Arduino data types and constants.

https://www.arduino.cc/reference/en/

Table of Contents

Constants

Floating Point Constants

Similar to integer constants, floating point constants are used to make code more readable. Floating point constants are swapped at compile time for the value to which the expression evaluates.

Example Code

```
n = 0.005;  // 0.005 is a floating point constant
```

Floating point constants can also be expressed in a variety of scientific notation. 'E' and 'e' are both accepted as valid exponent indicators.

floating-point constant	evaluates to:	also evaluates to:
10.0	10	
2.34E5	2.34 * 10^5	234000
67e-12	67.0 * 10^-12	0.000000000067

Integer Constants

Integer constants are numbers that are used directly in a sketch, like 123. By default, these numbers are treated as int but you can change this with the U and L modifiers (see below).

Normally, integer constants are treated as base 10 (decimal) integers, but special notation (formatters) may be used to enter numbers in other bases.

Base	Example	Formatter	Comment
10 (decimal)	123	none	
2 (binary)	B1111011	leading 'B'	only works with 8 bit values (0 to 255) characters 0&1 valid
8 (octal)	0173	leading "0"	characters 0-7 valid
16 (hexadecimal)	0x7B	leading "0x"	characters 0-9, A-F, a-f valid

Decimal (base 10)

This is the common-sense math with which you are acquainted. Constants without other prefixes are assumed to be in decimal format.

Example Code:

```
n = 101;      // same as 101 decimal    ((1 * 10^2) + (0 * 10^1) + 1)
```

Binary (base 2)

Only the characters 0 and 1 are valid.

Example Code:

```
n = B101;     // same as 5 decimal    ((1 * 2^2) + (0 * 2^1) + 1)
```

The binary formatter only works on bytes (8 bits) between 0 (B0) and 255 (B11111111). If it is convenient to input an int (16 bits) in binary form you can do it a two-step procedure such as:

```
myInt = (B11001100 * 256) + B10101010;    // B11001100 is the high byte
```

Octal (base 8)

Only the characters 0 through 7 are valid. Octal values are indicated by the prefix "0" (zero).

Example Code:

```
n = 0101;     // same as 65 decimal    ((1 * 8^2) + (0 * 8^1) + 1)
```

It is possible to generate a hard-to-find bug by (unintentionally) including a leading zero before a constant and having the compiler unintentionally interpret your constant as octal.

Hexadecimal (base 16)

Valid characters are 0 through 9 and letters A through F; A has the value 10, B is 11, up to F, which is 15. Hex values are indicated by the prefix "0x". Note that A-F may be syted in upper or lower case (a-f).

Example Code:

```
n = 0x101;    // same as 257 decimal    ((1 * 16^2) + (0 * 16^1) + 1)
```

U & L formatters:

By default, an integer constant is treated as an int with the attendant limitations in values. To specify an integer constant with another data type, follow it with:

- a 'u' or 'U' to force the constant into an unsigned data format. Example: 33u
- a 'l' or 'L' to force the constant into a long data format. Example: 100000L
- a 'ul' or 'UL' to force the constant into an unsigned long constant. Example: 32767ul

Defining Logical Levels: true and false (Boolean Constants)

false

`false` is the easier of the two to define. false is defined as 0 (zero).

true

`true` is often said to be defined as 1, which is correct, but true has a wider definition. Any integer which is non-zero is true, in a Boolean sense. So -1, 2 and -200 are all defined as true, too, in a Boolean sense.

Note that the `true` and `false` constants are typed in lowercase unlike `HIGH`, `LOW`, `INPUT`, and `OUTPUT`.

Defining Pin Levels: HIGH and LOW

When reading or writing to a digital pin there are only two possible values a pin can take/be-set-to: `HIGH` and `LOW`.

HIGH

The meaning of `HIGH` (in reference to a pin) is somewhat different depending on whether a pin is set to an `INPUT` or `OUTPUT`. When a pin is configured as an `INPUT` with pinMode(), and read with digitalRead(), the Arduino (ATmega) will report `HIGH` if:

- a voltage greater than 3.0V is present at the pin (5V boards)
- a voltage greater than 2.0V volts is present at the pin (3.3V boards)

A pin may also be configured as an INPUT with `pinMode()`, and subsequently made HIGH with digitalWrite(). This will enable the internal 20K pullup resistors, which will *pull up* the input pin to a `HIGH` reading unless it is pulled `LOW` by external circuitry. This is how `INPUT_PULLUP` works and is described below in more detail.

When a pin is configured to OUTPUT with `pinMode()`, and set to `HIGH` with `digitalWrite()`, the pin is at:

- 5 volts (5V boards)
- 3.3 volts (3.3V boards)

In this state it can source current, e.g. light an LED that is connected through a series resistor to ground.

LOW

The meaning of `LOW` also has a different meaning depending on whether a pin is set to `INPUT` or `OUTPUT`. When a pin is configured as an `INPUT` with `pinMode()`, and read with `digitalRead()`, the Arduino (ATmega) will report LOW if:

- a voltage less than 1.5V is present at the pin (5V boards)
- a voltage less than 1.0V (Approx) is present at the pin (3.3V boards)

When a pin is configured to `OUTPUT` with `pinMode()`, and set to `LOW` with `digitalWrite()`, the pin is at 0 volts (both 5V and 3.3V boards). In this state it can sink current, e.g. light an LED that is connected through a series resistor to +5 volts (or +3.3 volts).

Defining Digital Pins modes: INPUT, INPUT_PULLUP, and OUTPUT

Digital pins can be used as `INPUT`, `INPUT_PULLUP`, or `OUTPUT`. Changing a pin with `pinMode()` changes the electrical behavior of the pin.

INPUT

Arduino (ATmega) pins configured as `INPUT` with `pinMode()` are said to be in a *high-impedance* state. Pins configured as `INPUT` make extremely small demands on the circuit that they are sampling, equivalent to a series resistor of 100 Megohms in front of the pin. This makes them useful for reading a sensor.

If you have your pin configured as an `INPUT`, and are reading a switch, when the switch is in the open state the input pin will be "floating", resulting in unpredictable results. In order to assure a proper reading when the switch is open, a pull-up or pull-down resistor must be used. The purpose of this resistor is to pull the pin to a known state when the switch is open. A 10 K ohm resistor is usually chosen, as it is a low enough value to reliably prevent a floating input, and at the same time a high enough value to not not draw too much current when the switch is closed. See the Digital Read Serial tutorial for more information.

If a pull-down resistor is used, the input pin will be `LOW` when the switch is open and `HIGH` when the switch is closed.

If a pull-up resistor is used, the input pin will be `HIGH` when the switch is open and `LOW` when the switch is closed.

INPUT_PULLUP

The ATmega microcontroller on the Arduino has internal pull-up resistors (resistors that connect to power internally) that you can access. If you prefer to use these instead of external pull-up resistors, you can use the `INPUT_PULLUP` argument in `pinMode()`.

See the Input Pullup Serial tutorial for an example of this in use.

Pins configured as inputs with either `INPUT` or `INPUT_PULLUP` can be damaged or destroyed if they are connected to voltages below ground (negative voltages) or above the positive power rail (5V or 3V).

OUTPUT

Pins configured as `OUTPUT` with `pinMode()` are said to be in a *low-impedance* state. This means that they can provide a substantial amount of current to other circuits. ATmega pins can source (provide current) or sink (absorb current) up to 40 mA (milliamps) of current to other devices/circuits. This makes them useful for powering LEDs because LEDs typically use less than 40 mA. Loads greater than 40 mA (e.g. motors) will require a transistor or other interface circuitry.

Pins configured as outputs can be damaged or destroyed if they are connected to either the ground or positive power rails.

LED_BUILTIN

Most Arduino boards have a pin connected to an on-board LED in series with a resistor. The constant `LED_BUILTIN` is the number of the pin to which the on-board LED is connected. Most boards have this LED connected to digital pin 13.