Arduino Coding Cheat Sheet v11

Variables

Constants

Predefined expressions used to make the programs easier to read. `HIGH | LOW` `true | false` `INPUT | OUTPUT` `INPUT_PULLUP` `LED_BUILTIN`

```
Suffix:  long  |  unsigned  |  float
         65536L |    -1U     |  3.0F
Exponential form        4.2e1
Prefix: octal, hexadecimal  0 | 0x or 0X
Character constants:    char | octal | hex
                        'a'  |'\ooo' |'\xhh'
newline | cr | tab | backspace
  \n    | \r | \t  |    \b
Special characters      \\, \?, \', \"
```

Conversion

```
byte(x)      char(x)      float(x)
int(x)       long(x)      word(h,l)
x: a value of any type
h: the high-order (leftmost) byte of word
l: the low-order (rightmost) byte of word
Casting:
```

Variable Scope & Qualifiers

```
const // make a variable "read-only"
scope // any variable declared outside of a
    function (e.g.setup(), loop(), etc.), is a
    global variable.
static // persistant variables visable to only
    one function
volatile // used in code associated with
    interrupts (ISR)
```

Utilities

```
PROGMEM // put my info into flash memory,
    instead of SRAM
sizeof(variable)  // The number of bytes in a
    variable, or bytes occupied in an array.
    size_t is the usual return type
```

Functions (ANSI)

```
return_type function_name( parameter list ) {
   function body
}  // return_type is 'void' if nothing is returned.
```

Functions

Digital I/O

```
digitalRead(pin)
digitalWrite(pin, value(HIGH|LOW))
pinMode(pin,mode) INPUT|OUTPUT|INPUT_PULLUP
```

Analog I/O

```
analogRead(pin)
analogWrite(pin, value)
  value: the duty cycle: between 0 (always off)
  and 255 (always on). Allowed data types: int
analogReference(type)  Configures the reference
voltage:  See doc for type
```

Zero, Due & MKR Family

```
analogReadResolution(bits) (1-12*)
analogWriteResolution(bits)
```

Advanced I/O

```
tone(pin, frequency_hZ, {duration_ms})
noTone(pin)
pulseIn(pin, value, {timeout})
pulseInLong(pin, value, {timeout})
  pin/value(HIGH|LOW)/ms-default 1sec)
shiftIn(dataPin, clockPin, bitOrder)
shiftOut(dataPin, clockPin, bitOrder)
  input/toggle/(MSBFIRST|LSBFIRST)
```

Math

```
abs(x)     max(x, y)  min(x, y)
sqrt(x)    sq(x)      pow(base, exponent)
constrain(x, minrange, maxrange)
map(val, fromL, fromH, toL, toH)
```

Trigonometry

```
sin(rad)   cos(rad)   tan(rad)
```

Random Numbers

```
random()  // Pseudo Random Numbers
randomSeed() // Init. PRN Generator
```

Time

```
delay(ms)
delayMicroseconds(us) // 16383 Max
micros() // both return runtime since
millis() //  program started running
// resolution  varies per board type
```

Characters

[myChar = variable (int only)]

```
isAlpha(myChar)      isAlphaNumeric(myChar)
isAscii(myChar)      isControl(myChar)
isDigit(myChar)      isGraph(myChar)
isLowerCase(myChar)  isUpperCase(myChar)
isPunct(myChar)      isSpace(myChar)
isPrintable(myChar)  isWhitespace(myChar)
isHexadecimalDigit(myChar)
```

Bits and Bytes

```
bit(n)          bitRead(x,n) bitWrite(x,n,b)
bitSet(x,n)     bitClear(x,n)
highByte(v)     lowByte(v)
  x: the numeric variable      v: value of any type
  n: which bit, starting at 0 for the least-significant
     (rightmost) bit 7 for the most-significant (leftmost)
  b: the value to write to the bit (0 or 1)
```

Communication

```
Serial
Stream
```

USB

```
Keyboard
Mouse
```

Interupts: `interrupts() noInterrupts()`

Interrupts are useful for making things happen automatically in microcontroller programs

```
void loop() {
  noInterrupts();
       // critical, time-sensitive code here
  interrupts();
       // other code here
}  // Example:insure that a program always catches
the pulses from a rotary encoder, so that it never
misses a pulse.
```

External Interrupts: (recommended options)

```
attachInterrupt(
 digitalPinToInterrupt(pin),ISR,mode);
detachInterrupt(digitalPinToInterupt(pin));
```

Structure

Pointer Access Operators

& reference operator - get a pointer

If x is a variable, then &x represents the address of the variable x

***** dereference operator - follow a pointer

If p is a pointer, then *p represents the value contained in the address pointed by p.

```
int *p;      // declare a pointer to an int data type
int i = 5;
int result = 0;
p = &i;      // now 'p' contains the address of 'i'
result = *p;
  // 'result' gets the value at the address
  //  pointed by 'p'
  //  i.e., it gets the value of 'i' which is 5
```

Pointers are one of them more compicated topics, but they are handy to have in one's toolkit.

Structure

Basic Program Structure

```
void setup() { }      // Runs once
void loop() { }       // Runs repeatedly
```

{} curly braces: define the beginning and end of function / statement blocks

; semicolon: end statements and separate elements (i.e. in a for loop).

Control Structures

```
if (condition) {...} else {...}
for (int i = 0; i < 10; i++) {...}
while (condition) {...}
do {...} while (condition);
switch (var) { //must be integral not float
  case 1:
    ..... ;  break;
  default:
    ..... ;
}
goto; label // AVOID!
continue; // Go to next iteration
return x; // or "return;" for voids
```

Further Syntax

```
#define   Give a name to a constant value before
          compilation. -> No PROGMEM required.
          Localizes "Magic Numbers".
#include  (include)
/* */     block comment
//        single line comment
```

Arithmetic Operators

```
+ add        - subtract     / divide
* multiply   = assignment   % modulo
```

Comparison Operators

```
==  (equal to)              !=  (not equal to)
<   (less than)             >   (greater than)
<=  (less than or equal to) >=  (greater than or equal to)
```

Boolean Operators

```
&&  and      ||  or      !  not
```

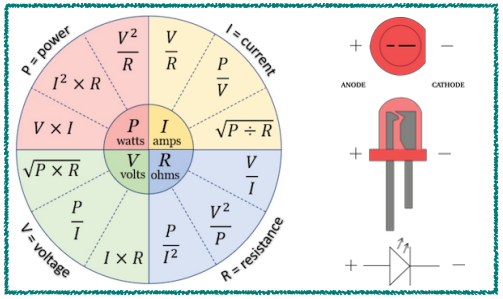
Bitwise Operators

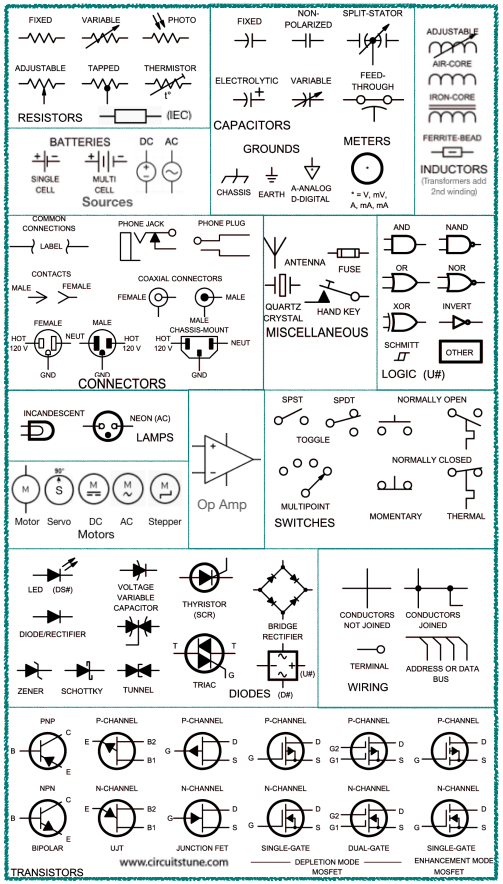
```
&  bitwise and    |  bitwise or    ^  bitwise xor
~  bitwise not    << shift left   >> shift right
```

Compound Operators

```
+=  addition       -=  subtraction    ++  increment
/=  division       *=  multiplication --  decrement
%=  remainder
^=  bitwise xor    &=  bitwise and    |=  bitwise or
```

Factor	10x	Prefix	Symbol
1,000,000,000	10^9	giga	G
1,000,000	10^6	mega	M
1,000	10^3	kilo	k
100	10^2	hecto	h
10	10^1	deka	da
0.1	10^{-1}	deci	d
0.01	10^{-2}	centi	c
0.001	10^{-3}	milli	m
0.000001	10^{-6}	micro	µ
0.000000001	10^{-9}	nano	n
0.000000000001	10^{-12}	pico	p



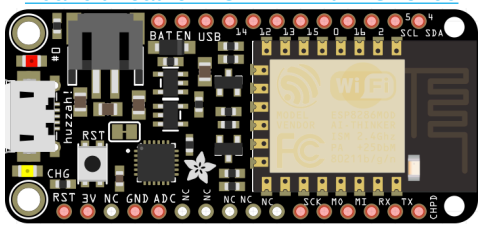


C Coding Style Guide

MUD: Here's How to write Clean Code

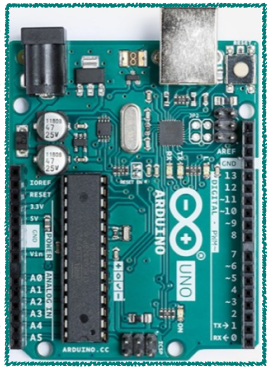
Whatever style you use, be Consistant!
KISS - Keep it Simple, Silly
Use Parenthesis, etc, even when you don't have to.
Use whitespace - use spaces and indentation, the compiler dosn't care. How to change default Tab behavior at the bottom of this discussion https://forum.arduino.cc/index.php?topic=111859.0
Comments - if you are explaning it to yourself, explanin it to everyone.
Meaningful identifiers.
(very subjective/ may change)
common/trivial variables - lower case
local variables camelCase
global variables - g_lower case (avoid)
constants and macros- All Caps
private functions - TitleCase
public functions - camel_Case with underscores
DRY - Don't repeat yourself! Use Functions/ encapsulate repeated tasks.
Functions should do precisely one thing only.
Explicit is better than implicit. Don't use language features that let you hide the obvious.
Limit line length to 80 Characters.
If the width of a int matters, use C99's portable fixed-width integer types. i.e. int8_t.
Initialize variables as soon as you know the initial value.
Avoide Magic Numbers. Declare constants using static const or enum. #define sometimes.
Use #IF Make the compiler help you enforce limits.
Minimize Global Variables.

Adafruit Feather HUZZAH with ESP8266



C Operator Precedence

Lvl	Details
1	() []→.(dot)
2	! ~ ++ -- +(unary) -(unary) *(deref) &(address of) (cast) sizeof
3	*(multiplication) / %
4	+ (binary) - (binary)
5	<< >>
6	< <= > >=
7	== !=
8	& (bitwise AND)
9	^
10	|
11	&&
12	||
13	?:
14	= += -= *= /= %= &= ^= |= <<= >>=
15	, (comma)



Thanks to: arduino.cc, Sicuramente, Nick Parlante: Essential C / Pointers and Memory, Gavin Smith, Mark Liffiton, et al. Design, Raph Levien: Inconsolata font
Compiled by Paul Ciarelli
Adafruit, Ever helpful!
Leeward Business Advisors

Coding Tips

`F()` Macro: stores strings in flash instead of SRAM
ex: `Serial.println(F("I'm stored in flash"));`

C ternary operator **?:** Make code more readable:
The expression `x ? y : z` evaluates to `y` if the value of `x` is true, otherwise to `z`

Arrays [Data Type] // INDEX STARTS AT 0

```
int myInts[6];
    // not initialized (OK)
int myPins[] = {2, 4, 8, 3, 6};
    // (auto sized by compiler)
int mySensVals[6] = {2, 4, -8, 3, 2};
    // both inialized and sized
char message[6] = "hello";
    // one more element than length of string is required
mySensVals[0] = 10; // Assignment
x = mySensVals[4];  // Retrieval

Array in for loop:
int i;
for (i = 0;   i < 5;   i = i + 1) {
  Serial.println(myPins[i]);
  }
```

Arduino Uno R3 (From: Components 101)

Pin Category	Details
Power	Vin: Input voltage using an external power source. 5V: Regulated power supply used to power microcontroller and other components 3.3V: 3.3V supply generated by on-board voltage regulator. Maximum current draw is 50mA. GND: ground pins.
Reset	Resets the Microcontroller
Analog Pins	A0 - A5 - Provide analog input in the range of 0-5V
I/O Pins	Digital Pins 0 - 13 - input or output pins.
Serial	0(Rx), 1(Tx) Receive and transmit TTL serial Data
Ext. Interupts	2.3 To trigger an interrupt.
PWM	3, 5, 6, 9, 11 - Provides 8-bit PWM output.
SPI	10 (SS), 11 (MOSI), 12 (MISO), 13 (SCK) - SPI comm.
Inbuilt LED	13 - To turn on the inbuilt LED.
TWI	A4 (SDA), A5 (SCA) - Used for TWI communication.
AREF	AREF - Provide reference voltage for input voltage.

32KB Flash(.5BL) 2KB SRAM, 1KB EEPROM at 16MHz
Operating Voltage 5V
Input Voltage Limits 6-20V / **7-12 Recommended**
DC Current on I/O Pins 40mA DC Current on 3.3V Pins 30Ma

Standard Libraries

EEPROM - reading and writing to "permanent" storage
Ethernet - for connecting to the internet using the Arduino Ethernet Shield, Arduino Ethernet Shield 2 and Arduino Leonardo ETH
Firmata - for communicating with applications on the computer using a standard serial protocol.
GSM - for connecting to a GSM/GRPS network with the GSM shield.
LiquidCrystal - for controlling liquid crystal displays (LCDs)
SD - for reading and writing SD cards
Servo - for controlling servo motors
SPI - for communicating with devices using the Serial Peripheral Interface (SPI) Bus
SoftwareSerial - for serial communication on any digital pins. Version 1.0 and later of Arduino incorporate Mikal Hart's NewSoftSerial library as SoftwareSerial.
Stepper - for controlling stepper motors
TFT - for drawing text , images, and shapes on the Arduino TFT screen
WiFi - for connecting to the internet using the Arduino WiFi shield
Wire - Two Wire Interface (TWI/I2C) for sending and receiving data over a net of devices or sensors.

Library Lists

https://www.arduinolibraries.info/libraries
https://www.arduinolibraries.info/libraries
https://circuitpython.readthedocs.io/en/3.x/docs/drivers.html

Arduino Data Types

Data Types	B	Min Size	Max Size
char	1	-128	127
unsigned char	1	0	255
bool	1	true|false	1|0
boolean[1]		Non standard. Just use bool	
byte	1	0	255
int (UNO)	2	-32768	32767
unsigned int	2	0	65535
short[1]	2	-32768	32767
long	4	-2147483648	2147483647
unsigned long	4	0	4294967295
float	4	-3.4028E+38	3.4028E+38
double[1]	4	Implemented the same as float 6-7 digits of precision only!	
string			
String()[1]		C++ Strings: more functionality at the cost of more memory	
array			
size_t		Capable of representing the size of any object in bytes	
void			

1 - not typically used in A-IDE

C99's Portable Fixed Width Interger Types
Gives integer types architecture-independent lengths.

int8_t	1	-128	127
uint8_t	1	0	255
int16_t	2	-32768	32767
uint16_t	2	0	65535
int32_t	4	-2147483648	2147483647
uint32_t	4	0	4294967295