int

We have said that an int is a 32-bit value interpreted as an integer directly from its binary representation. As it turns out, this is only half of the story—the positive half of the story. If we dedicate all 32 bits to expressing positive numbers, we can express 2^{32} values, from 0 up to 4,294,967,295. We request this interpretation by using the qualifier unsigned in the declaration, as shown in the second line of the figure below.

What about negative numbers? ints are actually represented using an encoding called two's complement, in which half of the 2^{32} possible bit patterns are used to express negative numbers and the other half to express positive ones. Specifically, all numbers with the most significant bit equal to 1 are negative numbers. A 32-bit int is inherently signed (i.e., can have both positive and negative values) and can express values from -2,147,483,648 to 2,147,483,647. Note that both unsigned and signed ints have 2^{32} possible values. For the unsigned int they are all positive; for the signed int, half are positive and half are negative.

In two's complement, the process for negating a number may seem a bit weird, but actually makes a lot of sense when you understand why it is setup this way. To compute negative X, you take the bits for X, flip them (turn 0s into 1s and 1s into 0s), and then add 1. So if you had 4-bit binary and took the number 5 (0101) and wanted negative 5, you would first flip the bits (1010) and then add 1 (1011). Why would computer scientists pick such a strange rule? It turns out that this rule makes it so that you can just add numbers naturally and get the right result whether the numbers are positive or negative. For example -5 + 1 = -4, and in binary 1011 + 0001 is 1100. To see that 1100 is -4, flip the bits (0011) and add 1 (0100) which is 4.

Another pair of qualifiers you may run into are short and long which decrease or increase the total number of bits dedicated a particular variable, respectively. For example, a short int (also referred to and declared in C simply as a short) is often only 16 bits long. Technically, the only requirement that the C language standard imposes is that a short int has fewer than or equal to as many bits as an int, and that a long int has greater than or equal to as many bits as an int.

Referring to the figure above, at first glance, c and x appear identical since they both have the binary value 65. However, they differ in both size (c has only 8 bits whereas x has 32) and interpretation (c's value is interpreted using ASCII encoding whereas x's value is

interpreted as a signed integer). Similarly, y and z are identical in hardware but have differing interpretations because y is unsigned and z is not.