C语言float、double的内存表示

在内存中，小数是以指数形式存在的。float、double 在内存中的形式如下所示：

float 的内存分布	符号位(1Bit)	指数部分(8Bits)	尾数部分(23Bits)
double的内存分布	符号位(1Bit)	指数部分(11Bits)	尾数部分(52Bits)

小数在被存储到内存前，首先转换为下面的形式：

$$a \times 2^n$$

其中 a 为尾数，是二进制形式，且 $1 \le a < 2$；n 为指数，是十进制形式。

例如对于 19.625，整数部分的二进制形式为：

$$19 = 1\times2^4 + 0\times2^3 + 0\times2^2 + 1\times2^1 + 1\times2^0 = 10011$$

小数部分的二进制形式为：

$$0.625 = 1\times2^{-1} + 0\times2^{-2} + 1\times2^{-3} = 101$$

将整数部分和小数部分合并在一起：

$$19.625 = 10011.101$$

再将小数点向左移动4位：

$$19.625 = 10011.101 = 1.0011101\times2^4$$

此时尾数为 1.0011101，指数为 4。

所有的小数被转换成指数形式后，尾数的整数部分都为1，无需在内存中提现出来，所以干脆将其截去，只把小数点后面的二进制放入内存中的尾数部分（23Bits）。对于 1.0011101，尾数部分就是 0011101。

C语言把整数作为定点数，而把小数作为浮点数。定点数必须转换为补码再写入内存，浮点数没有这个过程，直接写入原码。小数被转换成指数形式后，指数有正有负，在内存中不但要能表现其值，还要能表现其正负。而指数是以原码形式存储的，没有符号位，所以要设计一个巧妙的办法来区分正负。

对于 float，指数占用8Bits，能表示从 0~255 的值，取其中间值 127，指数在写入内存前先加上127，读取时再减去127，正数负数就显而易见了。19.625 转换后的指数为 4，4+127 = 131 = 1000 0011。

综上所述，float 类型的 19.625 在内存中的值为：0 - 10000011 - 001 1101 0000 0000 0000 0000。

下面我们使用代码来验证一下：

```c
#include <stdio.h>
#include <stdlib.h>
int main()
{
    typedef struct _FP_SIGLE{
        unsigned int nMantissa : 23;  //尾数部分
        unsigned int nExponent :  8;  //指数部分
        unsigned int nSign     :  1;  //符号位
    } FP_SINGLE;

    float a = 19.625;
    FP_SINGLE* p = (FP_SINGLE*)&a;
    printf("%d, %#X, %#X\n", p->nSign, p->nExponent-127, p->nMantissa);

    system("pause");
    return 0;
}
```

运行结果：
0, 0X4, 0X1D0000

> C语言不能直接输出二进制形式，一般输出十六进制即可，十六进制能够很方便地转换成二进制。

精度

精度指测量值与真实值的接近程度，在C语言中表现为输出值和真实值的接近程度。

float 和 double 的精度是由尾数的位数决定。内存中的尾数只保存了小数点后面的部分，其整数部分始终是一个隐含着的“1“，它是不变的，不会对精度造成影响。

float：2^23 = 8388608，一共七位，这意味着最多能有7位有效数字，但绝对能保证的为6位，也即 float 的精度为 6~7 位有效数字。

double：2^52 = 4503599627370496，一共16位，同理，double 的精度为 15~16 位。

取值范围和近似值

float 和 double 在内存中的指数和尾数的位数都是有限的，小数过大或过小都会发生溢出。float 的取值范围为 -2^128 ~ +2^128，也即 -3.40E+38 ~ +3.40E+38；double 的取值范围为 -2^1024 ~ +2^1024，也即 -1.79E+308 ~ +1.79E+308。

当小数的尾数部分过长时，多出的位数就会被直接截去，这时保存的就不是小数的真实值，而是一个近似值。在《C语言中的浮点数（float,double）》一节的示例中，我们看到 128.101 的输出结果就是一个近似值。

128.101 转换成二进制为 10000000.00011001110110110011000101101，向左移动7位后为 1.00000000001100111011011001000101101，由此可见，尾数部分为 000 0000 0001 1001 1101 1011 001000101101，将多出的二进制截去后为 000 0000 0001 1001 1101 1011。下面的代码有力地证明了这一点：

```
01.  #include <stdio.h>
02.  #include <stdlib.h>
03.  int main()
04.  {
05.      typedef struct _FP_SIGLE{
06.          unsigned int nMantissa : 23;  //尾数部分
07.          unsigned int nExponent :  8;  //指数部分
08.          unsigned int nSign     :  1;  //符号位
09.      } FP_SINGLE;
10.
11.      float a = 128.101f;
12.      FP_SINGLE* p = (FP_SINGLE*)&a;
13.      printf("%f\n", a);
14.      printf("%d, %#X, %#X\n", p->nSign, p->nExponent-127, p->nMantissa);
15.
16.      system("pause");
17.      return 0;
18.  }
```

运行结果：
128.100998
0, 0X7, 0X19DB

最后对 float 和 double 做一下总结：

类型说明符	比特数（字节数）	有效数字	数的范围
float	32(4)	6~7	-3.40E+38 ~ +3.40E+38
double	64(8)	15~16	-1.79E+308 ~ +1.79E+308