# 数据的存储

2024年7月24日 12:07

- ## 本章重点

1. 数据类型详细介绍
2. 整形在内存中的存储:原码、反码、补码
3. 大小端字节序介绍及判断
4. 浮点型在内存中的存储解析

## 一、数据类型介绍

- C语言类型：

  **内置类型**：char(字符数据类型/1个字节)、short(短整型/2个字节)、int(整形/4个字节)、long(长整形/4或8个字节)、long long(更长的整形/8个字节)、float(单精度浮点数/4个字节)、double(双精度浮点数/8个字节)；


  **自定义类型（构造类型）**：数组类型（int arr[10];将数组名arr去掉就是类型，类型就是int[10]，不同数组各不相同）、结构体类型struct、枚举类型enum、联合类型union

  **指针类型**：int*、char*、float*、void*(无具体类型的指针)（创建的指针变量都是4/8个字节）

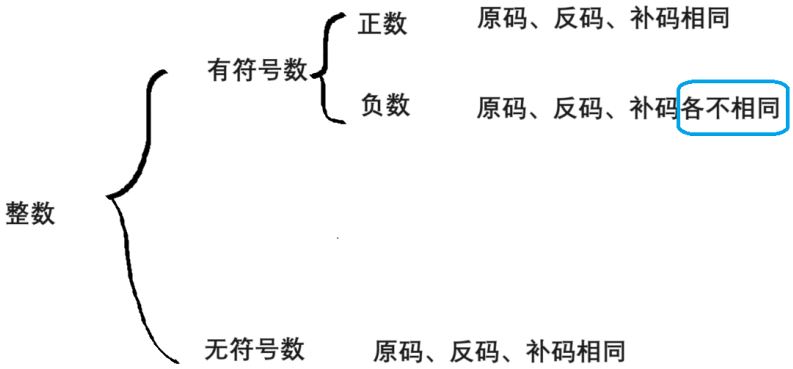
  **空类型**：void表示空类型（无类型），通常应用于函数的返回类型、函数的参数、指针类型。

- 类型的意义：使用这个类型开辟内存空间的大小(大小决定了使用范围)；如何看待内存空间的视角。
- 例1：见8.1：数据类型介绍

## 二、整形在内存中的存储

- 原码、反码、补码

  计算机中的有符号数（整数）有三种表示方法，即原码、反码和补码。（无符号数，原码、反码、补码相同）




三种表示方法均有**符号位**和**数值位**两部分，符号位都是用0表示“正”，用1表示“负”，而数值位三种表示方法各不相同。

原码：直接按照正负数的形式翻译成二进制就可以。

反码：将原码的符号位不变，其他位依次按位取反就可以得到了。

补码：反码+1就得到补码。

**结论**：对于整形来说，数据存放内存中其实存放的是补码（倒着存放的）。

**为什么呢?** 在计算机系统中，数值一律用补码来表示和存储。原因在于，使用补码，可以将符号位和数值域统一处理；同时，加法和减法也可以统一处理（CPU只有加法器）。此外，补码与原码相互转换，其运算过程是相同的，不需要额外的硬件电路。

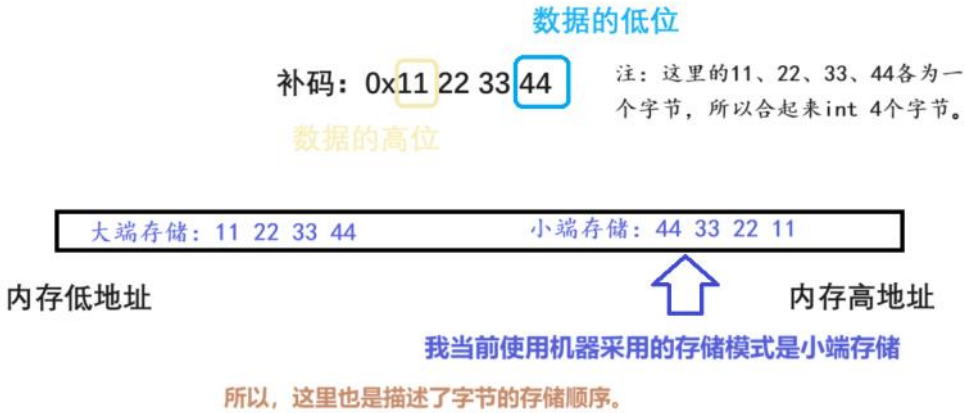
- 例2：见8.2：原码、反码、补码

- 存储的是补码，但是顺序有点不对劲（倒着存放的）。这是又为什么?

  **大小端介绍:**

  大端(存储)模式，是指数据的低位保存在内存的高地址中，而数据的高位，保存在内存的低地址中。（大端字节序存储模式）

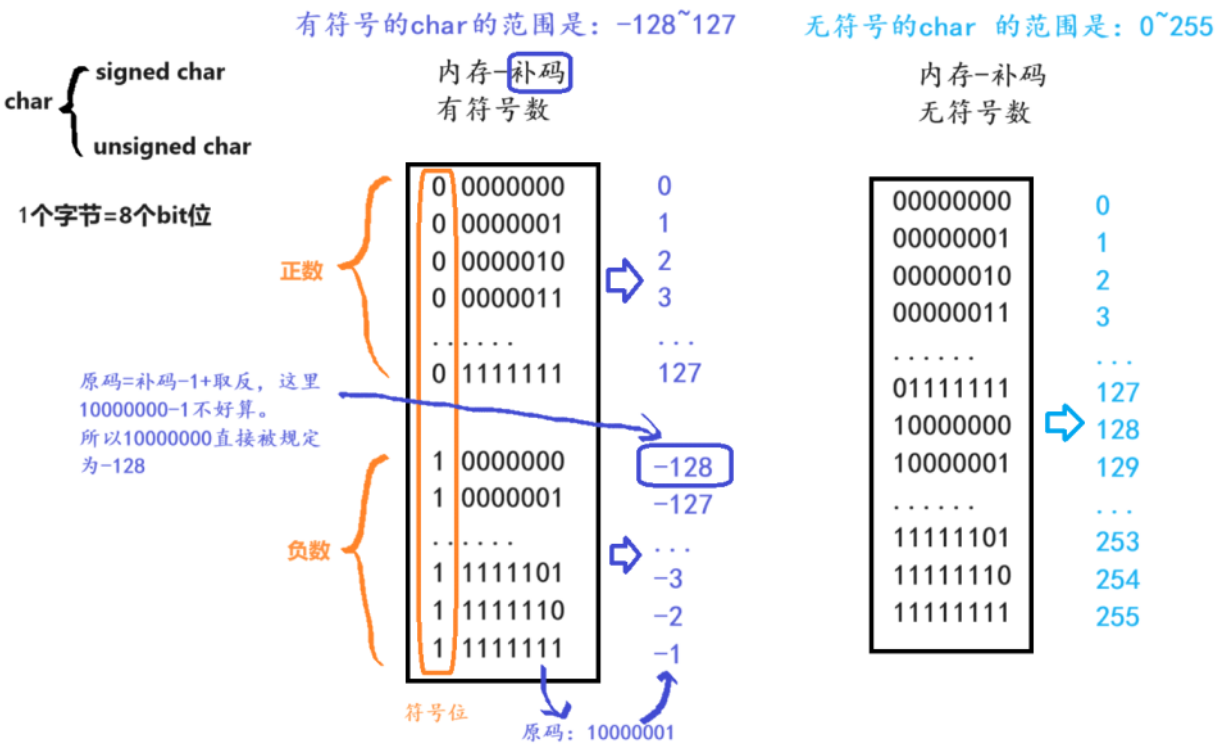
  小端(存储)模式，是指数据的低位保存在内存的低地址中，而数据的高位，，保存在内存的高地址中。（小端字节序存储模式）




**为什么有大端和小端:**

这是因为在计算机系统中，我们是以字节为单位的，每个地址单元都对应着一个字节，一个字节为8bit。但是在C语言中除了8bit的char之外，还有16bit的short型，32bit的long型(要看具体的编译器)；另外，对于位数大于8位的处理器，例如16位或者32位的处理器，由于寄存器宽度大于一个字节，那么必然存在着一个如何将多个字节安排的问题。因此就导致了大端存储模式和小端存储模式。

- 例3：见8.3：设计一个小程序来判断当前机器的字节序。
- 例4：见8.4：练习1
- 例5：见8.5：练习2




- 例6：见8.6：练习3
- 例7：见8.7：练习4
- 例8：见8.8：练习5
- 例9：见8.9：练习6

## 三、浮点型在内存中的存储

- 常见的浮点数：3.14159；1E10（科学计数法，1.0*10^10）
- 浮点数的类型包括：float、double、long double类型
- 整形家族范围可以在头文件<limits.h>中查到；浮点型家族范围可以在头文件<float.h>中查到

- 根据国际标准IEEE（电气和电子工程协会）754，任意一个二进制浮点数V可以表示成下面的形式:
  $(-1)^S \times M \times 2^E$
  $(-1)^S$表示符号位，当$S=0$，V为正数；当$S=1$，V为负数。
  M表示有效数字，大于等于1，小于2。（二进制）
  $2^E$表示指数位。
  例：-5.0（十进制）-> -（0101.0）-> -（1.01*2^2）-> $(-1)^1 \times 1.01 \times 2^2$
  S=1，M=1.01，E=2



**IEEE 754对有效数字M和指数E，还有一些特别规定。**前面说过，$1 \leq M < 2$,也就是说，$M$可以写成 1.xxxxxx 的形式，其中xxxxxx表示小数部分。

IEEE 754规定，在计算机内部保存M时，默认这个数的第一位总是1，因此可以被舍去，只保存后面的xxxxxx部分。比如保存1.01的时候，只保存01,等到读取的时候，再把第一位的1加上去。这样做的目的，是节省1位有效数字。以32位浮点数为例，留给M只有23位，将第一位的1舍去以后，等于可以保存24位有效数字。

**至于指数E，情况就比较复杂。**

**首先，E为一个无符号整数(unsigned int)。**这意味着，如果E为8位，它的取值范围为0~255；如果E为11位，它的

取值范围为0~2047。但是，我们知道，科学计数法中的E是可以出现负数的，所以IEEE 754规定，存入内存时E的真实值必须再加上一个中间数，对于8位的E，这个中间数是127；对于11位的E，这个中间数是1023。比如，2^10的E是10，所以保存成32位浮点数时，必须保存成10+127=137，即10001001。

- 例：0.5（十进制） -> 0.1 （二进制的第一个小数位的权重是2^-1即0.5，所以0.5转化为二进制0.1）-> $(-1)^0 \times 1.0 \times 2^{-1}$
  E=-1 -> 存入内存时为-1+127=126
- 例10：见8.10：浮点型在内存中的存储

- **然后，指数E从内存中取出还可以再分成三种情况**

1. **E不全为0或不全为1（常规情况）**
   指数E的计算值减去127（或1023），得到真实值；再将有效数字M前加上第一位的1。
2. **E全为0**
   这时，浮点数的指数E等于1-127(或者1-1023)即为真实值；有效数字M不再加上第一位的1，而是还原为0.xxxxxx的小数。这样做是为了表示+0，以及接近于0的很小的数字。
   例：E=00000000 -> E=-127 -> +/- * 1.xxxxxx * 2^-127 -> 直接写成 +/- * 0.xxxxxx * 2^-126即可
3. **E全为1**
   这时，如果有效数字M全为0，表示±无穷大(正负取决于符号位s):
   例：E=11111111(255) -> E+127=255 -> E=128 -> +/- * 1.xxxxxx * 2^128 -> 表示的是正负无穷大的数字

- 例11：见8.11：整形和浮点型的存储