第10讲：操作符

C语言非常灵活：丰富的操作符，甚至可以直接操作到二进制位。

1、二进制

10进制数：满10进1；每一位由0-9组成。

2进制数：满2进1；每一位由0、1组成。

1.1 2进制转10进制

每位数乘以权重值（2^n-1）再相加。

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  |  |  | 10进制数 |
| 10进制的位 | 1 | 2 | 3 |  |
| 权重 | 10^2 | 10^1 | 10^0 |  |
| 权重值 | 100 | 10 | 1 |  |
| 求值 | 1\*100 | 2\*10 | 3\*1 | 123 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  |  |  |  |  | 10进制数 |
| 2进制的位 | 1 | 1 | 0 | 1 |  |
| 权重 | 2^3 | 2^2 | 2^1 | 2^0 |  |
| 权重值 | 8 | 4 | 2 | 1 |  |
| 求值 | 1\*8 | 1\*4 | 0\*2 | 1\*1 | 13 |

1.2 10进制转2进制

短除法：由下往上依次所得余数。

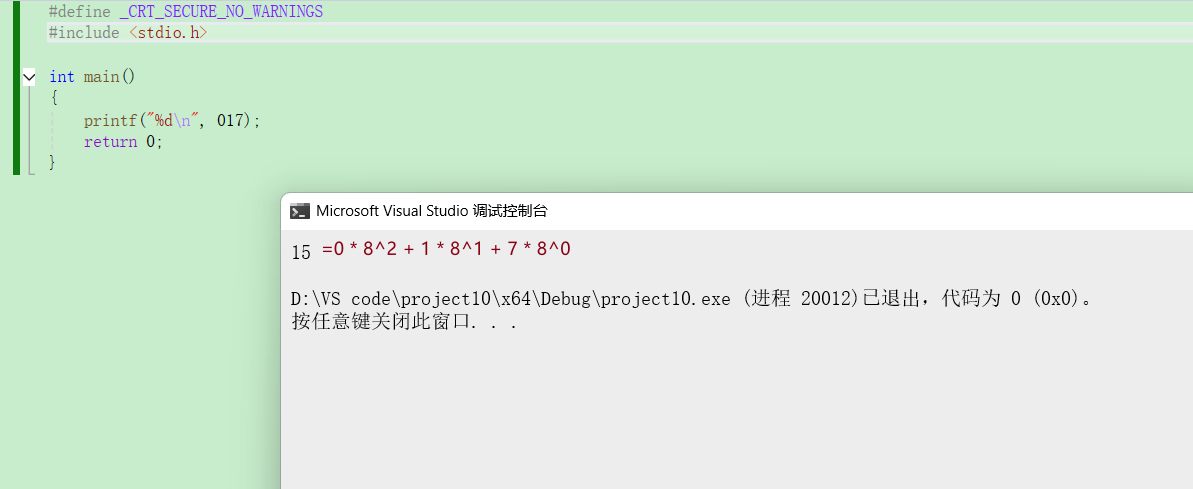
|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 2 | 125 | | | | | | | 余数为1 |
|  | 2 | 62 | | | | | | 余数为0 |
|  |  | 2 | 31 | | | | | 余数为1 |
|  |  |  | 2 | 15 | | | | 余数为1 |
|  |  |  |  | 2 | 7 | | | 余数为1 |
|  |  |  |  |  | 2 | 3 | | 余数为1 |
|  |  |  |  |  |  | 2 | 1 | 余数为1 |
| 10进制的125转换成2进制：1111101 | | | | | | | | |

1.3 2进制转8进制转16进制

1.3.1 2进制转8进制

8进制数（每一位0-7），各自写成2进制，最多有3个2进制位就够了，比如7的2进制是111，所以在2进制转8进制数的时候，从2进制序列中右边低位开始向左每3个2进制位会换算一个8进制位，剩余不够3个2进制位的直接换算。比如说：2进制的01101011换算成8进制就是0153（前面加上0，编译器会把它识别成8进制数）

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 10进制数 | 107 | | | | | | | |
| 2进制 | 0 | 1 | 1 | 0 | 1 | 0 | 1 | 1 |
| 8进制 | 1 | | 5 | | | 3 | | |



1.3.2 2进制转16进制

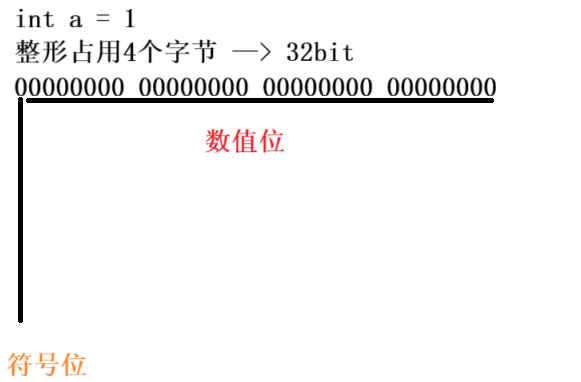
16进制的数字每一位是0~9,a~f（10-15）的，0~9,a~f的数字，各自写成2进制，最多有4个2进制位就足够了，比如f（15）的二进制是1111，所以在2进制转16进制数的时候，从2进制序列中右边低位开始向左每4个2进制位会换算一个16进制位，剩余不够4个2进制位的直接换算。比如说：2进制的01101011，换成16进制：0x6b，16进制表示的时候前⾯加0x。

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 10进制数 | 107 | | | | | | | |
| 2进制 | 0 | 1 | 1 | 0 | 1 | 0 | 1 | 1 |
| 8进制 | 6 | | | | b | | | |

2、原码、反码、补码

整数的2进制的三种表示方法（即原码、反码、补码），这里不涉及小数（浮点数）。

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0表示“负”。数值位最高位的一位是被当做符号位，剩余的都是数值位。



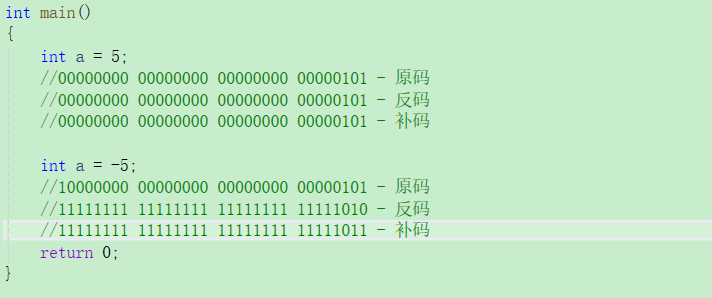
正整数的原码、反码、补码均一样！

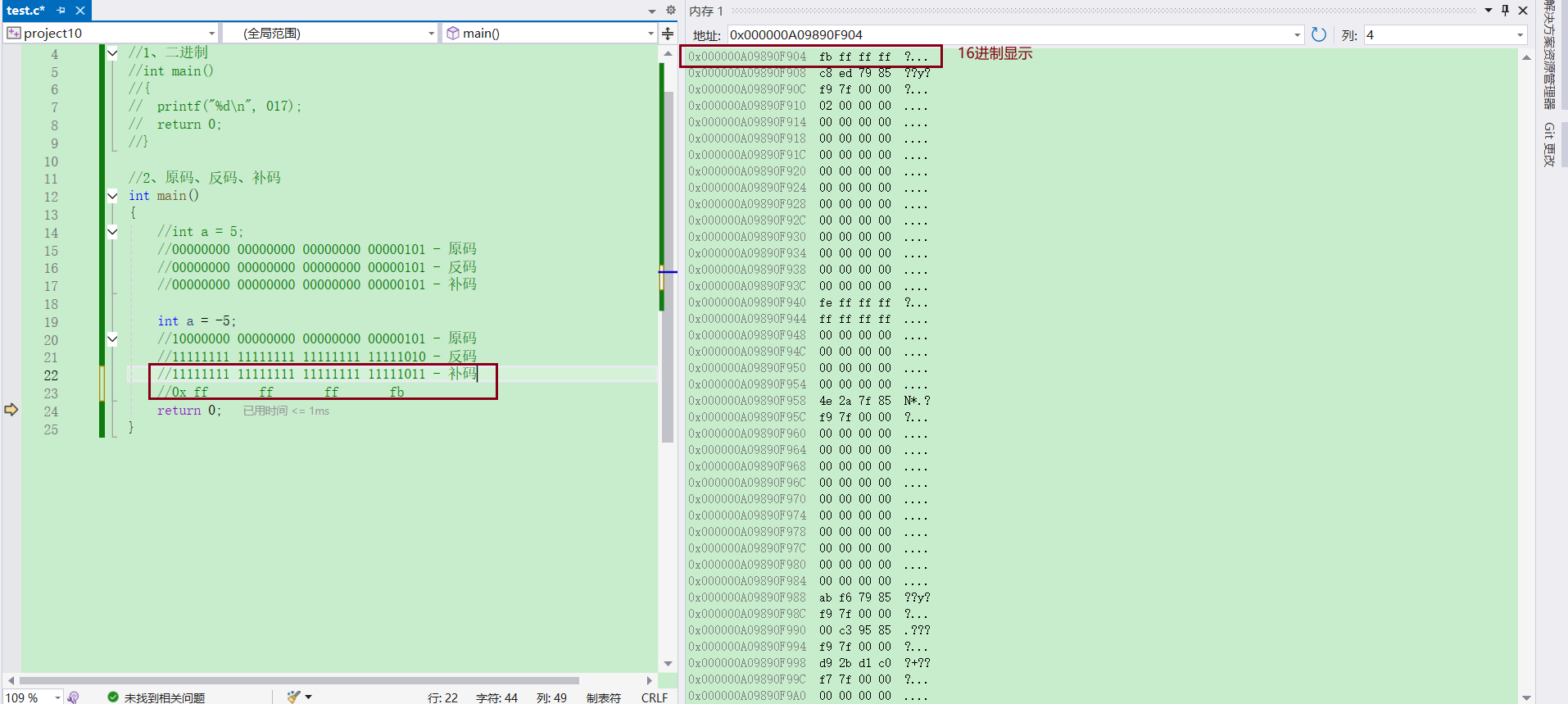
负整数的三种表示方法各不相同。

原码：直接将数值按照正负数的形式翻译成二进制得到的就是原码。

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反码。

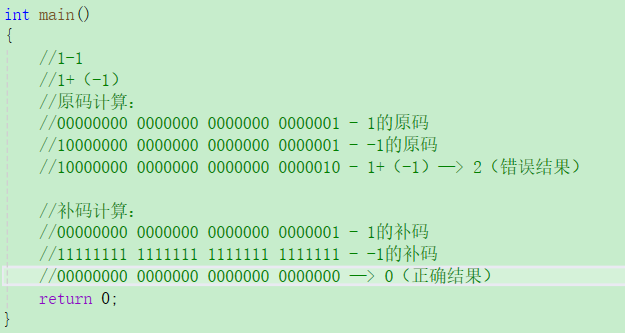
补码：反码+1就得到补码。



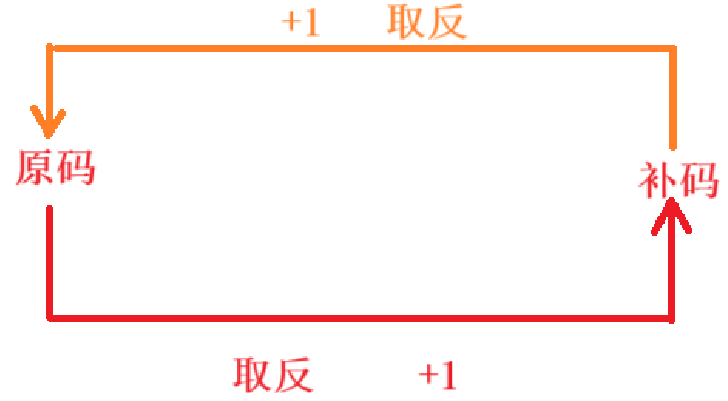


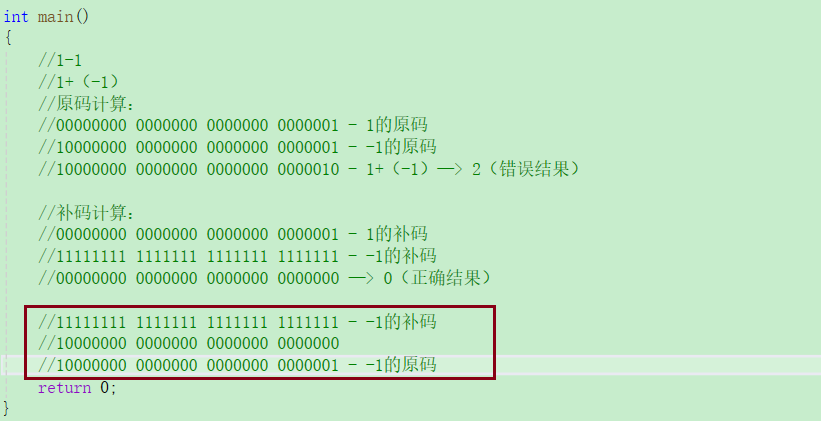
对于整形来说：数据存放内存中其实存放的是补码。为什么呢？

原因：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。



原码到补码、补码到原码换算：





整数：正整数、负整数。无符号整数：当成正整数处理。

3、移位操作符

<< 左移操作符

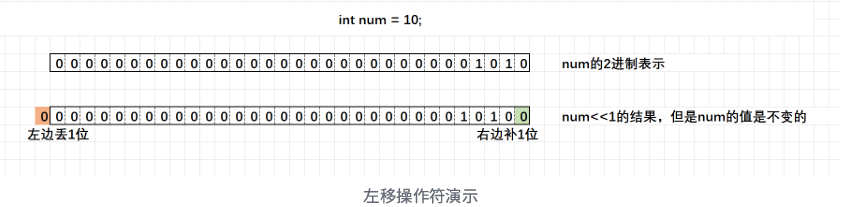
>> 右移操作符

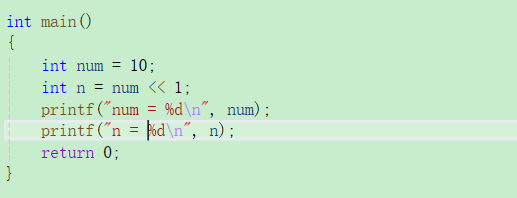
移位操作符的操作数只能是整数。

3.1 左移操作符

移位规则：左边抛弃、右边补0

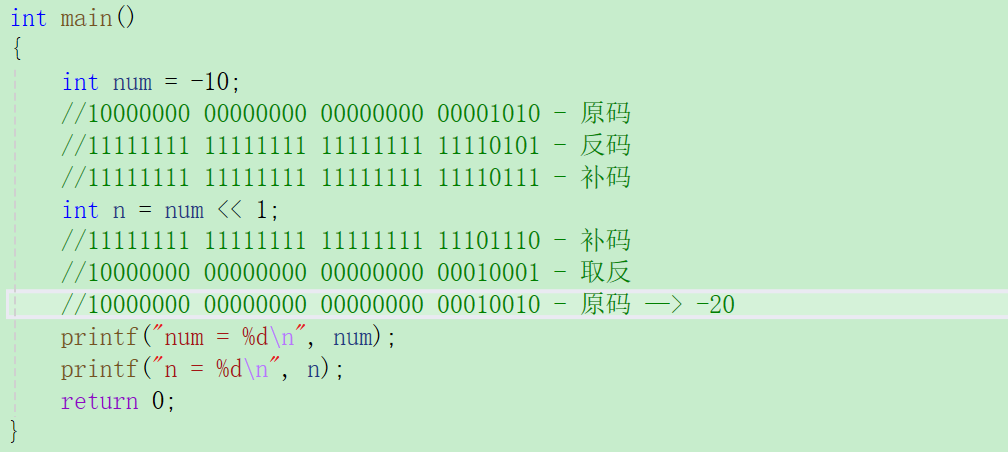
正整数左移：







负整数左移：





左移移动的是补码，打印出来是按照原码来打印。

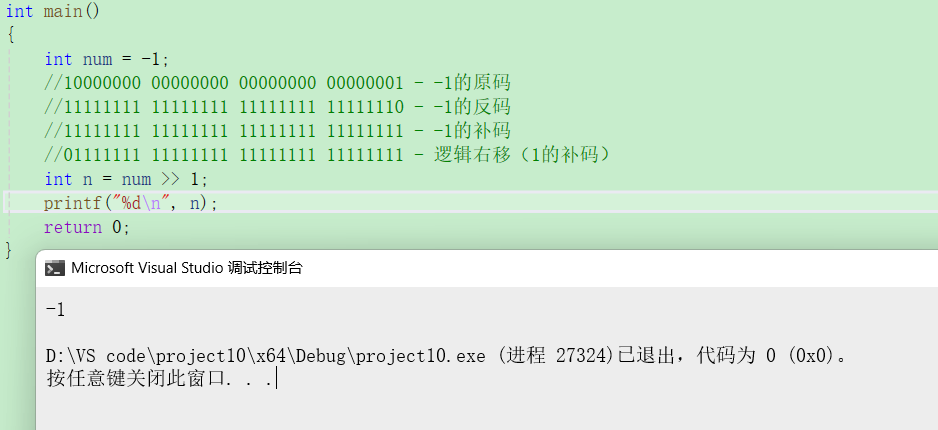
3.2 右移操作符

逻辑右移：左边用0填充，右边丢弃。

算术右移：左边用原该值的符号位填充，右边丢弃。

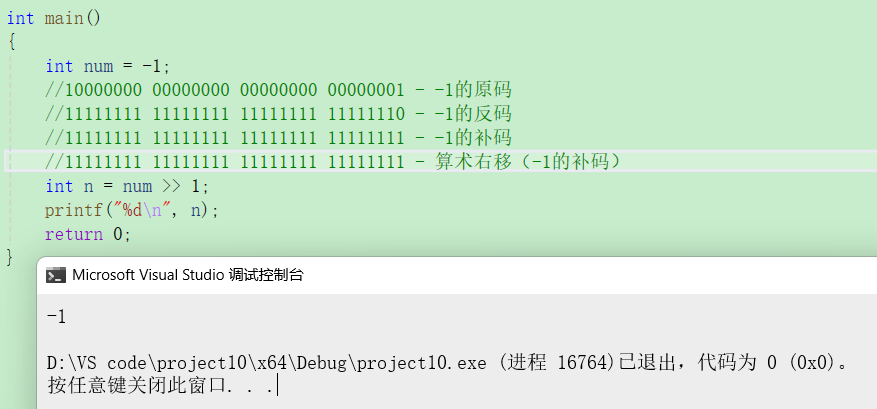
采用算术右移还是逻辑右移，是不确定的，取决于编译器。大部分编译器采用算术右移。

逻辑右移：



VS 2022编译器不支持逻辑右移，无法展示。

算术右移：



注意：对于移位运算符，不要移动负数位，这个是标准未定义的。