

为 1.0。另一方面，由于可能发生溢出，或者由于舍入而失去精度，它不具有可结合性。例如，单精度浮点情况下，表达式 $(1e20 * 1e20) * 1e-20$ 求值为 $+\infty$ ，而 $1e20 * (1e20 * 1e-20)$ 将得出 $1e20$ 。另外，浮点乘法在加法上不具备分配性。例如，单精度浮点情况下，表达式 $1e20 * (1e20 - 1e20)$ 求值为 0.0，而 $1e20 * 1e20 - 1e20 * 1e20$ 会得出 NaN。

另一方面，对于任何 a 、 b 和 c ，并且 a 、 b 和 c 都不等于 NaN，浮点乘法满足下列单调性：

$$a \geq b \text{ 且 } c \geq 0 \Rightarrow a *^f c \geq b *^f c$$

$$a \geq b \text{ 且 } c \leq 0 \Rightarrow a *^f c \leq b *^f c$$


此外，我们还可以保证，只要 $a \neq \text{NaN}$ ，就有 $a *^f a \geq 0$ 。像我们先前所看到的，无符号或补码的乘法没有这些单调性属性。

对于科学计算程序员和编译器编写者来说，缺乏结合性和分配性是很严重的问题。即使为了在三维空间中确定两条线是否交叉而写代码这样看上去很简单的任务，也可能成为一个很大的挑战。

2.4.6 C 语言中的浮点数

所有的 C 语言版本提供了两种不同的浮点数据类型：float 和 double。在支持 IEEE 浮点格式的机器上，这些数据类型就对应于单精度和双精度浮点。另外，这类机器使用向偶数舍入的舍入方式。不幸的是，因为 C 语言标准不要求机器使用 IEEE 浮点，所以没有标准的方法来改变舍入方式或者得到诸如 -0 、 $+\infty$ 、 $-\infty$ 或者 NaN 之类的特殊值。大多数系统提供 include(‘.h’) 文件和读取这些特征的过程库，但是细节随系统不同而不同。例如，当程序文件中出现下列句子时，GNU 编译器 GCC 会定义程序常数 INFINITY(表示 $+\infty$) 和 NAN(表示 NaN)：

```
#define _GNU_SOURCE 1
#include <math.h>
```

 **练习题 2.53** 完成下列宏定义，生成双精度值 $+\infty$ 、 $-\infty$ 和 0：

```
#define POS_INFINITY
#define NEG_INFINITY
#define NEG_ZERO
```

不能使用任何 include 文件(例如 math.h)，但你能利用这样一个事实：双精度能够表示的最大的有限数，大约是 1.8×10^{308} 。

当在 int、float 和 double 格式之间进行强制类型转换时，程序改变数值和位模式的原则如下(假设 int 是 32 位的)：

- 从 int 转换成 float，数字不会溢出，但是可能被舍入。
- 从 int 或 float 转换成 double，因为 double 有更大的范围(也就是可表示值的范围)，也有更高的精度(也就是有效位数)，所以能够保留精确的数值。
- 从 double 转换成 float，因为范围要小一些，所以值可能溢出成 $+\infty$ 或 $-\infty$ 。另外，由于精确度较小，它还可能被舍入。
- 从 float 或者 double 转换成 int，值将会向零舍入。例如，1.999 将被转换成 1，而 -1.999 将被转换成 -1。进一步来说，值可能会溢出。C 语言标准没有对这种情况指定固定的结果。与 Intel 兼容的微处理器指定位模式 $[10 \dots 00]$ (字长为 w 时的 $TMin_w$) 为整数不确定(integer indefinite)值。一个从浮点数到整数的转换，如果不能为该浮点数找到一个合理的整数近似值，就会产生这样一个值。因此，表达式 $(\text{int}) + 1e10$ 会得到 -21483648，即从一个正值变成了一个负值。