>>

[Bitwise Operators]

Description

The right shift operator >> causes the bits of the left operand to be shifted **right** by the number of positions specified by the right operand.

Syntax

variable >> number\_of\_bits;

Parameters

variable: Allowed data types: byte, int, long.  
number\_of\_bits: a number that is < = 32. Allowed data types: int.

Example Code

int a = 40; // binary: 0000000000101000

int b = a >> 3; // binary: 0000000000000101, or 5 in decimal

Notes and Warnings

When you shift x right by y bits (x >> y), and the highest bit in x is a 1, the behavior depends on the exact data type of x. If x is of type int, the highest bit is the sign bit, determining whether x is negative or not, as we have discussed above. In that case, the sign bit is copied into lower bits, for esoteric historical reasons:

int x = -16; // binary: 1111111111110000

int y = 3;

int result = x >> y; // binary: 1111111111111110

This behavior, called sign extension, is often not the behavior you want. Instead, you may wish zeros to be shifted in from the left. It turns out that the right shift rules are different for unsigned int expressions, so you can use a typecast to suppress ones being copied from the left:

int x = -16; // binary: 1111111111110000

int y = 3;

int result = (unsigned int)x >> y; // binary: 0001111111111110

If you are careful to avoid sign extension, you can use the right-shift operator >> as a way to divide by powers of 2. For example:

int x = 1000;

int y = x >> 3; // integer division of 1000 by 8, causing y = 125.