一、浮点数有没有移位和拓展功能？为什么？

浮点数在计算机中通常以 IEEE 754 标准表示，该标准规定了浮点数的表示方法、运算规则等。

在 IEEE 754 标准中，浮点数的表示形式包括符号位、指数位和尾数位。移位和拓展通常是针对整数类型的操作，而不是针对浮点数类型的操作。

移位操作是针对整数类型的，可以对整数进行左移或右移操作，通过移动位来改变整数的值。而浮点数的移位操作并没有被标准化或普遍实现，因为移动浮点数的位可能会破坏其表示形式和数值含义。

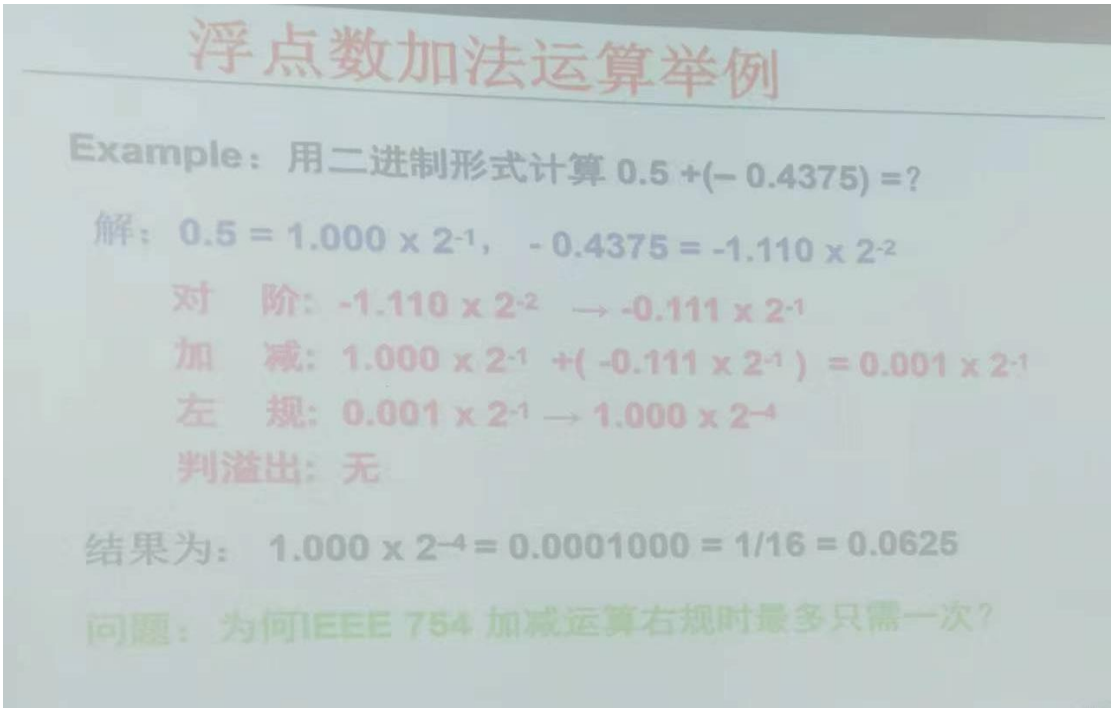
二、为什么整数除 0 会发生异常而浮点数除 0 不会？

它们的数值表示方式和数学运算规则不同，整数除以零违反了整数的除法规则，而浮点数除以零则符合 IEEE 754 标准的规定。

整数除以零会导致异常是因为在整数除法中，除数不能为零。当在程序中进行整数除法操作时，如果除数为零，会导致除法运算无法完成，这种情况通常会被编程语言的解释器或编译器捕获并报告为异常，例如“除零异常”或“被零除错误”。

浮点数在 IEEE 754 标准中定义了特殊的浮点数值，用来表示正无穷大、负无穷大和 NaN（Not a Number）。当浮点数除以零时，根据 IEEE 754 标准，会得到正无穷大（如果除数为正数）或负无穷大（如果除数为负数），而不会引发异常。

三、结合例子说明为何 IEEE 754 加减运算右规时最多只需一次？



0.5 可以表示为 1.000x2^-1，而 -0.4375 可以表示为 -1.110x2^2。

接下来，我们要对阶，将它们的指数部分对齐。-1.110x2^2 可以写为 -0.111x2^-1。

然后，我们进行加法运算：1.000x2^-1 + (-0.111x2^-1) = 0.001x2^-1。

再进行左规化，得到 0.001x2^1 - 1.000x2^-4。

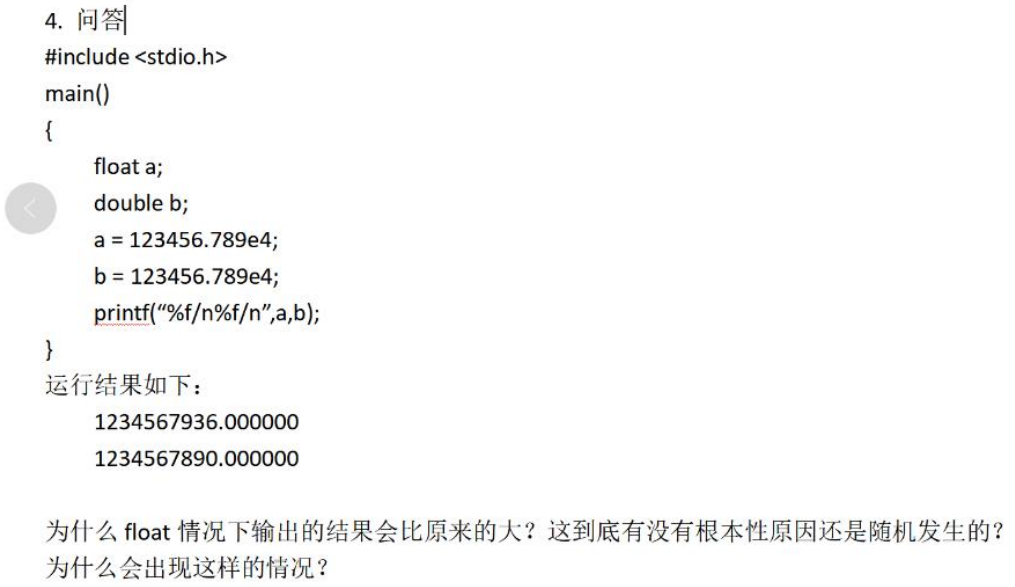
判断是否溢出，这里没有溢出。

最终结果为：1.000x2^-4 = 0.0001000 = 1/16 = 0.0625。

浮点数的尾数部分始终位于 [1, 2) 区间之间，加减运算时，不会出现尾数部分进位或借位超过一个单位

IEEE 754 加减运算右规时最多只需一次，因为在对阶和加减运算后，结果已经左规化了，不需要额外的规范化步骤。

四、为什么同一个实数赋值给 float 型变量和 double 型变量，输出结果会有所不同呢？


4. 问答
```
#include <stdio.h>
main()
{
    float a;
    double b;
    a = 123456.789e4;
    b = 123456.789e4;
    printf("%f/n%f/n",a,b);
}
```
运行结果如下：

1234567936.000000

1234567890.000000

为什么 float 情况下输出的结果会比原来的大？这到底有没有根本性原因还是随机发生的？为什么会出现这样的情况？

float 类型的输出结果比原始值大了,这是由于浮点数在计算机内部的存储方式导致的。

浮点数的存储是基于 IEEE 754 标准的，其中 float 类型使用 32 位来表示，而 double 类型使用 64 位来表示。由于 float 类型的存储空间较小，它只能表示有限的精度，不能完全准确地表示原始值。在进行浮点数的计算和存储过程中，会发生舍入误差，导致结果与原始值存在一定的差异。

在给定的代码中，原始值 123456.789e4 的精度超过了 float 类型的表示范围，因此在赋值给 float 类型变量 a 时，发生了舍入误差，导致结果比原始值稍大一些。

浮点数的舍入误差是由计算机的二进制表示和浮点数的存储方式决定的，这不是随机发生的，而是由于浮点数的有限精度所带来的必然结果,当需要更高精度的计算时，应选择使用 double 类型。