


旁注 Ariane 5——浮点溢出的高昂代价

将大的浮点数转换成整数是一种常见的程序错误来源。1996年6月4日，Ariane 5火箭初次航行，一个错误便产生了灾难性的后果。发射后仅仅37秒钟，火箭偏离了它的飞行路径，解体并且爆炸。火箭上载有价值5亿美元的通信卫星。

后来的调查[73, 33]显示，控制惯性导航系统的计算机向控制引擎喷嘴的计算机发送了一个无效数据。它没有发送飞行控制信息，而是送出了一个诊断位模式，表明在将一个64位浮点数转换成16位有符号整数时，产生了溢出。

溢出的值测量的是火箭的水平速率，这比早先的Ariane 4火箭所能达到的速度高出了5倍。在设计Ariane 4火箭软件时，他们小心地分析了这些数字值，并且确定水平速率决不会超出一个16位数的表示范围。不幸的是，他们在Ariane 5火箭的系统中简单地重用了这一部分，而没有检查它所基于的假设。

 **练习2.54** 假定变量 x 、 f 和 d 的类型分别是`int`、`float`和`double`。除了 f 和 d 都不能等于 $+\infty$ 、 $-\infty$ 或者`NaN`，它们的值是任意的。对于下面每个C表达式，证明它总是为真(也就是求值为1)，或者给出一个使表达式不为真的值(也就是求值为0)。

- A. `x == (int)(double) x`
- B. `x == (int)(float) x`
- C. `d == (double)(float) d`
- D. `f == (float)(double) f`
- E. `f == -(f)`
- F. `1.0/2 == 1/2.0`
- G. `d*d >= 0.0`
- H. `(f+d)-f == d`

2.5. 小结

计算机将信息编码为位(比特)，通常组织成字节序列。有不同的编码方式用来表示整数、实数和字符串。不同的计算机模型在编码数字和多字节数据中的字节顺序时使用不同的约定。

C语言的设计可以包容多种不同字长和数字编码的实现。64位字长的机器逐渐普及，并正在取代统治市场长达30多年的32位机器。由于64位机器也可以运行32位机器编译的程序，我们的重点就放在区分32位和64位程序，而不是机器本身。64位程序的优势是可以突破32位程序具有的4GB地址限制。

大多数机器对整数使用补码编码，而对浮点数使用IEEE标准754编码。在位级上理解这些编码，并且理解算术运算的数学特性，对于想使编写的程序能在全数数值范围上正确运算的程序员来说，是很重要的。

在相同长度的无符号和有符号整数之间进行强制类型转换时，大多数C语言实现遵循的原则是底层的位模式不变。在补码机器上，对于一个 w 位的值，这种行为是由函数 $T2U_w$ 和 $U2T_w$ 来描述的。C语言隐式的强制类型转换会出现许多程序员无法预计的结果，常常导致程序错误。

由于编码的长度有限，与传统整数和实数运算相比，计算机运算具有非常不同的属性。当超出表示范围时，有限长度能够引起数值溢出。当浮点数非常接近于0.0，从而转换成零时，也会下溢。

和大多数其他程序语言一样，C语言实现的有限整数运算和真实的整数运算相比，有一些特殊的属性。例如，由于溢出，表达式 $x*x$ 能够得出负数。但是，无符号数和补码的运算都满足整数运算的许多其他属性，包括结合律、交换律和分配律。这就允许编译器做很多的优化。例如，用 $(x < 3) - x$ 取代表达式 $7*x$ 时，我们就利用了结合律、交换律和分配律的属性，还利用了移位和乘以2的幂之间的关系。

我们已经看到了几种使用位级运算和算术运算组合的聪明方法。例如，使用补码运算， $\sim x + 1$ 等价于 $-x$ 。另外一个例子，假设我们想要一个形如 $[0, \dots, 0, 1, \dots, 1]$ 的位模式，由 $w-k$ 个0后面紧跟着 k