ARDUINO Code Language Basics

Code Language Basics for the ARDUINO Uno Processor

Note: This sheet reviews the basic C/C++ language used in Arduino IDE Platforms.
For additional information please see: http://www.arduino.cc/en/Reference

ARDUINO CODE STRUCTURE AND SYNTAX

ABOUT THE SKETCHES:
The Arduino UNO R3 is a microcontroller unit based on the 8-Bit RISC Atmel ATmega328P AVR that runs programs called "sketches". These text files consist of instructions – often called code – that the processor understands. They also typically also contain "comments" that explain what the code does. Comments and code will have different colors in the editor so you can tell them apart.
Please note that the IDE's Compiler is Case-Sensitive. Thus, **inPIN** and **inPin** are not the same constant.

COMMENTS:
Syntax: **/*** … ***/** (used for multi-line comments)

```
EXAMPLE:
/* This is a long comment – typically will be multi-line, it
must start and end with these characters. It is ignored when compiled */
```

Syntax: *//* … (used for single line comments)

```
EXAMPLE:
// This is a one line comment - anything on a line after "//" is ignored
// when compiled.
```

FUNCTION SEPARATORS: The **;** (semicolon)
The semicolon "**;**" is used after each declaration or step to let the microcontroller know there may be additional 'commands.'

CODE BLOCK SEPARATORS: The **{ }** (curly braces)
Curly braces are used to indicate to the microcontroller that there is a block of functions, etc., that follows. The opening brace "**{**" precedes these commands and the closing brace "**}**" must end them.

Defining Constants: #define constantName value
The **#define** is used to define one or more constant values. The compiler will substitute the value for the constant during compilation of sketch.
***DO NOT** use semicolons after or the "**=**" sign inside of a **#define** statement.*
Syntax: **#define constantName value**

```
EXAMPLE:    #define ledPin 3
// The compiler will replace any mention of ledPin in the entire sketch
with the value 3 at compile time.
```

Including Code Libraries: #include <avr/pgmspace.h>
The **#include** is used to include outside libraries (written in C programming language). A listing of some Arduino Libraries is at: **http://www.nongnu.org/avr-libc/user-manual/modules.html**
Like #define, above, we ***DO NOT** use semicolons after or the "**=**" sign inside of an **#include** statement.*
Syntax: **#include <avr/pgmspace.h>**

```
EXAMPLE: #include <library/pgmName.h>
// The compiler will retrieve the C program pgmName.h from the library
directory
```

Declaring a Function: The **void** command
The void keyword is used only in function declarations. It indicates that the function is expected to return no information to the function from which it was called.

ARDUINO Code Language Basics

THE STRUCTURE OF A SKETCH
All Arduino sketches ***MUST*** have two specific **functions**, named "**setup()**" and "**loop()**".
The Arduino runs these functions automatically when it starts up or if you press the reset button. A "function" is a 'named' block of code or "shell" that performs a specific function (often called a subroutine in other programming environments). Many useful functions are already built into the Arduino, see below, others you can name and write yourself for your own purposes.

THE SETUP FUNCTION: setup()
The **setup()** function runs once and only once when the sketch starts. Use it to initialize variables, pin modes, start using libraries, etc. The setup function will only run once, after each power up or reset of the Arduino board. The **void** command notes that there is no value that is expected as a result of completing the function. *NOTHING goes into the parenthesis of the setup function.* A curly brace "**{**" after the code **void setup ()** shows that we are including one or more items in the function. Once all the code has been included we use the closing brace "**}**" to note that **setup()** is concluded.

THE LOOP FUNCTION: loop()
After creating a **setup()** function which initializes and sets the program's initial values, the **loop()** function does precisely what its name suggests: it loops consecutively, allowing your program to change and respond. This is where the bulk of your program goes and where you actively control the Arduino board. Use loops to create shells for the actions you want to take during the program. *NOTHING goes into the parenthesis of the loop function.* A curly brace "**{**" after the code **void loop ()** shows that we are including one or more items in the function. Once all the code has been included we use the closing brace "**}**" to note that **loop()** is concluded. There can be several **loop()** functions and even other names nested inside of each other, for example to delay an action. As noted already, once all the loops are completed, the program returns to the **{** after **void loop ()** until it is turned off or the **reset** button is pressed.

Declaring DIGITAL I/O pins**: pinMode()**
Defining one of the Arduino's 13 digital pins as either input or output is done inside of the **setup()** function. Pins can be configured as either inputs or outputs. We define their usage with the built-in **pinMode()** function. The **pinMode()** function takes two values, which you type into the parenthesis after the function name. The first value is a pin number and the second value is the word **INPUT** or **OUTPUT**. After the function we use the semicolon "**;**" to indicate that we may have additional functions to declare.

```
EXAMPLE: pinMode(13, OUTPUT);
*/ Sets Digital I/O pin 13 (the one connected to the onboard "L" LED) to be an OUTPUT such that
we can send +5 volts "out" of the Arduino to the LED via that pin. /*
```

USING THE DIGITAL WRITE FUNCTION: digitalWrite()
Any of the Arduino's 13 digital pins can be used as an output set to **LOW** (zero volts) or **HIGH** (+5 volts). This is normally done inside a **loop()** function. We define their digital output level with the built-in **digitalWrite()** function which takes two values: the first value is a pin number and the second value is the word **HIGH** or **LOW** to indicate the logic state of that pin. After the function we use the semicolon "**;**" to indicate that we may have additional functions to declare.

```
EXAMPLE: digitalWrite(13, HIGH);
*/ Sets Digital I/O pin 13 (the one connected to the onboard "L" LED) to be HIGH turning the "L"
LED ON and sending +5 volts "out" of the Arduino via that pin. /*
```

Should we then at some later time set the pin 13 output to **LOW**, the *"L" LED* will turn OFF and there will be zero volts at pin 13.

Creating time DELAYs: delay()
The **delay()** function is an automatic built-in timer that pauses for a given amount of time. It takes one value, *the amount of time to wait*, measured in milliseconds. As there are 1000 milliseconds in a second, if you **delay(1000)**, it will pause the sketch loop for exactly one second:

```
EXAMPLE: delay(1000);
// Sets the delay to 1000 milliseconds = 1 second
```

ARDUINO Code Language Basics

Notes:

We have reviewed the following structural elements of a sketch in this lesson:
The use of comments (*//* and */* … */*), the need for the "**;**" after most commands and for the curly braces "**{**" and "**}**" to enclose each function; the **void** command, and **setup()** and **loop()** functions,.

We saw how to set up a digital I/O pin using **pinMode()**, how to write +5 or 0 volts to the pin using **digitalWrite()** and how to include some time delays between statements using the **delay()** statement.

Putting these all together we can write a sketch that turns the onboard "**L**" LED (***and Digital Pin 13***) ON and OFF with one second of delay between state changes:

```
EXAMPLE: Complete BLINK Sketch

// Sketch to Blink L LED ON/OFF once per second

void setup()          {
  pinMode(13, OUTPUT);      // "L" LED/Pin 13 set for digital output
   }

void loop()           {
  digitalWrite(13, HIGH);         // Turn on the "L" LED
  delay(1000);                          // Wait for one second
  digitalWrite(13, LOW);          // Turn off the "L" LED
  delay(1000);                          // Wait for one second
   }                          // REPEAT! FOREVER!

*/ END */                    // Not actually required, but good format!
```

See a listing of Arduino Language Structure, Variables and Functions on the following page…

'ARDUINO LANGUAGE' Command Reference

Arduino built-in programing elements can be divided in three main parts:
Structure, Values (variables and constants), and Functions.

• STRUCTURE

Comments: *// (Single comment line)* and */ … /* (Multiple Comment lines)*
Basic Structure Functions: setup() loop()
Control Structures: if if...else for switch case while do... while
break continue return goto
Arithmetic Operators: = (assignment operator) + (addition) - (subtraction)
* (multiplication) / (division) % (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)
Pointer Access Operators: * dereference operator & reference operator
Bitwise Operators: & (bitwise and) | (bitwise or) ^ (bitwise xor)
~ (bitwise not) << (bitshift left) >> (bitshift right)
Compound Operators: ++ (increment) -- (decrement) += (compound addition)
-= (compound subtraction) *= (compound multiplication) /= (compound division)
&= (compound bitwise and) |= (compound bitwise or)

• VALUES (VARIABLES and CONSTANTS)

Constants: HIGH | LOW INPUT | OUTPUT | INPUT_PULLUP LED_BUILTIN
true | false integer constants floating point constants
Data Types: void boolean char unsigned char byte int unsigned int word
long unsigned long short float double string-char array String-object array
Conversion: char() byte() int() word() long() float()
Variable Scope & Qualifiers: variable scope static volatile const
Utilities: sizeof() PROGMEM

• FUNCTIONS

Digital I/O: pinMode() digitalWrite() digitalRead()
Analog I/O: analogReference() analogRead() analogWrite() - PWM
Due & Zero only: analogReadResolution() analogWriteResolution()
Advanced I/O: tone() noTone() shiftOut() shiftIn() pulseIn()
Time: millis() micros() delay() delayMicroseconds()
Math: min() max() abs() constrain() map() pow() sqrt()
Trigonometry: sin() cos() tan()
Random Numbers: randomSeed() random()
Bits and Bytes: lowByte() highByte() bitRead() bitWrite() bitSet() bitClear() bit()
External Interrupts: attachInterrupt() detachInterrupt()
Interrupts: interrupts() noInterrupts()

For additional information please see:
http://www.arduino.cc/en/Reference

ARDUINO CHEAT SHEET

For more information visit: http://arduino.cc/en/Reference/

Structure

```
/* Each Arduino sketch must contain the
following two functions. */
void setup()
{
/* this code runs once at the beginning of
the code execution. */
}

void loop()
{
/* this code runs repeatedly over and over
as long as the board is powered. */
}
```

Comments

```
// this is a single line
/* this is
a multiline */
```

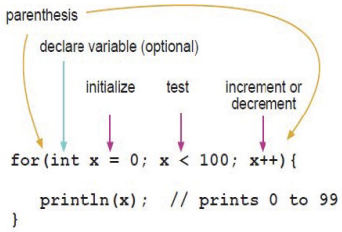
Setup

```
pinMode(pin, [INPUT \ OUTPUT \ INPUT_PUL-
LUP]);
/* Sets the mode of the digital I/O pin.
It can be set as an input, output, or an
input with an internal pull-up resistor.
*/
```

Control Structures

```
if(condition)
{
// if condition is TRUE, do something here
}
else
{
// otherwise, do this
}

for(initialization; condition; increment)
{
// do this
}
/* The 'for' statement is used to repeat
a block of statements enclosed in curly
braces. An increment counter is usually
used to increment and terminate the loop.
*/
```



Digital I/O

```
digitalWrite(pin, val);
/* val = HIGH or LOW write a HIGH or a LOW
value to a digital pin. */
int var = digitalRead(pin);
/* Reads the value from a specified digital
pin, either HIGH or LOW. */
```

Analog I/O

```
analogWrite(pin, val);
/* Writes an analog value to a pin.
val = integer value from 0 to 255 */
int var = analogRead(pin);
/* Reads the value from the specified
analog pin. */
```

Advanced I/O

```
tone(pin, freq);
/* Generates a square wave of the specified
frequency to a pin. Pin must be one of the
PWM (~) pins. */
tone(pin, freq, duration);
/* Generates a square wave of the specified
frequency to a pin for a duration in
milliseconds. Pin must be one of the PWM (~)
pins. */
noTone(pin);
// Turns off the tone on the pin.
```

Time

```
delay(time_ms);
/* Pauses the program for the amount of time
(in milliseconds). */
delayMicroseconds(time_us);
/* Pauses the program for the amount of time
(in microseconds). */
millis();
/* Returns the number of milliseconds since
the board began running the current program.
max: 4,294,967,295 */
micros();
/* Returns the number of microseconds since
the board began running the current program.
max: 4,294,967,295 */
```

Data Types

```
void     // nothing is returned
boolean  // 0, 1, false, true
char     // 8 bits: ASCII character
byte     // 8 bits: 0 to 255, unsigned
int      // 16 bits: 32,768 to 32,767, signed
long     /* 32 bits: 2,147,483,648
            to 2,147,483,647, signed */
float    // 32 bits, signed decimal
```

Constants

```
HIGH \ LOW
INPUT \ OUTPUT
true \ false
```

Mathematical Operators

```
=  // assignment
+  // addition
-  // subtraction
*  // multiplication
/  // division
%  // modulus
```

Logical Operators

```
==  // boolean equal to
!=  // not equal to
<   // less than
>   // greater than
<=  // less than or equal to
>=  // greater than or equal to
&&  // Boolean AND
||  // Boolean OR
!   // Boolean NOT
```

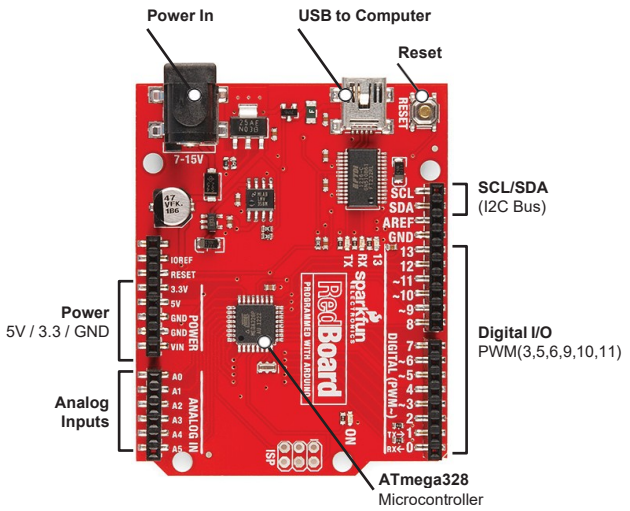
Bitwise Operators

```
&   // bitwise AND
|   // bitwise OR
^   // bitwise XOR
~   // bitwise INVERT
var << n    // bitwise shift left by n bits
var >> n    // bitwise shift right by n bits
```

Libraries

```
#include <libraryname.h>
/* this provides access to special
additional functions for things such as
servo motors, SD card, wifi, or bluetooth.
*/
```

RedBoard:



LilyPad ProtoSnap Simple: