

* The 32-bit signed integer initialized to zero and then decremented did just as we expected. It went from 0 to -1.
* The reason why the 32-bit **unsigned** integer initialized to zero and then decremented did not do as expected is because an “unsigned” integer has no way to represent negative numbers since we didn’t allocate that extra bit necessary to do so. Instead, the result is a wrap-a-round to the biggest value available (In this case 4,294,967,295 or 2^(32-1) ). This is called an underflow (the number is too small to be represented).
* In the next example we get to see a “overflow”. Here we have a 16-bit signed integer initialized to the maximum value that it can hold (2^(15-1)). Incrementing this value results in another wrap-a-round, but this time to the lowest (or most negative) number that it can hold.
* In this example we see another “overflow”. Because we know that unsigned integers can’t represent negative values and always start with 0, incrementing this particular 16-bit unsigned integer that was initialized to its highest possible value (2^(16-1)) results in it wrapping around to regurgitate the number 0.