计算机程序设计基础(1)
--- C语言程序设计(2)

孙甲松
sunjiasong@tsinghua.edu.cn
电子工程系 信息认知与智能系统研究所
罗姆楼6-104
电话: 13901216180/62796193

2022. 9.

第2章 C语言的基本数据类型

2.1 数据在计算机中的表示

- 2.1.1 计算机记数制
- 2.1.2 计算机中数的表示

2.2 常量与变量

2.3 基本数据类型常量

- 2.3.1 整型常量
- 2.3.2 实型(符点型)常量
- 2.3.3 字符型常量

2.4 基本数据类型变量的定义

- 2.4.1 整型变量的定义
- 2.4.2 实型变量的定义
- 2.4.3 字符型常量的定义

2.1 数据在计算机中的表示

2.1.1 计算机记数制

1 数制的概念

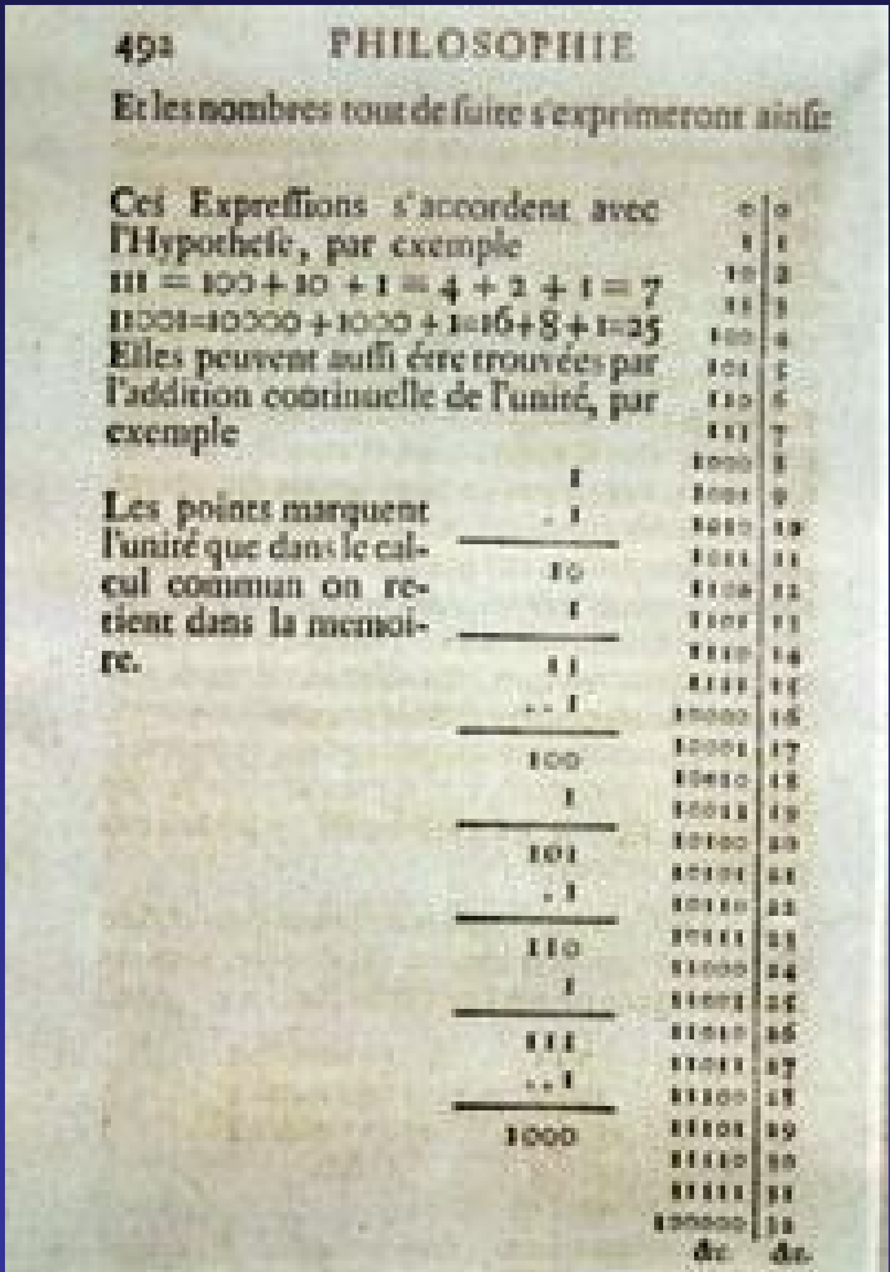
在日常生活中，人们习惯于用十进制记数。一个十进制数可以用位权表示。通常称某个固定位置上的计数单位为位权。计算机是由电子器件组成的，考虑到经济、可靠、容易实现、运算简便、节省器件等因素，在计算机中的数都用二进制表示而不用十进制表示。

2 二进制

二进制数中只有两个数字符号0与1，其特点是“逢二进一”。与十进制数一样，在二进制数中，每一个数字符号(0或1)在不同的位置上具有不同的值，各位上的权值是基数2的若干次幂。

世界上有10种人，一种懂二进制，
一种不懂二进制。

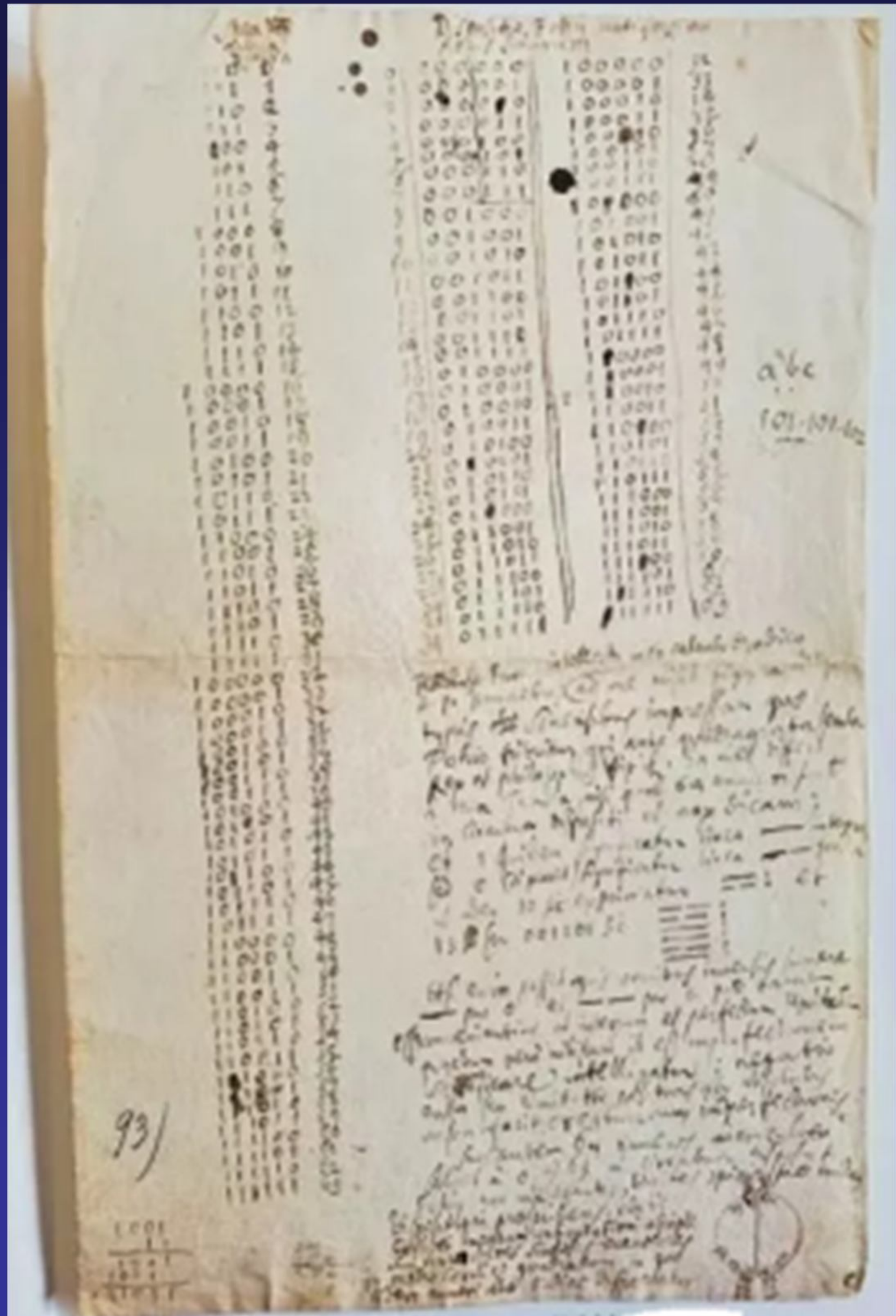
二进制是谁发明的？



- 在德国图灵的郭塔王宫图书馆里，保存着莱布尼兹一份珍贵的拉丁文手稿，其标题为《二进制算术》，此稿写于1679年3月15日。文中不仅给出了二进制的计数规则，而且给出了二进制加减乘除四则运算的规则，并与十进制作了比较。但这份手稿当时并没有公开发表。
- “1与0，一切数字的神奇渊源。这是造物美妙的典范，因为，一切无非都来自上帝。”
- 莱布尼兹首次提出了数理逻辑和计算机的概念。
- 牛顿PK莱布尼茨的对话的内容，请微信搜索：

  二进制的“前世今生”

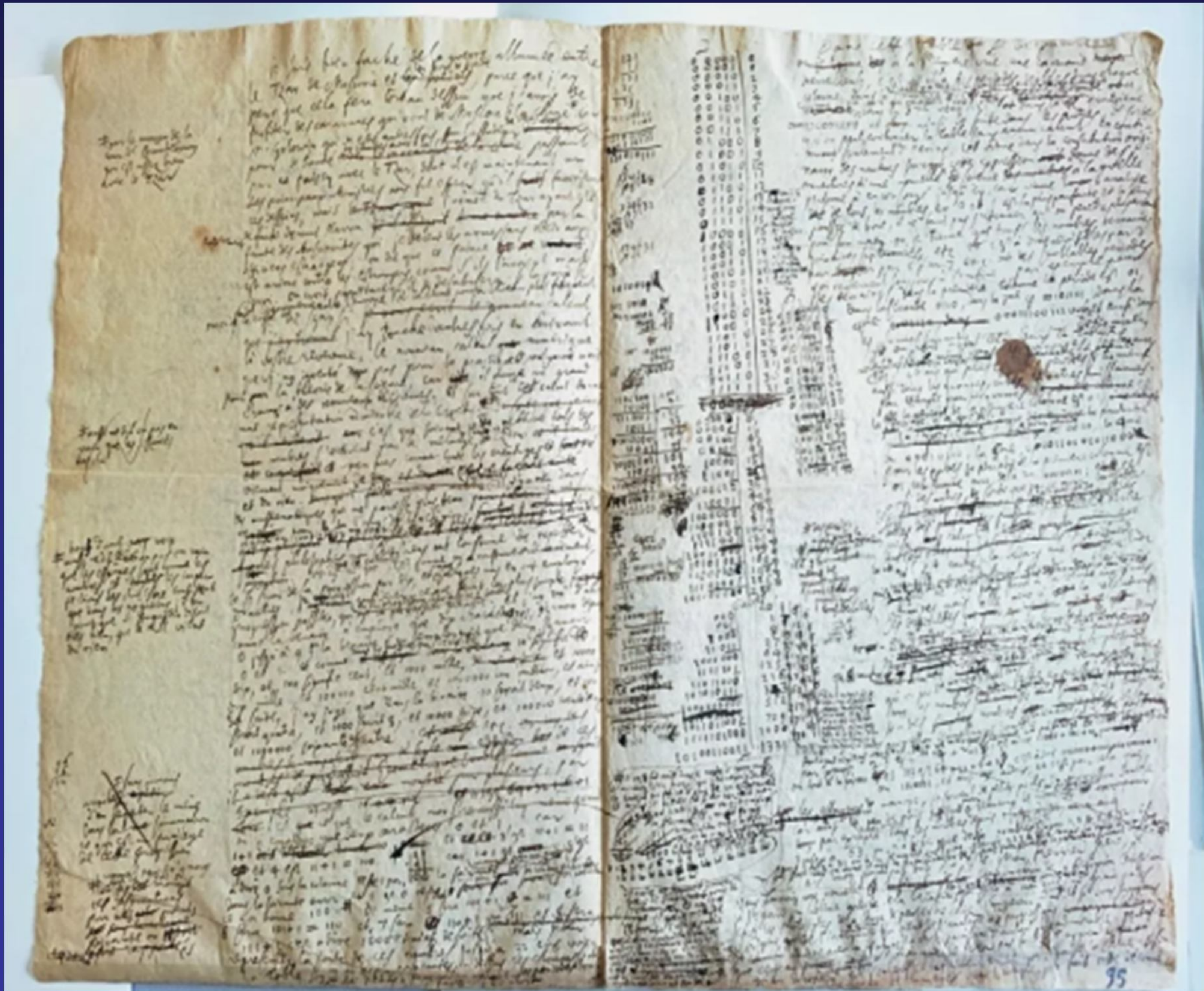
莱布尼茨二进制论文的手稿

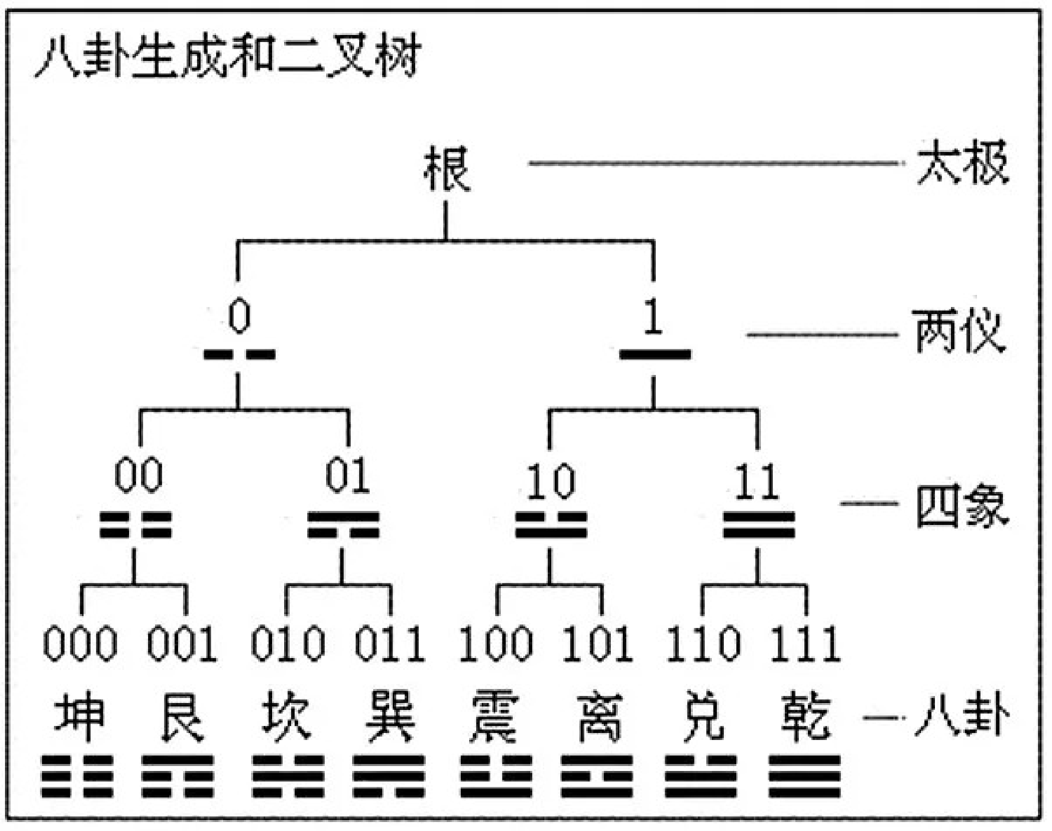


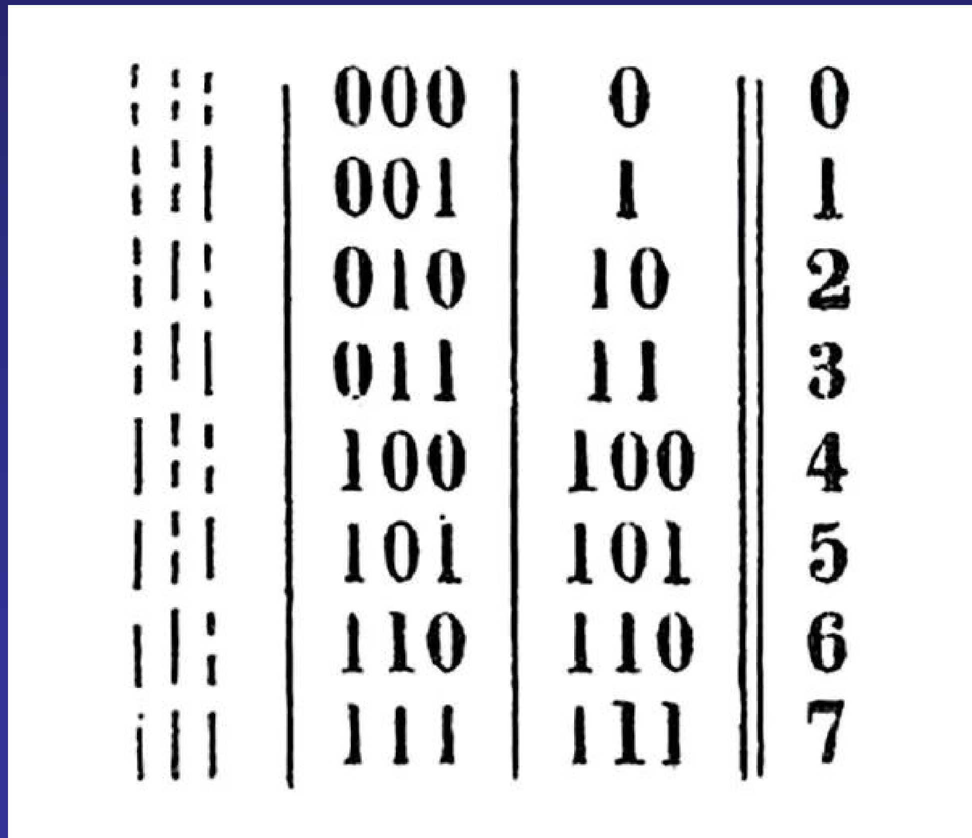
1697年元旦，莱布尼茨写了一封信给鲁道夫·奥古斯都公爵。写信的同时，他赠送了一枚自己制造的银币给公爵，这颗银币的出现，真正预示着二进制的诞生。

银币的正面当然是公爵帅气威风的肖像，这是为了获得“科研经费”必须做出的妥协。反面是一则创世故事：水面上笼罩着黑暗，顶部光芒四射……中间部分雕刻的是从1到17的二进制数学式。

莱布尼茨1701年2月15日致白晋信手稿，建议白晋将二进制介绍给中国皇帝康熙。白晋是一位法国神父，做过康熙的数学老师，他奉康熙之命回欧洲招募科学人才。让莱布尼茨更多了解伏羲卦图的人是白晋。

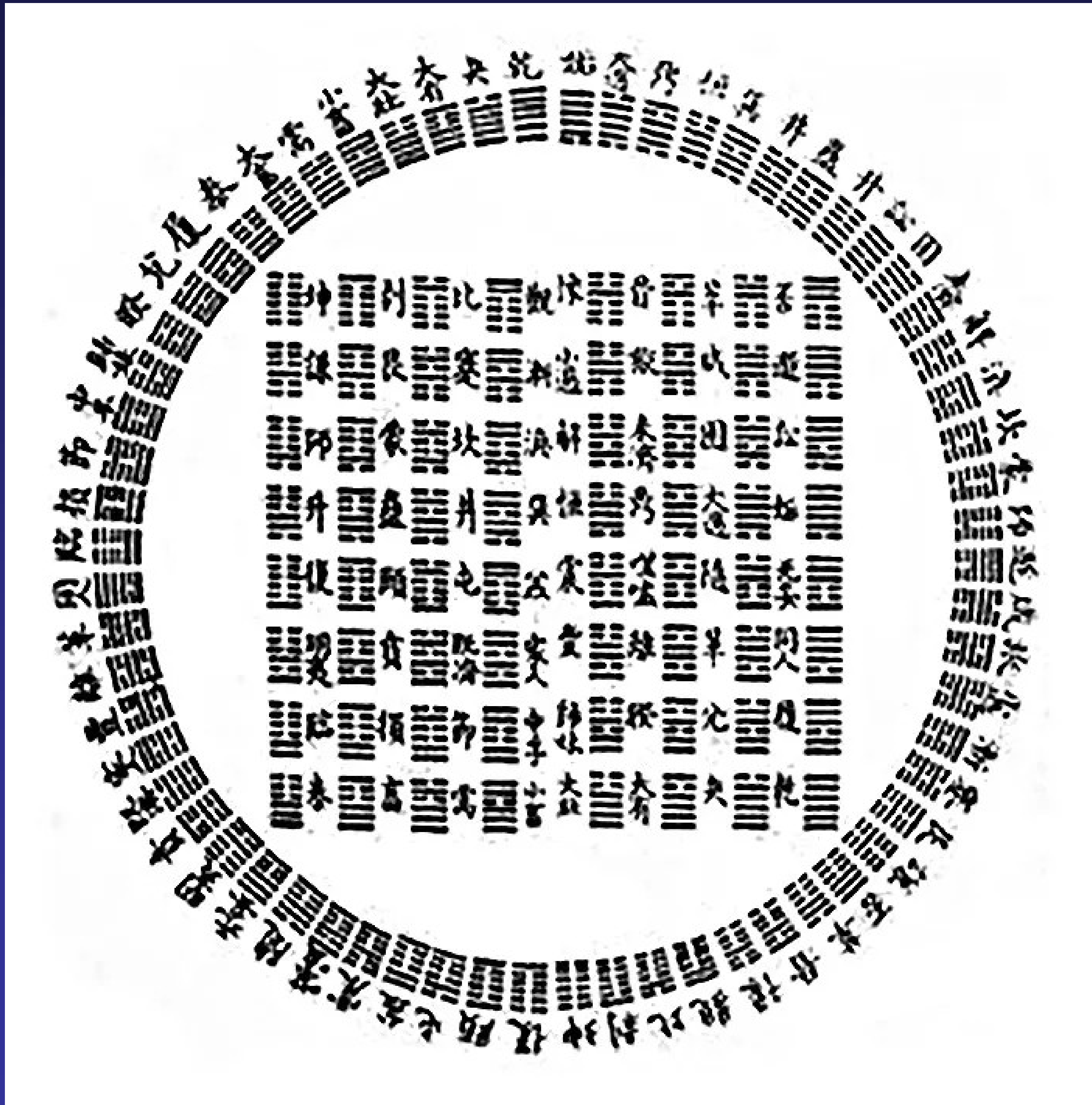




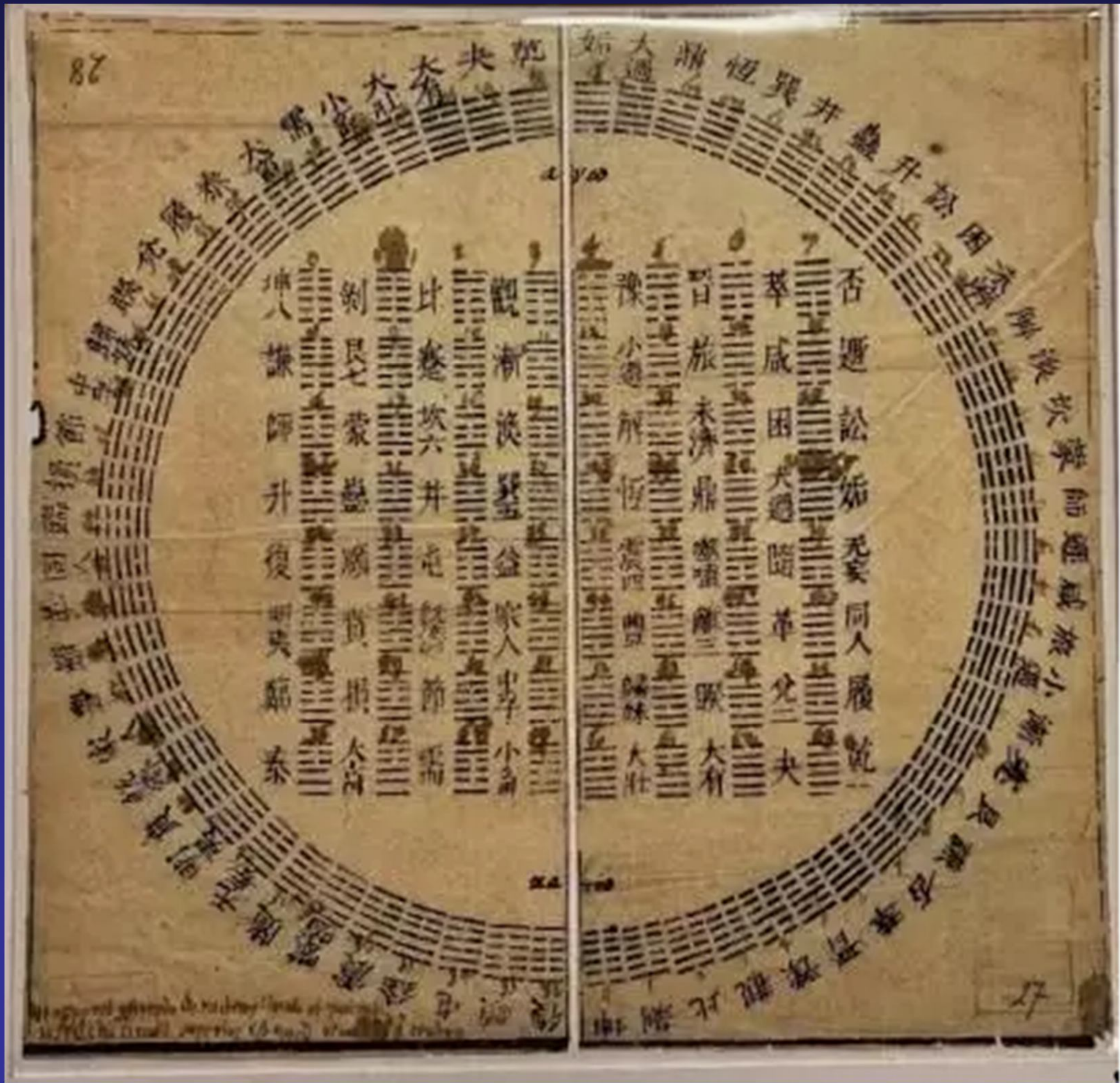


参考文章：
莱布尼茨、二进制和伏羲卦图

先天图（六十四卦图）



白晋寄给莱布尼茨的先天图（1703年）



二进制容易表示：(仅两个状态)

电灯：开、管

二极管：通、断

电平：高、低

坑：有、无　(VCD、DVD光盘)

逻辑：真、假

……………………

● 十进制整数转换成二进制整数　除2取余法

【例2-1】 将十进制数97转换成二进制数，其过程如下：

2	97	
2	48	余数为1，即a_0 =1
2	24	余数为0，即a_1 =0
2	12	余数为0，即a_2 =0
2	6	余数为0，即a_3 =0
2	3	余数为0，即a_4 =0
2	1	余数为1，即a_5 =1
	0	余数为1，即a_6 =1；商为0，结束

将十进制数除以2，得到一个商数和一个余数；再将商数除以2，又得到一个商数和一个余数；继续这个过程，直到商数等于零为止。每次得到的余数（必定是0或1）就是对应二进制数的各位数字。

注意：第一次得到的余数为二进制数的最低位，最后一次得到的余数为二进制数的最高位。

最后结果为:

$(97)_{10}=(a_6\ a_5\ a_4\ a_3\ a_2\ a_1\ a_0)_2=(1100001)_2$

二进制整数转换成十进制整数

【例2-1B】 将二进制数$(1100001)_2$转换成十进制数。

$(1\quad 1\quad 0\quad 0\quad 0\quad 0\quad 1)_2$

$2^6\quad 2^5\quad 2^4\quad 2^3\quad 2^2\quad 2^1\quad 2^0$

64 32 16 8 4 2 1

$(1100001)_2 = 64 + 32 + 1 = (97)_{10}$

将二进制数的第n位乘以位权2^n，将每一位的值求和。

$(a_n\ a_{n-1}\ \ldots\ldots a_2\ a_1\ a_0)_2 = (a_n * 2^n + a_{n-1} * 2^{n-1} + \ldots\ldots + a_2 * 2^2 + a_1 * 2^1 + a_0 * 2^0)_{10}$

乘2取整法

- 十进制小数转换成二进制小数

【例2-2】 将十进制小数0.6875转换成二进制小数，其过程如下：

用2乘十进制小数，得到一个整数部分和一个小数部分；再用2乘小数部分，又得到一个整数部分和一个小数部分；继续这个过程，直到余下的小数部分为0或满足精度要求为止。最后将每次得到的整数部分(必定是0或1)从左到右排列即得到所对应的二进制小数。

$$\begin{array}{rl}
0.6875 & \\
\times \quad\quad 2 & \\
\hline
1.3750 & \text{整数部分为1，即} a_{-1}=1 \\
0.3750 & \text{余下的小数部分} \\
\times \quad\quad 2 & \\
\hline
0.7500 & \text{整数部分为0，即} a_{-2}=0 \\
0.7500 & \text{余下的小数部分} \\
\times \quad\quad 2 & \\
\hline
1.5000 & \text{整数部分为1，即} a_{-3}=1 \\
0.5000 & \text{余下的小数部分} \\
\times \quad\quad 2 & \\
\hline
1.0000 & \text{整数部分为1，即} a_{-4}=1 \\
0.0000 & \text{余下的小数部分为0}
\end{array}$$

最后结果为 $(0.6875)_{10}=(0.a_{-1}\ a_{-2}\ a_{-3}\ a_{-4})_2=(0.1011)_2$

一个十进制小数不一定能完全准确地转换成二进制小数。在这种情况下，可以根据精度要求只转换到小数点后某一位为止。

例如，十进制小数0.32不能完全准确地转换成二进制小数。其转换过程如下：

0. 3 2
× 2
---
0. 6 4　整数部分为0，即a_{-1} =0
× 2
---
1. 2 8　整数部分为1，即a_{-2} =1
0. 2 8　余下的小数部分
× 2
---
0. 5 6　整数部分为0，即a_{-3} =0
× 2
---
1. 1 2　整数部分为1，即a_{-4} =1
0. 1 2　余下的小数部分
× 2
---
0. 2 4　整数部分为0，即a_{-5} =0

0. 2 4
× 2
---
0. 4 8　整数部分为0，即a_{-6} =0
× 2
---
0. 9 6　整数部分为0，即a_{-7} =0
× 2
---
1. 9 2　整数部分为1，即a_{-8} =1
0. 9 2　余下的小数部分
× 2
---
1. 8 4　整数部分为1，即a_{-9} =1
0. 8 4　余下的小数部分
× 2
---
1. 6 8　整数部分为1，即a_{-10} =1
…………

上述过程可以无休止地做下去，这说明十进制小数0.32不能准确地转换为二进制小数。在这种情况下，可以根据精度要求取到二进制小数点后的某一位为止，最后得到的只是近似的二进制小数。如果要求取到二进制小数点后第4位，则可以得到：

$$(0.32)_{10} \approx (0.0101)_2$$

实际上，这个二进制小数对应的十进制小数为：

$$(0.0101)_2 = (0.3125)_{10}$$

如果要求取到二进制小数点后第8位，则可以得到：

$$(0.32)_{10} \approx (0.01010001)_2$$

实际上，这个二进制小数对应的十进制小数为：

$$(0.01010001)_2 = (0.31640625)_{10}$$

二进制小数转换成十进制小数

【例2-2B】 将二进制小数$(0.1011)_2$转换成十进制小数。

$$
\begin{array}{ccccc}
(0. & 1 & 0 & 1 & 1\ )_2 \\
 & 2^{-1} & 2^{-2} & 2^{-3} & 2^{-4} \\
 & 0.5 & 0.25 & 0.125 & 0.0625
\end{array}
$$

$$
\begin{aligned}
(0.1011)_2 &= 0.5 + 0.125 + 0.0625 \\
&= 0.625 + 0.0625 \\
&= (0.6875)_{10}
\end{aligned}
$$

【例2-2B2】 将二进制小数$(0.01010001)_2$转换成十进制小数。

$$
\begin{array}{ccccccccc}
(0. & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1\ )_2 \\
 & 2^{-1} & 2^{-2} & 2^{-3} & 2^{-4} & 2^{-5} & 2^{-6} & 2^{-7} & 2^{-8} \\
 & 0.5 & 0.25 & 0.125 & 0.0625 & \cdots\cdots & & & 0.00390625
\end{array}
$$

$$
\begin{aligned}
(0.01010001)_2 &= 0.25+0.0625+0.00390625 \\
&= 0.3125 + 0.00390625 \\
&= (0.31640625)_{10}
\end{aligned}
$$

> 将二进制小数的第-n位乘以位权2^{-n}，将每一位的值求和。
>
> $(0.a_{-1}\ a_{-2}\ \ldots\ldots\ a_{-n})_2 = (a_{-1} * 2^{-1} + a_{-2} * 2^{-2} + \ldots\ldots + a_n * 2^{-n})_{10}$

一般的十进制数转换成二进制数

为了将一个既有整数部分又有小数部分的十进制数转换成二进制数，可以将其整数部分和小数部分分别转换，然后再组合起来。

例如：

$(97)_{10}=(1100001)_2$ $(0.6875)_{10}=(0.1011)_2$

由此可得 $(97.6875)_{10}=(1100001.1011)_2$

3 十六进制

- 十六进制　　逢16进1，用A~F表示10~15
- 十进制整数转换成十六进制整数　　除16取余法

【例2-3】 将十进制数986转换成十六进制数。

$$
\begin{array}{r|l}
16 & 986 \\
\hline
16 & 61 \\
\hline
16 & 3 \\
\hline
 & 0
\end{array}
$$

余数为10，即a_0 =A

余数为13，即a_1 =D

余数为3，　即a_2 =3；商为0，结束

最后结果为:
$(986)_{10}=(a_2\ a_1\ a_0)_{16}=(3DA)_{16}$

- 十六进制整数转换成十进制整数

将十六进制整数的第n位乘以位权16^n，然后将每一位的值求和。

【例2-3B】 将十六进制数$(3DA)_{16}$转换成十进制数。

(3	D	A $)_{16}$
16^2	16^1	16^0
256	16	1

$(3DA)_{16}=(3*16^2+13*16^1+10*16^0)_{10}$

$=(3*256+13*16+10*1)_{10}$

$=(986)_{10}$

● 十进制小数转换成十六进制小数

乘16取整法

【例2-4】 将十进制小数0.84375转换成十六进制小数。

```
  0.8 4 3 7 5
×         1 6
--------------
  5 0 6 2 5 0
+ 8 4 3 7 5
--------------
 13.5 0 0 0 0     整数部分为13，即a-1 =D
  0.5 0 0 0 0     余下的小数部分
×         1 6
--------------
  3 0 0 0 0 0
+ 5 0 0 0 0
--------------
  8.0 0 0 0 0     整数部分为8，即a-2 =8
  0.0 0 0 0 0     余下的小数部分为0，结束
```

整数部分为13，即a_{-1} =D

整数部分为8，即a_{-2} =8

$(0.84375)_{10} = (0.D8)_{16}$

同样，一个十进制小数不一定能完全准确地转换成十六进制小数。在这种情况下，可以根据精度要求只转换到小数点后某一位为止。

例如，十进制小数0.32不能完全准确地转换成十六进制小数。其转换过程如下：

```
     0.3 2
 ×     1 6
 ─────────
     1 9 2
 +   3 2
 ─────────
     5.1 2        整数部分为5，即a_{-1} =5
     0.1 2        余下的小数部分
 ×     1 6
 ─────────
       7 2
 +   1 2
 ─────────
     1.9 2        整数部分为1，即a_{-2} =1
     0.9 2        余下的小数部分
 ×     1 6
 ─────────
         ⋮
```

上述过程可以无休止地做下去，这说明十进制小数0.32不能准确地转换为十六进制小数。在这种情况下，可以根据精度要求取到十六进制小数点后的某一位为止，最后得到的只是近似的十六进制小数。如果要求取到十六进制小数点后第1位，则可以得到：

$$(0.32)_{10} \approx (0.5)_{16}$$

实际上，0.5这个十六进制小数对应的十进制小数为：

$$(0.5)_{16} = (0.3125)_{10}$$

如果要求取到十六进制小数点后第2位，则可以得到：

$$(0.32)_{10} \approx (0.51)_{16}$$

实际上，0.51这个十六进制小数对应的十进制小数为：

$$(0.51)_{16} = (0.31640625)_{10}$$

十六进制小数转换成十进制小数

将十六进制小数的第-n位乘以位权16^{-n}，将每一位的值求和。

【例2-4B】 将十六进制小数 $(0.D8)_{16}$转换成十进制小数。

$(0.\quad D \quad 8 \quad)_{16}$

$16^{-1} \quad 16^{-2}$

$0.0625 \quad 0.00390625$

最终结果：

$$(0.D8)_{16} = (13*16^{-1} + 8*16^{-2})_{10}$$

$$= (13*0.0625 + 8*0.00390625)_{10}$$

$$= (0.84375)_{10}$$

一般的十进制数转换成十六进制数

在将一个十进制数转换成十六进制数时，需要将整数部分和小数部分分别进行转换。

【例2-5】十进制数986.84375转换成十六进制数的过程如下：

先转换整数部分

$$(986)_{10}=(3DA)_{16}$$

再转换小数部分

$$(0.84375)_{10}=(0.D8)_{16}$$

最后合起来结果为：

$$(986.84375)_{10}=(3DA.D8)_{16}$$

4 八进制

- 八进制　逢8进1，八进制数中只会出现0-7，没有8和9
- 十进制整数转换成八进制整数　**除8取余法**

【例2-6】将十进制数277转换成八进制数。

8	277	
8	34	余数为5，即 $a_0=5$
8	4	余数为2，即 $a_1=2$
	0	余数为4，即 $a_2=4$
		商为0，结束

最后结果为: $(277)_{10}=(425)_8$

【例2-7】十进制小数0.140625转换成八进制小数。

- 十进制小数转换成八进制小数　**乘8取整法**

0.140625	
× 8	
1.125000	整数部分为1，即 $a_{-1}=1$
0.125000	余下的小数部分
× 8	
1.000000	整数部分为1，即 $a_{-2}=1$
0.000000	余下的小数部分为0

最后结果为: $(0.140625)_{10}=(0.11)_8$

● 八进制整数转换成十进制整数

将八进制整数的第n位乘以位权8^n，将每一位的值求和。

【例2-6B】 将八进制整数425转换成十进制整数。

$(4\quad 2\quad 5\quad)_8$

$8^2\quad 8^1\quad 8^0$

$64\quad 8\quad 1$

$$(425)_8 =(4*8^2 +2*8^1 + 5*8^0)_{10}$$
$$=(4*64 +2*8 + 5*1)_{10}$$
$$=(256+16+5)_{10}$$
$$=(277)_{10}$$

● 八进制小数转换成十进制小数

将八进制小数的第-n位乘以位权8^{-n}，将每一位的值求和。

【例2-7B】 将八进制小数$(0.11)_8$转换成十进制小数。

$(0.\quad 1\quad 1\quad)_8$

$8^{-1}\quad 8^{-2}$

$0.125\quad 0.015625$

$$(0.11)_8 =(1*8^{-1} + 1* 8^{-2})_{10}$$
$$= (0.125 + 0.015625)_{10}$$
$$= (0.140625)_{10}$$

5 各种计算机记数制之间的转换

	十 进 制	二 进 制	八 进 制	十六进制
基 数	10	2	8	16
位 权	10^K	2^K	8^K	16^K
数字符号	0～9	0, 1	0～7	0～9与A～F

十进制以及计算机常用记数制的表示法

表2.2　计算机常用记数制的表示

十进制	二进制	八进制	十六进制
0	0	0	0
1	1	1	1
2	10	2	2
3	11	3	3
4	100	4	4
5	101	5	5
6	110	6	6
7	111	7	7
8	1000	10	8
9	1001	11	9
10	1010	12	A
11	1011	13	B
12	1100	14	C
13	1101	15	D
14	1110	16	E
15	1111	17	F
16	10000	20	10

● 十六进制数与八进制数转换成二进制数

【例2-8】 十六进制数$(2BD.C)_{16}$转换成二进制数为

$$\begin{array}{ccccc} 2 & B & D & . & C \\ \downarrow & \downarrow & \downarrow & \downarrow & \downarrow \\ 0010 & 1011 & 1101 & . & 1100 \end{array}$$

即 $(2BD.C)_{16}=(1010111101.11)_2$

每位十六进制数用相应的四位二进制数代替

【例2-9】 八进制数$(315.27)_8$转换成二进制数为

3	1	5	.	2	7
↓	↓	↓	↓	↓	↓
011	001	101	.	010	111

即$(315.27)_8=(11001101.010111)_2$

每位八进制数用相应的三位二进制数代替

● 二进制数转换成十六进制数或八进制数

【例2-10】 二进制数$(1101001101.01)_2$转换成十六进制数为

11	0100	1101	.	0100
↓	↓	↓	↓	↓
3	4	D	.	4

即 $(1101001101.01)_2=(34D.4)_{16}$

从小数点开始，向前每四位一组构成一位十六进制数；向后每四位一组构成一位十六进制数，当最后一组不够四位时，应在后面添0补足四位

【例2-11】二进制数$(1101001101.01)_2$转换成八进制数为

1 101 001 101 . 010

↓ ↓ ↓ ↓ ↓ ↓

1 5 1 5 . 2

即 $(1101001101.01)_2=(1515.2)_8$

从小数点开始，向前每三位一组构成一位八进制数；向后每三位一组构成一位八进制数，当最后一组不够三位时，应在后面添0补足三位

2.1.2 计算机中数的表示

1. 正负数的表示

在计算机中，一个数的正、负号也是用一个二进制位来表示。一般将整个二进制数的最高位定为二进制数的符号位。符号位为“0”时表示正数，符号位为“1”时表示负数。

2^8-1

8位 最大无符号数为255 有符号的整数范围为-127～127 2^7-1

16位 最大无符号数为65535 有符号的整数范围-32767～32767 $2^{15}-1$

$2^{16}-1$

32位 最大无符号数为4294967295 $2^{32}-1$

有符号的整数范围-2147483647～2147483647 $2^{31}-1$

64位 最大无符号数为18446744073709551615 $2^{64}-1$

有符号的整数范围-9223372036854775807～9223372036854775807

$2^{63}-1$

说明：

如果用8个二进制位表示一个无符号的数，由于不考虑数的符号问题，该8位都可以用来表示数值，因此，8个二进制位可以表示的最大无符号数为255(即8位全是“1”)。

如果用8个二进制位表示一个有符号的整数，由于最高位为符号位，具体表示数值的只有7位。在这种情况下，所能表示的数值范围为-127～127。

例如，十进制数+50和-50用8位二进制数表示以及转换成相应的十六进制数分别为

$$(+50)_{10}=(00110010)_{2}=(32)_{16}$$

$$(-50)_{10}=(10110010)_{2}=(B2)_{16}$$

其中二进制表示中最左边的二进制位(称为最高位)为符号位，0表示正，1表示负。如果用十六进制表示，则只要每四位作为一组，每一组分别用十六进制表示对应一个十六进制位。

如果用16个二进制位表示一个无符号的数，由于不考虑数的符号问题，该16位都可以用来表示数值，因此，16个二进制位可以表示的最大无符号数为65535(即16位全是1)。

如果用16个二进制位表示一个有符号的整数，由于最高位为符号位，具体表示数值的只有15位。在这种情况下，所能表示的数值范围为-32767～32767。

例如，十进制数+513和-513用16位二进制数表示以及转换成相应的十六进制数分别为

$$(+513)_{10}=(0000001000000001)_{2}=(0201)_{16}$$

$$(-513)_{10}=(1000001000000001)_{2}=(8201)_{16}$$

其中二进制表示中最左边的二进制位(称为最高位)为符号位，0表示正，1表示负。如果用十六进制表示，则只要每四位作为一组，每一组分别用十六进制表示对应一个十六进制位。显然，用8位二进制数是无法表示这两个数的。由此可以看出，如果使用的二进制位数越多，则能表示的数值的范围就越大。

上面的例子，十进制数+50和-50用8位二进制数表示以及转换成相应的十六进制数分别为：

$$(+50)_{10}=(00110010)_2=(32)_{16}$$

$$(-50)_{10}=(10110010)_2=(B2)_{16}$$

十进制数+50和-50如果用16位二进制数表示以及转换成相应的十六进制数分别为：

$$(+50)_{10}=(0000000000110010)_2=(0032)_{16}$$

$$(-50)_{10}=(1000000000110010)_2=(8032)_{16}$$

十进制数+50和-50如果用32位二进制数表示以及转换成相应的十六进制数分别为：

$$(+50)_{10}=(00000000000000000000000000110010)_2=(00000032)_{16}$$

$$(-50)_{10}=(10000000000000000000000000110010)_2=(80000032)_{16}$$

2．定点数

所谓定点数是指小数点位置固定的数。在计算机中，通常用定点数来表示整数与纯小数，分别称为定点整数与定点小数。

（1）定点整数

在定点整数中，一个数的最高二进制位是符号位，用以表示数的符号；而小数点的位置默认为在最低(即最右边)的二进制位的后面，但小数点不单独占一个二进制位。因此，在一个定点整数中，符号位右边的所有二进制位数表示的是一个整数值。

（2）定点小数

在定点小数中，一个数的最高二进制位是符号位，用以表示数的符号；而小数点的位置默认为在符号位的后面，它也不单独占一个二进制位。因此，在一个定点小数中，符号位右边的所有二进制位数表示的是一个纯小数。

3．原码、反码、补码与偏移码

（1）原码

● 所谓原码就是前面所介绍的二进制定点数表示。即原码的符号位在最高位，0表示正，1表示负，数值部分按一般的二进制形式表示。

例如：$(+50)_{10}$的8位二进制原码为: $(00110010)_{原}$

$(-50)_{10}$ 的8位二进制原码为: $(10110010)_{原}$

$(+33)_{10}$的8位二进制原码为: $(00100001)_{原}$

● 在二进制原码中，使用的二进制位数越多，所能表示的数的范围就越大。一般来说，如果用n位二进制来存放一个定点整数的偏移码，能表示的整数值范围为:

$-2^{n-1}+1 \sim 2^{n-1}-1$。

（2）反码

- 正数的反码和原码相同，负数的反码是对该数的原码除符号位外其余各位都取反(即将0变为1，1变为0)。

例如： $(+50)_{10}$的8位二进制反码为：$(00110010)_{反}$

$(-50)_{10}$ 的8位二进制反码为：$(11001101)_{反}$

$(+33)_{10}$的8位二进制反码为：$(00100001)_{反}$

$(-33)_{10}$ 的8位二进制反码为：$(11011110)_{反}$

$(00000000)_{反}$所表示的十进制数为： 0

$(11111111)_{反}$ 所表示的十进制数为：-0

（3）补码

- 正数的补码和原码相同，负数的补码是在该数的反码的最后(即最右边)一位上加1。
- 一个数的补码的补码就是这个数的原码本身。
- 一般来说，如果用n位二进制来存放一个定点整数的补码，能表示的整数值范围为 $-2^{n-1} \sim 2^{n-1}-1$ 。

【例2-13】 采用二进制补码计算

$(33)_{10}-(50)_{10}$ 和 $(33)_{10}+(50)_{10}$

由于 $(33)_{10}-(50)_{10}=(-50)_{10}+(+33)_{10}$

因此

	1 1 0 0 1 1 1 0	$(-50)_{10}$的8位二进制补码
+)	0 0 1 0 0 0 0 1	$(+33)_{10}$的8位二进制补码
	1 1 1 0 1 1 1 1	

最后得到 $(-50)_{10}+(+33)_{10}=(11001110)_{补}+(00100001)_{补}$

$=(11101111)_{补}$

$=(10010001)_{原}$ (对上述补码"除符号位外各位求反末位加1"后得到)

$=(-17)_{10}$

下面计算$(33)_{10}+(50)_{10}$

$$
\begin{array}{r l}
0\,0\,1\,0\,0\,0\,0\,1 & (+33)_{10}\text{的8位二进制补码} \\
+)\ 0\,0\,1\,1\,0\,0\,1\,0 & (+50)_{10}\text{的8位二进制补码} \\
\hline
0\,1\,0\,1\,0\,0\,1\,1 &
\end{array}
$$

最后得到 $(+33)_{10}+(+50)_{10}=(00100001)_{补}+(00110010)_{补}$

$=(01010011)_{补}$

$=(01010011)_{原}$ (正数的补码与原码相同)

$=(83)_{10}$

提示： 补码使得正负数的加减法和普通数一样而不必考虑符号位，因此计算机中使用补码存整数，以便于计算。

（4）偏移码

不管是正数还是负数，其补码的符号位取反即是偏移码。由此可知，定点数用偏移码表示后，其最高位也为符号位，但符号位的取值刚好和原码与补码相反，1表示正，0表示负；而其数值部分与相应的补码相同。

$(33)_{10} = (00100001)_{补} = (10100001)_{偏移码}$

$(-50)_{10} = (11001110)_{补} = (01001110)_{偏移码}$

$(0)_{10} = (00000000)_{补} = (10000000)_{偏移码}$

$(-128)_{10} = (10000000)_{补} = (00000000)_{偏移码}$

$(127)_{10} = (01111111)_{补} = (11111111)_{偏移码}$

一般来说，如果用n位二进制来存放一个定点整数的偏移码，能表示的整数值范围为 $-2^{n-1} \sim 2^{n-1}-1$

定点数用偏移码表示后，也可以执行加减运算，必须将结果的符号位取反后才是偏移码形式的结果。例如，

0 1 0 0 1 1 1 0	$(-50)_{10}$的8位二进制偏移码
+) 1 0 1 0 0 0 0 1	$(+33)_{10}$的8位二进制偏移码
1 1 1 0 1 1 1 1	

其结果(11101111)并不是$(-17)_{10}$的偏移码，$(-17)_{10}$的偏移码为$(01101111)_{偏移码}$。即

$(-50)_{10}+(+33)_{10}=(01001110)_{偏移码}+(10100001)_{偏移码}$

$=(01101111)_{偏移码}$

$=(10010001)_{原}$ (对上述偏移码“各位求反（包括符号位）末位加1”后得到)

$=(-17)_{10}$

最后要指出，在二进制定点数的四种表示中，原码比较直观，但不能用于具体运算；补码与偏移码可用于具体运算；反码只起到由原码转换为补码或偏移码的中介作用。

4. 浮点数

一个既有整数部分又有小数部分的十进制数R可以表示成如下形式：

$R = Q \times 10^n$

其中Q为一个纯小数，n为一个整数。

对于既有整数部分又有小数部分的二进制数P也可以表示成如下形式：

$P = S \times 2^N$

其中S为一个二进制定点小数，称为P的尾数；N为一个二进制定点整数，称为P的阶码，它反映了二进制数P的小数点的实际位置。

在计算机中，通常用一串连续的二进制位来存放二进制浮点数，它的一般结构为：

尾数S（定点小数原码或补码）	阶码N（定点整数原码或偏移码）

二进制最高位　　　　　　　　二进制最低位

【例2-14】 用16位二进制定点小数补码以及8位二进制定点整数补码表示十进制数-254.75。

首先将$(-254.75)_{10}$转换成二进制数，即$(-254.75)_{10}=(-11111110.11)_2$

$=(-0.1111111011)_2\times 2^8$

然后将尾数化成16位二进制定点小数，即

$S=(-0.1111111011)_2=(\,1\,1\,1\,1\,1\,1\,1\,1\,0\,1\,1\,0\,0\,0\,0\,0\,)_2$

小数点位置

其反码为 $S=(1000000010011111)_{反}$

补码为 $S=(1000000010100000)_{补}=(80A0)_{16}$

最终十进制数-254.75用十六进制表示为：$(80A008)_{16}$

将阶码8也转换成二进制数，即 $(+8)_{10}=(+1000)_2$

化成8位二进制定点整数为 $N=(+1000)_2=(\,0\,0\,0\,0\,1\,0\,0\,0\,)_2$

小数点位置

其补码为（正数的补码是原码本身） $N=(00001000)_{补}=(08)_{16}$

2.2 常量与变量

常量 ● 在程序执行过程中其值不变的量（不能改变的量）

变量 ● 在程序执行过程中其中的值可以改变的量

其中C语言中常用的基本数据类型有以下四种：

1）整型：包括有符号基本整型(int或signed int)、无符号基本整型(unsigned int)、有符号长整型(long或signed long)、无符号长整型(unsigned long)、有符号短整型(short或signed short)、无符号短整型(unsigned short)、有符号超长整型(long long 或 _int64)、无符号超长整型(unsigned long long)

2）实型：分为单精度实型(float)与双精度实型(double)

3）字符型：分为有符号字符型（char或signed char）、无符号字符型（unsigned char）

4）空类型：void类型

除了上述四种基本数据类型外，C语言还有枚举类型、构造类型、指针类型等复合数据类型。

C语言规定，程序中的每一个变量都有一个唯一的名字，称为变量名。变量在使用前必须首先定义。所谓定义一个变量，就是系统根据变量的数据类型为该变量分配存储空间，变量名即代表其存储空间，以便在程序执行过程中在这个存储空间中存取数据。

- 变量名的命名要符合下列两个规则：

1）变量名必须以字母或下划线开头，后面可以跟若干个字母、数字或下划线。区分大小写字母。 例如：_a _A a123 A123 __

2）不同的编译系统对变量名中的字符总个数有不同的规定。

在某些编译系统中，允许使用长达31个字符的变量名。还有的编译系统不限制变量名的长度。变量名最好起的有意义，做到“望名生意”，现在常用的变量命名法有微软的 *匈牙利命名法* 等。

在C语言变量名中，英文字母是区分大小写的。Day与day是两个不同的变量名。

2.3 基本数据类型常量

注：以下B代表字节(Byte)

2.3.1 整型常量

- 整型常量的分类

四种类型的整型常量:

- 有符号与无符号基本整型常量
- 有符号与无符号短整型常量
- 有符号与无符号长整型常量
- 有符号与无符号超长整型常量

在一般编译器上，长整型=基本整型

有符号

短整型常量	2B	-32768～32767(-2^{15}～$2^{15}-1$)
基本整型常量	4B	-2147483648～2147483647(-2^{31}～$2^{31}-1$)
超长整型常量	8B	-9223372036854775808～9223372036854775807(-2^{63}～$2^{63}-1$)

无符号

短整型常量	2B	0～65535($2^{16}-1$)
基本整型常量	4B	0～4294967295($2^{32}-1$)
超长整型常量	8B	0～18446744073709551615($2^{64}-1$)

- 整型常量的表示

（1）十进制表示　可以使用的符号有十个数字符号0～9以及+与-。

（2）十六进制表示　整型常量以0x或0X开头，符号有0～9与A～F(或a～f)

（3）八进制表示　整型常量以0(零)开头，可以使用的符号有0～7。

在用十进制表示的整型常量中，

● 对于正整数，前面的“+”号可以省略。例如，+123，123，756，-234都是合法的整型常量，其中+123与123是等值的。

● 对于长整型常量，一般要在长整型常量的后面加一个英文字母L或l。

● 对于无符号整型常量，一般要在无符号整型常量的后面加一个英文字母U或u。

● 对于64位的超长整型，一般要在超长整型常量的后面加i64或两个英文字母LL或ll。

例如，123u、123L、123LL、123i64虽然其数值是相同的，但123u是一个无符号基本整型常量，在计算机中用4个字节存放它。而123L是一个长整型常量，在计算机中也要用4个字节存放它。而123LL和123i64是一个64位超长整型常量，在计算机中要用8个字节存放它。

2.3.2 实型(浮点型)常量

必须要有

（1）十进制表示　包括符号“+”与“-”，0～9十个数字以及小数点“.”。

（2）指数形式(科学记数法)

包括符号“+”与“-”，0～9十个数字，小数点“.”以及e(或E)。其中e（或E）后面应为整数。

注意

① 在符号e的前面必须要有数字。

② 在符号e的后面必须为整数，即不能是带有小数点的实型数。

实型常量　$P=S\times 2^{N}$

其中S为一个二进制定点小数，称为P的尾数；N为一个二进制定点整数，称为P的阶码，它反映了二进制数P的小数点的实际位置。

63	62 … 52	51 … 0
	(阶码N-2)的偏移码(11位)	尾数S原码数值部分的后52位
尾数符号		加$(0.1)_2$

double数的存储形式(8B)　　IEEE 754 double 标准格式

例如，十进制实型数97.6875，其尾数原码与阶码如下：

$$(97.6875)_{10}=(1100001.1011)_2$$

$$=(0.11000011011)_2\times 2^7$$

因此，其54位尾数原码S为：

$$S=(\textcolor{red}{01}1000001101100000000000000000000000000000000000000000)_{原码}$$

11位(阶码N-2)的偏移码为：

$$(7-2)_{10}=(10000000101)_{偏移码}$$

0 10000000101 1000001101100000000000000000000000000000000000000000

0100 0000 0101 1000 0110 1100 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000

用十六进制表示为 40 58 6C 00 00 00 00 00，共占8个字节（64位）。

由于计算机系统分配给一个数据的存储空间是有限的。一般来说，一个实型常量无法转换成与之等值的有限位的二进制数据，其有限位以后的数字将被舍去，由此就会产生<u>舍入误差</u>。

实型常量 $P=S\times 2^N$

其中：S为P的尾数是一个二进制定点小数，

N为P的阶码是一个二进制定点整数。

31	30 … 23	22 … 0
	(阶码N-2)的偏移码(8位)	尾数S原码数值部分的后23位

尾数符号　　加$(0.1)_2$

float数的存储形式(4B)　　IEEE 754 float 标准格式

例如，十进制实型数97.6875，其尾数原码与阶码如下：

$(97.6875)_{10}=(1100001.1011)_2=(0.11000011011)_2\times 2^7$

因此，其25位尾数原码S为：

$S=(0110000110110000000000000)_{原码}$

8位(阶码N-2)的偏移码为：$(7-2)_{10}=(10000101)_{偏移码}$

0 10000101 10000110110000000000000

0100 0010 1100 0011 0110 0000 0000 0000

用十六进制表示为 42 C3 60 00，共占4个字节（32位）。

验证：打印出double和float型变量内部表示的十六进制值：

```
#include  <stdio.h>
main( )
{double x=97.6875;   float y=97.6875f;
  unsigned char *p;
  p = (unsigned char *)&x;    /* p指向双精度数8个字节的第1个字节 */
  printf("%02X %02X %02X %02X %02X %02X %02X %02X\n",
        *p, *(p+1), *(p+2), *(p+3), *(p+4), *(p+5), *(p+6), *(p+7));
  p = (unsigned char *)&y;    /* p指向单精度数4个字节的第1个字节 */
  printf("%02X %02X %02X %02X\n", *p, *(p+1), *(p+2), *(p+3));
}    /* 说明：%02X 是以两位十六进制方式输出指针所指的字节中的值  */
```

在win7+VS2008上的运行结果：

00 00 00 00 00 6C 58 40

00 60 C3 42

因为Intel CPU x86是小端对齐，一个变量的2/4/8字节颠倒存放。
请自己了解：大端（big-endian)对齐 与 小端（little-endian)对齐

浮点数阶码为什么存的时候取N-2呢？

因为IEEE 754标准中，一个规格化的浮点数x的真值表示为：

$$x = (-1)^S \times (1.M) \times 2^E$$

因为尾数的第一个1要隐藏。

IEEE 754规定：浮点数阶码E采用“移码-1”表示，请记住这一点。

为什么指数移码要减去1？这是IEEE 754对阶码的特殊要求，以满足特殊情况，比如对正无穷的表示。

我们的教程上是把尾数变成0.1M的纯小数形式，阶码与1.M的阶码差了一个1，连同规定减去的1，因此需要把阶码N减去2。

【例2-15】 下列C程序的功能是将10个实型数0.1进行累加，然后将累加结果输出。

```
#include  <stdio.h>
main( )
{    int k;                              /*定义整型变量k*/
     double x, z;                        /*定义双精度是型变量x与z*/
     z=1.0;                              /*实数1.0赋给变量z*/
     x=0.0;
     for (k=0; k<10; k++)
         x=x+0.1;                        /* 10个0.1累加到变量x中 */
     printf("z= %20.17f\n", z);          /*输出变量z的值*/
     printf("x= %20.17f\n", x);          /*输出变量x的值*/
}
```

运行这个程序后，输出的结果如下：

z= 1.00000000000000000

x= 0.99999999999999989

有积累误差。

2.3.3 字符型常量

- 转义字符

'\n'	换行
'\r'	回车（不换行）
'\b'	退格
'\t'	制表(横向跳格)
'\''	单引号(单撇号)
'\"'	双引号(双撇号)
'\ddd'	1～3位八进制数所代表的ASCII码字符
'\xhh'	1～2位十六进制数所代表的ASCII码字符
'\f'	走纸换页
'\\'	反斜杠字符
'\a'	响铃一次

说明：

- **字符常量**： **'A'** , '\n', '0', '\0'， 占一个字节。单个字符用单引号引起来。
- 其中\为转义符，使字符表示别的意思，表示一些无法直接书写的字符。

（1）'\n'不再表示字符'n'，而是表示回车换行

（2）'\r'不再表示字符'r'，**表示**回车不换行

（3）'\t'不再表示字符't'，而是表示制表符Tab

（4）'\0'不再表示字符'0'，而是表示字符串结束符，其ASCII内码为0

（5） '\'' 表示单引号字符

（6） '\ddd' 为八进制的ASCII字符。 最大'\377'

（7） '\xdd' 为十六进制的ASCII字符, 最大'\xFF'

（8） '\a' 响铃一次（等价于'\007' ）

注意：

(1) 'ab' 是错误的，单引号中只能是单个字符。

但'\xab'是正确的。

(2) '377'是错误的，但'\377'是正确的。

2.4 基本数据类型变量的定义

2.4.1 整型变量的定义

（1）基本整型变量

定义基本整型变量的形式如下： int 变量表列；

（2）长整型

定义长整型变量的形式如下： long [int] 变量表列；其中int可以省略。

（3）短整型

定义短整型变量的形式如下：short [int] 变量表列；其中int可以省略。

（4）无符号整型

定义无符号基本整型变量的形式如下： unsigned [int] 变量表列；
其中int可以省略。 也可以是： unsigned long 或 unsigned short

（5）超长整型

定义超长整型变量的形式如下： long long 变量表列；

说明

● 一个类型说明语句可以同时定义多个同类型的变量，各变量之间用逗号“,”分隔。多个同类型的变量也可以用多个类型说明语句定义。

● 用类型说明语句定义的变量只是说明了为这些变量分配了存储空间，以便用于存放与之相同类型的数据，在未对这些变量赋值前，这些变量中（即存储空间中）的值是随机的。

● C语言允许在定义变量的同时为变量赋初值。

● 在为长整型变量初始化或赋值时，如果被赋数据为基本整型常量，则C编译系统自动将被赋数据转换成与相应变量的类型一致。

● 由于各种整型变量所占的字节数有限，因此，它们所能存放的整数有一定的范围。

【例2-16】 阅读下列C程序：

```
#include  <stdio.h>
main( )
{   long x, y, z;
    x= -0xffffL;   y= -0xffL;   z= -0xffffffffL;
    printf("x=%6ld   y=%6ld   z=%6ld\n", x, y, z);
    x= -0xffff;   y= -0xff;   z= -0xffffffff;
    printf("x=%6ld   y=%6ld   z=%6ld\n", x, y, z);
}
```

该程序运行后，输出的结果为

```
x=-65535    y=  -255    z=     1
x=-65535    y=  -255    z=     1
```

【例2-17】 有如下C程序：

```
#include  <stdio.h>
main( )
{ short  x;
  unsigned short  y;
  long z;
  x=65535;
  y=65535;
  z=65535;
  printf("x=%d\n", x);
  printf("y=%u\n", y);
  printf("z=%ld\n", z);
}
```

它们在计算机中用二进制表示如下：

变量x	1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
变量y	1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
变量z	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
	1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1

%d为基本整型输出格式说明符，
%u为无符号基本整型输出格式说明符，
%ld为长整型输出格式说明符

输出结果：
x=-1
y=65535
z=65535

若把【例2-17】程序改为：

```
#include  <stdio.h>
main( )
{ short  x;
  unsigned short  y;
  long z;
  x=75535;
  y=75535;
  z=75535;
  printf("x=%d\n", x);
  printf("y=%u\n", y);
  printf("z=%ld\n", z);
}
```

输出结果：

x=9999

y=9999

z=75535

对于短整型变量x和y来说，2个字节存放不下75535的二进制值，而只能存放后16位的二进制值，其十进制值为9999。

2.4.2 实型变量的定义

单精度

单精度型具有6～7位有效数字

单精度实型变量的定义形式如下： float 变量表列;

定义一个单精度实型变量，实际上给分配了4B的存储空间

双精度

双精度型具有15～16位有效数字

双精度型变量的定义形式如下： double 变量表列;

定义一个双精度实型变量，实际上给分配了8B的存储空间

一个实型类型说明语句可以同时定义多个同类型的变量，各变量之间用逗号“,”分隔，多个同类型的变量也可以用多个类型说明语句定义。

例如：

```c
#include <stdio.h>
main( )
{ float a=1.0f, b=2.5f, c;
  double d=1.0, e, f;
  c=755.35f;
  e=5.35e-5;
  f=1e2;
  printf("a=%f\n", a);
  printf("b=%f\n", b);
  printf("c=%f\n", c);
  printf("d=%f\n", d);
  printf("e=%f\n", e);
  printf("f=%f\n", f);
}
```

```
输出结果：
a=1.000000
b=2.500000
c=755.349976
d=1.000000
e=0.000053
f=100.000000
请按任意键继续. . .
```

%f 默认输出的浮点数6位小数，不足补0

2.4.3 字符型变量的定义

字符型变量用于存放字符型常量。

字符型变量的定义方式如下： **char 变量表列；**

【例2-18】 字符型数据与整型数据的输出。

```
#include <stdio.h>
main( )
{   int  x;
    char  y;
    x = 65;
    y = 'B';
    printf("x=%c\n", x);
    printf("y=%c\n", y);
    printf("y=%d\n", y);
}
```

C编译系统为字符型变量分配一个字节，存放字符的ASCII码。因此，所谓字符型变量存放字符常量，实际上就是存放该字符的ASCII码。

运行结果是：

```
x=A
y=B
y=66
```

一个int型数据占4B，而字符型数据只占1B，因此，在将整型数据以字符形式输出时，只取低字节中的数据作为ASCII码字符输出。

```
#include  <stdio.h>
main( )
{  int x;
   x=1348;  /* 0x544 */
   printf("x=%c\n", x);
}
```

该程序运行后输出结果为

x=D   /* ASCII值：0x44 */

由于字符型数据只占1B，它只能存放0～255范围内的整数。

由于C语言中的字符数据与整数之间可以通用，因此，字符型数据与整型数据之间可以进行混合运算，可以将字符型数据赋给整型变量，也可以将整型数据赋给字符型变量。

【例2-19】英文大小写字母的转换。

```c
#include  <stdio.h>
main( )
{   char   x, y, c1, c2;
     x='A';
     y='B';
     c1=x+32;
     c2=y+32;
     printf("x=%c, y=%c\n", x, y);
     printf("c1=%c, c2=%c\n", c1, c2);
}
```

每一个英文小写字母比它相应的大写字母的ASCII码大32

这个程序的输出结果为

x=A, y=B

c1=a, c2=b

c1=x+32; 可以写成： c1= x - 'A' + 'a' ;

c2=y+32; 可以写成： c2= y - 'A' + 'a';

通过相对位置的计算进行字符大小写转换。

总结一下C的基本类型

包括：算术类型、枚举类型和void类型

其中算术类型包括字符型、整型和实型。（枚举类型和void类型后面再讲）

(a) 字符型　占一个字节

char：取值范围为-128～127 (-2^7～2^7-1)。

unsigned char：取值范围0～255 (0～2^8-1)。

注意:

1) 处理中文信息要使用unsigned char类型。GBK表用两个unsigned char存放一个汉字。

2) 字符型可以作为整型使用，但取值范围小，小心不要溢出。

（b） 整型

short： 16位和32位编译系统都占两个字节，

取值范围都为-32768~32767(-2^{15}~2^{15}-1)。

unsigned short：16位和32位编译系统占两个字节，

取值范围都为 0~65535 (0~2^{16}-1)。

int: 16位系统占两字节，取值范围为

-32768~32767 (-2^{15}~2^{15}-1)。

32位系统占四字节, 取值范围为

-2147483648~2147483647(-2^{31}~2^{31}-1)。 2G

unsigned int：16位系统占两字节，取值范围都为

0~65535 (0~2^{16}-1)。

32位系统: 四字节, 取值范围为

0~4294967295 (0~2^{32}-1)。 4G

long：16位和32位编译系统都占四字节，取值范围为

-2147483648~2147483647 (-2^{31}~$2^{31}-1$)。 2G

unsigned long：16位和32位编译系统都占四个字节，

取值范围为

0~4,294,967,295 (0~$2^{32}-1$)。 4.3×10^{9} 4G

long long：64位，占八个字节，VC++6.0以上编译系统

有此类型，取值范围为 -2^{63}~$2^{63}-1$

(最大数 9,223,372,036,854,775,807 大约9.2×10^{18}） 9E

注意：

VC++6以上版本编译器有写法：

_int64(long long) _int32(long, int), _int16(short), _int8(char)

附加说明：

计算机存储单位一般用B，KB，MB，GB，TB，PB，EB，ZB，YB，BB来表示，它们之间的关系是：

1KB (Kilobyte 千字节) = 1024 B， 10^3

1MB (Megabyte 兆字节 简称“兆”) =1024 KB， 10^6

1GB (Gigabyte 吉字节 又称“千兆”) =1024 MB， 10^9

1TB (Trillionbyte 万亿字节 太字节) =1024 GB， 10^{12}

1PB (Petabyte 千万亿字节 拍字节) =1024 TB， 10^{15}

1EB (Exabyte 百亿亿字节 艾字节) =1024 PB， 10^{18}

1ZB (Zettabyte 十万亿亿字节 泽字节) = 1024 EB, 10^{21}

1YB (Yottabyte 一亿亿亿字节 尧字节) = 1024 ZB, 10^{24}

1BB (Brontobyte 一千亿亿亿字节) = 1024 YB. 10^{27}

其中 $1024=2^{10}$ (2 的10次方)

（c） 实型

float: 占四个字节，取值范围为：

$-10^{38} \sim 10^{38}$, 6-7位有效数字。

最小正小数10^{-38} 1e-38

double: 占八个字节，取值范围为：

$-10^{308} \sim 10^{308}$, 15-16位有效数字。

最小正小数10^{-308} 1e-308

第2次作业：

教材p.43 习题3，4

第2次上机实验

1. 编写程序打印查看97.6875和-97.6875的double型值在计算机内的**IEEE 754**存储格式**按字节的十六进制值，验证是否与教材所给结果一致。同时也打印查**看97.6875和-97.6875的float型值在计算机内的**IEEE 754**存储格式**按字节的十六进制值。**
2. 编写程序分别**打印100与-100**的double、float、long、int、short、char型值在计算机内的**存储按字节的十六进制值，分析结果为什么各自不同。**

附：打印double和float型变量内部表示十六进制值的程序示例：

```
#include <stdio.h>
main( )
{double x=97.6875;    float y=97.6875;
 unsigned char *p ;
 p = (unsigned char *)&x;
 printf("%02X %02X %02X %02X %02X %02X %02X %02X\n",
      *p, *(p+1), *(p+2), *(p+3), *(p+4), *(p+5), *(p+6), *(p+7));
 p = (unsigned char *)&y;
 printf("%02X %02X %02X %02X\n", *p, *(p+1), *(p+2), *(p+3));
}
```

顺序打印出指针所指每个字节的十六进制值。也可以用%llX输出double数的IEEE 754存储内码的十六进制值。

```
printf("%llX\n", x);
```

但不能用%lX输出float数的IEEE 754存储内码的十六进制值。

printf("%lX\n", y); 输出结果是0，

printf("%llX\n", y); 输出结果与printf("%llX\n", x); 完全一样。