高等学校教学用书

数据结构

冶金工业出版社

高等学校教学用书

数据结构

东北工学院 姚天顺 编

冶金工业出版社

高等学校教学用书

数据结构

东北工学院 姚天顺 编

*

冶金工业出版社出版

（北京灯市口74号）

新华书店北京发行所发行

冶金工业出版社印刷厂印刷

*

787×1092 1/16 印张 13 1/4 字数 315千字

1981年4月第一版 1981年4月第一次印刷

印数 00,001～9,000 册

统一书号：15062·3645 定价 **1.40** 元

前　言

计算机科学的迅速发展，迫切地提出了这样一个问题：如何把技术性的课题提高到一定的理论水平。为了解决这一问题，重要任务之一是研究计算机科学的最基本对象——数据以及数据间的关系。因此，数据结构这一新兴学科，被认为是计算机专业的基础科学，近年来日益引起计算机界的重视。

本书是根据冶金部所属院校的计算机专业教学计划编写的，原列入《计算机应用数学》第三册，考虑到《数据结构》已单独设课，因此改为单独出版。本书可作为本课程80学时的教材，内容着重讨论数据结构的分析和符合计算机运行要求的算法，为编译系统、操作系统和数据库等课程打基础。同时也注意到计算机语言的实现以及数据结构在文本编辑、分时系统、计算机图形学等方面的应用，书中列举了一些例题。

本书脱稿后经杨箴引教授审阅，作了很多宝贵的指导，繁重的收尾工作是李景银同志完成的，特在此表示衷心的感谢。

本书在编写中主要参照了J. P. Tremblay & P. G. Sorenson: An Introduction to Data Structures with Applications一书，但限于篇幅，取舍可能不当，编者对本课程的教学实践经验不多，又限于水平和时间，书中的缺点错误在所难免，敬请读者批评指正。

编　者

1980年2月

目　录

绪　论

什么是数据结构呢？所谓结构，就是一组不可再分割的元素之间的关系；而数据结构则是在计算机科学中，一门研究数据项之间关系的学科。

用计算机求解数值问题，最初无所谓结构问题。但是随着计算机科学的飞速发展，应用的范围日益广阔，结构问题就显得十分突出，成为必须研究的基本课题。

例如一个大型复杂的程序系统，需要几百上千个“人年”才能完成。这样的大型问题，总需要有系统程序员统一调度，分配任务，规定全程变量进行系统设计。这里涉及到输入的源语句是什么，语句中有哪些基本元素，它们之间有些什么关系。存入内存以后，具体的是什么表示。一个大型系统具备有多种功能，每种功能又可划分为子功能，子功能还可以再划分为它的子功能。每个子功能可以用规定的程序段来完成，称为程序模块。每一类模块要求格式统一，相互之间的软接口要尽可能地清楚而简单。很多个模块组合在一起，一般总会有一个主模块，由高到低分层分级或者组成一个网络，相互之间的关系也是相当复杂而值得研究的。总之，组织和定义某些问题，建立起它们之间的结构关系，是件十分复杂和艰巨的任务。本书就是试图研究结构关系中最基本的问题，为进一步讨论复杂的程序系统提供必要基础知识。

按照这样的考虑，借助于计算机研究程序系统，至少面临有下列四个子问题：

（1）彻底地理解与求解问题有关的数据元素之间的关系，即数据结构问题。

这个问题是随着计算机应用的日益发展而逐渐提出来的。最初，计算机用于数值计算时，问题并不突出。一个数值问题求解的基本数据结构，例如变量、向量和数组等，在一般情况下已经足够了。但是对于非数值问题就不行，它将需要更加复杂的结构关系。

在某些问题中，数据结构涉及到两个方面，即数据项本身和数据项之间的逻辑关系。

（2）要确定逻辑相关的数据元素之间所必须执行的运算。

这就是数据结构上的运算问题。数据结构上的运算包括：生成和消除一个数据结构的运算；从一个数据结构中插入和删去元素的运算；从一个数据结构中存取元素的运算等等。这类运算主要是非数值运算。结合给定的数据结构，这些运算可以由特定的语言用一组相当完善的算法过程来描述，也可以用程序设计语言的基本语句来实现。

（3）要考虑数据元素在计算机内存和外存中的表示问题。

这就是数据结构在计算机内的表示问题。它要求很好地保存原数据项之间的逻辑关系，并能方便而有效地完成对数据元素的运算。一般说来，对应于某个特定的数据结构存在有多种内存构造或存储结构。存储结构在外存中表示，又称为文件结构。

（4）在问题的求解过程中选择最好的解决问题（程序设计）的语言。

这是对所解的问题选择合适的程序设计语言的问题。有的时候对于给定的数据结构，存在有某些程序设计语言没法表示的情形，例如像企业管理、财经支付等问题应该使用COBOL语言，如果选用FORTRAN语言就不合适，即使可行也将十分复杂。

第一章　信息及其存储表示

第一节　信息及其度量

在计算机科学中任何方面的研究，可以说都涉及到信息的处理、存储和检索。例如，一个学生的档案文件，在期末进行分数登记时就是信息的存入；一个编译程序从一个程序的源文件翻译成另一个源语言指令时，是信息的检索；两个数在计算机的运算器中相加并得出结果时，又可以理解为信息的处理。因此，学生的分数、程序的指令和数值都是信息。在这个意义上，我们把信息理解为具有与其符号相联系的记录或通讯资料。

在自然科学的发展过程中，在二十世纪初，主要研究的是能量。到本世纪四十年代以后，信息日益成为另一个主要的研究对象。

信息的传送有五个环节：信息源、发送器、通道、接受器和目标，如图 1-1 所示。例如在两个人的通话过程中，某人（或人的大脑）就是一个信号源，他的嘴就是发送器，空气是通道，另外一个人的听觉系统是一个接受器而接收者的大脑就是个目标。在一个计算机系统里，一张卡片可以认为信息源，卡片读入机是个发送器，从卡片读入机到计算机主机的传送线是个通道，通道选择器控制部件是个接受器而计算机的内存就是目标。

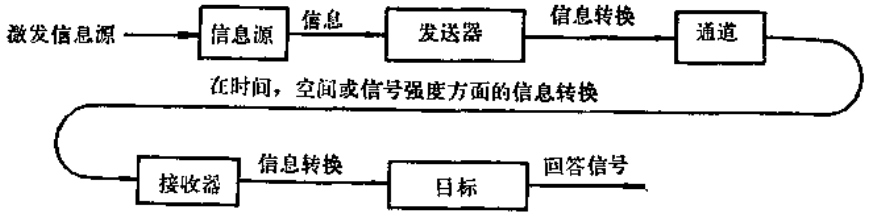


图 1-1　信息传送过程示意图

在讨论信息传送过程时，有三个方面的问题必须考虑，即语法、语义和实效。语法方面涉及到信息发送的实际形式。例如我们接收一个英语句子时，先是从英文字母开始，并由此组成一个词，再组成一个句子。在句子的形成过程中，一个个概念都要通过句子来表达，而其意义则与信息的语法形式有关。但是，语法上正确的并不一定有意义。例如“Data evade lowly”是个语法上正确的英文句子，但没有什么意义。因此还应该有语义方面的要求。信息传送的实效是指防止噪声干扰和实际传送效果等等。

信息能否测量？例如一个学生的分数中存在多少信息？程序 A 是否比程序 B 含有更多的信息？回答这些问题对于计算机科学是非常基本的。

信息论与统计学类似，都是研究一个集合中各元素的差异性的。但是，统计学认为差异性是一种妨碍，因此希望撇开它来确定可以说明什么或者做什么。信息论则把差异性作为一种有价值的东西，它通过一定程序的差异性来确定可以获得些什么。

从信息论的观点来考虑，信息是鉴别不肯定性的。假定我们给出任何系统的几个组态

（或数据项）的集合，其中每个组态出现的相互独立的概率为 p_i，$\sum_{i=1}^{n} p_i = 1$。对于这样一个系统，我们不可能预言哪一个组态一定出现，因此我们说系统有先验不肯定性。自然，如果 n 越大，那么某一个组态出现的不肯定程度也就越大。这就是说信息这个概念是与不肯定性密切相关的，信息量随不肯定性的增大而增加。因此我们对信息进行估量，必须考虑如何用数量来表示不肯定性。

我们知道，当有一个 p_i 等于 1 时，其余的自然都等于 0。这个时候系统就不存在什么不肯定性。

其次，我们在系统中添加或消去几个概率为 0 的组态，自然不会影响不肯定性的数值。

再其次，当系统中所出现的组态的概率都相等时，即 $p_i = \frac{1}{n}$ 时，则系统的不肯定性最大。因为在这个时候，任何一种组态出现的可能性都一样，不能预言哪一种更可能出现，自然，这时候的不肯定性也就最大。因此我们定义任何系统 n 个组态集合的不肯定性为

$$H = -\sum_{i=1}^{n} p_i \log(p_i) \qquad (1)$$

其中对数的底数可以取不等于 1 的正常数。一般情况是取以 2 为底的对数，即

$$H = -\sum_{i=1}^{n} p_i \log_2(p_i) \qquad (2)$$

假定我们有 8 张编了从 1 到 8 号码的卡片，把它们翻过来混匀放在桌面上，则取到某张号码卡片的概率都是1/8，那末选取这种卡片的不肯定性为

$$H = -\sum_{i=1}^{8} \frac{1}{8} \log_2\left(\frac{1}{8}\right) = -\log_2 \frac{1}{8} = \log_2 8 = 3\text{位}$$

值得注意的是，我们在这里的度量单位取名为“位”。它是在表示断-开或真-假等两状态条件下提出来的，我们可以说位既是不肯定性也是信息的度量单位。为了说明这一点，我们仍以上述选取编号卡片为例。

设图1-2是选取编号卡片的问题树。利用公式（2）计算出来的信息量是 3 位，意味着任何一张编号卡片可以通过三个“是-否问题”找到。一个“是-否问题”可以由一个 0

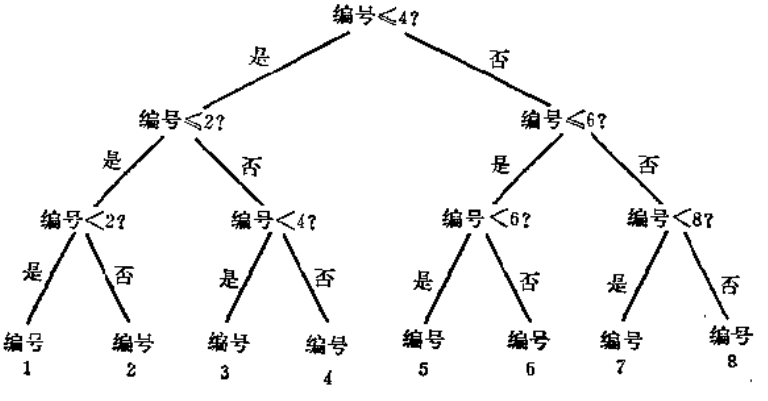


图 1-2　编号卡片的问题树

或1表示，三个问题就可以由三个0或1的数来表示。这就是说，8张不同编号的卡片，结合寻找卡片的次数，可以由长度为3的二进制数来表示。表1-1给出了两种表示8张卡片的不同信息方案。

表 1-1 表示8个编号卡片的信息方案

编号	信息方案 A	信息方案 B
1	1 1 1	0 0 1
2	1 1 0	0 1 0
3	1 0 1	0 1 1
4	1 0 0	1 0 0
5	0 1 1	1 0 1
6	0 1 0	1 1 0
7	0 0 1	1 1 1
8	0 0 0	0 0 0

由此可见，信息的度量H提供了不肯定性的大小，能帮助我们对给定的数据结构确定一个好的存储结构。H通常称为熵。

习　　题

1．假定某个控制系统有编号为2～12的11个开关，打开一次某个开关的概率（打开后自动合上）如表1-2所示。试计算它的不肯定性H的值。

2．试考虑一个固定长度的二进制码来表示第1题中的每个开关（所谓“固定长度”是指所有代码都是固定长度的），并把这些代码的长度与第1题计算的值作比较。

3．试设计一个变长度的二进制码来表示第1题中的每个开关，并要求它的平均长度小于第2题所给出的长度。

表 1-2

开关编号	概率	开关编号	概率
2	1/36	8	5/36
3	1/18	9	1/9
4	1/12	10	1/12
5	1/9	11	1/18
6	5/36	12	1/36
7	1/6		

第二节　初始数据结构

这一节，我们讨论计算机的初始数据结构。所谓初始，是因为这种数据结构是最基本的，其它的数据结构都是以它们为元素组合起来的。

一、整数及其表示

一个整数集合，无非是

$$\{\cdots, -(n+1), -n, \cdots, -1, 0, 1, 2, \cdots, n, n+1, \cdots\}$$

它们在计算机里用的是所谓符号-尾数方法。在一个二进制数里，符号用最左侧的一位表示，其余部分表示尾数。例如整数+7和−6的符号-尾数表示为

数　　值	符　号　位	尾　　数
+7	0	0 0…0 1 1 1
−6	1	0 0…0 1 1 0

但是这种表示方法并不好，例如作最简单的加法，两数相加首先要判别一下符号，如果两数同号，则数值相加符号不变；如果是异号，那末就要做减法，而且还要比较两数绝对值的大小，才能确定谁减谁。这些事在手算时是容易解决的，但是让计算机完成就不方便了。

整数的更为普遍和有效的表示法是**基数补码**表示，或称**2的补码**表示。在这种表示中，我们取模M，$M=R^N$，其中R为基数，N为正整数。

取模运算是这样进行的：设有两个数x和y，进行运算$*$，如果不取模，结果$z=x*y$；同样的进行运算$*$，但取模M，我们把它记作$*_M$，则$x*_My$的结果$z_M=z\bmod M$。即z_M是z被M除后的余数。

在模M的算术运算中，如果x是正数，自然有

$$x=x+M \tag{1}$$

而x的负数，则有

$$-x=M-x \tag{2}$$

这两个式子实际上是统一的。例如取$M=16$（$\because 2^4=16$，$\therefore N=4$），-2的补码是$16-2=(10000)_2-(10)_2=(1110)_2$而2的补码是$2+16=(10)_2+(10000)_2=(0010)_2$。

例 1　试在模32系统中，求-38的2的补码表示。

解　首先，我们必须把-38表示为模32的数。利用等式（1），

$$-38\bmod(32)=-38+32=-6$$

然后再找等价的2的补码表示，应用等式（2），就有

$$32-6=26=(11010)_2$$

或

$$(100000)_2-(110)_2=(11010)_2$$

利用基数补码表示的一个主要优点是，无论是进行加法还是减法，只需要用加法和求补就可以，不必再作任何判断。下面我们再举几个例题，在计算过程中我们把某数x的2的补码简写成2的COMP（x）。

例 2　试利用2的补码表示并按模16计算$3+4$，$3-4$，$-3+4$，$7+7$，$-7-7$等。

（a）　$3+4=(0011)_2+(0100)_2=(0111)_2=7$

（b）　$3-4=(0011)_2+2$的COMP$((0100)_2)=(0011)_2+(1100)_2=(1111)_2=-1$

（c）$-3+4=2$的COMP$((0011)_2)+(0100)_2=(1101)_2+(0100)_2=(0001)_2=1$

（d）　$7+7=(0111)_2+(0111)_2=(1110)_2=-2$

（e）$-7-7=2$的COMP$((0111)_2)+2$的COMP$((0111)_2)$

$=(1001)_2+(1001)_2=(0010)_2=2$

注意，例2中（d）和（e）所得的结果是错误的。这是因为在模16的系统中，它们都超过了允许表示的范围（$-8\sim+7$），产生溢出，计算结果自然就不正确。

在数值系统中，还有一种称为基数减1补码（或反码）。设基数为R，给定数值d，则d的基数R减1补码为$R-1-d$。如果我们希望取n位，整数I的减1补码，记作DRCOMP

(I),为

$$\mathrm{DRCOMP}(I)=R^n-1-I$$

其中R^n在基数R的数值系统中，等于一个1后面跟着n个0，共$n+1$位。

基数补码形式和基数减1补码形式之间的区别，是相对于同一个模（即R^n），后者比前者多减个1。整数的基数减1补码形式，在二进制中通常称为**1的补码**。例如整数6的1的补码，在模16的系统中是

$$16-1-6=9$$

或

$$(10000)_2-1-(0110)_2=(1111)_2-(0110)_2=(1001)_2$$

一个数x如果是二进制的，1的COMP(x)是很容易计算的，只要把原数x的所有0改成1，所有的1改成0。这是很方便的，缺点是+0和−0是不一样的。例如在模16的系统中，1的COMP(+0)=0000，而1的COMP(−0)=1111。

利用基数减1补码表示，在算术运算上无论是加法或减法也只需要用加法和求补，再加上“加个进位”。下面我们举例说明

例 3 试利用1的补码表示和模16计算3+4，3−4，−3+4，7+7，−7−7等。

解 (a) $3+4=(0011)_2+(0100)_2=(0111)_2$

(b) $3-4=(0011)_2+1$的COMP$((0100)_2)=(0011)_2+(1011)_2$

$=(1110)_2=-1$

在这里，$(1110)_2$的1的补码形式为−1。

(c) $-3+4=1$的COMP$((0011)_2)+(0100)_2=(1100)_2+(0100)_2$

$=(0000)_2=0$

(d) $7+7=(0111)_2+(0111)_2=(1110)_2=-1$

(e) $-7-7=1$的COMP$((0111)_2)+1$的COMP$((0111)_2)$

$=(1000)_2+(1000)_2=(0000)_2=0$

从例3中可以看到，第（c）个小题在运算过程中并没有得到正确的结果，但是如果第（a），（b），（c）各题把进位值加进最后的结果，就分别得到正确的答案7、−1和1。因此，利用这种表示法，必须要注意加上进位值。

第（d）和（e）等两个小题，由于溢出，所以结果是错的。

对于溢出，我们有个简单的检查方法：

（1）一个正数和一个负数相加不会引起溢出。

（2）两个正数或两个负数相加，如果其和与这两个运算数有相同的符号也不会引起溢出。

判断溢出的问题是比较复杂的，读者有兴趣可以参阅Trembley和Manohar著的“应用于计算机科学的离散数学结构”和Stone著的“计算机组织与数据结构导论”。

习　　题

1．试利用2的补码表示下列整数（N为6）：

a．−33，b．−52，c．$(-33)_8$，d．$(-E)_{16}$，e．−241。

2．试利用2的补码表示和模16计算下列表示式：

a．6−1，b．7−(−2)，c．−3−3。

3．试利用1的补码表示及模32计算下列算式：

a．8+8，b．6−1，c．7−11。

二、实数及其表示

我们知道，基数为R的系统中，实数A的定点表示为

$$A=\pm(a_{N-1}a_{N-2}\cdots a_1a_0\cdot a_{-1}\cdots a_{-(M-1)}a_{-M})_R$$

其和式表示为

$$A=\pm\sum_{i=-M}^{N-1}a_i R^i \tag{3}$$

实数的上述表示，在计算机上是经常使用的。但是需要表示很大的或很小的实数时，仅用定点表示就不方便了。例如像16,800，000，000，000这样的大数和.000,000，000，083，2这样的小数。这就迫使我们采用新的表示方法。

在基数为R的系统中，一个数都可以表示为

$$f\times R^E$$

或者

$$f_{-1}f_{-2}\cdots f_{-M}\times R^E$$

其中$f_{-1}f_{-2}\cdots f_{-M}$称为小数或尾数，$E$一般为整数，称为指数，这就是基数$R$的系统中，实数的浮点表示。那末上面提到的大数和小数可分别表示为$.168\times10^{14}$和$.832\times10^{-10}$。

但是这种表示法也有问题，如果所要表示的数是16，817，210，391，704。而尾数的长度又限定为5位，其浮点表示只能写成$.16817\times10^{14}$。引起的误差称为**舍入误差**。这种舍入误差在实际应用时，只能忽略不计。因此，我们只能说可表示数的范围扩大了，但并不是所有的实数都能表示，两个相邻实数之间是可以有很多数的。为了提高实数的有效数字，可以有三种方案：

第一种方案，是在一台大型计算机上进行计算，某些大型机的精度可以达到20位十进制数字。但是这样做，有时有具体困难，费用也太高了。

第二种方案，是采用多精度的程序包，一个实数用几个存储字来表示，客观上成倍地增长了位数，它们完全靠程序包解释执行。这种方法比较好，缺点是降低了计算机的速度和容量。

第三种方案，是采取舍入的方法。这是一种常用的方法，它根据计算机的字长，把需要舍入的数字采取四舍五入的办法。例如有实数21.833652，按5位十进制数字精度进行舍入，即

$$21.833652+.0005=21.834152$$

那末舍入的答案是21.834。

三、数值数据的存储结构

上面我们讨论了两种类型的初始数据结构，整数和实数，并且讨论了它们之间的简单算术运算。这一节，我们将进一步从计算机存储结构的观点来研究整数和实数。它们是基于下列计算机系列的实际表示提出来的。即DJS-130，IBM360-370,CDC6000和CYBER 70-170，Burroughs B5000和B6000，Univac1108，PDP-10，Hewlett-Packard2100和DJS-183等。我们先讨论整数的存储表示。

1．整数的存储表示

一个整数在计算机里通常只占一个存储字，负数用2的补码形式表示，而不用1的补

码表示，以避免＋0和－0之间不好区分。对于国内外各种计算机的存储表示，如表1-3所示。

一般说来，一个整数的二进制表示，第一位是符号位，余留部分是按补码形式的数的尾数。图1-3就是整数13的32位2补的存储表示

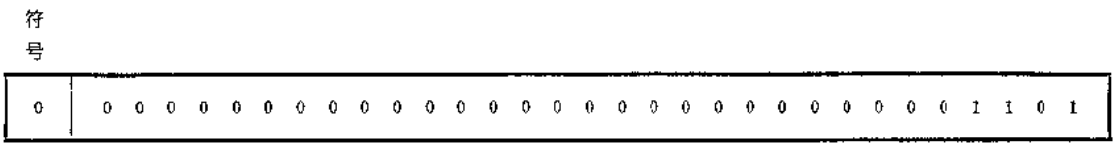


图 1-3　整数13的32位2的补的存储表示

用 n 位存储表示所能表示的整数的数值范围有多大是很重要的。由于 $2^0+2^1+\cdots+2^{n-1}$ 的和式为

$$\sum_{i=0}^{n-1}2^i=2^n-1$$

因此一个整数的 n 位 2 补的存储表示，当第一位是符号位时，正整数的范围是 0 到 $2^{n-1}-1$，负数的范围是 -2^{n-1} 到 -1，因此用 n 位 2 补表示的整数 N 满足下列不等式

$$-2^{n-1}\leqslant N\leqslant 2^{n-1}-1$$

表 1-3　某些计算机整数数据的存储表示

计　算　机	二 进 制 表 示	十 进 制 表 示
IBM 360-370	16和32位数的 2 补	每32位字组成 7 位十进制数
CDC 6000和CYBER70-170系列	18和60位数的 1 补	每60位字组成14位十进制数
Burroughs B5000和B6000系列	符号和40位的尾数	每48位字按字符形式（6 位/字符）组成 8 个十进制数字
Univac 1108	36位和72位数的 1 补	不存在
PDP-10	36位数的 2 补	不存在
Hewlett-Packard系列	16位数的 2 补	不存在
DJS-183	16位数的 2 补	不存在
DJS-130	16位数的 2 补	不存在

对于 n 位 1 补的存储表示，因为它有正零和负零，即＋0和－0。比 2 补少一个数，也就是整数 N 的范围为

$$-2^{n-1}+1\leqslant N\leqslant 2^{n-1}-1$$

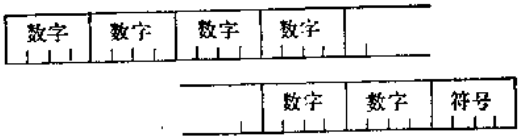
利用二进制的整数存储表示，还有一种常用的方法称为8421码，或二-十进制码，简记为BCD。按照这种方法，一个十进制数用 4 位二进制码表示。例如整数9613就可以表示为

1 0 0 1 0 1 1 0 0 0 0 1 0 0 1 1

9　　6　　1　　3

这种表示方法的一个特点是四位为一组，非常容易识别。另一个特点是四位数码中每一位对应一个固定的常数，自左至右分别是 8，4，2，1。这些固定常数就是以前所说的位值或权。

在IBM360～370系列里有一种纯BCD表示，称为组合式十进制数。它的存储形式为



用组合式十进制表示的某些整数，如表1-4所示，其中右侧4位的1100和1101码分别表示正和负的符号。还有一些其它的十进制表示，例如在Burroughs系列的计算机里，用6位的字符编码表示等等。

表 1-4 组合式十进制表示

整数	组合式十进制表示
0149	0 0 0 0 0 0 0 1 0 1 0 0 1 0 0 1 1 1 0 0
-0149	0 0 0 0 0 0 0 1 0 1 0 0 1 0 0 1 1 1 0 1
935	1 0 0 1 0 0 1 1 0 1 0 1 1 1 0 0
-6	0 1 1 0 1 1 0 1

2．实数的存储表示

关于实数，我们已经讨论了两种表示方法，即浮点表示和定点表示。在这节里，我们先讨论浮点表示的存储结构，然后再研究定点表示问题。

一个用浮点形式表示的实数，一般形式为

其中第一位是符号位，0表示正数，1表示负数。特征或指数利用余表示法。假定它的长度为7位，余表示法就成为余64表示法，特征部分每取一个数，一律都减64，因此它的范围为$-64 \leqslant$特征数$\leqslant +63$。最后是尾数或小数部分。它以小数形式出现，在运算过程中，为了尽可能少地丢失有效数字，提高运算的精度，采用规格化的表示法。按这种表示法，尾数部分F要求满足$R^{-1} \leqslant F < 1$。如果我们所选的基数R是2，那末任何浮点存储结构的非零小数部分F必须满足$2^{-1} \leqslant F < 1$。目前，国际上逐渐流行基数为八进制和十六进制的数值系统。例如十六进制的，尾数必须满足$16^{-1} \leqslant F < 1$或$(.0001)_2 \leqslant F < 1$。

在有些机器里，例如Burroughs的产品，小数点放在尾数的右端，尾数既然不是小数也就不必规格化。这种表示的优点是实数和整数使用同样的存储结构。有的机器像DJS-130和DJS-183，它的基数取2，优点是规格化方便，只需要尾数的第一位为1就可以了。在CDC6000系列的机器里，对于规格化设置了专门的指令，规格化的工作就能自动进行。在IBM的机器中，可以有两种浮点指令，一种是规格化的；一种是不规格化的。

表1-5说明利用32位浮点表示的机器中，某些实数的存储结构。它们由一位符号位，7位特征值（余64表示）和24位小数部分（规格化十六进制数）组成。

实数的浮点表示一般都采用两个字，称为**单精度浮点表示**。有的时候还用4个字称为**双精度浮点表示**。表1-6是一些机器双精度浮点表示的情况。

除了浮点存储表示之外，实数还可以有定点存储表示。在一般情况下，一个定点数可以理解成小数或整数。如理解成整数，那么运算不能简单地进行。例如，在一个“十进制的

表 1-5 浮点存储形式举例

十进制数	十六进制数	32位浮点数
$.0\times10^{0}$	$.(0)_{16}\times16^{0}$	0 1000000 00000000000000000000000
$-.17307\times10^{5}$	$-.(439B)_{16}\times16^{4}$	1 1000100 01000011100110110000000
$.28692\times10^{2}$	$.(1CB126E9)_{16}\times16^{2}$	0 1000010 00011100101100010010110
$-.75\times10^{0}$	$-.(C)_{16}\times16^{0}$	1 1000000 11000000000000000000000
$.8317\times10^{-3}$	$.(36819C4)_{16}\times16^{-2}$	0 0111110 00110110100000011011100

机器”中，3.782和93.2分别可以表示为00…03782和00…00932，如果希望把这两个数相加，必须把小数点对齐，也就是把存入的数00…0932左移2位。对齐之后就可以相加和得出正确结果00…096982，小数点在自右向左的第三位。

上面我们引出了初始数据结构——整数和实数，以及它们的存储表示。在这一章里我们还要继续研究非数值的数据结构问题。

表 1-6 某些计算机中实数的存储表示

计算机	指数	指数的基数	尾数 单精	尾数 双精	小数点位置	规格化
IBM 360～370系列	7位（余64）	16	24位	56位	尾数的左侧	是/否
CDC 6000系列	11位（余1024）	2	48位	—	尾数的右侧	由指令
Burroughs B5500&B6500	符号+6位	8	39位	78位（只对B6500）	尾数的右侧	否
PDP-10	8位①	2	27位	—	尾数的左侧	是
Univac 1108	8位(对单精度余128) 11位（对双精度余1024）	2	27位	61位	尾数的左侧	是
PDP-11(45型)	8位（余128）	2	23位	55位	尾数的左侧	是
DJS-130	7位	16	24位	56位	尾数的左侧	是
Hewlett Packard 2100系列	在硬件里没有浮点表示，但可配备有软件包或添个微程序。					

① 对正指数以余128表示，对负指数以1补表示。

习　　题

1. 试按16位表示下列整数，（i）1的补码表示，（ii）2的补码表示：

a .27，b .−256，c .−7，d .44

2. 试给出第1题各数的组合式十进制存储表示。
3. 试把.250以浮点数存入（i）IBM360-370系列计算机和（ii）CDC6000系列计算机。
4. 把.625表示成Univac1108中的浮点数。

四、字符信息

早期的计算机只用于处理数值数据，编写程序用的也只是机器码，程序是非常难读和难写的。为了改变这种使用方法，出现了表示信息项的符号码，这就是字符。以字符形式提供的数据一般用于地址码和操作码的助记符，它导致了汇编语言及以后的面向过程的语言的发展。

国际上有两种最大的和应用最广泛的字符集。它们是EBCDIC（扩充的二进制编码的十进制交换码）和ASCII（美国信息交换标准代码）。EBCDIC是个字符的编码系统，主要用于IBM360～370系列计算机。ASCII逐渐发展成为计算机工业的标准编码方案，很多非IBM机器都用这种代码。我们现在把这两种编码方案说明如下：

1．ASCII码字符集

（1）大写的和小写的26个英文字母｛a，b，…，z，A，B，…，Z｝。

（2）十进制数字集｛0，1，2，3，…，9｝。

（3）运算和特殊字符集｛+，－，*，/，>，=，<，|，空格（SP），!，"，{，}，'，#，$，%，&，'，（，），，，·，：，；，?，@，〔，\，〕，↑，→，～｝。

（4）控制符，如：DEL（删除），STX（文本起动）ETX（文本结束），ACK（应答），HT（水平制表），VT（垂直制表），LF（换行），CR（回车），NAK（不应答），SYN（对同步传送的同步空闲），ETB（传送数据块的结束），FS（文件分隔符），GS（成组分隔符），RS（记录分隔符）。

2．EBCDIC字符集

这种字符集同样也有上述（1）～（3）的子字符集，至于控制符集，虽然助记符和名字不同，但也具有同样的控制功能。

在这些字符中，大量使用的是英文字符、十进制数值字符和运算符，还有对程序员很重要的控制符。图1-4所示为含有几个记录的数据块的典型传送格式，从图中可看出各种字符的应用。

在存储信息时，控制符是很重要的，主要是用于构造信息，而与具体问题的求解无关。例如控制符FS、GS和RS，它们是分别用来分隔信息文件、成组数据和记录的，虽然不是文件、数据或记录的一部分，但是没有它们也是不行的。

字符集也可以为某些专用计算机而建立。例如在计算机图形系统中，可以用某些字符对坐标点进行控制，在屏幕上指明标志、平移、放大或缩小等。

五、字符数据的存储结构

字符在内存里表示成一个位的序列，不同的字符指定不同的位序列。定长度的位序列比变长度位序列处理起来效力高一些，所以一般都采用定长度位序列的编码。

长度为 n 的位序列可以表示2^n种字符的编码，有很多种不同的编码方案，FORTRAN字符集的某些常用编码如表1-7所示。

这些编码分别由这样一些计算机采用：

EBCDIC——IBM360～370系列。

ASCII——Burroughs 5000～6000系列，PDP-10，DJS-183，Hewlett-Packard 2100系列，TOCBAC-40，DJS-100系列和DJS-200系列。

Univac CPU码——Univac1108。

6600显示码——CDC6600。

扩充BCD码是一种6位码，用在磁带上存储信息。Hollerith码已成为穿孔卡片表示信息的标准代码。

有关字符数据存储的另一个问题是数值和字符间的转换。这是计算机语言中经常要处