Here, both zero and null are 0, and both one and numero_uno are 1. In earlier versions of C++, you could assign only int values (or values that promote to int) to enumerators, but that restriction has been removed so that you can use type long values.

**Value Ranges for Enumerations**

Originally, the only valid values for an enumeration were those named in the declaration. However, C++ now supports a refined concept by which you validly can assign values by a type cast to an enumeration variable. Each enumeration has a range, and you can assign any integer value in the range, even if it's not an enumerator value, by using a type cast to an enumeration variable. For example, suppose that bits and myflag are defined this way:

```c
enum bits{one = 1, two = 2, four = 4, eight = 8};
bits myflag;
```

Then, the following is valid:

```c
myflag = bits(6);    // valid, because 6 is in bits range
```

Here 6 is not one of the enumerations, but it lies in the range the enumerations define.

The range is defined as follows. First, to find the upper limit, take the largest enumerator value. Find the smallest power of two greater than this largest value, subtract one, and that is the upper end of the range. For example, the largest bigstep value, as previously defined, is 101. The smallest power of two greater than this is 128, so the upper end of the range is 127. Next, to find the lower limit, find the smallest enumerator value. If it is zero or greater, the lower limit for the range is zero. If the smallest enumerator is negative, use the same approach as for finding the upper limit, but toss in a minus sign. For example, if the smallest enumerator is -6, the next power of two (times a minus sign) is -8, and the lower limit is -7.

The idea is that the compiler can choose how much space to hold an enumeration. It might use one byte or less for an enumeration with a small range and four bytes for an enumeration with type long values.

**Pointers and the Free Store**