Memory and Types

If you're new to programming, then the concept of variable types is likely new to you as well. In this lecture, we'll look at how the Arduino allocates memory for each variable type, and how it interprets the value it saves based on the variable type. In this lecture you'll learn how computers really do use ones and zeros to represent almost *anything*.

Bits and Bytes

Lets start at the most basic level: bit logic. A bit can be either 0 or 1; there are no other possible values. This is what is known as a binary system. A bit by itself can only be used to express two possible numbers, but when we combine them together into groups, we can express far more numbers. For example, with two bits we have the combinations of 00, 01, 10, and 11, which is twice as many as before. If we add a third bit to the front, then we have all four of the previous combinations with a 0 or 1 added to the beginning, again twice as many, so 8 combinations (000, 001, 010, 011, 100, 101, 110, 111). Do you see a pattern developing? Every time we add a bit, we double the number of combinations, thus the quantity of numbers we can express with a group of bits is equal to 2^n where n is the number of bits. For instance, an 8 bit number can express 2^8 or 256 possible numbers.

The most common grouping of bits is to put 8 of them together. We refer to this grouping as a byte. Now, instead of only being able to express 2 possible numbers, we can express 256 possible numbers. We do this by giving each position in the byte a different multiplier based on a power of 2.

2^0	2^1	2^2	2^3	2^4	2^5	2^6	2^7

We then add all these bits together to get some value for the byte. For instance, lets say we wanted to use a byte to save the number 83. We could express that as the following byte:

2^0	2^1	2^2	2^3	2^4	2^5	2^6	2^7
1	1	0	0	0	1	1	0

Lets add up all those values to confirm that our expression is valid:

2^0 x 1 = 1	2^4 x 0 = 16
2^1 x 0 = 0	2^5 x 1 = 0
2^2 x 1 = 2	2^6 x 1 = 64
2^3 x 0 = 0	2^7 x 0 = 0

64+16+2+1 = 83

One result of this is that every integer from 0 to infinity can be expressed as a sum of powers of 2. Feel the power? Now, the power is yours!

Data Types

Now that we have some understanding of how we numbers are represented in binary, lets look at some of the data types we use on the Arduino, and how it allocates storage for the numbers in memory.

boolean (1 byte) – This data type can only be either true or false. Strictly speaking, we really only need a single bit to be able to express these two possible states. However, since most memory is allocated in bytes, we use a whole byte to store this data type.

int (2 byte) – This data type represents an integer number. Since we are using 16-bits (2 bytes) to store this number, we get a total of 2^{16} (65,536) possible numbers. We let these numbers range from -32,768 to 32,767.

unsigned int (2 byte) – This data type represents an integer number. Since we are using 16-bits (2 bytes) to store this number, we get a total of 2^{16} (65,536) possible numbers. This time, we only use positive numbers and let them range from 0 to 65,535.

long (4 byte) – This is a data type that represents an integer number. We are using 32-bits (4 bytes) to store this number, so we get a total of 2^{32} (4,294,967,296). We let these numbers range from from -2,147,483,648 to 2,147,483,647.

float (4 byte) – This is a data type that represents a number with a decimal place (called a floating point number). The float type allows about 6-7 digits of decimal precision depending on much precision is used for the part of the number to the left of the decimal.

char (1 byte) – This data type is used to represent a single character. Of course, we only store numbers in bytes, so we use a system called ASCII (American Standard Code for Information Interchange) to map those values to letters. For instance, the character ‘A’ happens to correspond with the value 65. Casting a variable as a char is what tells the Arduino to use the ASCII translation instead of leaving it as a number.