点数虽然可以编码一个较大的数值范围，但是这种表示只是近似的。

通过研究数字的实际表示，我们能够了解可以表示的值的范围和不同算术运算的属性。为了使编写的程序能在全部数值范围内正确工作，而且具有可以跨越不同机器、操作系统和编译器组合的可移植性，了解这种属性是非常重要的。后面我们会讲到，大量计算机的安全漏洞都是由于计算机算术运算的微妙细节引发的。在早期，当人们碰巧触发了程序漏洞，只会给人们带来一些不便，但是现在，有众多的黑客企图利用他们能找到的任何漏洞，不经过授权就进入他人的系统。这就要求程序员有更多的责任和义务，去了解他们的程序如何工作，以及如何被迫产生不良的行为。

计算机用几种不同的二进制表示形式来编码数值。随着第 3 章进入机器级编程，你需要熟悉这些表示方式。在本章中，我们描述这些编码，并且教你如何推出数字的表示。

通过直接操作数字的位级表示，我们得到了几种进行算术运算的方式。理解这些技术对于理解编译器产生的机器级代码是很重要的，编译器会试图优化算术表达式求值的性能。

我们对这部分内容的处理是基于一组核心的数学原理的。从编码的基本定义开始，然后得出一些属性，例如可表示的数字的范围、它们的位级表示以及算术运算的属性。我们相信从这样一个抽象的观点来分析这些内容，对你来说是很重要的，因为程序员需要对计算机运算与更为人熟悉的整数和实数运算之间的关系有清晰的理解。

旁注　怎样阅读本章

本章我们研究在计算机上如何表示数字和其他形式数据的基本属性，以及计算机对这些数据执行操作的属性。这就要求我们深入研究数学语言，编写公式和方程式，以及展示重要属性的推导。

为了帮助你阅读，这部分内容安排如下：首先给出以数学形式表示的属性，作为原理。然后，用例子和非形式化的讨论来解释这个原理。我们建议你反复阅读原理描述和它的示例与讨论，直到你对该属性的说明内容及其重要性有了牢固的直觉。对于更加复杂的属性，还会提供推导，其结构看上去将会像一个数学证明。虽然最终你应该尝试理解这些推导，但在第一次阅读时你可以跳过它们。

我们也鼓励你在阅读正文的过程中完成练习题，这会促使你主动学习，帮助你理论联系实际。有了这些例题和练习题作为背景知识，再返回推导，你将发现理解起来会容易许多。同时，请放心，掌握好高中代数知识的人都具备理解这些内容所需要的数学技能。

C++ 编程语言建立在 C 语言基础之上，它们使用完全相同的数字表示和运算。本章中关于 C 的所有内容对 C++ 都有效。另一方面，Java 语言创造了一套新的数字表示和运算标准。C 标准的设计允许多种实现方式，而 Java 标准在数据的格式和编码上是非常精确具体的。本章中多处着重介绍了 Java 支持的表示和运算。

旁注　C 编程语言的演变

前面提到过，C 编程语言是贝尔实验室的 Dennis Ritchie 最早开发出来的，目的是和 Unix 操作系统一起使用(Unix 也是贝尔实验室开发的)。在那个时候，大多数系统程序，例如操作系统，为了访问不同数据类型的低级表示，都必须大量地使用汇编代码。比如说，像 malloc 库函数提供的内存分配功能，用当时的其他高级语言是无法编写的。

Brian Kernighan 和 Dennis Ritchie 的著作的第 1 版[60]记录了最初贝尔实验室的 C 语言版本。随着时间的推移，经过多个标准化组织的努力，C 语言也在不断地演变。1989