The blessing of the leading zero

devblogs.microsoft.com/oldnewthing/20210518-00

May 18, 2021

Raymond Chen

Some time ago, I noted the curse of the leading zero. But sometimes the leading zero can be a blessing.

For example, there is a debugger extension we use internally that accepts an integer on the command line. Sometimes, this integer can be negative, but if you just type the negative number the usual way, the extension's command line parser thinks that the leading hyphen is a command line switch.

```
0:001> !widget 2
Widget 2 is not in use.

0:001> !widget -4
Invalid switch "-4"
```

The trick for passing a negative number is to add a leading zero:

```
0:001> !widget 0-4
Widget -4 is in use by Bob.
```

The leading zero is also handy for avoiding warning C4146: "unary minus operator applied to unsigned type, result still unsigned." You can run into this when you are using the unary minus operator as part of some fancy bit-twiddling scheme.

```
uint32_t twiddle(uint32_t selector, uint32_t value)
{
  uint32_t mask = -(selector & 1);
  return value & mask;
}
```

The computation of `-(selector & 1)` looks at the bottom bit of `selector`. If the bit is clear, then the mask is zero. If the bit is set, then the mask is `-1`, which according to the C++ rules for unsigned arithmetic produces the value which is all-bits-set. The resulting mask is then *and*'ed against the second value.

The result is that `twiddle` returns zero if the selector is even, and returns the `value` if the selector is odd.

Another example is the bit-twiddling trick:

```
uint32_t lowest_set_bit(uint32_t value)
{
  return value & -value;
}
```

This magic expression extracts the lowest set bit of the value.

Both of these trigger warning C4146. The warning is trying to tell you, “Hey, so it looks like you’re trying to take the negative of an unsigned number. You might think that this gives you a negative number, but it doesn’t.” It’s trying to warn you about this:

```
void f(uint32_t value)
{
 if (-value < -2) too_low();
 ...
}
```

The test `-value < -2` is a comparison between an unsigned and a signed value, and the rules for C++ say that both sides are converted to unsigned values, and the values are then compared as unsigned. Therefore, this test is really

```
 if (-value < 0xFFFFFFFE) too_low();
```

which is probably not what you intended.

But in the case where you’re doing bit-twiddling, you know that you’re getting another unsigned value. You’re really after the bit pattern, not the mathematical negative. You can appease the compiler by changing the unary minus to a binary subtraction:

```
uint32_t lowest_set_bit(uint32_t value)
{
  return value & (0-value);
}
```

Subtracting a value from zero is the same as taking its negative, but using the binary subtraction operator avoids the warning about taking the negative of an unsigned value.

Raymond Chen

Follow