最　新

C 语言程序设计教程

（第 2 版）

刘正林　谢永锋
祝　宏　郭　胜　编著

华中科技大学出版社
中国・武汉

内 容 简 介

本书以 ISO/ANSI C++标准为准则,以国内外广为流行的美国 Microsoft 公司开发的 Visual C++ 为语言蓝本，系统地讲授 C++模块化语言的基础部分，即 C 语言作为 ISO/ANSI C++内核的基本语法和程序设计方法，扬弃了 C 语言老版本的一些非标准内容。

根据教育部高等学校计算机基础课程教学指导委员会制订的白皮书精神，本书按照“循序渐进、突出重点、深入浅出、融会贯通”的教学原则编写，并以全国计算机等级考试中的 C 和 C++两个科目作为参照体系。每章都有小结，归纳出必须掌握的重点内容，并附有大量的习题，以加深读者对重点内容的理解。

本书可作为大专院校理工科各专业，特别是独立学院学生学习“C 语言程序设计”的教科书，也可供广大电脑爱好者作为自学 C 语言和 C++的教材和参考书。

前　言

崭新的 21 世纪，以现代电子信息产业为龙头的全球经济一体化浪潮正席卷世界，这是当今人类所面临的巨大挑战，人们将认真面对挑战的内涵和挑战所带来的机遇。而以 IT(Information Technology)技术为基础的信息产业正深入到人类社会生活的方方面面，无论是生产制造、国防和科技等领域，还是第三产业，计算机软件现已成为担当重任的核心力量，互联网和软件已成为推动新经济发展的重要基础。因此，计算机软件技术将是各类专业的大专生、本科生和研究生所必备的基础知识。 面向对象的程序设计是软件技术的一场革命，必将成为 21 世纪普遍采用的软件开发方法。C++是面向对象程序设计的第一个大众化设计语言版本，是当前学习面向对象程序设计方法的首选语言。C++是著名 C 语言的面向对象扩展。C 语言最初设计时，是作为一种面向系统软件（操作系统 Operating System 和语言处理系统）的开发语言，即是用来代替汇编语言的，但是由于它强大的生命力，在事务处理、科学计算、工业控制和数据库技术等各个方面都得到了广泛应用。即便进入到以计算机网络为核心的信息时代，C 语言仍然是作为通用的汇编语言使用，用以开发软（件）、硬（件）结合的程序，如实时监控程序、控制程序和设备驱动程序等。而 C++是 C 语言的超集，它保留了 C 语言的所有组成部分而与其完全兼容，既可以做传统的结构化程序设计，又能进行面向对象程序设计，也是当今世界比较流行的程序设计语言。

本书以 ISO/ANSI(American National Standards Institute) C++标准（ISO 14882）为准则，以美国 Microsoft 公司开发的 Visual C++ 6.0 为语言蓝本，重点介绍 C++程序设计语言的内核即模块结构化基础部分，并讲解 ANSI C 老标准和 ISO/ANSI C++标准的不同点，使学生能自觉扬弃 C 语言老版本的一些非标准内容，将 C 语言老版本程序快捷地修改成符合 ISO/ANSI C++标准的实用程序，其目的是便于学生尽快掌握像 Visual C++和 C++ Builder 这些优秀可视化的应用程序开发工具，快速冲向计算机应用领域的前沿，成为各类专业从事计算机应用、具有相当的应用软件开发能力的技术人才，为进一步学习面向对象程序设计技术打下牢固的基础。为了使学生便于学习，我们总结了多年教学和科研的实践经验，根据“ 循序渐进、突出重点、深入浅出、融会贯通 ”的教学原则，对内容进行了精选，强化如函数及其参数、指针变量及应用和结构体等重要概念，并编写成自成体系的 C++教科书形式。

本书以“ 实用、创新、深入浅出、通俗易懂 ”为特色，讲述思路清晰，层次分明，逻辑性强，可作为目前各大专院校已开设的“ C 语言程序设计 ”课程的教材，取名为《最新 C 语言程序设计教程》，以取代那些内容老化的教材。为便于理解，本书不生硬翻译国外的语言手册，力戒使用晦涩难懂的语言，对于日新月异的计算机领域的许多新专业术语则采用通俗易懂的大众化语言讲述，对核心概念做到图文并茂并举实例加以说明。

作者发扬“ 蜜蜂采蜜 ”的精神，一方面总结多年教学和科研的成果，另一方面广泛收集国内外最新发展动态和应用实例，并经过消化和吸收，在理解的基础上，按照“ 循序渐

进、重点突出”的教学原则，以“面向对象方法及其理论”为基础，酿造成“思路新颖、重点突出、逻辑清晰、易学易用”的知识奉献给读者，这是作者的创新之举。

应用软件的开发是从事计算机应用的各类专业技术人员的大舞台，它是由从事计算机应用的各类专业人员来完成的。这部分人员多数毕业于非计算机专业，不仅人数众多，而且分布在广阔的各类应用领域内，他们既具有扎实的本专业基础理论知识和丰富的实践经验，又熟悉计算机应用技术方面的知识，善于用计算机作为强有力的辅助工具去完成本应用领域内的各种任务，在计算机应用领域内发挥着其他人才无法替代的重要作用，是应用领域不可缺少的基本力量。本教材以此为编写宗旨，因此，适用于大专院校理、工科各类专业学生作为“C 语言程序设计”课程的教科书，也适合于广大电脑爱好者选作自学 C 语言的教材和参考书。在内容安排上也有深有浅，适用于各个层次的读者，既适合于以前从未接触过 C 语言的初学者，也适合于具有一定编程基础的读者作为学习 C 语言，提高编程能力的教材和参考书。

进入 21 世纪，我国高等教育得到了前所未有的蓬勃发展，特别是民办的独立学院如雨后春笋般涌现，华中科技大学文华学院就是其中一员，这一发展势态使得我国高等教育从精英型跨入到大众化型时代。独立学院的办学宗旨应是培养高水平应用型人才，培养出来的学生是从事工程项目的实用型人才。按“高等教育法”和“民办教育促进法”的规定，独立学院也应保证教学质量，所培养的学生仍应达到国家规定的本科生标准，不应该以任何原因作为借口降低国家规定的标准。因此，本科生计算机文化基础课程仍然应与时俱进地跟踪世界先进应用技术，把先进的计算机应用技术传授给学生。同时，考虑到独立学院学生的特点，不能全部照搬公办母体学校的教学体制，其中包含教材和教学方法这两个核心部分。此外，教育部举办的全国计算机等级考试是满足人才市场服务的需求，考核应试人员对计算机和软件实际掌握能力的权威考试，其证书对独立学院学生在毕业后求职将是重要凭证之一。本书第 2 版兼顾上述因素，保留 C++程序设计语言的内核即模块结构化基础部分的完整框架，并以全国计算机等级考试中的 C 和 C++两个科目作为参照体系，针对独立学院学生的特点，在本书第 1 版的基础上进行修改，更详细地、深入浅出地讲述核心内容，并增加各种图示加以说明，便于学生理解。因此，它是特别适于独立学院本、专科生作为学习“C 语言程序设计”课程的好教材。我们也诚恳地恭候广大师生，特别是从事计算机文化基础教育的教师提出宝贵意见，共同探讨！

华中科技大学文华学院
刘正林　谢永锋　祝　宏　郭　胜
2006 年 6 月

目　　录

第1章　概　论

1.1　C语言的入门知识

C语言和由它发展而来的C++是当今计算机界最流行的程序设计语言，也是重要的编程工具。C语言最初设计时，是作为一种面向系统软件（操作系统和语言处理系统）的开发语言，是用来代替汇编语言的，但是由于它强大的生命力，后来在事务处理、科学计算、工业控制和数据库技术等各个方面都得到了广泛应用。即便进入到以计算机网络为核心的信息时代，C语言仍然是作为通用的汇编语言使用，用以开发软、硬件结合的程序，如实时监控程序、系统控制程序和设备驱动程序等。随着计算机软件的飞速发展，为了尽快与国际接轨，作为理工科各类专业本科生的计算机基础课程“C语言程序设计”应该以ISO/ANSI（America National Standard Institute，美国国家标准局）C++标准（ISO 14882，通常称为“ISO/ANSI C++标准”）的基础内核为准则来讲授，重点介绍C++程序设计语言的主要语法和模块结构化基础知识，引导学生自觉扬弃C语言老版本的一些非标准内容，将C语言老版本的程序快捷地修改成符合ISO/ANSI C++标准的实用程序。为此，我们首先介绍一些计算机的基础知识，为讲解C和C++做好必要的准备。

1.1.1　计算机中的数据和编码系统

人们最习惯和熟悉的计数和运算方式是十进制，即逢十进一。但在计算机内，数是以电子器件的物理状态来表示的，而这些器件只具有两种不同且又能相互转换的稳定状态，例如，晶体管的断开（OFF）和接通（ON），因而采用二进制数（1和0）比较方便。

1. 数的多项式表示

数是表示现实世界中事物的量的基本数学概念，任何数都能用以R为基数的多项式表示为：

$$N_R = K_n \times R^n + K_{n-1} \times R^{n-1} + \cdots + K_i \times R^i + \cdots + K_1 \times R + K_0 \times R^0 + K_{-1} \times R^{-1} + \cdots + K_{-i} \times R^{-i} + \cdots + K_{-m} \times R^{-m}$$

其中，R称为基数，表示R进制。如R = 2为二进制，R = 8为八进制，R = 10为十进制，R = 16为十六进制等。$K_{\pm i}$为多项式的系数，它的取值范围为0 ~ （R - 1）。例如：二进制基数R为2，多项式的系数$K_{\pm i}$取值范围为0和1；八进制基数R为8，$K_{\pm i}$取值范围为0 ~ 7；十进制基数R为10，$K_{\pm i}$取值范围为0 ~ 9等。而n和m为幂指数，均为正整数。

为了简化问题，假定N_R是一个整数，则上式变为：

$$N_R = K_n \times R^n + K_{n-1} \times R^{n-1} + \cdots + K_i \times R^i + \cdots + K_1 \times R + K_0 \times R^0 \quad (1\text{-}1)$$

为适应不同的应用场合，计算机可分别采用二进制、八进制、十进制和十六进制等。为方便人们的习惯，计算机输入数据或者输出数据均采用十进制。然而计算机内存放数据

都采用二进制，但由于表示同一个数二进制比十进制的位数要长得多，人们在研究计算机原理时很不方便，因而多采用十六进制，十六进制与二进制之间的转换非常简单，但位数却缩短了很多。式（1-1）是表示正整数的一个通用表达式。

为了区别各种进制，对于二进制（Binary，R = 2），我们把多项式系数 K_i 改为用 Binary 的第 1 个字母 B_i 表示，则式（1-1）变为：

$$N_R = B_n \times 2^n + B_{n-1} \times 2^{n-1} + \cdots + B_i \times 2^i + \cdots + B_1 \times 2 + B_0 \times 2^0 \qquad (1\text{-}2)$$

对于十进制（Decimal，R = 10），将多项式系数 K_i 改为用 Decimal 的第 1 个字母 D_i 表示，则式（1-1）变为：

$$N_R = D_n \times 10^n + D_{n-1} \times 10^{n-1} + \cdots + D_i \times 10^i + \cdots + D_1 \times 10 + D_0 \times 10^0 \qquad (1\text{-}3)$$

数是对客观存在的数量的描述，而各种进制是人们计数的一种方法，因此，同一个数可用不同的进制表示，其数量值理所当然是相等的。

2. 二进制数

(1) 二进制数的特点。十进制数的每个位有 0~9 十个不同的数字符号（也称为“数码”），且“逢十进一”。而二进制数具有如下两个特点：一是每个位具有两个不同的数字符号，即 1 和 0；二是逢二进一。如图 1.1 所示的二进制数 B 为：

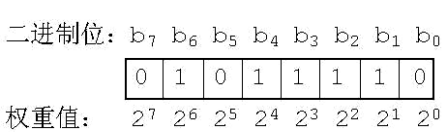


图 1.1 整数 94 的二进制码

$$01011110_B$$

下标 B 为 Binary 的第 1 个字母，表示二进制数，由式（1-2）则有（并且为了与微型计算机原理保持一致，多项式系数用小写字母 b_i 表示）：

$$\begin{aligned} B &= b_7\times2^7 + b_6\times2^6 + b_5\times2^5 + b_4\times2^4 + b_3\times2^3 + b_2\times2^2 + b_1\times2^1 + b_0\times2^0 \\ &= 0\times2^7 + 1\times2^6 + 0\times2^5 + 1\times2^4 + 1\times2^3 + 1\times2^2 + 1\times2^1 + 0\times2^0 \\ &= 64 + 16 + 8 + 4 + 2 = 94 \end{aligned}$$

通常，二进制数也称为二进制码，把每个二进制位叫做一个 bit，并用 b_x 表示，其下标 x（=0，1，2，···，7）是从右至左的位序号，即 b_0、b_1、b_2、b_3、b_4、b_5、b_6、b_7，它们都只能取两个数字符号 1 和 0。但同一个数字符号在不同的二进制数位上具有不同的数值，称为该二进制位的权重值。如 b_0 的权重值为 1，b_1 的权重值为 2，b_2 的权重值为 4，b_3 的权重值为 8，b_4 的权重值为 16 ····· b_7 的权重值为 128 等。显然，各位数的权重值是基数 2 的正次幂，幂指数就是位序号。这就像用 1 克、2 克、4 克和 8 克的砝码在天平上可以秤质量为 1 克 ~ 15 克的物体，b_0 位相当于 1 克砝码，b_1 位相当于 2 克砝码，b_2 位相当于 4 克砝码 ····· 那么，7 克的物体需要加上 1 克、2 克和 4 克的砝码，与此对应就是 b_0、b_1 和 b_2 为 1，其他二进制位为 0；而 12 克的物体要加上 4 克和 8 克的砝码，与此对应 b_2 和 b_3 为 1，其他二进制位为 0。对于十进制各位数的权重值就是 10（基数）的正次幂，个位的权重值为 10^0 等于 1，十位的权重值为 10^1 等于 10，百位的权重值为 10^2 等于 100 等等，如此类推。

(2) 二翻十运算。所谓二翻十运算就是把二进制数转换成十进制数。对于一个数 dec，由式（1-2）可写成：

$$dec = b_n \times 2^n + b_{n-1} \times 2^{n-1} + \cdots + b_1 \times 2 + b_0 \qquad (1\text{-}4)$$

式中 b_n，b_{n-1}，···，b_1，b_0 均为二进制数。

若二进制数的位数为 m(如 8 位)，m 个 bits 可表示 2^m 个二进制数，其表示的数值范围是 0 ～ $(2^m - 1)$，如 0 ～ $(2^8 - 1) = 255$。若表示最大数值 255 的二进制码，其 8 个 bits 都是 1，即：

$$\overbrace{1111,1111}^{8\text{个bits}} = 2^8 - 1$$

一般来说，m 位二进制数所表示的最大数值的二进制码，其 m 个 bits 都是 1，即：

$$\overbrace{1111,\cdots,1111}^{m\text{个bits}} = 2^m - 1$$

$$\begin{array}{rl} & 1111,\cdots,1111 \\ + & \qquad\qquad 1 \\ \hline 2^m & 0000,\cdots,0000 \end{array}$$

溢出

位数越多表示数的范围越大。若最大数值再加 1 则 m 位都变成 0，并向 m+1 位进位，这种现象称为“溢出”。通俗地说，就是 m 位二进制数已装满了，则溢出一个 2^m 数值后，再从零开始计数，该数值就是 m 位二进制计数的容量，即可表示二进制数的个数为 2^m 个。

(3) 十翻二运算。十翻二运算就是把十进制数转换成二进制数。其方法是：将式 (1-4) 用 2 除的整数商（丢掉小数部分保留整数部分的商），即第 1 次用 2 除的商为：

$$dec/2 = b_n \times 2^{n-1} + b_{n-1} \times 2^{n-2} + \cdots + b_1$$

所得的余数为：　　$dec \% 2 = b_0$

式中%为取余数运算符，即两整数相除取其余数（下同）。第 2 次用 2 除的商为：

$$dec / 2 = b_n \times 2^{n-2} + b_{n-1} \times 2^{n-3} + \cdots + b_2$$

所得的余数为：　　$dec \% 2 = b_1$

……

直到求得 b_n。例如：

① $94 / 2 = 47$，$94 \% 2 = b_0 = 0$；

② $47 / 2 = 23$，$47 \% 2 = b_1 = 1$；

③ $23 / 2 = 11$，$23 \% 2 = b_2 = 1$；

④ $11 / 2 = 5$，$11 \% 2 = b_3 = 1$；

⑤ $5 / 2 = 2$，$5 \% 2 = b_4 = 1$；

⑥ $2 / 2 = 1$，$2 \% 2 = b_5 = 0$；

⑦ $1 / 2 = 0$，$1 \% 2 = b_6 = 1$；

⑧ $0 / 2 = 0$，$0 \% 2 = b_7 = 0$。

所以得 94 = 0101,1110，因此，十翻二运算的法则是“用 2 除取余数，直到商为零，逆序取余数为二进制的每个位，即第 1 次所取得的余数为最低位，最后的余数为最高位”。在此提醒读者，由于微型计算机结构的特点，各种数据类型的二进制位数通常是 2^3、2^4、2^5、2^6 等，即 8 个二进制位，或是 16、32 或 64 等个二进制位，为此，通常只研究 8、16、32 和 64 等个二进制位的数据类型。虽然有些例题可能用 2 除不到第⑧次商就为零，

如 26 除到第⑥次商就为零得到二进制数为 011010,我们仍然在高位用零补到 8 个二进制位即 0001,1010。

表 1.1　十进制、二进制和十六进制对照表

十进制	二进制	十六进制	八进制
0	0000	0	0
1	0001	1	1
2	0010	2	2
3	0011	3	3
4	0100	4	4
5	0101	5	5
6	0110	6	6
7	0111	7	7
8	1000	8	10
9	1001	9	11
10	1010	A	12
11	1011	B	13
12	1100	C	14
13	1101	D	15
14	1110	E	16
15	1111	F	17

顺便指出，存放在计算机中的数值都是二进制形式，称为二进制码，即以二进制形式的数码来表示数值信息。用键盘敲入的十进制数需经十翻二运算转换成二进制数存入计算机，而计算机内的二进制形式计算结果则需经二翻十运算变成十进制数，输出显示在计算机的屏幕上，这个过程是由计算机的操作系统自动完成的。

3. 十六进制数

(1) 十六进制数的特点。由于二进制数较长，因而阅读和书写困难，且容易出错。然而 4 位二进制数可以用一位十六进制数来表示，它们之间具有直接的、唯一的对应关系，如表 1.1 所示。由此可知，十六进制数具有如下两个特点。

① 具有 16 个不同的数字符号，除了数字 0 ~ 9 外，还有 A、B、C、D、E、F 等 6 个英文字母，采用英文大小写字母都可以。

② 逢十六进一。从表 1.1 可知，二进制数和十六进制数之间的转换非常简捷方便，即 4 位二进制数可以用一位十六进制数来表示。例如：

94 = 0101,1110 = 5E（H）

其中，H 是 Hexadecimal（十六进制的）第 1 个字母，在计算机领域中用以标明十六进制数。由此可见，十六进制数是二进制数的简洁记法，表示同一数量的位数大幅度减少，每位十六进制数又能按表 1.1 所示的对应关系方便地转换成 4 位二进制数。以后本书对二进制数都是从最低位开始把每 4 位用逗号加以分隔，若读者熟练地记住了表 1.1 所示的对应关系也可以方便地转换成十六进制数。

(2) 十六进制数转换成十进制数。例如，4 位十六进制数转换为十进制数的转换公式为

$$dec = hex_3 \times 16^3 + hex_2 \times 16^2 + hex_1 \times 16 + hex_0 \qquad (1\text{-}5)$$

例如：

$$80AF(H) = 8 \times 16^3 + 0 \times 16^2 + A \times 16 + F$$
$$= 32768 + 0 + 160 + 15 = 32943$$

(3) 十进制数转换成十六进制数。与二进制类似，转换方法是把十进制数不断地用 16 去除，得到整数商和余数，第 1 次所得的余数作为最低有效位数值 hex_0；接着，用所得的整数商除以 16，所得的余数则为第一位有效位数值 hex_1 ····· 直到所得的整数商等于零为止，得到最高有效位数值 hex_3。例如：

① 32943 / 16 = 2058，32943 % 16 = 15 = F(H)；

② 2058 / 16 = 128， 2058 % 16 = 10 = A(H)；

③ 128 / 16 = 8，128 % 16 = 0(H)；

④ 8/16 = 0，8 % 16 = 8(H)。

因此，十进制数转换成十六进制数的运算法则是“用 16 除取余数直到商为零，逆序取余数为十六进制的每个有效位，即第 1 次所取得的余数为最低位，最后的余数为最高位”。需要指出的是，尽管在计算机领域广泛地采用十六进制数来简捷地表示二进制数，但存放在任何计算机内的数据仍然是二进制数形式。

(4) 借用十六进制数简化十翻二运算。如前所述，4 位二进制数可以用一位十六进制数表示，因此，可以说十六进制数是二进制数的简洁记法。如果读者在理解的基础上，熟练地记住了表 1.1 所列的十六进制数和二进制数的对应关系，则把十进制数转化成二进制数，即十翻二运算可以先把十进制数转换成十六进制数，然后，从最高位开始（从左至右）把每一位十六进制数利用表 1.1 的对应关系转换成 4 位二进制，这样计算不仅大大简化了计算步骤且不容易出错。例如：

```
32989  =   8     0     D     D(H)
           ↓     ↓     ↓     ↓ 十六进制数转换成二进制数
       = 1000,0000,1101,1101
```

4. 二进制数的原码、反码和补码

(1) 符号位。以上的整数都是无符号整数，每个二进制位（bit）进行加法运算的规则为：0 + 0 = 0，1 + 0 = 1，0 + 1 = 1，1 + 1 = 0（向高一位进一），1 + 1 + 1（低一位的进位）= 1（向高一位进一）。

带符号的整数在计算机内如何表示呢？通常取一个二进制数的最高位为符号位，1 表示负号，0 表示正号。如图 1.2 所示，一个 8 位二进制数的最高位是 b_7，其后 7 个位是它的数值。如此类推，一个 16 位二进制数的最高位是 b_{15}，其后 15 位是它的数值。一般来说，一个 m 个二进制位的数，其最高位是 b_m，其后 m-1 个二进制位数是它的数值。

这种表示法称为原码，原码表示法中+94 和-94 的数值各位都相同，而符号位不同。原码表示法简单易懂，但对带符号数的运算却不方便，如一个正整数加上一个负整数，一般来说，两异号数相加（或说两同号数相减），就要做减法。为了把减法运算转换成加法运算就引入了反码和补码的概念。

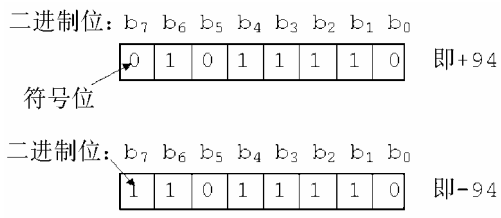


图 1.2　94 和–94 的原码表示法

(2) 反码。一个二进制数逐位取反，即把 0 变成 1，把 1 变成 0，所得到的二进制数就是原来二进制数的反码。如一个 8 位二进制数 B，其数值为 94，其原码用 $B_{原}$ 表示，反码用 $B_{反}$ 表示可得：

$$
\begin{array}{rl}
 & B_{原} = 0\ 1\ 0\ 1\ ,\ 1\ 1\ 1\ 0 \\
+ & B_{反} = 1\ 0\ 1\ 0\ ,\ 0\ 0\ 0\ 1 \\
\hline
 & 2^8 - 1 = 1\ 1\ 1\ 1\ ,\ 1\ 1\ 1\ 1
\end{array}
$$

由此可以看出原码和反码之间的关系为：

$$B_{反} = (2^8 - 1) - B_{原}$$

一般地说，一个m位二进制数原码和反码之间的关系为：

$$B_{反} = (2^m - 1) - B_{原} \qquad (1\text{-}6)$$

(3) 补码。将反码再加1则就得到该二进制数的补码。由式(1-6)可得：

$$B_{补} = B_{反} + 1 = 2^m - B_{原} \qquad (1\text{-}7)$$

式(1-7)是经过数学严格证明的重要公式。

5. 利用补码将减法运算转换成加法运算

二进制运算法则规定：一个正整数的补码就是它的原码，一个负整数的补码是把它的绝对值（肯定是一个正整数）变成二进制数后再变反加1即变补，并且，把符号位也看成一位二进制数，在进行各种运算（包括变补运算、加法运算等）时一起参加运算。由数学严格证明可知，按上述二进制运算法则所得的计算结果，其最高位仍然是符号位，不需对它另行处理，计算得1表示结果为负，得0则表示结果为正。因此，可以说参加运算的数（称为运算量）都以补码形式表示，所得的结果也是以补码形式表示的。例如：

$$(-94)_{补} = (-(94))_{补} = (-(94)_{原})_{补}$$
$$= (0101,1110)_{补} = 1010,0010$$

即可以把-94看成由两部分（负号和绝对值94）组成，负号可看成是求该数绝对值补码的运算符，这种运算称为“变补”运算。由于绝对值肯定是正整数，其补码就是原码。

因此，若二进制数的位数为m（如8位），m个bits可表示2^m个二进制数，其表示的数值范围是$-2^{m-1} \sim 2^{m-1} - 1$，如一个8位二进制数的表示的数值范围为：$-2^7 \sim 2^7 - 1 = -128 \sim 127$。

如此类推，一个16位二进制数的数值范围为：$-2^{15} \sim (2^{15} - 1)$，即-32768～32767。

一般来说，一个m个二进制位的数，其数值范围为：$-2^{m-1} \sim (2^{m-1} - 1)$。

这样一来，就可以把减法运算变成加法运算，如一个正整数减去另一个正整数可以看成加上一个负正整数，即A - B = A + (-B)，由式(1-7)则有：

↑ 溢出

$$A_{原} - B_{原} = A_{原} - (2^m - B_{补}) = A_{原} + B_{补} - 2^m \qquad (1\text{-}8)$$

例如：$94 - 1 = 94 + (-1) = 94 + (-1)_{补} - 2^m$

$1_{原}$ = 0 0 0 0 , 0 0 0 1

$94_{原}$ = 0 1 0 1 , 1 1 1 0　　变补

$+ (-1)_{补}$ = 1 1 1 1 , 1 1 1 1

$94 - 1$ = 0 1 0 1 , 1 1 0 1 = 93

又如：-A + B = (-A) + B，即：

↑ 溢出

$$-A_{原} + B_{原} = -(2^m - A_{补}) + B_{原} = A_{补} + B_{原} - 2^m$$

综上所述，在一个运算式中每当遇到一次负号，则把整数运算量的二进制码变补一次。显然，如果将原码变补两次其结果仍为原码，如一个正整数18，在它前面加一个负号则变为-18，求-18的补码即将18的二进制码变反加1，即：

$18_{原}$ = 0 0 0 1 , 0 0 1 0

变反　　1 1 1 0 , 1 1 0 1

加 1 后得$(-18)_{补}$ = 1　1　1　0 ，1　1　1　0

若再变补一次：

变反　　0　0　0　1 ，0　0　0　1

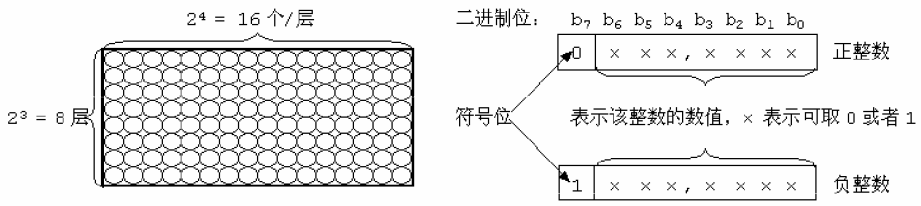
+1　　0　0　0　1 ，0　0　1　0

这就相当于：-(-18) = 18。一般来说，

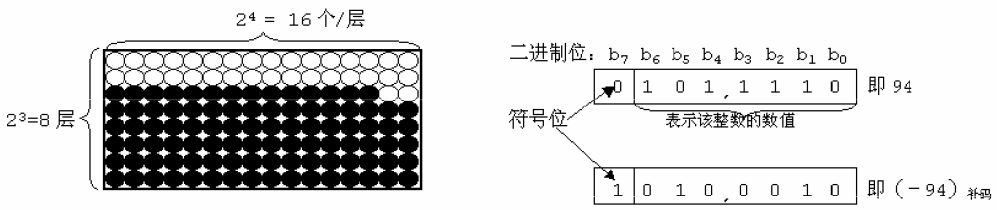
$$A - (-B) = A + B$$

因此，把一个带符号整数的符号位也看成一位二进制数，在进行各种运算（包括变补运算、加法运算等）时一起参加运算，运算结果的符号位不需另行处理，并且可以把减法运算用加一个补码的加法运算来代替，这与代数学中的“减一个正数等于加一个负数”的规律是一致的。因此，一个整数运算量每遇到一个负号时就将它的二进制码变补一次，而实际做的都是加法运算，只要在整数运算量所能表示的数值范围内都会得到正确的结果。这种把减法运算转换成加法运算的例子在日常生活中是常见的。例如当标准时间是 6 点整而时钟却指在 10 点时，有两种校准办法：一种是把时针从 10 点反时针方向拨回 4 个小时校准到 6 点，这显然是一种减法运算，即 10 – 4 = 6；另一种是将时针从 10 点顺时针方向拨 8 个小时也同样可以校准到 6 点，这种校准办法是加法运算，即 10 + 8 = 12 + 6。由于钟表上 12 点和 0 点是重合的，所以，时针拨到 12 点再顺时针拨 6 个小时同样能校准到 6 点。这两种校准办法能等效的原因是可以把 12 看成 0，这在计算机数学中称为“对 12 取模”，而把 12 称为模数（Modulus，常简写为 Mod），它类似于前述的一个 m 位二进制数的容量 2^m。因此，“10 – 4”可以用“10 + 8”代替。显然，加上的数 8 和减去的数 4 具有这样一种关系，即两数之和必须等于模数 12，即 8 + 4 = 12（模数），8 就称为 4 对模数 12 的补码，因此，10 减 4 可以用“加上 4 对模数 12 的补码 8”来代替。一般来说，当一个数 A 减去小于模数 K 的某数 B 时，就可以用加上数 B 对模数 K 的补码 $B_{补}$来代替，即：$A_{原} - B_{原} = A_{原} + B_{补}$。在计算机中要使用补码运算规则，就必须确定模数 K，通常，计算机的一些语言处理系统（如 C、C++和 Java 等）的整数类型数据长度，即二进制码的位数 m，指定为 8 位、16 位、32 位和 64 位等，因此，这些整数类型的模数分别取为 2^8、2^{16}、2^{32} 和 2^{64} 等。

这种利用补码将减法运算转换成加法运算对于初学者有点难于理解，下面用实际例子加以说明。如图 1.3 (a)所示，有一种摆放苹果的箱子可以摆放 8（2^3）层，每层放 16（2^4）个苹果，即最多可放 $8 \times 16 = 2^7 = 128$ 个，即该箱子的容量为 128，它相当于一个字节（8 个 bits）长度的带符号整数类型，最高位 b_7 是符号位，其后 7 个二进制位 $b_0 \sim b_6$ 是它的数值，因此，它的存放数据的容量也是 $2^7 = 128$。当装完 128 个苹果后就填满了，这时必须换一个空箱子，这与上述钟表上 12 点和 0 点是重合的类似，即 128 和 0 重合，这与时钟问题一样在计算机数学中都抽象为“对某正整数取模”，本例是“对 128 取模”，而把 128 称为模数。用图 1.3(b)就可形象地说明求（－94）$_{补}$的实际含义，（－94）的绝对值 94 是苹果箱内已经放有 94 个苹果，箱内还有 128 － 94 = 34 个空位置，这个空位置的个数 34 就是（－94）$_{补}$，即对绝对值 94 = 0101,1110 进行变反加 1 运算后得 1010,0001，其符号位后面的 $b_0 \sim b_6$ 为 010,0001= 34 是它的数值。显然，这个数值的含义是一个容量为 $2^7 = 128$ 个的箱子，已经放有 94 个苹果后还剩 34 个空位置的数目。在计算机数学中则称为“34 是 94 对 128 取模的补码”。同样，如图 1.3(c)所示，若求－1 的补码（－1）$_{补}$，其绝对值是 1 表示箱内只放有一个苹果，对绝对值

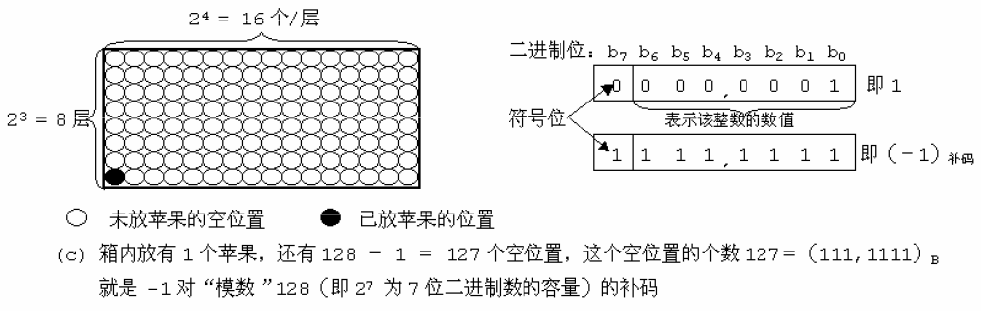


(a) 箱内有 $2^3 \times 2^4 = 2^7 = 128$ 个放苹果的位置即箱子的容量，计算机数学把它称为“模数”



○ 未放苹果的空位置　　● 已放苹果的位置

(b) 箱内放有 94 个苹果，还有 128 － 94 = 34 个空位置，这个空位置的个数 34 = (010,0010)$_B$
就是 -94 对“模数”128（即 2^7 为 7 位二进制数的容量）的补码



● 未放苹果的空位置　　○ 已放苹果的位置

(c) 箱内放有 1 个苹果，还有 128 － 1 = 127 个空位置，这个空位置的个数 127 = (111,1111)$_B$
就是 -1 对“模数”128（即 2^7 为 7 位二进制数的容量）的补码

图 1.3　用补码把减法运算转换成加法运算

1 = 0000,0001 变补运算后得 1111,1111，而表示数值的 $b_0 \sim b_6$ 为 111,1111 = 127，则表示 127 个空位置，称为“127 是 1 对 2^7 = 128 取模的补码”，且 1 + 127 = 模数 128。因此，减法算式“94 － 1”可转换成：

$$94 + (-1)_{补} = 94 + (128 - 1) = 94 + 127 = 221$$

由于箱子的容量只能放 2^7 = 128 个苹果，装满了 128 个就必须换一个空箱子，即把 128 可看成 0，这就是（1-8）式所表示的 2^m 溢出，本例将溢出模数 2^7 = 128，则有：

$$221 - 128 = 93$$

所以，减法算式“94 － 1”可用加法算式“94 + 127”代替，即在容量为 128 的箱子内，将原来减去的 1 个苹果变成加上空位置的个数 127，因模数的溢出即达到 128 就看成 0 使得运算的结果是一样的，这在数学上称为“减 1 运算”，可转换成加上－1 的补码 127，这与前述的时钟问题是完全一致的。

1.1.2　计算机的基础知识

首先考察一下会计是如何用算盘来算账的。如有这样一道算式：

$$(450 + 7 \times 25) \times 130 \times 22 / 32$$

计算时必须先把算式和运算量 450、7、25 等记忆下来，再按计算步骤用珠算四则运算的口诀进行运算，先做圆括号内的算式(450 + 7 × 25)，把算盘上计算的结果记忆在大脑中，再用乘法口诀做乘以 130 的运算并把所得的乘积记忆下来……直到求出最后结果。计算机就是模仿人脑的计算过程来处理实际计算问题的，人们设计出的具有记忆、计算和控制等功能的电子器件和设备构成了计算机的硬件，仅有硬件的计算机称为“裸机”，还不能做任何工作，必须有各种层次的软件，如操作系统、语言处理系统和应用软件等的配合，才能按照人们的意图完成所规定的任务，这就像有了算盘但不会珠算四则运算口诀的人，不可能完成算账任务一样。因此，计算机只有由硬件和软件两大部分组成，才能构成一个能按照人的意图自动进行运算和处理操作的系统，通常称之为“计算机系统”。

1. 二进制编码系统

如前所述，计算机只能识别和处理二进制码，它不仅要用二进制码表示数值，还要用它表示其他各种信息，例如字符、大小写英文字母、运算符和标点符号。而各种字符只能用若干个二进制位按一定规则进行不同的排列和组合所构成的二进制码来表示，这就是二进制编码。对英文字母、运算符和标点符号等字符现都采用 ASCII 码（American Standard Code for Information Interchange，美国国家标准信息交换码），详见附录。该代码由 7 个 bits 组成，可表示 127 个字符。由于微型计算机内存的一个存储单元是 8 个 bits，称为一个字节（Byte），因此，一个字节只能表示一个 ASCII 码字符，而把最高位设置为 0。图 1.4 举出了几个重要的 ASCII 码，其中，b_5 和 b_4 为 1 的字符（十六进制高位为 3）是数字字符，b_6 为 1 的字符（十六进制高位为 4）是英文大写字符，b_5 和 b_6 为 1 的字符（十六进制高位为 6）是英文小写字符。

对于多媒体计算机，把文本、图像和声音等信息按某种标准格式进行二进制编码，就可解决文本、图像和声音等的控制管理、存储、变换和传送等问题。行使这些功能的系统都统称为二进制编码系统。

	b_7	b_6	b_5	b_4	b_3	b_2	b_1	b_0	
字符 NULL:	0	0	0	0	0	0	0	0	= 00(H)
数字字符 0:	0	0	1	1	0	0	0	0	= 30(H)
数字字符 9:	0	0	1	1	1	0	0	1	= 39(H)
英大字母 A:	0	1	0	0	0	0	0	1	= 41(H)
英大字母 B:	0	1	0	0	0	0	1	0	= 42(H)
英小字母 a:	0	1	1	0	0	0	0	1	= 61(H)
英小字母 b:	0	1	1	0	0	0	1	0	= 62(H)

图 1.4　'0'、'9'、'A'和'a'等字符的 ASCII 码

2. 微型计算机硬件的基本组成

即使计算机发展到今天，引起了多次世界技术革命，改变着人类生活的方方面面，但从它的基本运行原理看，与 1946 年问世的第一台电子计算机大同小异，都可统称为冯·诺依曼（Von Neumann）型计算机，仍然由机器的硬件顺序地执行一条一条指令来完成所规定的任务，能完成某项工作所规定任务的一组指令称为“程序”，它是用某种计算机语言编写的、能指挥计算机自动地执行去完成某项工作。例如下面的一个计算任务：

$$(36 + 72) \times 2 + 128$$

计算机模仿人脑自动地完成这项工作，首先它必须像人的大脑一样具有记忆功能，不仅要记住这些运算的数据，还需要记住加、减、乘、除的运算规则和该任务的计算步骤，即需要把它们设计成“程序”再保存在计算机内。因此，冯·诺依曼型计算机必须具有两个基本功能，一是能保存程序和数据，二是能自动地执行程序完成所规定的任务，这就是