TECHNOVUS

Build-It-Yourself Shops Document : Module 1

Intended Use & Scope

This document is developed by Technovus as an aid for members attending the workshops. The scope and context of the topics covered here is limited to embedded programming (C/C++), especially programming Arduino microcontrollers. The Build-It-Yourself Shops consist of six modules. Each module ends with a building exercise which will provide a way to practically use the information provided in the module.

Module 1 is designed to quickly introduce programming to beginners. We will also take a look at the Serial command available in the Arduino IDE. Module 2 covers reading and writing digital data. Module 3 progresses to reading and writing analog data along with working with interrupts. Module 4 will introduce servo-motor control using Pulse Width Modulation (PWM). Module 5 and 6 will allow members to put their newly acquired knowledge to test by building a mini-project that encompasses topics covered in previous modules.

The ultimate objective of this document and the Build-It-Yourself Shops is to get members equipped for better and bigger projects.

Contents

Module 1: Functions, loops, and statements

Arduino Board (UNO)

Arduino is the developer and official manufacturer for the microcontroller board used in this workshop. In simple terms, a microcontroller is a combination of a processing core, memory, and programmable I/O (input/output) peripherals. The I/O peripherals allow users to control external peripherals (such as motors, LEDs etc.) and acquire data from sensors (such as pushbuttons, switches etc.). In all modules of this workshop, we will be using the Arduino Uno microcontroller board. The Uno is only one of the microcontrollers produced by Arduino; the following web link provides more information about different Arduino microcontrollers: https://www.arduino.cc/en/Main/Products.

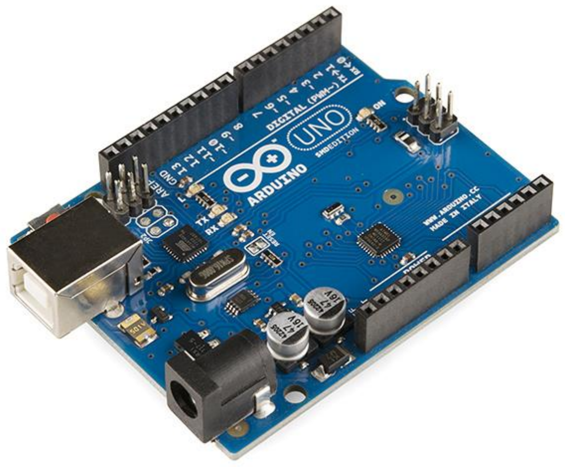
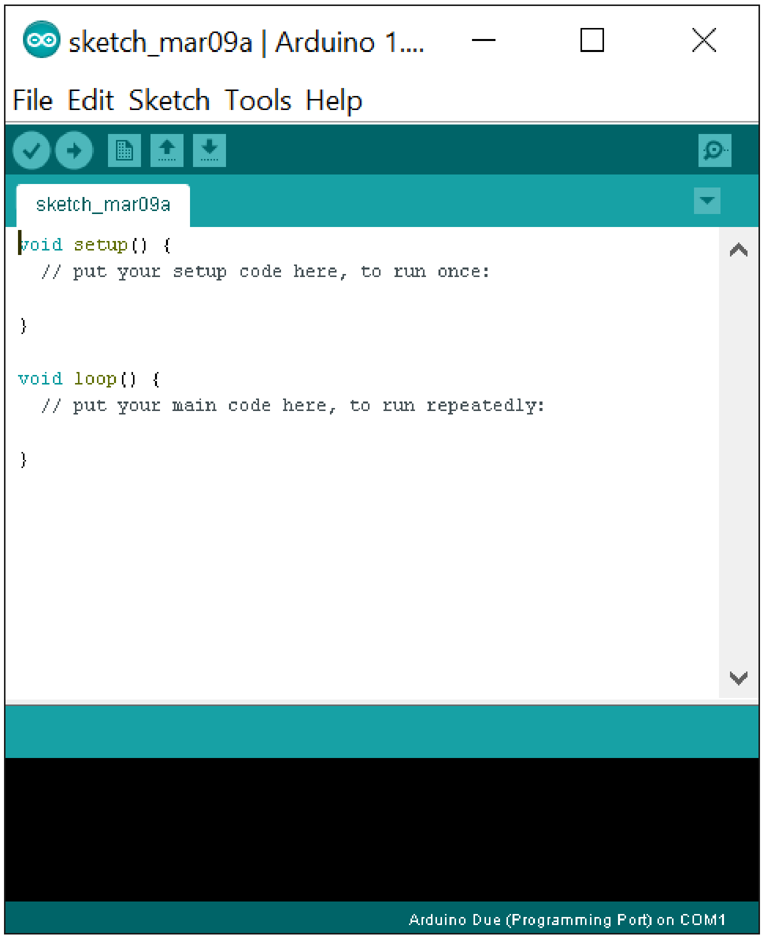


Figure 1: Arduino Uno Microcontroller

Arduino IDE

Arduino IDE (Integrated Development Environment) is a software tool supplied by Arduino that is used to program the Arduino microcontroller board. The IDE is simply a piece of software that we will be using to write code for programming the microcontroller in all workshop modules. The figure below shows a screenshot depicting what the IDE looks like.



```
void setup() {
  // put your setup code here, to run once:

}

void loop() {
  // put your main code here, to run repeatedly:

}
```

Arduino Due (Programming Port) on COM1

Figure 2: The Arduino IDE

We can write, compile, and upload code to the microcontroller using the IDE. Before uploading code to any board, first we should check that the IDE is properly setup for uploading code to the correct board through the correct port number. More on the IDE will be discussed during the workshop.

Download (Windows/Mac OS/ Linux): https://www.arduino.cc/en/Main/Software

Loops

We will limit the discussion of loops here to the two most commonly used types of loops:

While Loop

```
while(loop condition){

      //code within the while loop

}
```

A while loop evaluates the loop condition and executes the code within the loop as long as the loop condition is true. The loop condition is checked before each iterated execution of the loop code.

Example

Here is a piece of code that will run the code within the while loop 5 times.

```
int value = 0;

While(value < 5){

      // do something

      value++;

}
```

For Loop

```
for (initialization; condition; instruction) {

      //code within the for loop

}
```

A for loop requires a variable initialized to some value, a condition for that variable, and an instruction (typically increment/decrement of the value of that variable).

The initialization happens once. Before each iteration of the loop, the condition is tested, if it is true, then the code within the loop runs, and the instruction is applied (for example, variable is incremented), thus completing one iteration. If the condition is false, the loop ends.

Example

Here is a piece of code that will run the code within the for loop 5 times.

```
for (int x = 0; x < 5; x++) {

      //do something }
```

If-Else

```
if(condition){

	//code to be run if condition is true

}

else{

	//code to be run if condition is not true

}
```

The ‘if’ statement runs a piece of code if the condition is true. The ‘else’ statement will run the code enclosed within its curly braces if the ‘if’ statement before it is not true.

Example

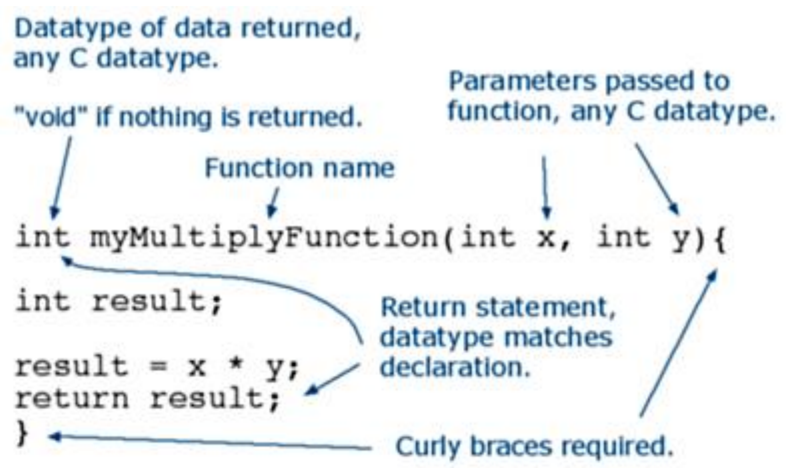
Here is a piece of code that will run the code enclosed by the ‘if’ statement if the value is negative. The code enclosed by the ‘else’ statement will run if the ‘if’ statement is false, i.e. the value is either 0 or positive.

```
if(value < 0){

	//do something

}

else{

	//do something else

}
```

Functions

Functions are typically used for performing repetitive tasks in a program. Such repetitive tasks can be defined as functions, this helps keep the main program short in terms of number of lines, and also makes the code modular.

Anatomy of a C function:


Datatype of data returned, any C datatype.
"void" if nothing is returned.
Function name
Parameters passed to function, any C datatype.
Return statement, datatype matches declaration.
Curly braces required.

```
int myMultiplyFunction(int x, int y){
int result;
result = x * y;
return result;
}
```

[1]

An Arduino sketch contains two required functions, setup() and loop(). The setup() function allows us to initialize pin modes, libraries etc. It is important to note that setup() function only runs once. As the name suggests, the loop() function loops infinitely, allowing us to control the Arduino board in real-time.

In the next sections, we take a look at the serial.println(), pinMode(), and digitalWrite() functions.

Print

The print statement provides a convenient way to print a value or string of text on the screen. This can be particularly useful for debugging. In this Module, we will take a quick look at the 'Serial.println' function.

```
Serial.println(val);
```

The above statement will print on the serial monitor the value of variable 'val' in decimal. We can also print text as shown below.

```
Serial.println("Hello World!");
```

Note: Print is tied to the serial communication in microcontrollers, which will be covered in depth in more advanced modules. For now, we use print for displaying text or variables on the serial monitor of the Arduino IDE.

[1] Source: https://www.arduino.cc/en/Reference/FunctionDeclaration

Example

The following piece of code will print “Hello World!” on the serial monitor 3 times.

```
for (int x = 0; x < 3; x++) {

      Serial.println(“Hello World!”);

}
```

Try using the ‘Serial.println’ with a while loop or an ‘if’ statement.

pinMode

```
pinMode(Pin number, Mode);// Mode can be INPUT or OUTPUT
```

The pinMode() function is typically used within setup() to configure a specified pin as an input pin or output pin. Configuring a pin as an input pin allows us to use that pin to acquire data from a sensor, whereas configuring a pin as an output pin allows us to use that pin to control actuators (such as LEDs and motors).

Digital Write

```
digitalWrite(pin number, value);
```

We can write a value (HIGH or LOW) to a pin using the digitalWrite function. A value of HIGH turns the pin on (pin outputs a 5V amplitude digital signal) and LOW corresponds to off.

For this function to work, we first need to set the mode for the pin number as output using the pinMode function within setup().

```
pinMode(pin number, OUTPUT);
```

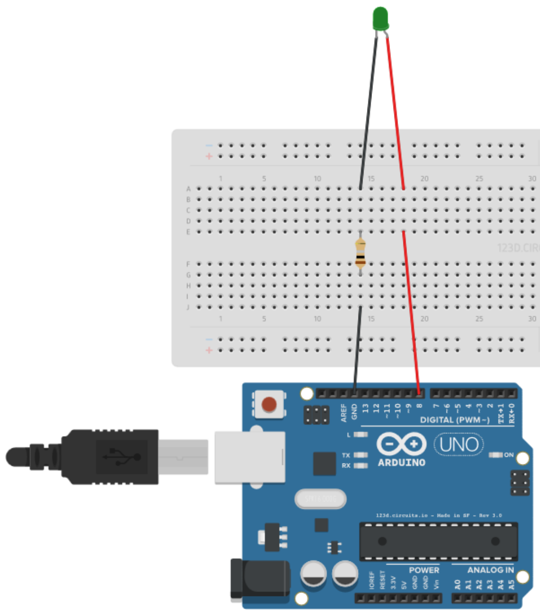
Build!

This building exercise involves making an LED blink.

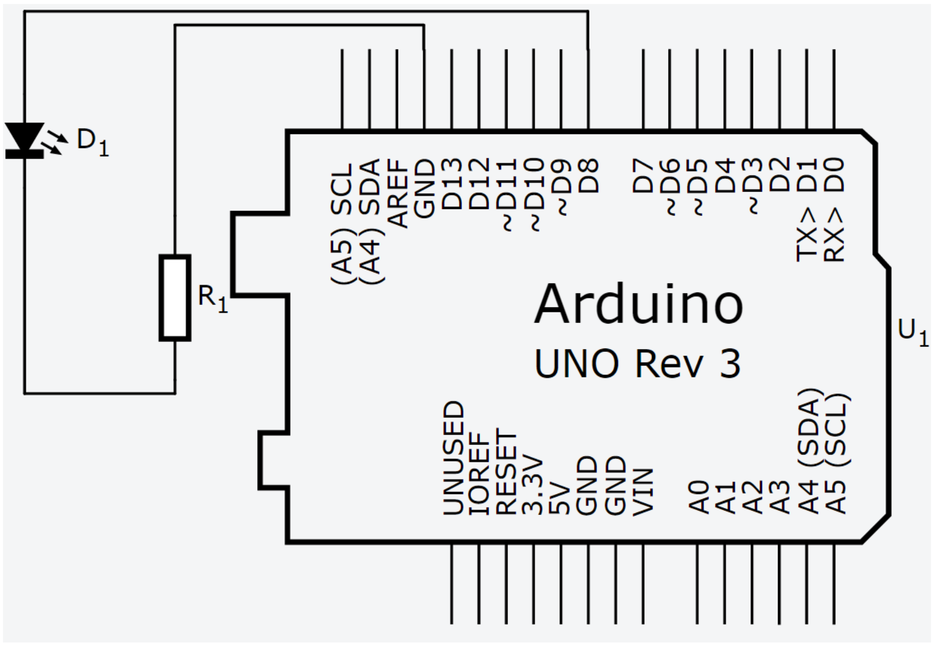
Items required:

1. Microcontroller
2. LED
3. 220 ohm Resistor

Circuit



Schematic



Code

```
const int waitTime = 1;     // Delay time for LED [seconds]

const int LEDPin = 8;       // Pin number for turning LED on or off


// the setup function runs once when you press reset or power the
board
void setup() {
  pinMode(LEDPin, OUTPUT); // initialize LEDPin as an output.
}

// the loop function runs over and over again forever
void loop() {
  digitalWrite(LEDPin, HIGH);    // turn the LED on
  delay(waitTime*1000);                        // wait
  digitalWrite(LEDPin, LOW);     // turn the LED off
  delay(waitTime*1000);                        // wait
}
```

Good to know

Breadboard Basics

A breadboard is useful tool for quickly making electronic circuits. In order to understand how to use a breadboard, we need to get familiar with the layout of the breadboard.

On the left side of the figure below, we can see what a breadboard looks like when no wires are plugged into it. We can see two continuous strips of holes on each long end of the breadboard. These strips are known as the bus strips (shown in red on the right side of figure), and the holes in each strip are connected to each other as shown on the right side of the figure. The red line going over the holes indicates that these holes are connected internally; therefore, if a wire is connected to one end of a strip and another wire is connected to the other end of the same bus strip, the two wires will be connected to each other through the breadboard.

In the middle of the breadboard, we can see strips of holes running along the short side of the breadboard; these are known as the terminal strips. Again, we can see how the terminal strips are connected internally in the right side of the figure by looking at the green lines.

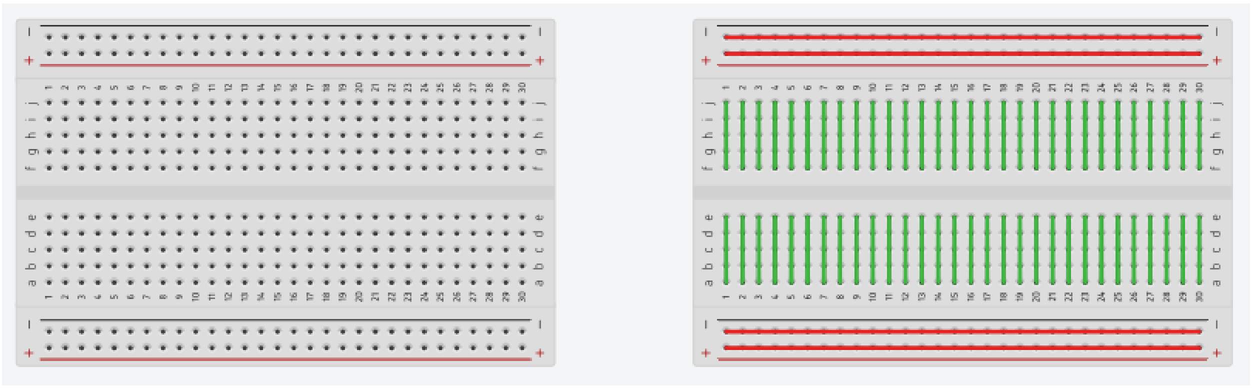
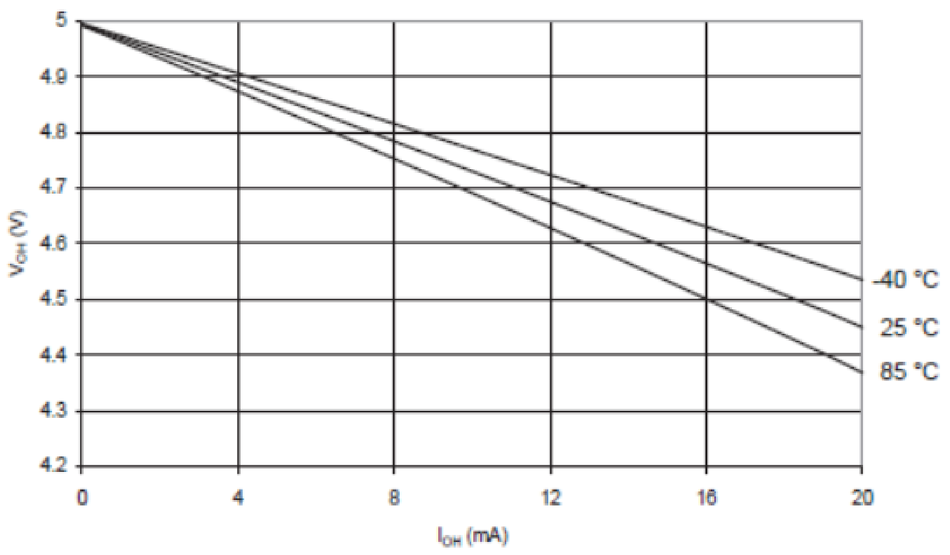


Figure 3: LEFT: A small breadboard; RIGHT: Green lines indicating internal connections of a breadboard.

Why do I need a resistor in the circuit? Why not just an LED?

The output pins on Arduino microcontrollers can only output a max current of 40 mA safely, if something draws more current than that, the microcontroller may suffer damage. The resistance of an LED is so small, it's almost negligible. The output pins typically supply a voltage of 5V. However, as the current drawn increases, the output voltage decreases.
The graph below from the microcontroller's datasheet (ATmega) can be used to approximate the internal resistance.



We can see that the voltage drops from 5V to ~4.45V from no load (0 mA) to 20mA. Using Ohm's Law, we can approximate the internal resistance to be 27.5 ohm. The voltage drop across an LED is typically 2V. Knowing that the resistance of an LED is negligible, we can apply ohm's law to the complete circuit as follows to find the current

$$I = \frac{5V - 2V}{27.5\Omega} = 109mA$$

We can see that this current is much higher than the max rated current of 40mA, therefore it is not safe. Having a resistor in the circuit increases the total resistance of the circuit, thereby limiting the current. This solution to limiting the current is convenient, although not perfect because the added resistor reduces the energy efficiency of the circuit. Current limiting ICs are a standard solution used in commercial products with LEDs in them.