

第2章 基本数据及其运算

●要求：

- 1) 掌握C语言的常用数据类型；
- 2) 掌握C语言的常用运算符；
- 3) 熟练掌握输入/输出函数的使用。

1

2.1 基本类型数据

基本数据有三种：

■ 整数

■ 实型

◆ (单精度实型、双精度实型)

■ 字符型

◆ ASCII码

有无穷多的不同整数、浮点数、字符，但计算机只能用有限的二进位表示不同的数据。所以，计算机只能表示有限的整数、有限的浮点数和有限的字符。

C语言还提供了几种聚合类型 (aggregate types)，包括数组、指针、结构、共用体（联合）、位域和枚举。这些复杂类型(数组、指针、结构)在以后的章节中讨论。

2

2.1.1 整型数据

整型数据：

是不带小数点和指数符号的数据，按值内部的最高位不同理解又分两类：

► 有符号数(存储单元最高位作为符号位)

整型 [signed] int, 简写为int

短整型 [signed] short [int], 简写为short

长整型 [signed] long [int], 简写为long

注：它的最高位是整数位。

3

► 无符号数(存储单元最高位作为数据)

◆ 整型 unsigned [int], 简写为unsigned

◆ 短整型 unsigned short [int]

简写为unsigned short

◆ 长整型 unsigned long [int],

简写为unsigned long

例如：int i, j; unsigned short k;
long m, n;

注意：凡方括号中的内容均可省略。

4

整数字长

◆ 字长：指数据在内存中占用的字节数。

1字节(byte)=8个二进制位(bit)。

◆ long为short的2倍字长，int要么和short相同，要么和long相同。

不同的系统字长可能不同。比如，在我们用的系统中，字长为：

◆ char 1字节 [11111111]

◆ short 2字节 [11111111] [11111111]

◆ int 4字节 [11111111] [11111111] [11111111] [11111111]

◆ long 4字节 [11111111] [11111111] [11111111] [11111111]

5

小结：

► 有符号数=1位符号位+n-1位数据位

0/1	n-1
-----	-----

$$-2^{n-1} \sim 2^{n-1} - 1$$

设整数用16位二进位表示

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

↑最高位是符号位：0表示正，其余各位是数据位

带符号整数，值为32767 (即 $2^{15}-1$)

1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1

↑最高位是符号位：1表示负，其余各位是数据位

带符号整数，值为-1 (负数用补码表示)

► 有符号数表示负数，无符号数只能表示正数

6

➤无符号数

$$n \\ 0 \sim 2^n - 1$$

11111111111111111111

16位二进位都是数据位

无符号整数，值为65535 (即 $2^{16}-1$)

■无符号数表示的正数比有符号数大一倍

整型常量默认为int类型，而加上后缀(L或L)后，就是long int类型。在本系统中都占据4个字节

7

整型范围

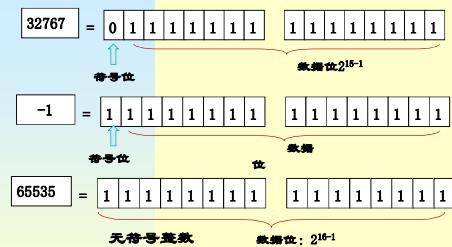
	位数	字节数	取值范围
short	16	2	-32768 ~ 32767
int	32	4	-2147483648 ~ 2147483647
long	32	4	-2147483648 ~ 2147483647
Unsigned short	16	2	0 ~ 65535
unsigned int	32	4	0 ~ 4294967295
unsigned long	32	4	0 ~ 4294967295

- 采用short来表示较小的整数，以节省内存；采用int，long来表示较大的整数，以防止溢出(Overflow)；
- 在常数后面加上L或l表示长整型，加上U或u表示无符号数。例如：1321(L), 122U(u)

8

整型在内存中的存储方式

二进制形式



9

整型变量

C规定在程序中所有用到的变量都必须在使用前指定其类型，即“定义”。例如：

```
void main()
{
    int a, b, c, d; /* 指定a, b, c, d为整型变量 */
    unsigned u; /* 指定u为无符号整型变量 */
    a=12; b=-24; u=10;
    c=a+u; d=b+u;
    printf("a+u=%d, b+u=%d\n", c, d);
}
```

运行结果为：
a+u=22, b+u=-14

可以看到不同类型的整型数据可以进行算术运算。在本例中是int型数据与unsigned int型数据进行相加运算。

10

溢出概念 (Overflow)

各种数据编码都有其数据表示范围，如果在运算过程中出现的数据超出这个表示的范围，称为溢出。

如8位二进制数原码表示的范围是
-127到+127；

如8位二进制数补码表示的范围是
-128到+127。

如16位二进制数原码表示的范围是
-32767到+32767

如16位二进制数补码表示的范围是
-32768到+32767

11

例1：

```
#include <stdio.h>
void main()
{ short a, b;
  a = 32767; /* 32767为16位数的最大数 */
  b = a + 1; /* 加1后发生溢出 */
  printf("%d, %d\n", a, b); // ?
}
验证： //a=-32768;b=a-1;?
32767 (补码) 0 1111111 11111111 最大数
+ 1 (补码) 0 0000000 00000001
=) 1 0000000 00000000 => -32768 最小数
```

12

例2:

```
#include <stdio.h>
void main()
{ short a, b;
  a = 32767; /* 32767为16位数的最大数 */
  a = a + a;
  printf("%d\n", a); //?
}

验证:
32767 (补码) 0 1111111 11111111 最大数
+ 32767 (补码) 0 1111111 11111111
=) 1 1111111 11111110 => -2
```

13

例3:

用补码计算: $-18 - 11 = (-18 + (-11))$

-18:	1 1111111 11101110 (补)
+ -11:	1 1111111 11110101 (补)
=) X:	1 1111111 11100011 (补)
弃溢出1,	1 1111111 11100011 (补) -29

由例1可知: 当带符号的数采用补码形式进行相加时, 可把符号位也当作普通数字一样与数值部分一起进行加法运算, 若符号位上产生进位时, 则自动丢掉, 所得的结果为两数之和的补码形式。如果想得到运算后原码的结果, 可对运算结果再求一次补码即可。

1 0000000 00011101 (原码) -29

14

实型数据(又称浮点数)

■ 实型常量

- ◆ 小数形式: 0.123、.123, -123.0、0.0
- ◆ 指数形式: (科学计数形式)

$\pm Ne \pm A$ 或者 $\pm NE \pm A$ A 为10的幂指数

N 和 A 的数值和符号位, 均采用补码表示
 N 不可省, $+$ 可省, A 必须为整数

正确: 2e3, 3.45e3 错误: e3, -2e3.5

15

实型常量的类型

◆ 默认double型

◆ 在实型常量后加字母f或F, 认为它是float型

实型变量

单精度型

双精度型

长双精度型

float

double

long double

16

例如:

```
float x, y; /* 指定x, y为单精度实数 */
double z; /* 指定z为双精度实数 */
```

在一般系统中, 一个float型数据在内存中占4个字节(32位), 一个double型数据占8个字节(64位)。单精度实数提供7位有效数字, 双精度提供15~16位有效数字, 数值的范围随机器系统而异。

17

	位宽	字节数	实数的绝对值范围	有效数字
单精度float	32	4	$-1.17 \times 10^{-38} \sim 3.4 \times 10^{38}$	7
双精度double	64	8	$-2.2 \times 10^{-308} \sim 1.79 \times 10^{308}$	15 ~ 16

18

■ 说明：

1. 程序实际接受的浮点数和接受书写的浮点数有一定的误差。

```
#include <stdio.h>
void main()
{
    float a; //7位有效数字
    a=111111.111;
    printf("%f", a); 111111.109375
}
```

由于float型变量只能接收7位有效数字，因此最后两位小数不起作用。如果将a改为double型，则能全部接收上述9位数字并存储在变量a中。

19

2. 浮点数运算有一定的计算误差

```
#include <stdio.h>
void main()
{
    float a, b; //double a, b;
    a=100000.0;
    b=99999.999;
    if((a-b)==0.0)printf("a=b: %f\n", a-b);
    else    printf("a > b: %f\n", a-b);
}
```

运行结果：a=b: 0.000000

20

误差的来源

■ 字长有限

■ 比较两个实数是否相等的方法：

绝对误差 $\text{fabs}(x-y) < 1e-6$

相对精度 $\text{fabs}(x-y) \leq \text{fabs}(x*1e-6)$

21

```
/* Func: 解决实数计算精度舍入误差的办法示例 */
#include <stdio.h>
void main()
{
    double a, b;
    a=10000.0;
    b=9999.999;
    if ( (a-b) < 1E-5)
        printf("a = b: %f\n", a-b);
    else
        printf("a > b: %f\n", a-b);
}
```

22

实型数据的存储方式



23

字符型数据

字符常量

在C语言中，字符是按其所对应的ASCII码值来存储的，一个字符占一个字节。例如：字符ASCII码值（十进制）。

1. 单引号括起来的单个字符。

◆ 'a'、'D'、'\$'、'!'

数字编码从48到57, (0 ~ 9)

大写字母编码从65到90, (A ~ Z)

小写字母编码从97到122, (a ~ z)

感叹号！ 33

注意字符'9'和数字9的区别，前者是字符常量，后者是整型常量，它们的含义和在计算机中的存储方式都截然不同。

24

2. 转义字符 (P21, 表2-1) :

C语言还允许使用一种特殊形式的字符常量，就是以反斜杠“\”开头的转义字符。

转义字符及其含义

转义字符	含义	转义字符	含义
\n	换行	\t	水平制表
\v	垂直制表	\b	退格
\‘	单引号	\f	换页
\“	双引号	\\\	反斜线
\ddd	3位8进制数代表的字符	\xhh	2位16进制数代表的字符

25

说明：

(1) 字符的ASCII码可以用八进制或十六进制表示，不能省略前缀X。如一个换行符可用下面任一形式表示：’\n’、’\012’、’\12’、’\xa’

(2) 制表符’\t’的作用是，使当前位置横向移到下一个输出区的开始列位置。如，当前位置是1~8列的某个位置，则用’\t’后，将当前位置移动到第9列。

(3) 如反斜杠之后不是表2-1所列出的字符，则不进行转义。如’\w’就不是转义字符，系统把’\w’当作字符’w’看待。

26

(4) 打印机与显示屏输出的组织方法稍有不同 打印机：

仅当一行字符填满或遇换行符时才输出，即整行一次性输出。当输出空格符或制表符时，作跳格处理，不用空格符填充。

显示器

逐个字符输出，空格符及制表符经过位置都用空格符输出。以上区别，仅当输出字符列中有回车符时才会发生差异

27

ASCII Code (附录B)

- American Standard for Coded Information Interchange, 美国标准信息交换代码
- 采用8位二进制数(1字节)来表示256个字符
- 采用16位二进制数(2字节)来表示65536个字符(汉字)

从ASCII代码表中可以看到每一个小写字母比大写字母的ASCII码大32。即’a’的字母是’A’+32。

28

字符变量

字符变量用字符类型标识符char来定义，字符变量占一个字节的存储单元，只能存放一个字符。

如：char c1, c2; // c1, c2各占1字节

定义了两个字符型变量c1、c2，各可以存放一个字符，可以用下面语句对c1、c2赋值：

c1 = 'A'; c2 = 'B';

值得注意的是，将一个字符常量赋值给一个字符变量，并不是将字符本身放到内存单元中，而是将该字符的ASCII码存储到内存单元中。比如，’A’的ASCII码为65，’B’的ASCII码为66。

29

字符型在内存中的存储方式

如： ’A’

0 1 0 0 0 0 0 1

’B’

0 1 0 0 0 0 1 0

在char前面可以加上unsigned或signed，
unsigned char a; //变量a为无符号字符类型
[signed] char b; //变量b为有符号字符类型
signed char和unsigned char的含义与signed int
和unsigned int相仿，但它只有一个字节。

30

有些系统将字符变量中的最高位作为符号位，也就是将字符处理成带符号的整数，即signed char型，它的取值范围是-128 ~ 127。如果使用ASCII码为0 ~ 127间的字符，由于字节中最高位为0，因此用%d格式符输出时，输出一个正整数。如果使用ASCII码为128 ~ 255间的字符，由于字节中最高位为1，因此用%d格式符输出时，就会得到一个负整数。例如

```
char c=130;
printf("%d\n", c);
```

1	0	0	0	0	0	1	0
---	---	---	---	---	---	---	---

-126

31

字符型数据可与整型数据混合运算

字符型数据以 ASCII 代码的二进制形式存储，与整数的存储形式相类似。因此，在 C 程序中，字符型数据和整型数据之间可以通用，字符型数据与整型数据可混合运算。

32

【例2.1】字符型数据与整型数据通用。

```
#include <stdio.h>
void main()
{ char c1, c2; /* 定义两个字符型变量 */
  c1 = 97; /* 'a' 的 ASCII 码值为 97 */
  c2 = c1+1; /* 字符型与整型数据混合运算 */
  printf("c1 = %c, c2 = %c\n", c1, c2);
  printf("%c's ASCII code = %d\n", c2, c2);
}
```

程序输出：

```
c1 = a, c2 = b
b's ASCII code = 98
```

33

字符串常量

- 用双引号括起来的字符序列，如：

- ◆ "Hello\n"

H	e	l	l	o	\n	\0
---	---	---	---	---	----	----

- ◆ "a" 占2个字节

- ◆ "\$123. 45" 占8个字节

- ◆ ""：称为空字符串常量，占1个字节

- 由若干个字符组成，字节数=字符数+1，即每个字符串尾自动加一个'\0'作为字符串结束标志。

"abc"

a	b	c	\0
---	---	---	----

字符串变量

- ◆ 没有专门的形式，要用数组或指针来表示。

34

2.2 数据输入和输出

C语言本身不提供输入/输出语句。数据的输入和输出功能由C语言的标准I/O库函数提供。

输入：

将要输入数据从输入设备读入到存储器中

输出：

将存储器中的数据写到输出设备中

详见课本p22 和附录E.

35

输入输出函数

- 库函数

- 头文件

- ◆ stdio.h

- ◆ 文件中有标准I/O的变量定义和宏定义。

- 文件包含

- ◆ #include <stdio.h>

- ◆ #include "stdio.h"

- ◆ “包含”的含义就是copy一份。

36

字符输出函数

格式： int putchar(int);

功能： 向终端输出一个字符。

参数： char c = '\n'; (ASCII码)

头文件： #include <stdio.h>

例如： putchar(c);

说明：

1. c 是实参，可以是字符型常量、整型常量（包括控制字符和转义字符）、字符型变量、整型变量等。

2. 使用字符输入输出函数，在程序首必须书写

#include <stdio.h>

37

【例2.2】使用putchar()示例

```
#include <stdio.h>
int main()
{ char ch= 'h'; int i = 'i';
  putchar(67); /* 输出字符 C */
  putchar(ch); /* 输出字符 h */
  putchar(i); /* 输出字符 i */
  putchar('n'); /* 输出字符 n */
  putchar('141'); /* 输出字符 a (141为八进制) */
  putchar('\n'); /* 输出一个回车符 */
  return 0;
}
```

运行该程序将输出 China

38

字符输入函数

格式：

int getchar(void);

功能：

从标准输入设备（一般为键盘）读入一个字符，返回该字符的ASCII码值。

说明：

1. 该函数没有参数，它只能接受一个输入字符。
2. getchar() 得到的字符可以赋给一个字符变量或整型变量，也可以不赋给任何变量。

例如： putchar(getchar())

39

【例2.3】使用getchar()示例

```
#include <stdio.h>
int main()
{ char c ;
  c = getchar(); /* 调用getchar(), 无参数 */
  putchar(c); /* 输出读入的字符 */
  putchar('n'); /* 输出一个回车符 */
  return 0;
}
```

若： 程序运行时从键盘键入字符 z 和回车

则： 程序输出

z (其中变量 c 的值为 'z')

40

格式输出函数printf()

形式： printf("格式控制字符串", 输出项表);

功能： 按照一定的格式向终端输出任意个任意类型的数据。

格式控制字符串 - 用双引号括起来的字符串。

它包含三类字符：

1. **普通字符**：要求按原样输出的字符。

例： int a=3, b=5;

printf("a=%d, b=%d, a+b=%d\n", a, b, a+b);

结果： a=3, b=5, a+b=8

41

2. **转义字符**：要求按转义字符的意义输出。如'\n'表示输出 时回车换行，'\b'表示退格等。

3. **格式转换说明**：由“%”和格式字符组成，如%d、%f、%c、%s等。

输出项表 - 由若干个常量、变量或表达式组成的表列，每个格式对应一个输出项。

42

格式字符

格式字符(%)	说 明
d, i	十进制整数
o	八进制整数(补码)
x, X	十六进制整数(补码)
u	无符号十进制整数
c	字符
s	字符串
f	单、双精度小数，缺省6位小数
e, E	单、双精度指数形式
%	字符%

43

附加格式字符

字符	说 明
l	用于长整型，加于d, o, x, u之前
h	用于短整型
w	代表一个整数，表示数据最小宽度
.p	代表一个整数，表示小数位数或字符串字符个数
-	左对齐
+	输出带符号(包括+)
#	输出带0、0x、小数点

44

几点说明：

1. 域宽说明，w

- ◆ 数据长度>w，则输出实际字符数；
- ◆ 数据长度<w，左补空格(若-w，则在右边补空格)
- ◆ 若w有前导0，左边空位用0填补。

示例：

```
void main()
{
    int a=888;
    printf("a=%6d\n",a);
    printf("a=%06d\n",a);
}
```

程序运行结果为：

```
a= 888
a=000888
```

45

```
#include <stdio.h>
```

```
void main()
```

```
{
```

```
printf("%8d\n",12345);
```

程序运行结果为：

```
12345
```

```
printf("%8d\n",12);
```

```
12
```

```
printf("%-8d\n",12345);
```

```
12345
```

```
printf("%-8d\n",12);
```

```
12
```

```
}
```

46

```
void main()
```

```
{
```

```
printf("%+08d\n",12345);
printf("%+8d\n",12);
printf("%-8d\n",12345);
printf("%-8d\n",12);
```

程序运行结果为：

```
+0012345
```

```
      +12
```

```
12345
```

```
12
```

47

◆ w为*，则域由下一个输出项的整数值指出。

即域是可变的。

例如：printf("%*c", 10, 'A');

则等同与%10c, printf("%10c", 'A');

先输出9个空格，再输出一个A。

48

2. 精度说明: .p

◆**用于浮点数%f, %e, %E:** 表示“小数点后最多显示的位数”，默认值为6

◆**用于字符串%s:** 指定输出的字符数

◆**用于整数:** “必须显示的最小位数”，不足时补前导0。

```
void main()
{
    printf("%.4f\n", 123.1234567);
    printf("%3.8d\n", 1234);
    printf("%10.15s\n", "This is a simple test");
}
```

程序运行结果为
123.1235
00001234
This is a simpl

49

【例2.4】

```
(1) printf("%d, %+6d, %-6d, %ld\n", 1234, 1234, 1234,
1234567L)
```

1234, +1234, 1234 ,1234567

```
(2) printf("%#o, %4o, %6lo\n", 045, 045, -1);
```

045, 45, 37777777777

```
(3) printf("%#x, %4x, %6lX\n", 045, 045, -1);
```

0x25, 25, FFFFFFFF

50

```
(4) printf("%d, %4u, %lu\n", 4294967295u, 4294967295u, -1);
-1, 4294967295, 4294967295

(5) printf("%c, %3c, %2c\n", 045, 'a', 'a');
// 八进制045对应的字符是%
%, a ,a

(6) printf("%f, %8.3f, %-7.2f, %.7f\n",
123.4567f, 123.4567f, 123.4567f, 123.456789f);
123.456703, 123.457, 123.46 ,123.4567890
```

51

格式输入函数

格式:`scanf ("格式控制字符串", 地址表列, ...)`**功能:**

从标准设备读入数据，并按格式存储到对应的数据存储地址中。

其中:**格式控制字符串:**

是用双引号括起来的字符串

地址表列:

变量的地址（变量前加取地址运算符&），或指针。

例如: `scanf ("%d%d%d", &a, &b, &c);`

52

格式字符

格式字符	说 明
d, i	十进制整数
u	无符号十进制整数
o	八进制整数
x	十六进制整数
c	字符
s	字符串，以'0'为结束字符
f、e	单精度小数或指数形式

53

附加格式字符

字符	说 明
l	长整型 (%ld、%lo、%lx)，以及双精度型 (%lf、%le)
h	短整型 (%hd、%ho、%hx)
w	输入数据宽度
*	本输入项读入后不赋给相应的变量

54

scanf()：要点

1. 格式控制字符串之后给出的是变量地址，而不是变量名（除非是指针）。

例如：为整型变量 n 输入数据

写成： scanf("%d", n); 是不正确的

应写成： scanf("%d", &n);

2. 在格式控制字符串中，如果有普通字符或转义序列，则照原样输入。

例如： scanf("%d,%d", &i, &j)

正确输入： 1,2

错误输入： 1 2

55

3. 在用 "%c" 格式输入字符时，空白类字符和转义字符都作为有效字符输入。若要取输入的一串空白类字符之后的第一个非空白类字符，可采用格式 "%s"。

4. 在输入数值数据和字符串时，遇以下情况，就认为该数据结束：

(1) 遇空白类字符：空格、 TAB (制表符)、 换行符。

(2) 遇宽度结束：如 "%4d" 多至 4 个数字符。

(3) 遇非法输入：对输入数值数据，下一个字符不能构成正确的数据格式。

56

【例2.5】

对应下列输入代码，要让变量 i 和 j 值分别为 12 和 234，试指出合理的输入。

(1) scanf("%d,%d",&i,&j);
scanf("%d,%d",i,j); // 不对

(2) scanf("%d%d",&i,&j);

对于(1)，"%d,%d"中间的逗号是普通字符，必须按原样输入。所以，输入是：12,234

对于(2)，两个输入格式之间没有其它字符，输入时，数据以一个或多个空格符分隔，也可以用 Tab 键、 Enter 键分隔。所以，可以输入：12 234

或： 12

234

57

(3) scanf(" %2d%3d",&i,&j); // 指定数据输入的数字符个数，分别是 2 个和 3 个，输入数据可以用空格符分隔，也可以有前 2 个数字符为变量 i 输入，后 3 个数字符为变量 j 输入。例如，输入 12234 也能满足要求，将 12 赋值给变量 i，将 234 赋值给变量 j。

(4) scanf("%d%*d%d",&i,&j);

格式中的第 2 个输入格式有赋值抑制符，所以要输入 3 个整数，其中第二个整数用于输入不赋值的要求。只要 3 个整数有空格符分隔即可，

例如，输入：12 0 234

58

进一步的例子：

```
int i; char c; float x;
scanf("%d%c%f", &i, &c, &x)
```

若输入字符为：123a123x.26

则：变量 i 为 123，变量 c 为字符 a，变量 x 为 123.0。

```
int i, j;
scanf("%3d%*4d%d", &i, &j)
```

若输入字符为：123456 78

则：变量 i 为 123，j 为 78。其中数据 456 因赋值抑制符 * 的作用被跳过。

59

输入输出小结

掌握以下输入输出函数：

- ◆ putchar()
- ◆ getchar()
- ◆ printf()
- ◆ scanf()

掌握基本的输入输出格式字符（以大纲为准）

60

作业与上机

■ 作业

◆ 教材

P39: 3, 7

■ 上机

◆ 教材

P39-40: 5, 8

61

1.7 数制

■ 整型常量（整常数）

◆ 三种形式：

- 十进制整数：由数字0~9和正负号表示，但第一位不能是0。
- 八进制整数：由数字0开头，后跟数字0~7表示。
如0123, 011
- 十六进制整数：由0x开头，后跟0~9, a~f, A~F表示。如0x123, 0xffff

注意，空白字符不可出现在整数数字之间。

整常数在不加特别说明时总是正值。如果需要的是负值，则负号“-”必须放置于常数表达式的前面。

62

示例：

$$0571 = (377)_{10}$$

$$0x179 = (377)_{10}$$

$$0X179 = (377)_{10}$$

下面将介绍数制间的转换。

63

数制转换

常用数制

- 二进制 Binary 0 ~ 1, 2个数码
- 八进制 Octal 0 ~ 7, 3个数码
- 十进制 Decimal 0 ~ 9, 10个数码
- 十六进制 Hexdecimal 0 ~ 9, A ~ F(a ~ f)
16个数码

其中a ~ f, A ~ F分别对应数值10 ~ 15。

64

常用计数制的对应数值

Dec	Bin	Oct	Hex
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

65

数的表示方法

$$\begin{aligned}S &= k_{n-1}k_{n-2}\cdots k_1k_0.k_{-1}k_{-2}\cdots k_{-m} \\&= \sum_{i=-m}^{n-1} k_i \cdot B^i \\&= k_{n-1} \times B^{n-1} + k_{n-2} \times B^{n-2} + \cdots + k_1 \times B^1 + k_0 \times B^0 \\&\quad + k_{-1} \times B^{-1} + \cdots + k_{-m} \times B^{-m}\end{aligned}$$

k_i 第*i*位的值

B 某种进制的基数

66

十进制 - 二进制

- 整数部分：除2取余；
- 小数部分：乘2取整。（精度？）

$$\begin{array}{r}
 \begin{array}{c} 12 \\ \hline 2 & 6 \\ & \hline 2 & 3 \\ & \hline 1 & \end{array} \quad \begin{array}{l} \text{取余: 0 低位} \\ \text{取余: 0} \\ \text{取余: 1} \\ \text{取余: 1 高位} \end{array} \\
 \begin{array}{c} 0.7 \\ \hline \times 2 \\ 1.4 \\ \hline \times 2 \\ 0.8 \\ \hline \times 2 \\ 1.6 \\ \hline \times 2 \\ 1.2 \end{array} \quad \begin{array}{l} \text{取整: 高位} \\ \text{取整: 0:} \\ \text{取整: 1:} \\ \text{取整: 1: 低位} \end{array}
 \end{array}$$

转换结果为：
 $12.7_{(D)} = 1100.1011 \cdots_{(B)}$

67

二进制 - 八进制

- 正向：以小数点为中心，左右每三位一段，不足补0。
- 反向：按位展开。

$$\begin{aligned}
 & 010 \underline{011} \underline{101}. \underline{111} \underline{010}_{(B)} \\
 & = 235.72_{(O)}
 \end{aligned}$$

68

二进制 - 十六进制

- 正向：以小数点为中心，左右每四位一段，不足补0。
- 反向：按位展开。

$$\begin{aligned}
 & 1001 \underline{1101}. \underline{1110} \underline{1000}_{(B)} \\
 & = 9D.E8_{(X)}
 \end{aligned}$$

69

八进制 - 十六进制

- 先转换到二进制；
- 再从二进制出发进行转换。

70

八、十六进制 - 十进制

- 正向：按数的表示方法展开。

$$0123 = 1*8^2 + 2*8^1 + 3*8^0 = 83_{(D)}$$

$$\begin{aligned}
 1A7_{(X)} &= 1*16^2 + A*16^1 + 7*16^0 \\
 &= 256 + 160 + 7 \\
 &= 423_{(D)}
 \end{aligned}$$

注意，空白字符不可出现在数字之间。

71

十进制 - 八进制、十六进制

- 先转换到二进制；
- 再从二进制转换到八、十六进制。

72

数的补码表示

准备

- 以下讨论均假设数据为2字节长度（16位），则可知有符号数范围：

$$-2^{15}(-32768) \sim 2^{15}-1(32767)$$

73

原码

以最高位（1位）表示数值的正负，0表示正数，1表示负数。其他位表示为数字的二进制码。

- 正数：符号位为0，数据位不变；
- 负数：符号位为1，数据位不变。

$$10 \text{ (原码)} = 0\ 0000000\ 00001010$$

$$-10 \text{ (原码)} = 1\ 0000000\ 00001010$$

74

反码

- 正数：符号位为0，数据位不变；
- 负数：符号位为1，数据位取反。

$$10 \text{ (反码)} = 0\ 0000000\ 00001010$$

$$-10 \text{ (反码)} = 1\ 1111111\ 11110101$$

75

补码

- 正数：符号位为0，数据位不变；
- 负数：反码+1。

$$10 \text{ (补码)} = 0\ 0000000\ 00001010$$

$$-10 \text{ (反码)} = 1\ 1111111\ 1111\ 0101$$

$$-10 \text{ (补码)} = 1\ 1111111\ 1111\ 0110$$

76

原码的缺陷（1）

- 0有两个不同的码字：+0和-0。

$$+0 \text{ (原码)} = 0\ 0000000\ 00000000$$

$$-0 \text{ (原码)} = 1\ 0000000\ 00000000$$

77

原码的缺陷（2）

- 最小的负数（-32768）没有码字。

$$-32767 \text{ (原码)} = 1\ 1111111\ 11111111$$

$$-32768 \text{ (原码)} = ?$$

78

原码的缺陷 (3)

- 数据的编码不连续，运算不方便。

+1 (原码) = 0 0000000 00000001
+0 (原码) = 0 0000000 00000000
-0 (原码) = 1 0000000 00000000
-1 (原码) = 1 0000000 00000001

显然，+0和-0不相等，使得从-1到+1不连续。

79

补码解决了原码的3个缺陷

- 0有唯一的表示：+0和-0相同

+0(补码) = 00000000 00000000
-0(反码) = 11111111 11111111
-0(补码) = 00000000 00000000

80

- 可以表示最小的负数(-32768)

-32767(原码) = 11111111 11111111
-32767(反码) = 10000000 00000000
-32767(补码) = 10000000 00000001
-32768(补码) = 10000000 00000000
(= -32767-1)

81

- 数据的编码连续，运算方便

+0(补码) = 00000000 00000000
+1(补码) = 00000000 00000001
+32767(补码) = 01111111 11111111
(最大数)

-0(补码) = 00000000 00000000 (= -1+1)
-1(补码) = 11111111 11111111 (= -0-1)
-32768(补码) = 10000000 00000000
(最小数)

82

十六进制

+32767 (补码) = 7FFF
...
+1 (补码) = 0001
+0 (补码) = 0000
-0 (补码) = 0000 (= +1-1)
-1 (补码) = FFFF (= ± 0-1)
...
-32767 (补码) = 8001
-32768 (补码) = 8000



83