterstring的值域。虽然数目是巨大的，但并不是无限的。

在specification 1.2中给出了5种操作。当然，所有的操作种数大大地超过5种，我们这里只不过给出几个例子。Element的意义为组成元素，即大小写英文字母及空格。

Specification 1.2 Abstract Data Type letterstring

Elements: The component elements are the characters 'a'—'z','A'—'Z',and the space character. We refer to them as letters.

Structure: There is a linear relationship(structure)among the letters in each value of the letterstring

Domain: There are between 0 and 80 letters in any such letterstring. The domain of the type *letterstring* is all such letter strings that satisfy these rules.

Operations:In specifying the operations, we occasionally have to refer to ther value of a letter string before and after execution of an operation. We call the former s-pre and the latter s-post.

leftletter(var s:letterstring):letter

pre — The number of letters in the input letter string is greater than 0.

post — leftletter is the first (leftmost) letter in the input letter string (s-pre).s-post is s-pre less its leftmost letter.

append(l:letter; var s:letterstring)

pre — The number of letters in s-pre is less than 80.

post — The string s (s-post) is longer by one letter than s-pre, and the letter in l is its new last (rightmost) letter.

empty(s:letterstring):boolean

pre — None.
post — If s contains 0 letters
then empty is true
else empty is false.

full(s:letterstring):boolean
pre — None.
post — If s contains 80 letters
then full is true
else full is false.

reverse(var s:letterstring)
pre — None.
post — The sequence of the letters in the string is reversed so that the first and last have changed places,the second and next-to-last have changed places, and so on.

注意，leftletter 操作产生两个值，其中之一是一个新的字符串仍放在 s 之中，另一个是类型为 letter 的字符。操作 reverse 仅仅产生类型为 letterstring 的一个字符串值。

对于结构的抽象类型来说，更加需要作精确的说明。但是和原子抽象类型不一样，它的值域通常无法精确地给出。

从某种意义上来说，ADT 的说明相应于建筑物的蓝图。蓝图是建筑师对于所设计的建筑物的想象的精确描述。建筑队必须受蓝图的制约和指导，才能建造出合乎要求的建筑物来。但是，蓝图通常非常详尽，甚至要告诉建筑者们用什么材料。但 ADT 的说明，则与此相反，基本上是一些功能性的说明，对具体的实现不作过多的约束。在具体实现时，所要求的程序的逻辑，所选择的程序设计语言的固有数据类型，完成同样的任务选取什么方法是自由的。

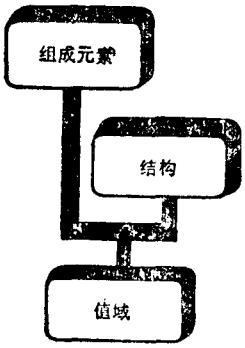


图1.5 ADT的值域的构成

在 color 和 letterstring 类型的说明中，都是一些功能性的说明。同样，可以写出 ADT 的其他方面的说明。例如，对于执行速度的限制，要求该程序必须在什么时间内完成以及存储空间的限制等等。但是，在本书中我们的注意力仍然集中在功能说明上。

1.3.2 实现

上一节中，我们已经给出了抽象类型 letterstring 的说明。本节将讨论它的实现问题，换句话说，我们将用虚拟数据类型完成 ADT 的具体实现。

表示(representation)

我们必须用某种方法保存类型 letterstring 的值。仍然采用 Pascal 作为具体实现的语言。假定我们按照下述方法使用 Pascal 的固有数据类型：

type letterstring = **record**