Embedded Systems for Non Electrical Engineers

Suranaree University of Technology

Workshop on March 11-12, 2017

Dusadee Treeumnuk

Outline

- Day 1
  - Morning
    - Introduction to Embedded Systems
    - Introduction to Arduino & Review of C language
    - Getting started
    - Exercises
  - Afternoon
    - GPIOs
    - Switches & Interrupts
    - Analog Inputs & PWM
    - Library, Timer, Counter
    - Devices

Outline

- Day 2
  - Morning
    - Ring buffer
    - Multitasking, State Machines, Scheduling
    - Projects
  - Afternoon
    - Projects
    - DIY project

Workshop files can be found from here:

https://www.dropbox.com/sh/3oeoryk88otp37i/AABOIpljD7Wv-vptty7YyIN7a?dl=0

Day 1 - Introduction

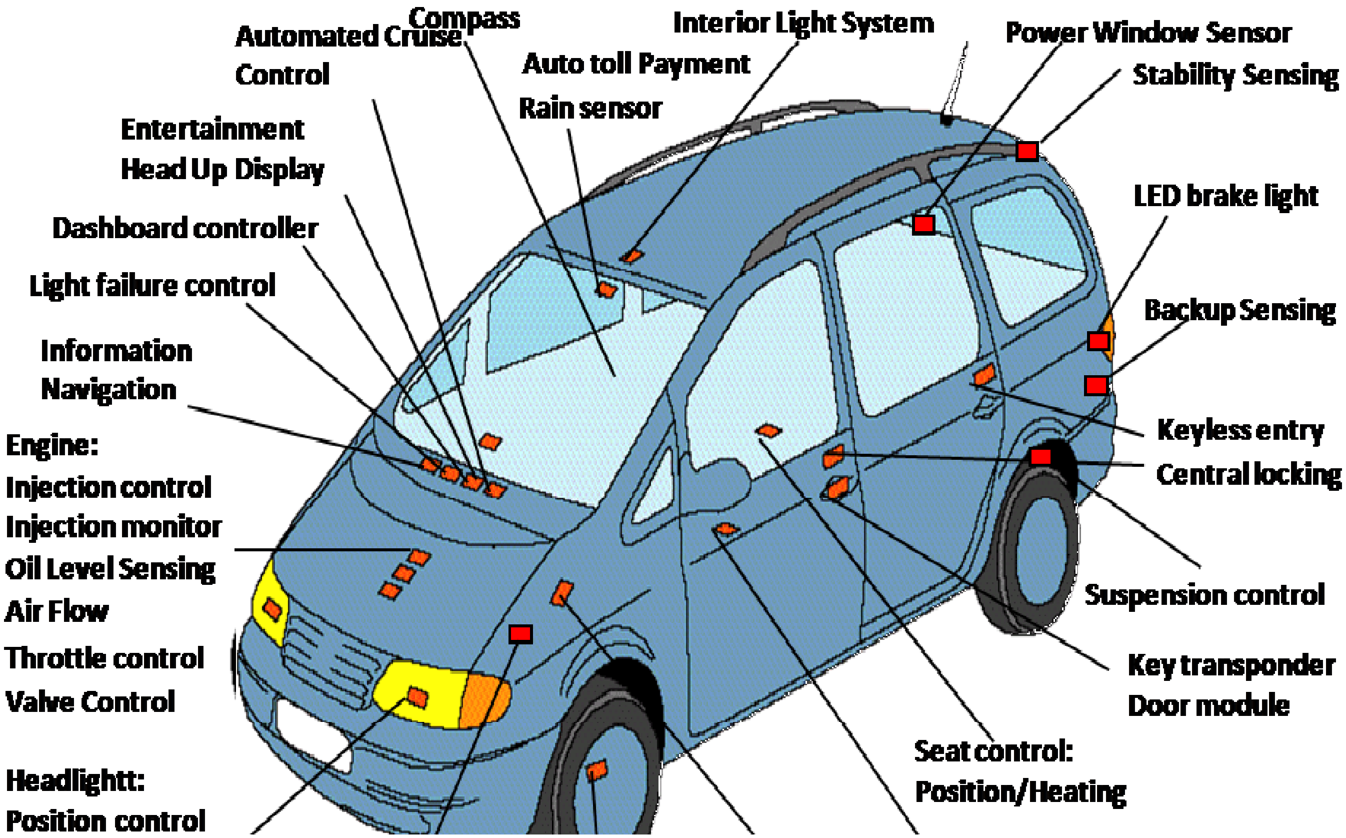
- What is an embedded system?

An embedded system can be broadly defined as a device that:

- contains tightly coupled hardware and software components to perform a single function.
- forms part of a larger system.
- is not intended to be independently programmable by the user.
- is expected to work with minimal or no human interaction.

An embedded system is one that has a dedicated purpose software embedded in a computer hardware.

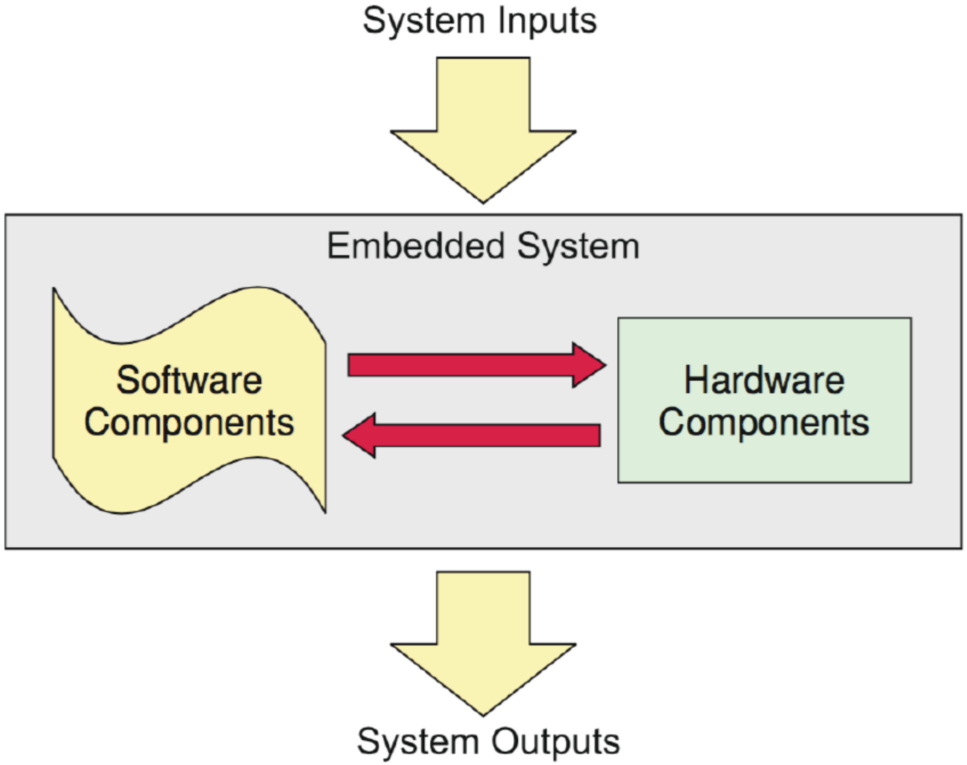
Embedded Systems



http://www.theengineeringprojects.com/wp-content/uploads/2016/11/automotive.png

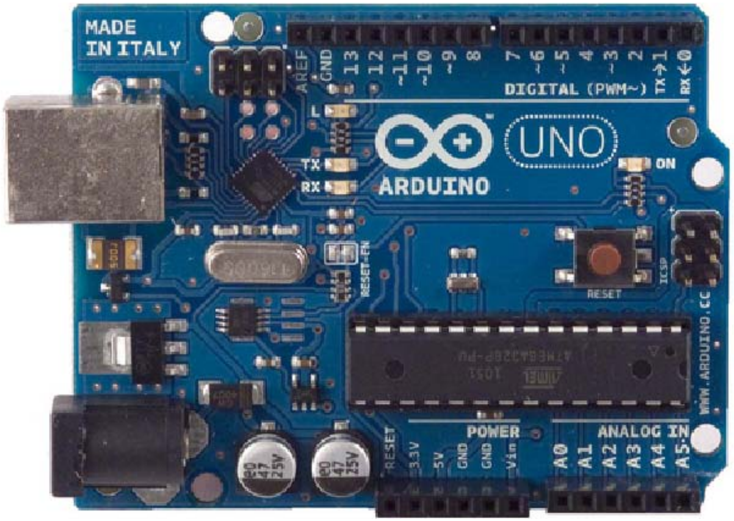
Embedded Systems

General View of an Embedded System



Arduino

Arduino is a single board microcontroller to make using electronics in multidisciplinary projects more accessible.

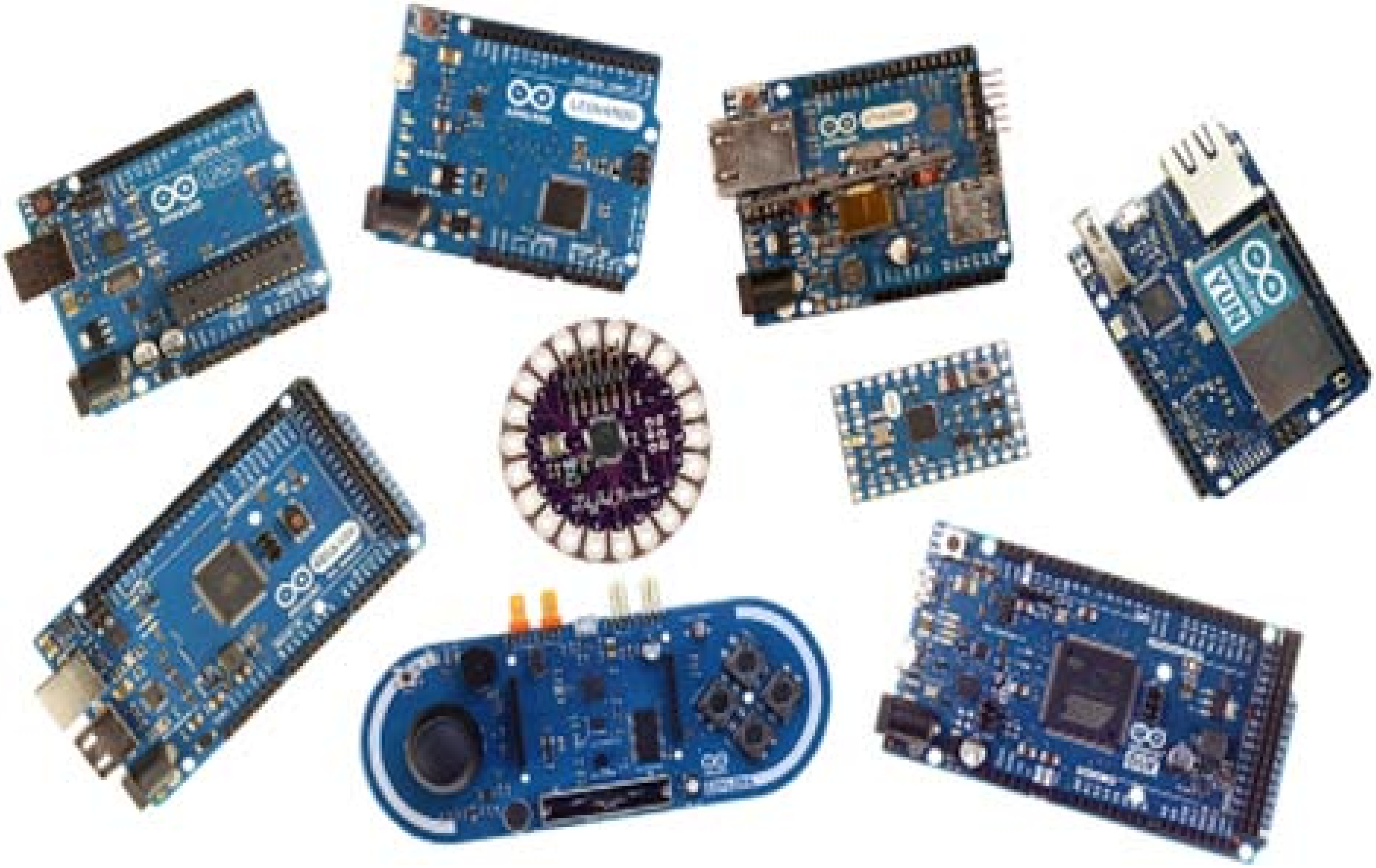


The hardware consists of a simple open source hardware board designed around an 8-bit Atmel AVR microcontroller, or a 32-bit Atmel ARM.

The software consists of a standard programming language compiler and a boot loader that executes on the microcontroller.

More products: https://www.arduino.cc/en/Main/Products

Arduino

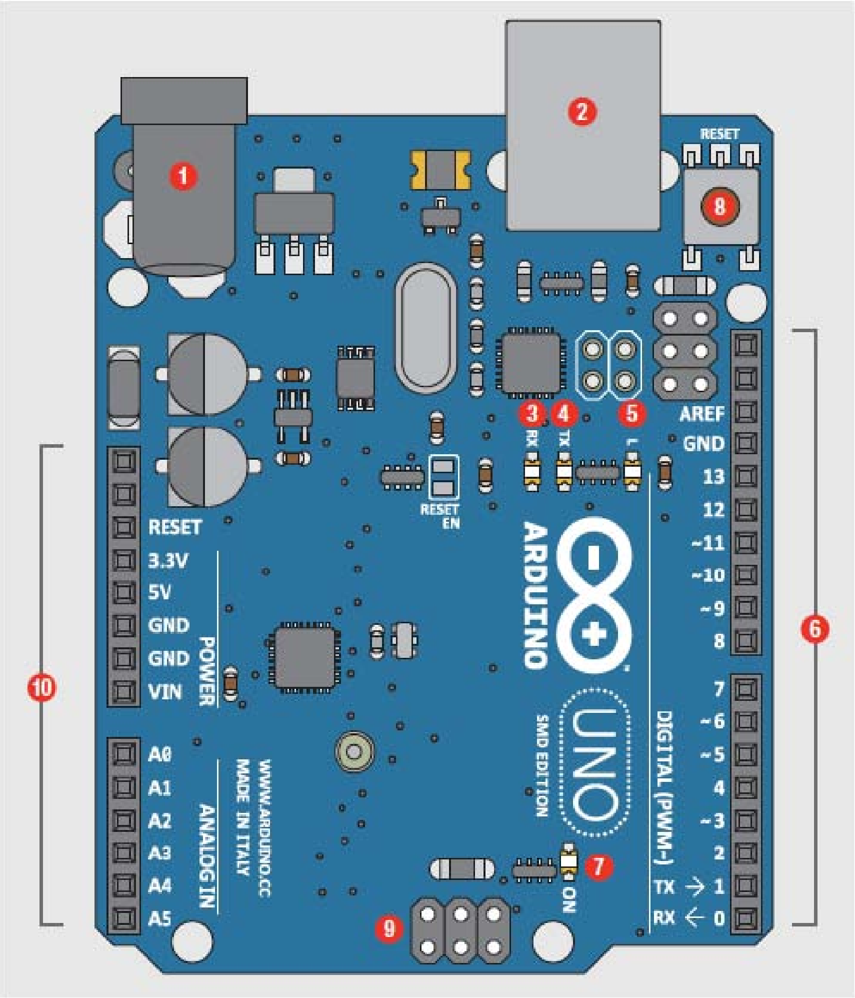


Arduino UNO

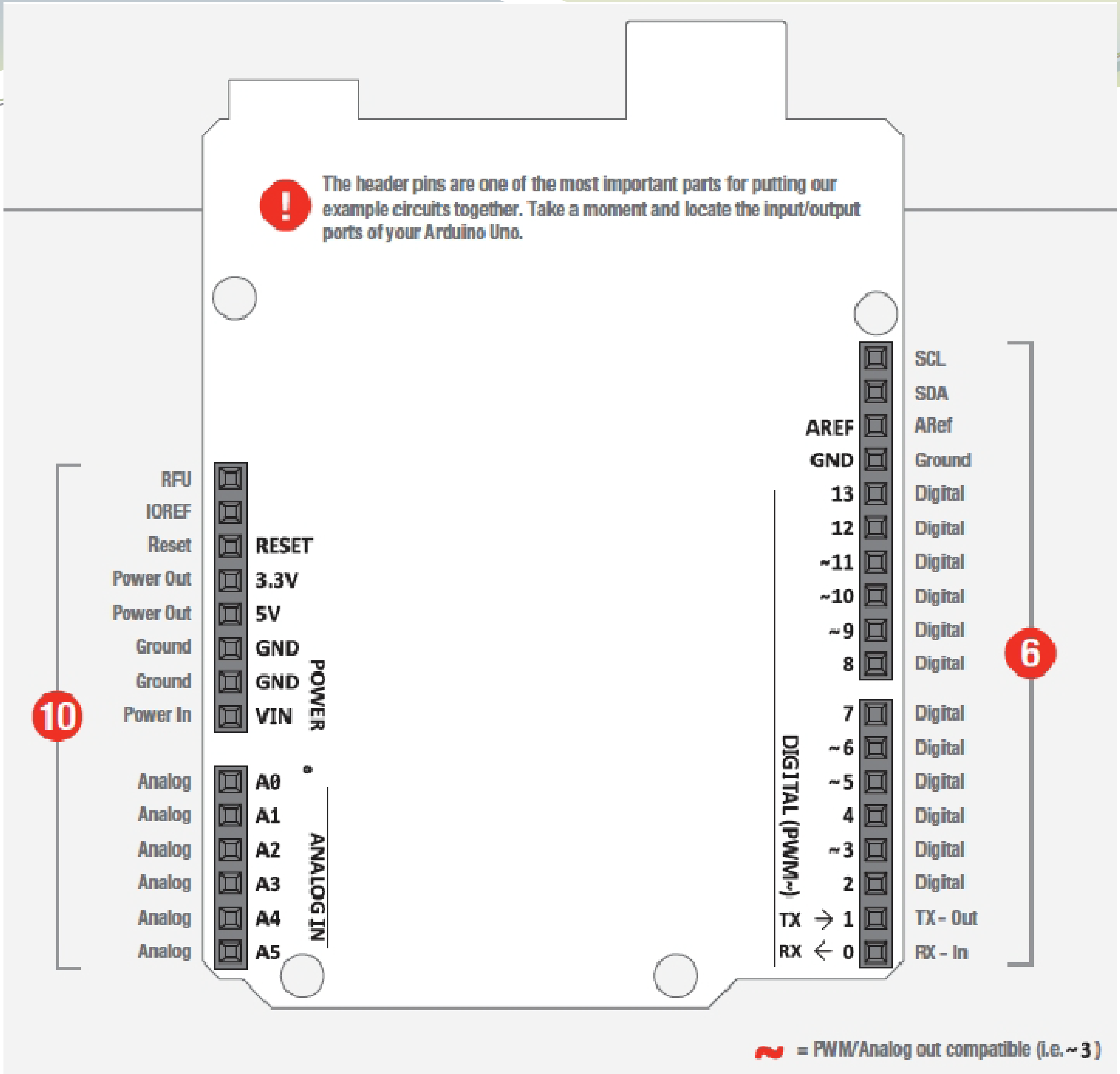
- Microcontroller: ATmega328
- Operating Voltage: 5V
- Input Voltage (recommended): 7-12V
- Input Voltage (limits): 6-20V
- Digital I/O Pins: 14 (of which 6 provide PWM output)
- Analog Input Pins: 6
- DC Current per I/O Pin: 40 mA
- DC Current for 3.3V Pin: 50 mA
- Flash Memory: 32 KB (ATmega328)
- SRAM: 2 KB (ATmega328)
- EEPROM: 1 KB (ATmega328)
- Clock Speed: 16 MHz

ARDUINO
OPEN-SOURCE
COMMUNITY

Arduino UNO

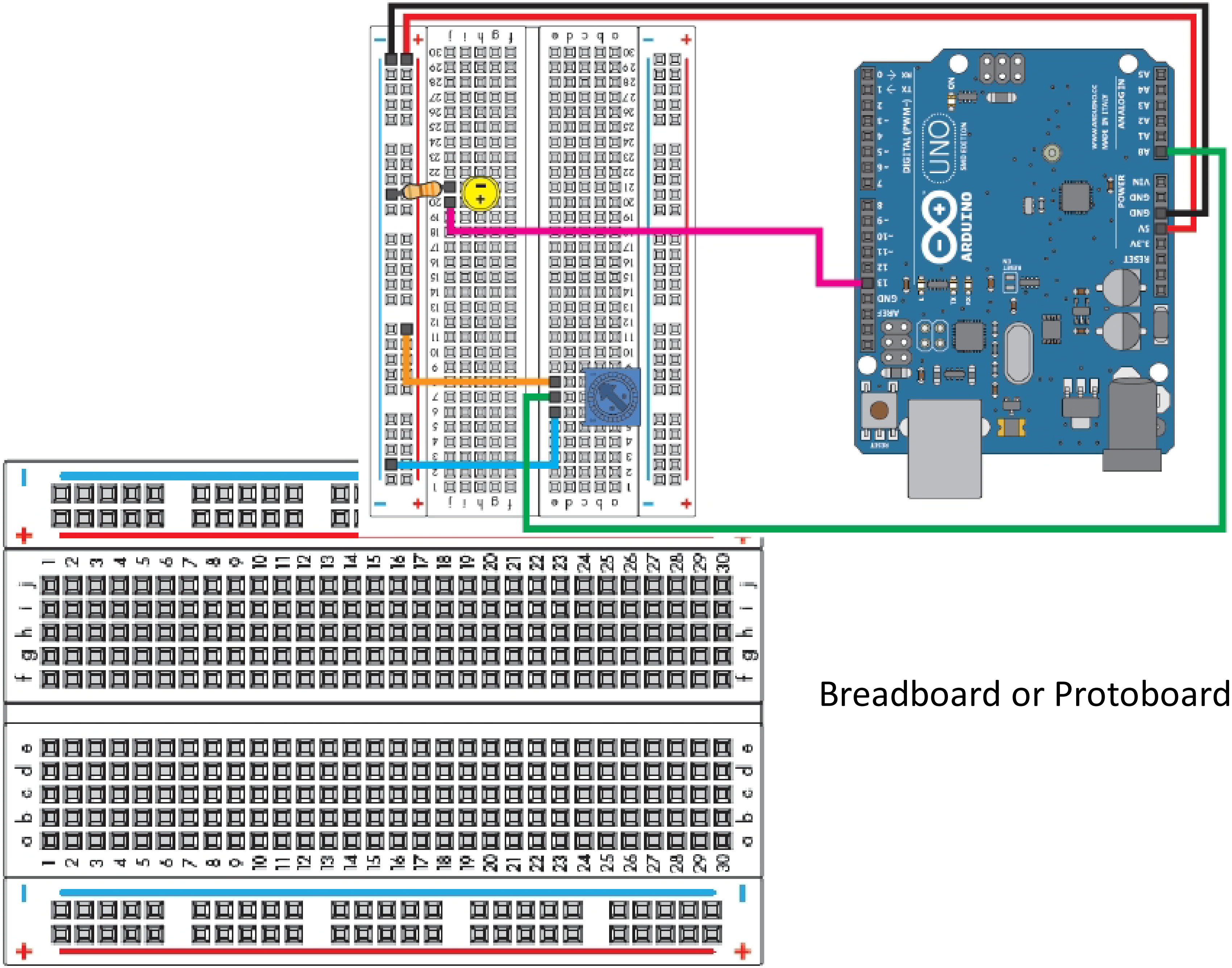


1 – Power In (9V-12V)
2 – Power In (USB)
3 – LED (Rx receiving)
4 – LED (Tx transmitting)
5 – LED (for programming)
6 – Pins (for programming)
7 – LED (power on status)
8 – Reset Button
9 – ICSP Pins (in-circuit serial programming)
10 – Pins (for programming)



The header pins are one of the most important parts for putting our example circuits together. Take a moment and locate the input/output ports of your Arduino Uno.

~ = PWM/Analog out compatible (i.e. ~3)



Breadboard or Protoboard

Kit Contents

1. Arduino UNO	1
2. LCD + I2C Driver	1
3. Bread board	1
4. Female-Male Jumper Wires	1 (40)
5. Male-Male Jumper Wires	1 (40)
6. 4x4 Matrix Keypad	1
7. Ultrasonic Range Finder	1
8. Relay Module	1
9. Photo Interrupter	1
10. Stepper Motor + Driver	1
11. DC Motor + Driver	1
12. Humid-Temp Sensor (DHT22)	1
13. Real Time Clock (RTC)	1
14. Micro SD Card Reader	1
15. Micro SD Card	1

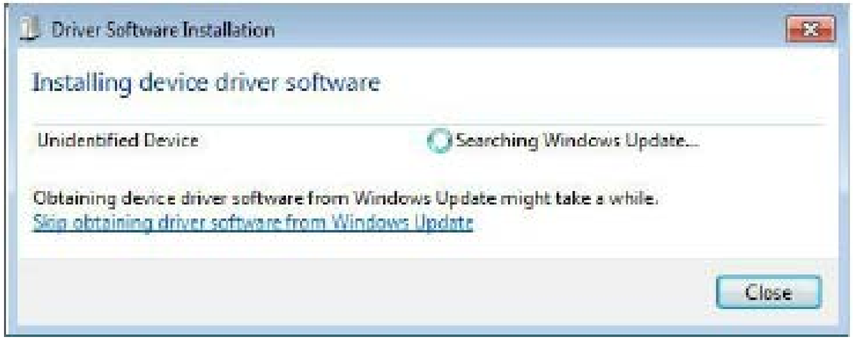
16. LEDs (red, green, yellow)	3
17. Resistor 300Ω	3
18. Tact Switches	3
19. Resistor 10kΩ	3
20. LDR	1
21. POT 10kΩ	1

Please take care of all components in your box.

Arduino IDE

- To install, first simply go to http://arduino.cc/en/Main/Software

  and download the current version of the software for your operating system. It is available for Windows, Mac and Linux.
- On Windows, the software is packed into a zip file. This will need to be unzipped with a program such as WinZip. Once unzipped, simply run the setup executable file. This will install almost all of the files needed in short order.
- You also need to install the driver for your board. Once the software is loaded, plug your Arduino board into your PC via a USB A to USB B cable. In a moment Windows will inform you that it found new hardware and is looking for a driver for it.



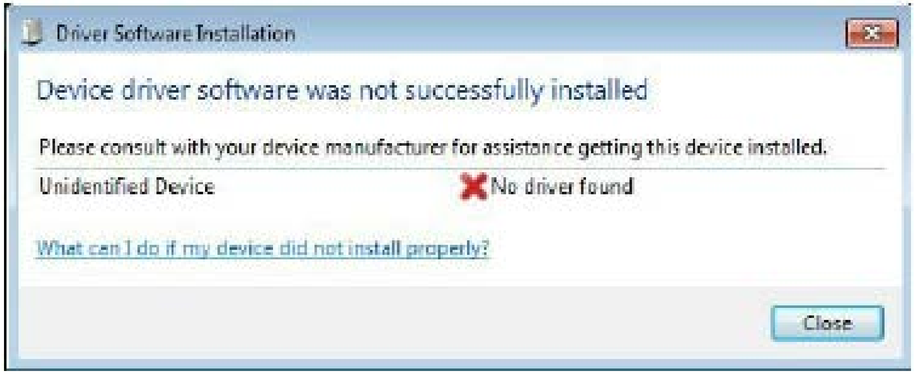
Installing device driver software

Unidentified Device Searching Windows Update...

Obtaining device driver software from Windows Update might take a while.
Skip obtaining driver software from Windows Update

Arduino IDE

- If Windows couldn’t find the driver, you’ll be greeted with something like this:



Device driver software was not successfully installed

Please consult with your device manufacturer for assistance getting this device installed.

Unidentified Device ❌ No driver found

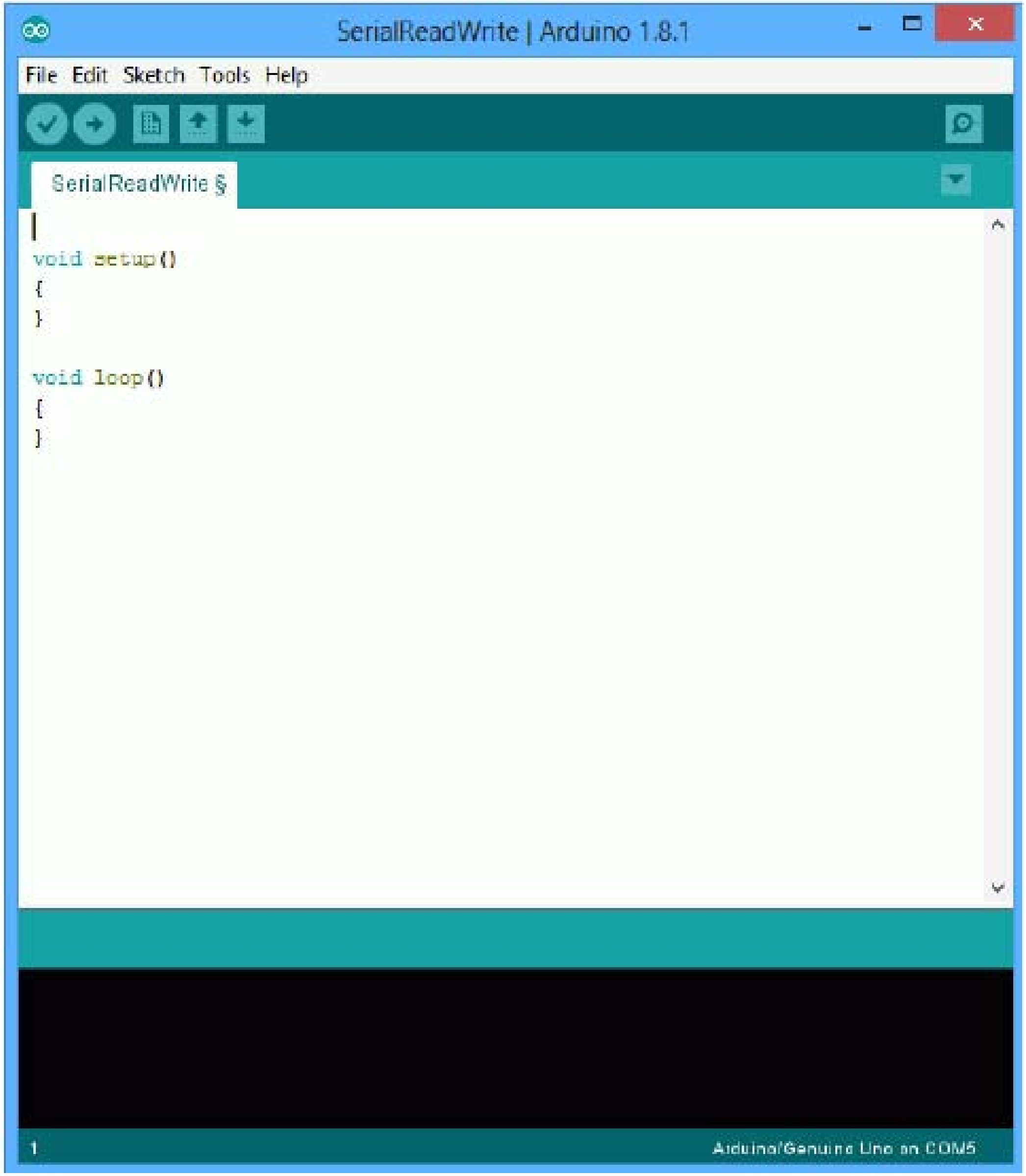
What can I do if my device did not install properly?

- You can install the driver manually. First, open the Device Manager (via Control Panel>>System>>Hardware). Scroll down to “Ports”. You should see entry like “Arduino Uno R3 (COM1)”. (You might instead find it under “Unknown Devices” instead of “Ports”.)
- Select this and go to the Driver tab. Click on Update/Install Driver. Browse to find your driver. It will be located in the “drivers” directory of the Arduino software install directory. For the Uno, it will be named “Arduino Uno R3.inf”. Select it and let Windows install it.

A step-by-step guide is available here: http://arduino.cc/en/Guide/Windows

Arduino IDE

- Once the board is installed, it's time to open the Arduino IDE!



SerialReadWrite | Arduino 1.8.1
File Edit Sketch Tools Help
SerialReadWrite §

```
void setup()
{
}

void loop()
{
}
```

1
Arduino/Genuino Uno on COM5

Review of C

NOTE:
/* */   denotes a comment.
//   denotes a single line comment.

- The Sizes and Ranges of Variables.

Variable Type	Bytes Used	Minimum	Maximum
char	1	-128	127
unsigned char	1	0	255
short int	2	-32768	32767
unsigned short int	2	0	65535
long int	4	≈ -2 billion	≈ 2 billion
unsigned long int	4	0	≈ 4 billion
float (6 significant digits)	4	± 1.2 E -38	± 3.4 E +38
double (15 significant digits)	8	± 2.3 E -308	± 1.7 E +308

- Variables must be declared before used! Examples:

```
char  x;            //This declares a signed 8 bit integer called x.
unsigned char  y;   //This declares an unsigned 8 bit integer called y.
short  z, a;        //This declares two signed 16 bit integers named z and a.
float  b = 1.0;     //This declares a real number named b and sets its initial value to 1.0
```

Functions

Format:

```
return_value  function_name( function argument list )
{
                ...statements...
}
```

Examples:

```
float my_function( int x, int y )
{
                ...appropriate statements here...
}
```

```
void other_function( void )
{
                ...appropriate statements here...
}
```

```
float add_mult_div( float a, float b ) {
      float answer;

      answer = a*b/(a+b);
      return( answer );
}
```

If Conditions

```
if ( test condition(s).. ) {
                ...do stuff...
}
```

```
if ( test condition(s).. ) {
                ...do stuff...
} else {
                ...do other stuff...
}
```

Example:

```
if ( a==b )
        do_x();
else
        do_y();
```

The test condition may check for numerous possibilities. The operators are:

==	equality
!=	inequality
>	greater than
<	less than
>=	greater than or equal to
<=	less than or equal to

You may also use Boolean (logic operators):

||	OR
&&	AND
!	NOT

Switch Cases

Format:

```
switch( test_variable )
{
        case value_1:
                ...do stuff...
                break;
        case value_2:
                ...do other stuff...
                break;
        default:
                ...do stuff for a value not in the list...
                break;
}
```

Example:

```
switch( my_choice )
{
        case 1:
                fRun1();            // function
                break;
        case 2:
                fRun2();            // function
                break;
        default:
                fRunDefault();  // function
                break;
}
```

Looping

- There are three looping constructs in C.

```
for ( initialization(s); termination test(s); increment(s) )
{
        ..statements to iterate..
}
```

```
while( test condition(s).. ) {
        ..statements to iterate..
}
```

```
do {
         ..statements to iterate..
} while( test condition(s).. )
```

Example:

```
for( a=0; a<10; a++ )
{
    /* stuff to do ten times */
}
```

Examples:

```
a=0;
while( a<10 ) {
       a++;
}
```

```
a=0;
do {
       a++;
} while( a<10 )
```

Structure

- **C** allows compound data called structures, or struct for short. The idea is to use a variety of the basic data types such as float or int to describe some sort of object. Structures may contain several of each type along with pointers, arrays, and even other structures.

```
typedef struct {
      double       currentgain;
      double       breakdown;
      double       maxpower;
      short int    manufacturer;
      char         model[20];
} transistor;

transistor my_transistor;
my_transistor.currentgain = 200.0;
my_transistor.maxpower = 50.0;
my_transistor.manufacturer = 23;
```

Array and Pointer

- Arrays:

```
float results[10];      // An array of 10 floats
long int x[20];         // An array of 20 longs, or 80 bytes
char y[10];             // An array of 10 chars or 10 bytes
double a[5] = {1.0, 2.0, 4.7, -177.0, 6.3e4};
char m[20] = {‘m’, ‘y’, ‘ ’, ‘d’, ‘o’, ‘g’, 0};
char n[20] = {“Bill the cat”};
```

- Pointers:

```
char        a;
char        *b;                // pointer
char        c[5]={“Hello”};    // array = string

a = 10;
b = &a;                        // now b = a’s address
*b = 11;                       // now a = 11
b=&c[0];                       // now *b=‘H’
b++;                           // now *b=‘e’
```

#define and #include

- #define

  Very often it is desirable to use symbolic constants in place of actual values.

```
#define PI   3.14159              // We define PI=3.14159
```

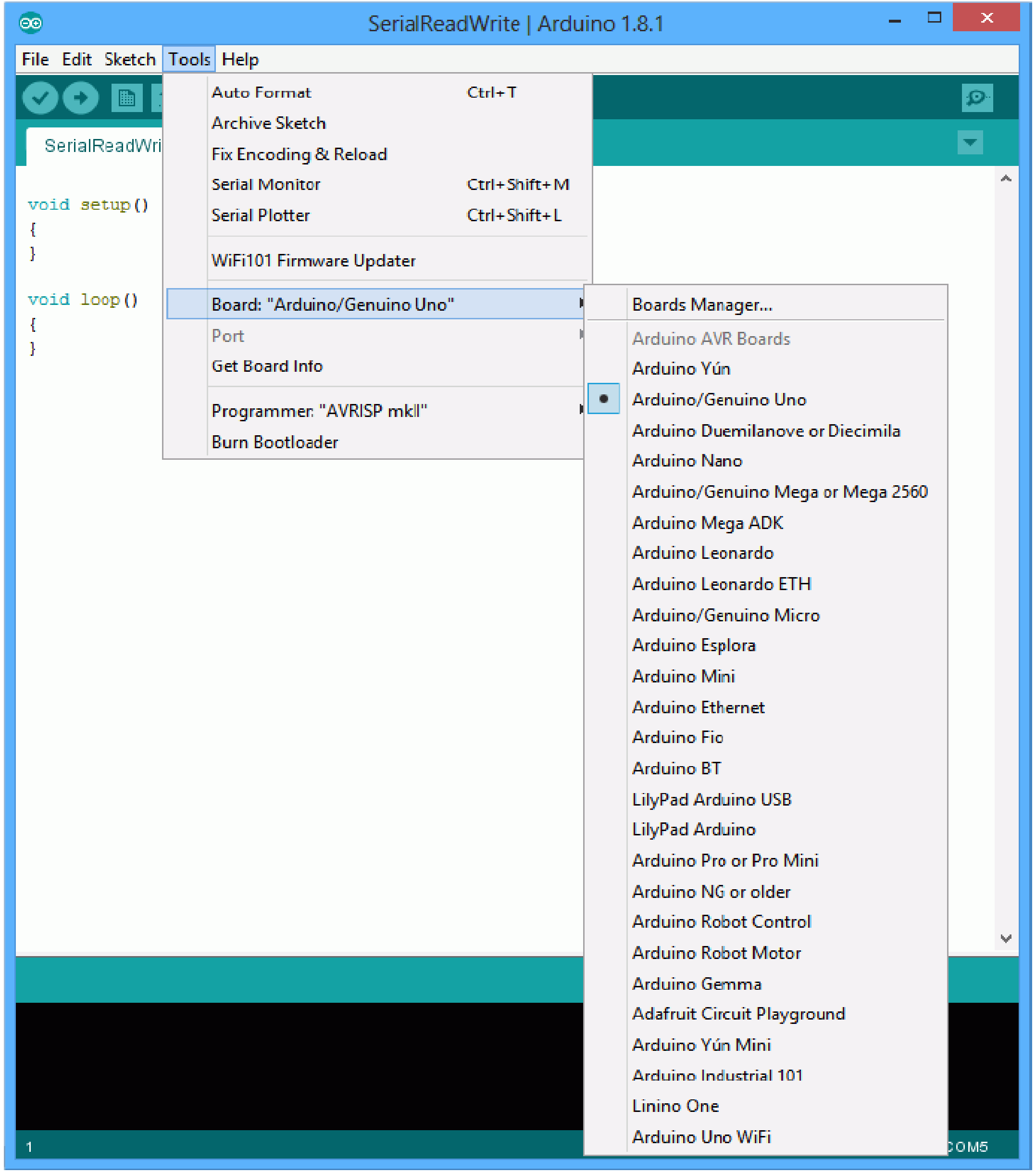
- #include

  We can tell the compiler to look into a special file called a header file to find this information. Every library has an associated header file (usually of the same name) and it will normally end with a **.h** file extension. The compiler directive is called an include statement.

```
#include <stdio.h>              // To include library stdio
#include “myheader.h”           // To include my self-defined header
```

Getting Started

- The two most important items here are Board (your board) and Serial Port (from which your board is connected). Failure to set these correctly will result in unsuccessful programming of your board.
- **Note** that the current board and COM port selections are printed at the lower right corner of the IDE window.

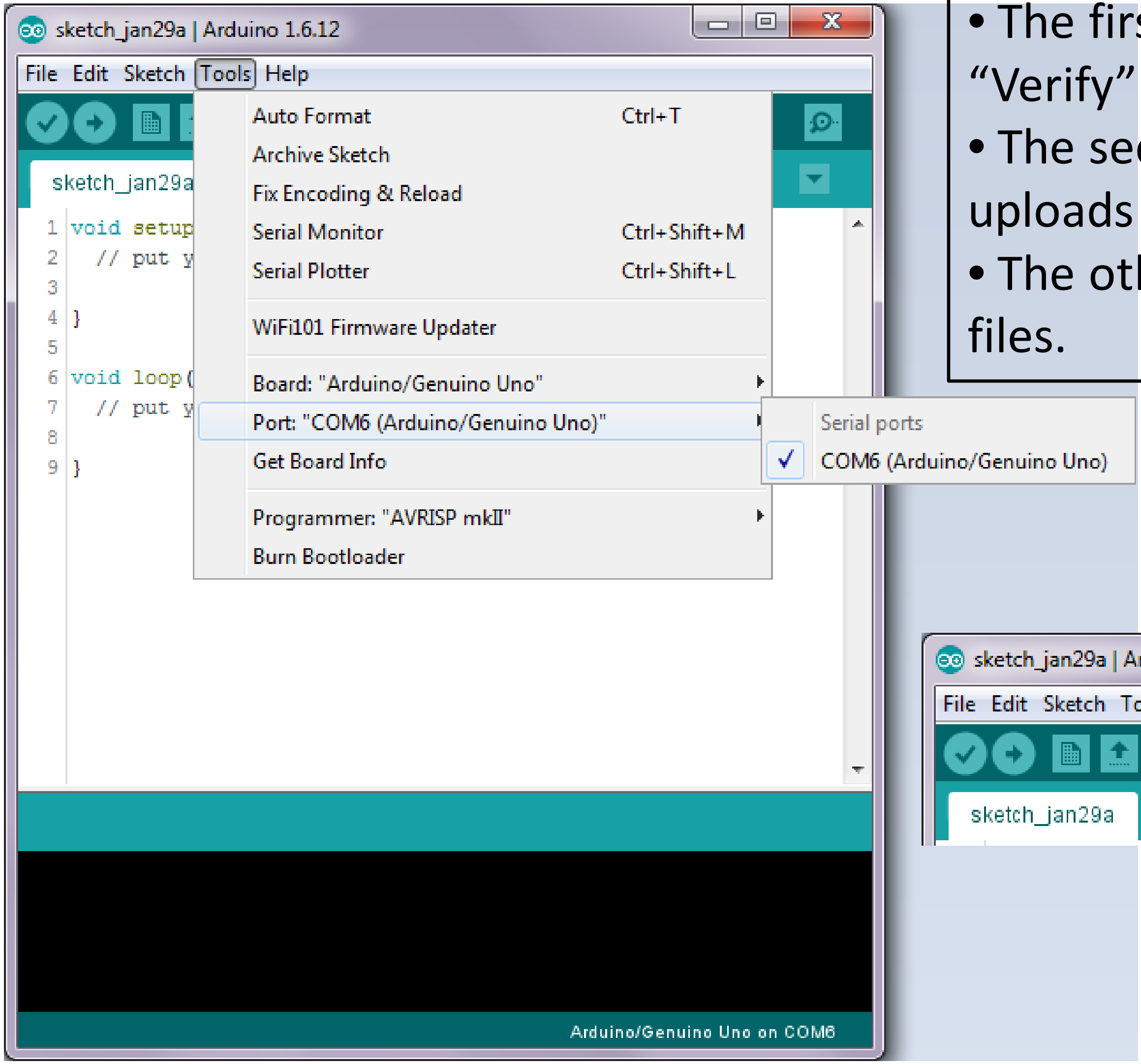


```
void setup()
{
}

void loop()
{
}
```

Getting Started

There is a simple toolbar below the menus.

- The first item (checkmark) is called "Verify" and compiles your code.
- The second item (right arrow) uploads the code to the target board.
- The other items new, open, and save files.


sketch_jan29a | Arduino 1.6.12
File Edit Sketch Tools Help
sketch_jan29a
Auto Format Ctrl+T
Archive Sketch
Fix Encoding & Reload
Serial Monitor Ctrl+Shift+M
Serial Plotter Ctrl+Shift+L
WiFi101 Firmware Updater
Get Board Info
Burn Bootloader
Serial ports
Arduino/Genuino Uno on COM6

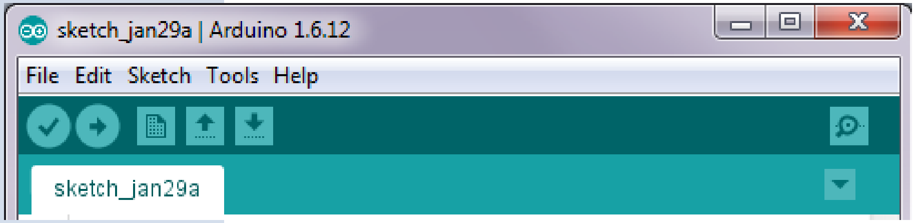
Board: "Arduino/Genuino Uno"

Port: "COM6 (Arduino/Genuino Uno)"

✓ COM6 (Arduino/Genuino Uno)

Programmer: "AVRISP mkII"

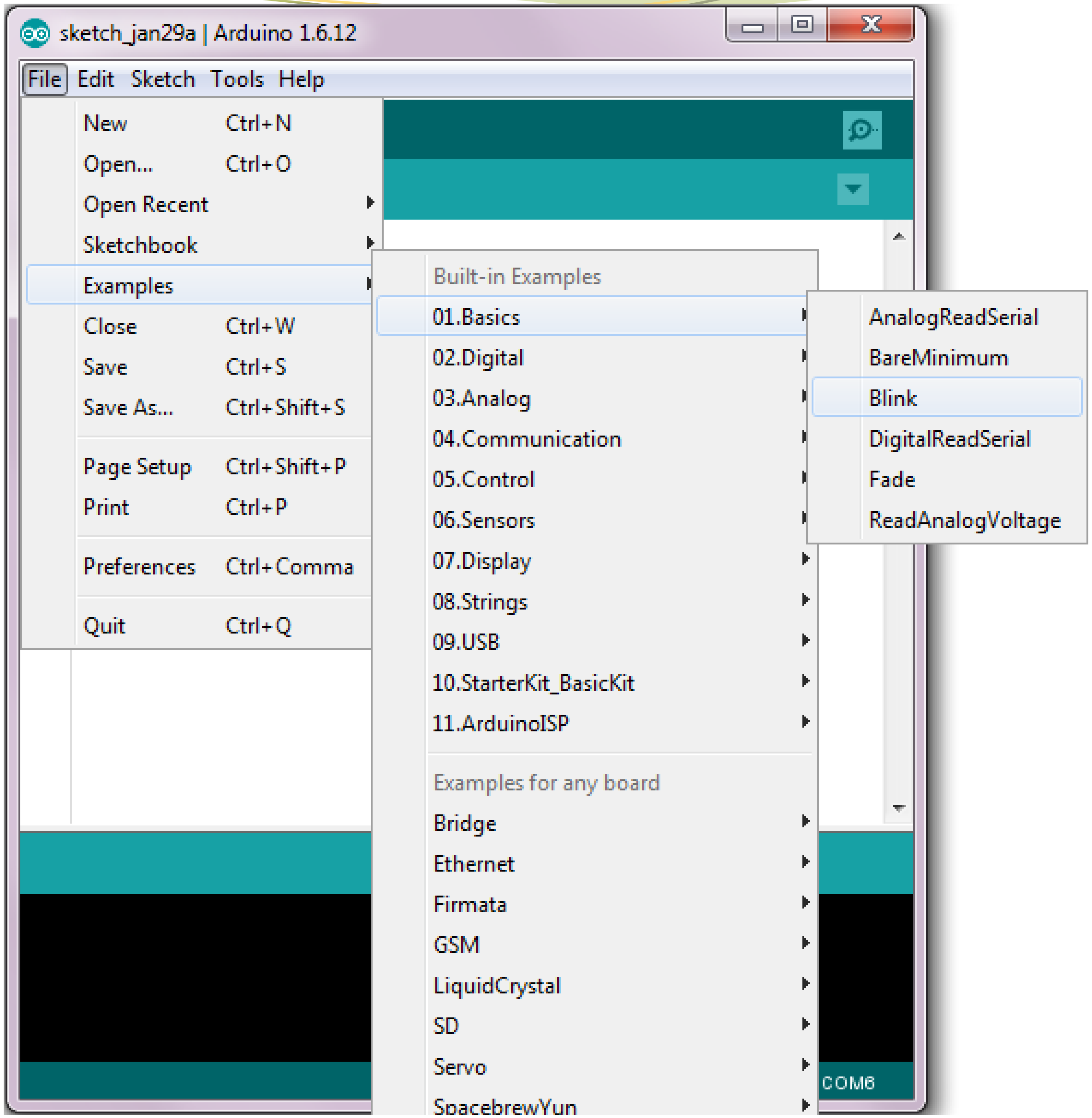
```
void setup
  // put y
}
void loop(
  // put y
}
```

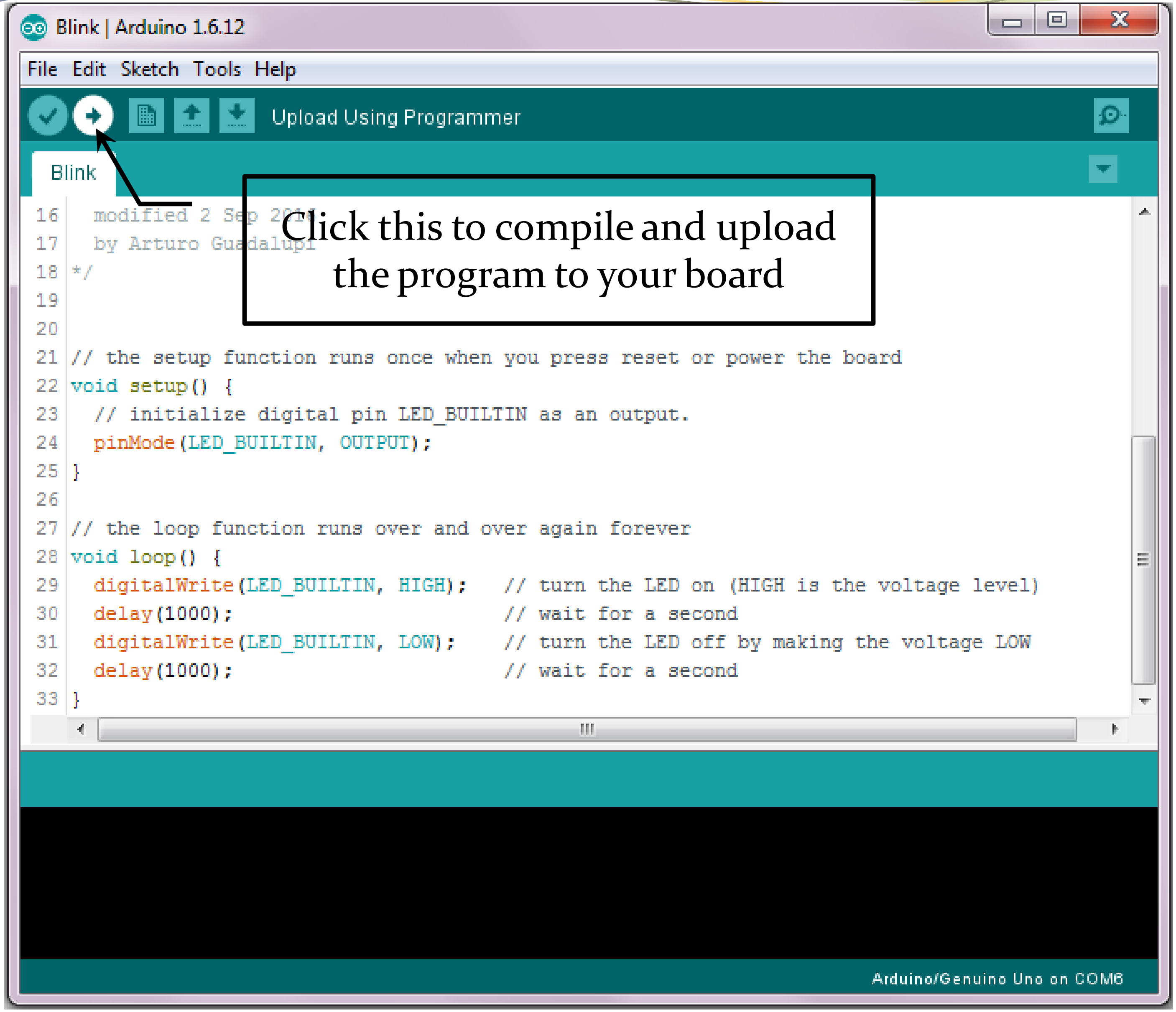


Blinking an LED

- Open **Blink** by clicking:

  File >> Examples >> 01.Basics >> Blink





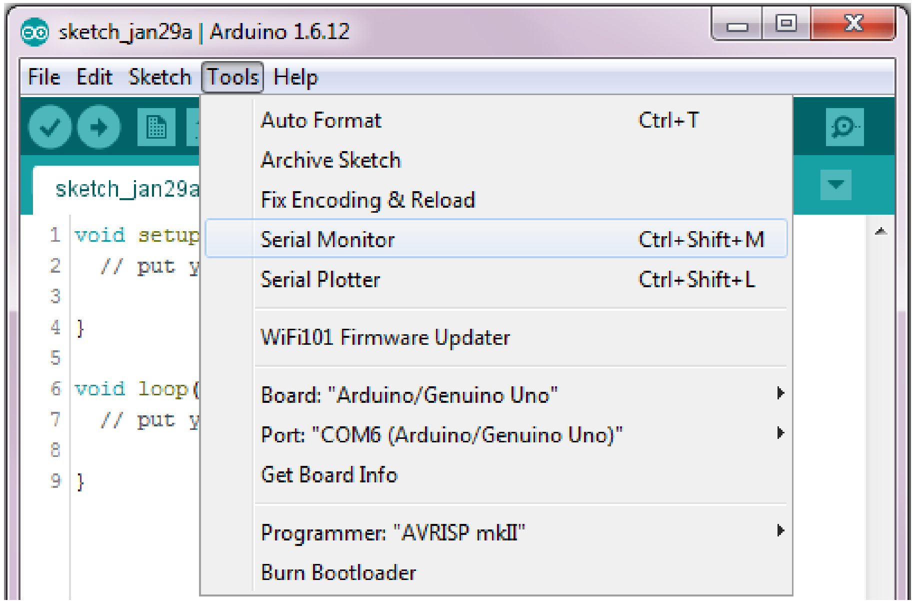
Click this to compile and upload the program to your board

```
  modified 2 Sep 2016
  by Arturo Guadalupi
*/


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

// the loop function runs over and over again forever
void loop() {
  digitalWrite(LED_BUILTIN, HIGH);   // turn the LED on (HIGH is the voltage level)
  delay(1000);                       // wait for a second
  digitalWrite(LED_BUILTIN, LOW);    // turn the LED off by making the voltage LOW
  delay(1000);                       // wait for a second
}
```

Hello World

- Create your new sketch and write the following code.

```
void setup()
{
      Serial.begin(9600);
}

void loop()
{
      Serial.print("Hello World\n");
      delay( 1000 );
}
```

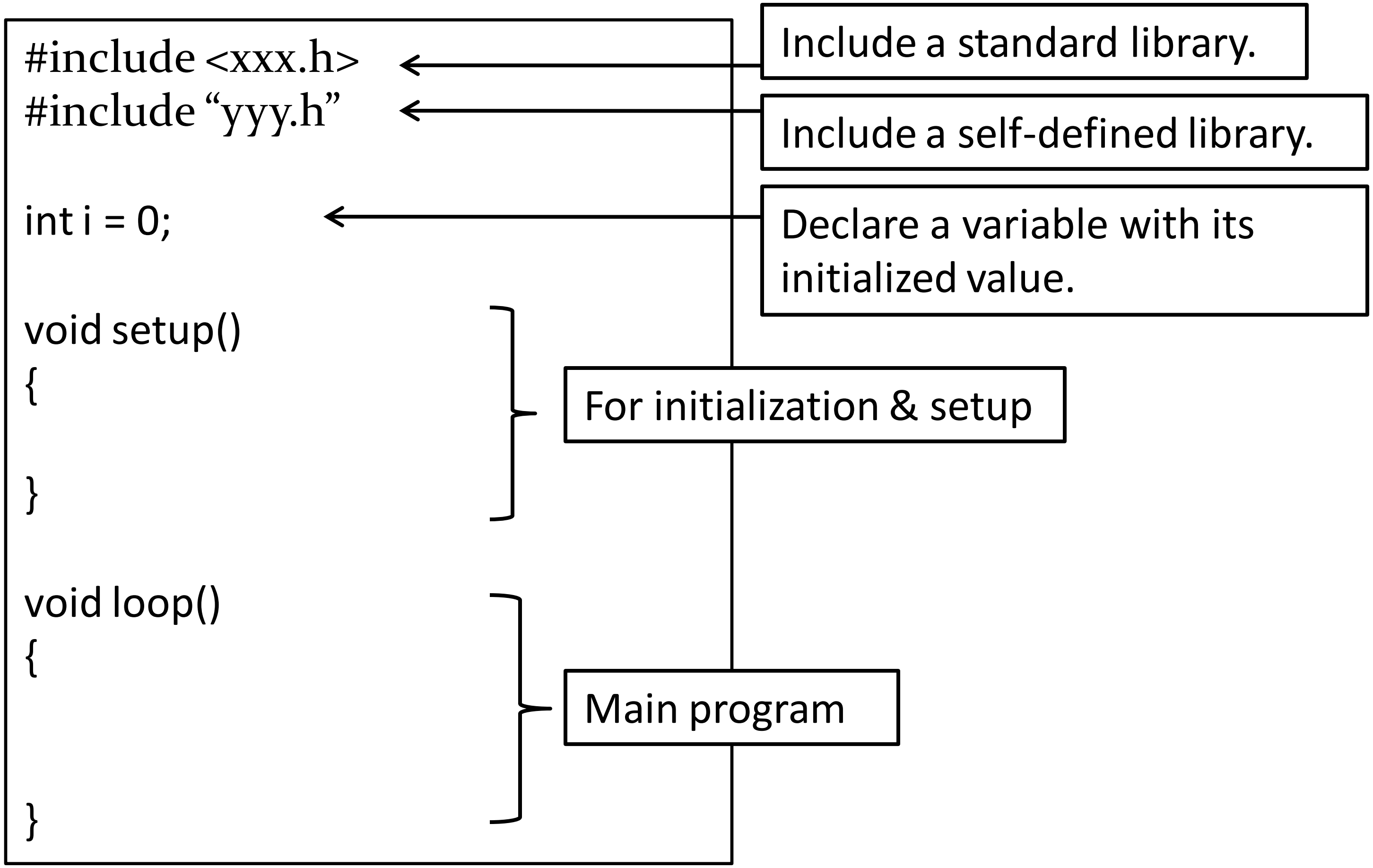


```
void setup
  // put y

}

void loop(
  // put y

}
```

sketch_jan29a | Arduino 1.6.12
File Edit Sketch Tools Help
sketch_jan29a
Auto Format Ctrl+T
Archive Sketch
Fix Encoding & Reload
Serial Monitor Ctrl+Shift+M
Serial Plotter Ctrl+Shift+L
WiFi101 Firmware Updater
Board: "Arduino/Genuino Uno"
Port: "COM6 (Arduino/Genuino Uno)"
Get Board Info
Programmer: "AVRISP mkII"
Burn Bootloader

- Compile and upload your program by clicking the right arrow.
- See your result by opening the serial monitor as shown in the above figure.

Coding



```
#include <xxx.h>
#include "yyy.h"

int i = 0;

void setup()
{

}

void loop()
{


}
```

Include a standard library.

Include a self-defined library.

Declare a variable with its initialized value.

For initialization & setup

Main program

Coding

delay(ms)

Description

Pauses the program for the amount of time in miliseconds.

Ex: delay(1000) = 1 second delay time.

delayMicroseconds(us)

Description

Pauses the program for the amount of time in microseconds.

Ex: delay(1000000) = 1 second delay time.

If possible, avoid the use of delay functions!!

Serial.print() & Serial.println()

Serial.begin()

It is to set the communication speed.

Ex:

```
Serial.begin(9600);
Serial.begin(14400);
Serial.begin(19200);
```

Serial.print(78)	gives "78"
Serial.print(1.23456)	gives "1.23"
Serial.print('N')	gives "N"
Serial.print("Hello world.")	gives "Hello world.“
Serial.print(78, BIN)	gives "1001110"
Serial.print(78, OCT)	gives "116"
Serial.print(78, DEC)	gives "78"
Serial.print(78, HEX)	gives "4E"
Serial.println(1.23456, 0)	gives "1"
Serial.println(1.23456, 2)	gives "1.23"
Serial.println(1.23456, 4)	gives "1.2346“

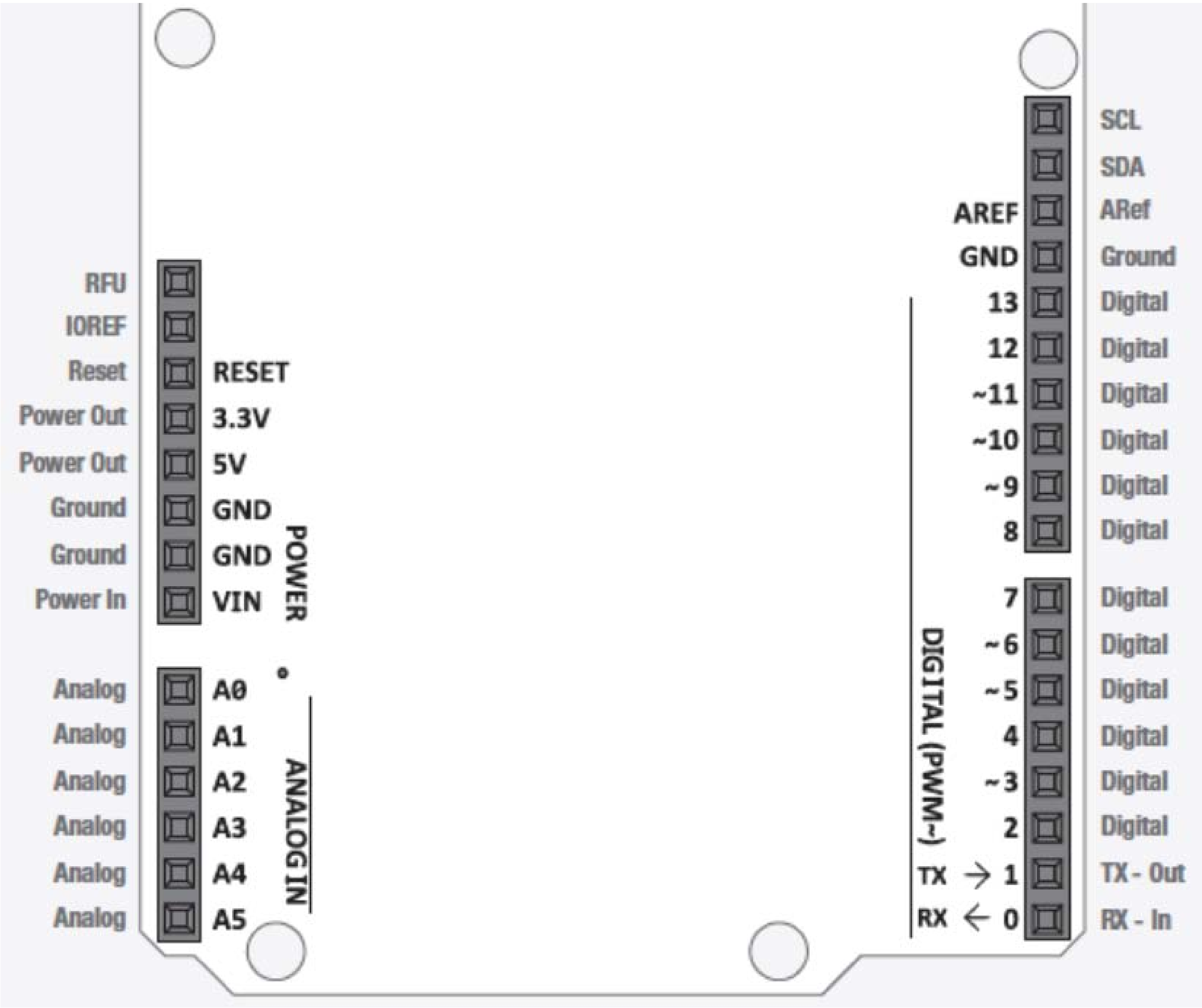
NOTE: println is to print with an added newline character.

TIPs: You may use Serial.print()/Serial.println() for debugging!
More information, see: https://www.arduino.cc/en/Serial/Print

GPIO – General Purpose Input & Output

PIN MODES

- Digital Output
- Digital Input
- Analog Input
- PWM
- Interrupt Input
- Communication Interface (Serial)



GPIO – General Purpose Input & Output

pinMode(pin, mode)

Configures the specified pin to behave either as an input or an output.

Parameters:

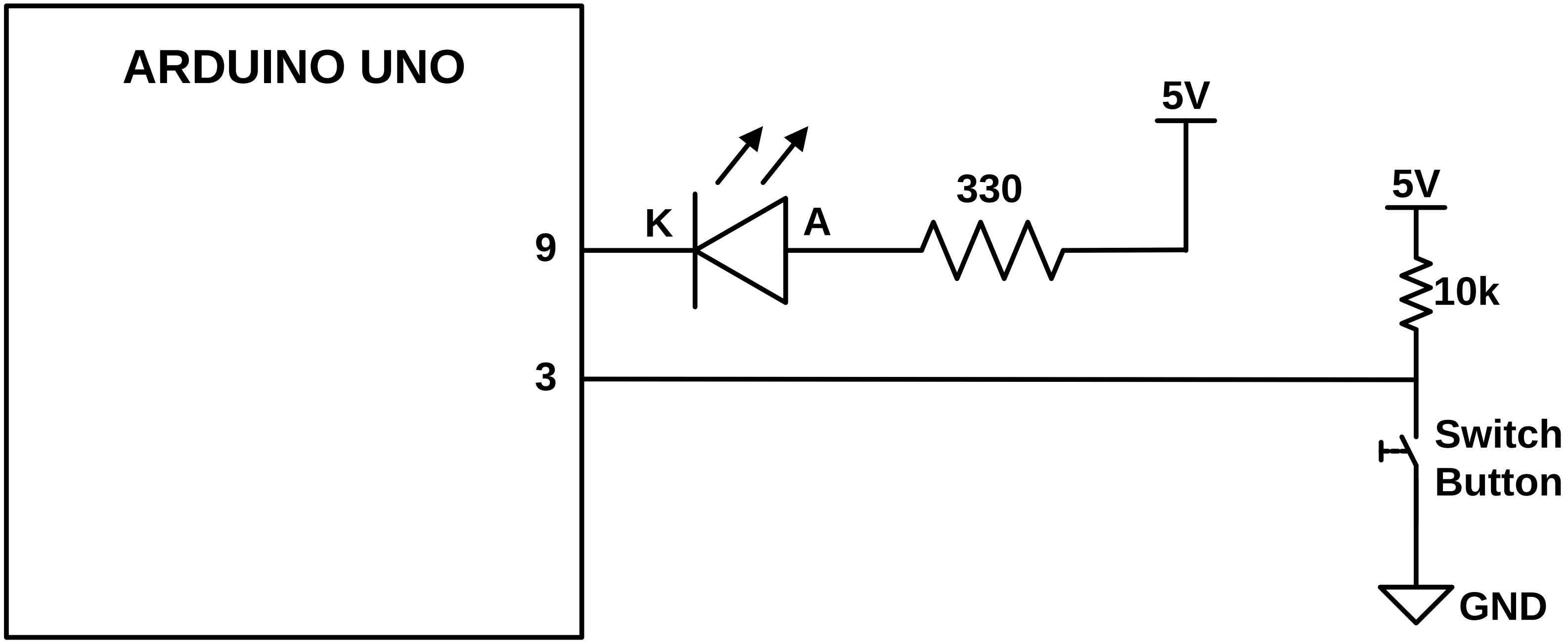
pin: the number of the pin whose mode you wish to set

mode: INPUT, OUTPUT, or INPUT_PULLUP

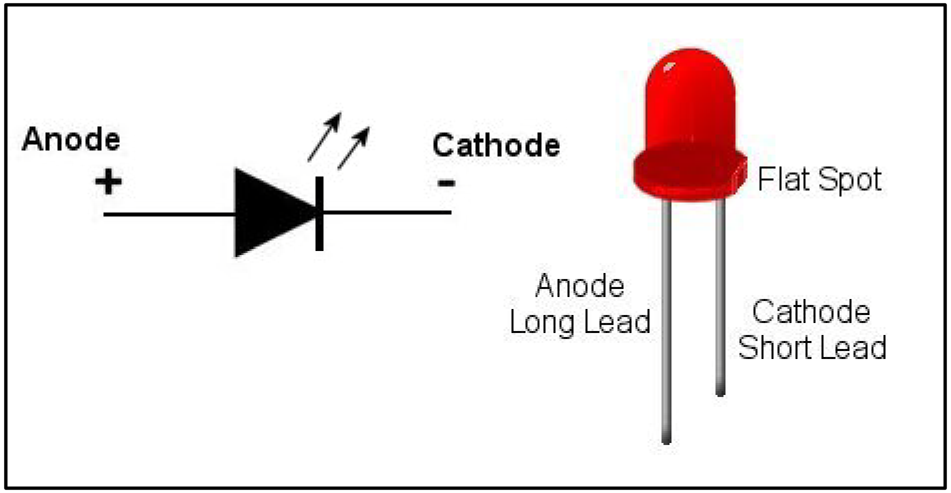
INPUT_PULLUP enables a built-in 20k-pullup resistor on that pin.

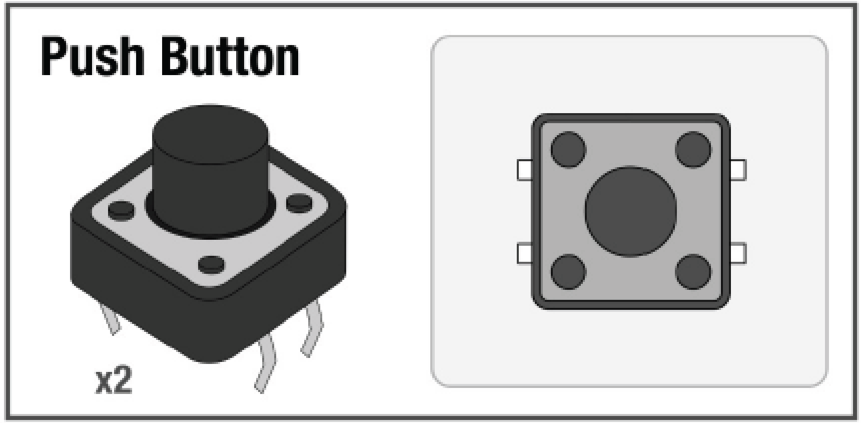
GPIO – General Purpose Input & Output

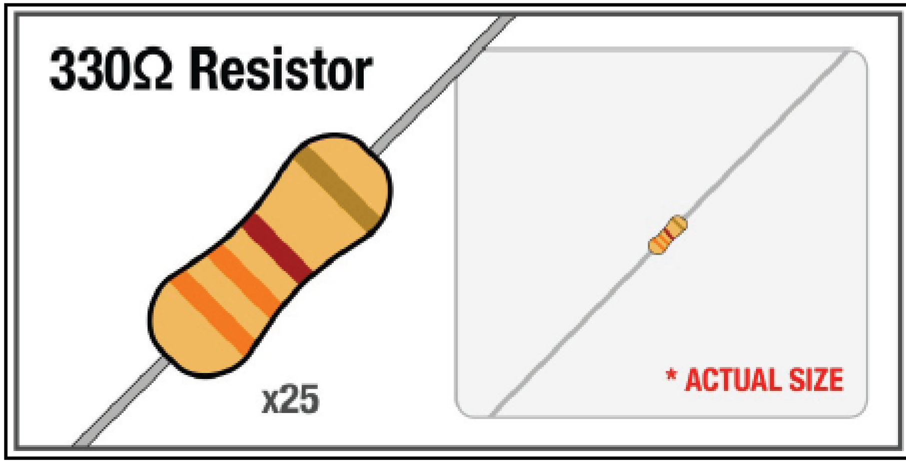
Make connections as the picture.

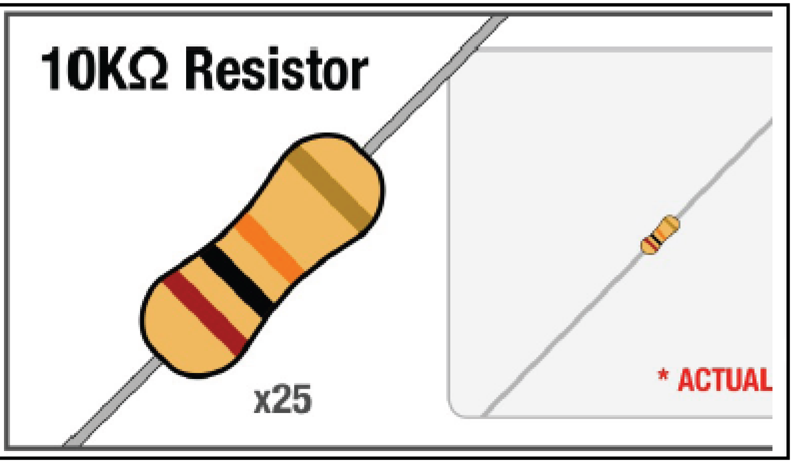


GPIO


Anode
+
Cathode
-
Flat Spot
Anode
Long Lead
Cathode
Short Lead

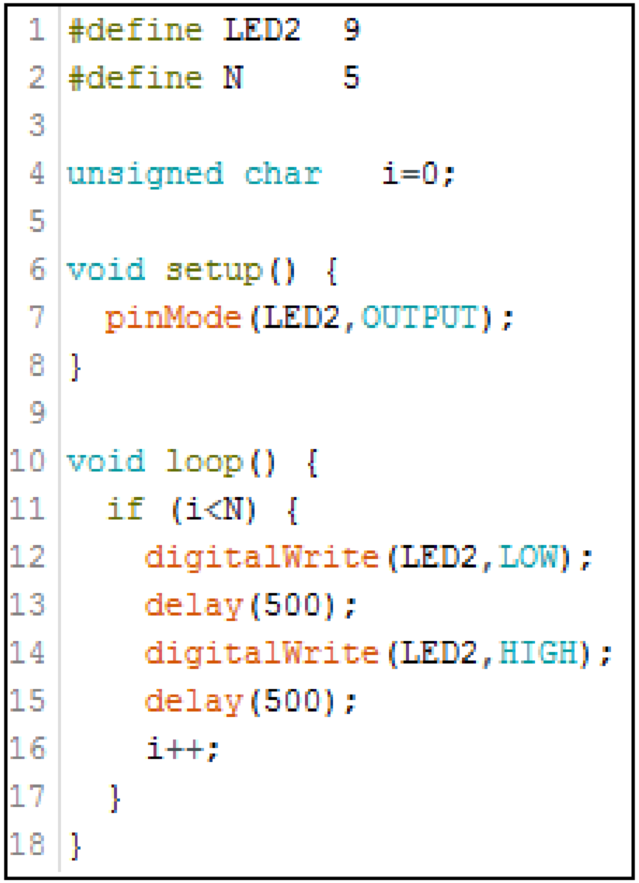

Push Button
x2


330Ω Resistor
x25
* ACTUAL SIZE


10KΩ Resistor
x25
* ACTUAL

GPIO

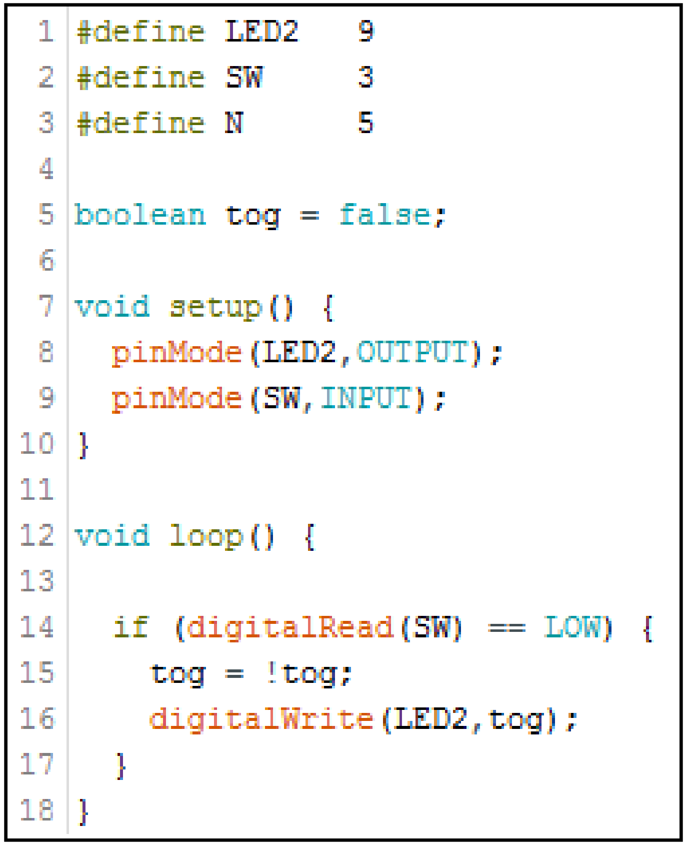
Exercise 1: Blinking a LED N times.



```
#define LED2  9
#define N     5

unsigned char   i=0;

void setup() {
  pinMode(LED2,OUTPUT);
}

void loop() {
  if (i<N) {
    digitalWrite(LED2,LOW);
    delay(500);
    digitalWrite(LED2,HIGH);
    delay(500);
    i++;
  }
}
```

GPIO

Exercise 2: Toggle a LED each time switch is pressed.



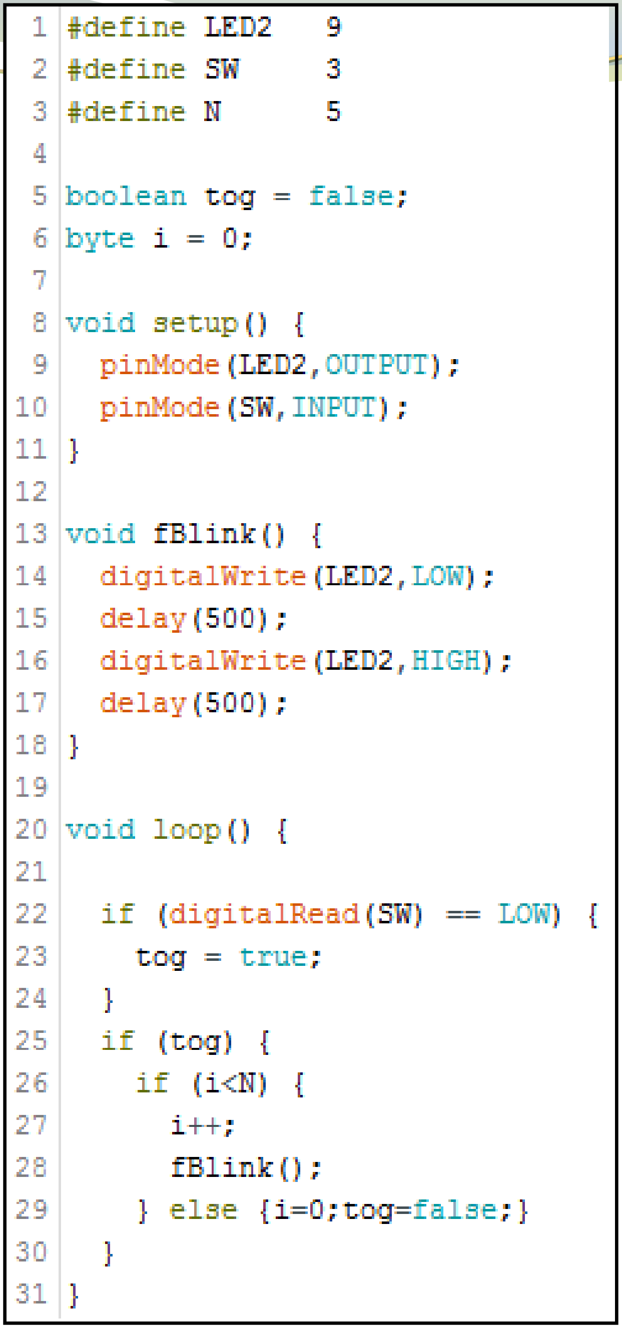
```
#define LED2   9
#define SW     3
#define N      5

boolean tog = false;

void setup() {
  pinMode(LED2,OUTPUT);
  pinMode(SW,INPUT);
}

void loop() {

  if (digitalRead(SW) == LOW) {
    tog = !tog;
    digitalWrite(LED2,tog);
  }
}
```

Q1: What can you see each time you press the switch?

GPIO

Exercise 3:

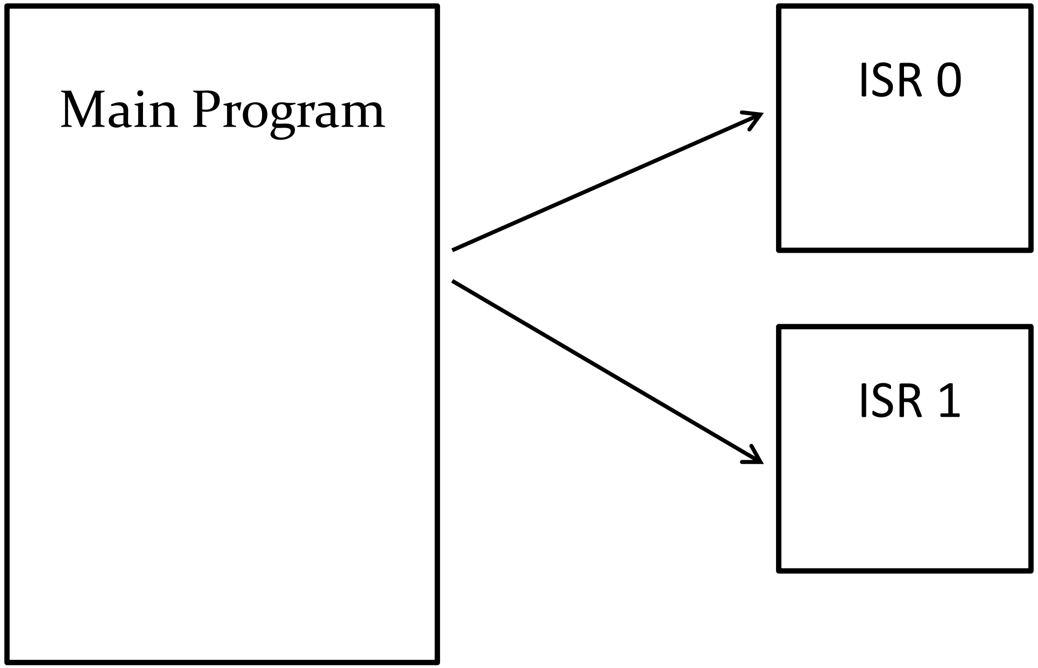
Blinking a LED N times when press the switch.



```
#define LED2   9
#define SW     3
#define N      5

boolean tog = false;
byte i = 0;

void setup() {
  pinMode(LED2,OUTPUT);
  pinMode(SW,INPUT);
}

void fBlink() {
  digitalWrite(LED2,LOW);
  delay(500);
  digitalWrite(LED2,HIGH);
  delay(500);
}

void loop() {

  if (digitalRead(SW) == LOW) {
    tog = true;
  }
  if (tog) {
    if (i<N) {
      i++;
      fBlink();
    } else {i=0;tog=false;}
  }
}
```

GPIO – General Purpose Input & Output

INTERRUPT



ISR = Interrupt Service Routine

GPIO

Arduino UNO has 2 digital pins usable for interrupts, which are pin 2 and 3.

How to attach an ISR to your program:

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

pin: the pin number

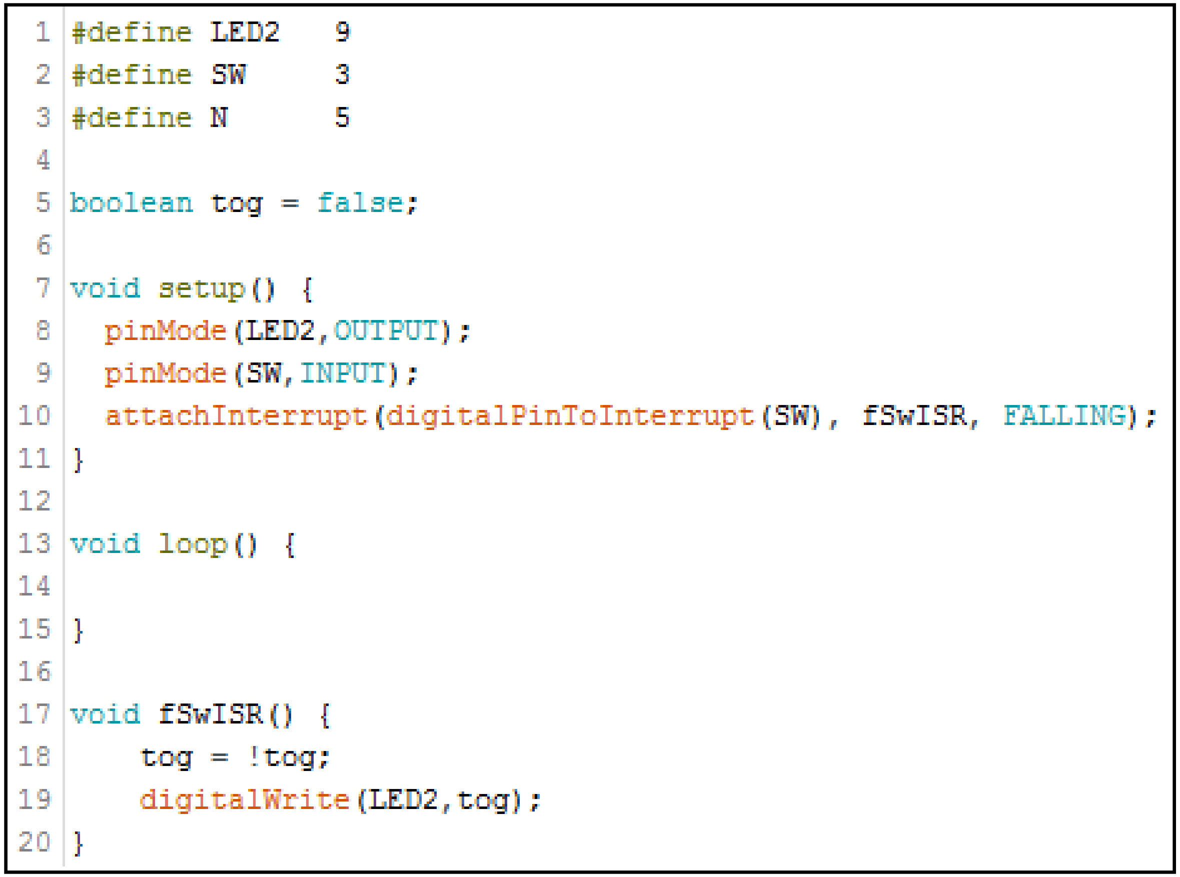
ISR: the interrupt service routine

mode:

LOW to trigger the interrupt whenever the pin is low,
CHANGE to trigger the interrupt whenever the pin changes value,
RISING to trigger when the pin goes from low to high,
FALLING for when the pin goes from high to low.

GPIO

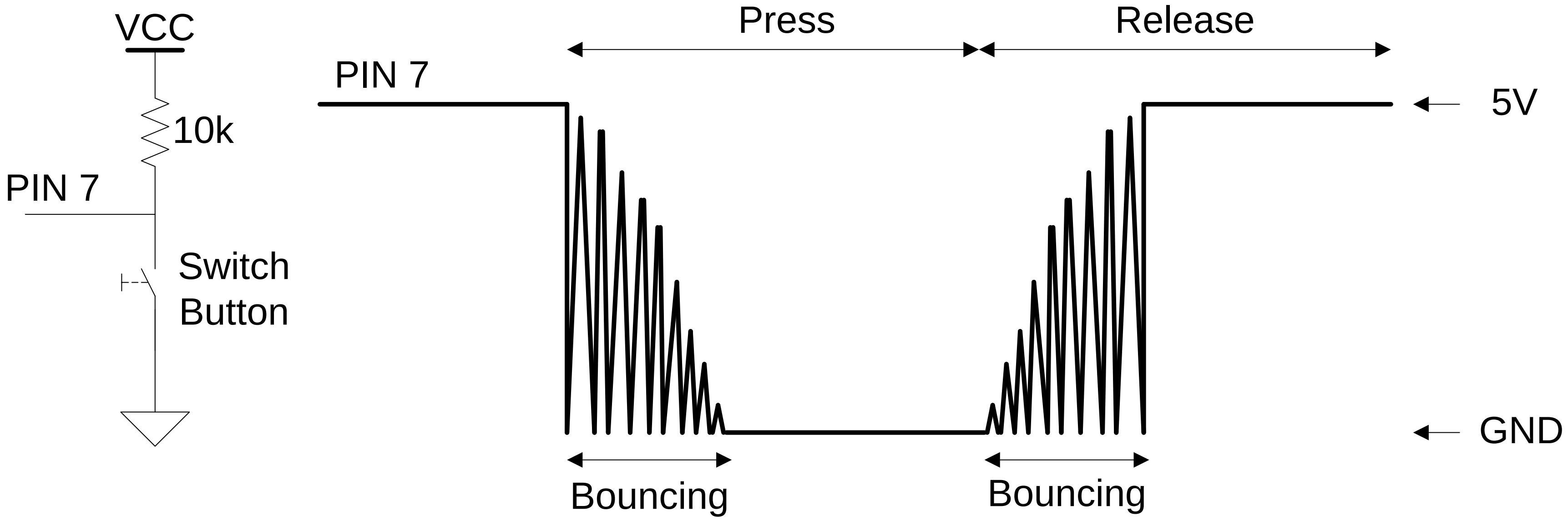
Exercise 4: Switch’s interrupt



```
#define LED2   9
#define SW     3
#define N      5

boolean tog = false;

void setup() {
  pinMode(LED2,OUTPUT);
  pinMode(SW,INPUT);
  attachInterrupt(digitalPinToInterrupt(SW), fSwISR, FALLING);
}

void loop() {

}

void fSwISR() {
    tog = !tog;
    digitalWrite(LED2,tog);
}
```

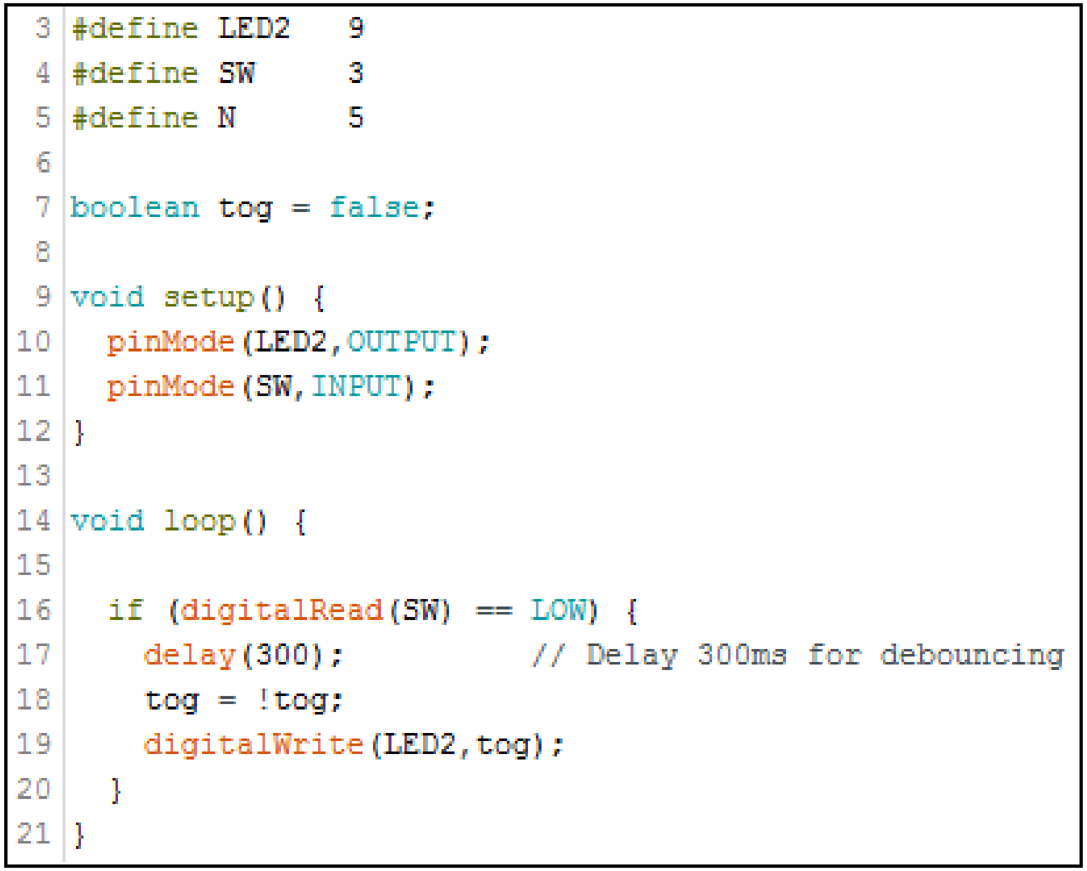
GPIO – General Purpose Input & Output

Bouncing Signals



GPIO

Exercise 5: Debouncing with delay

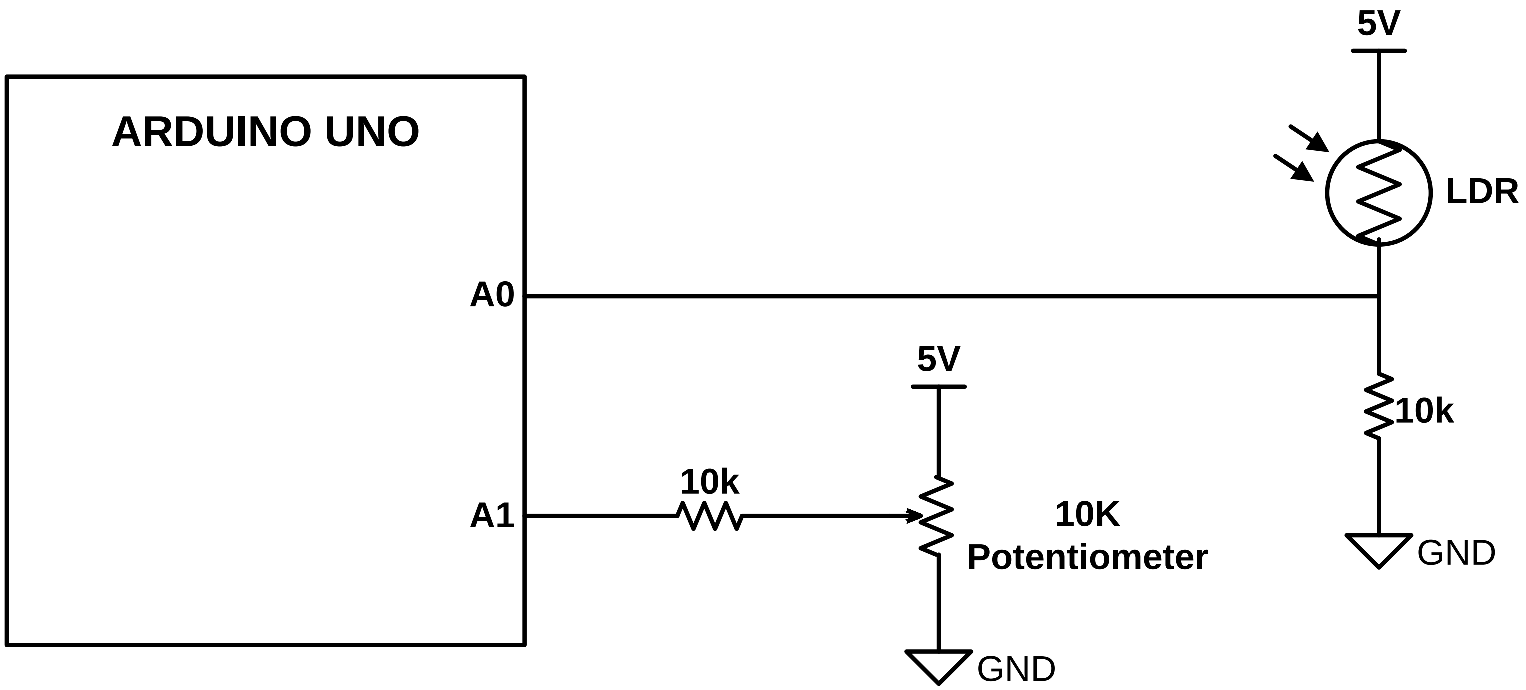


```
#define LED2   9
#define SW     3
#define N      5

boolean tog = false;

void setup() {
  pinMode(LED2,OUTPUT);
  pinMode(SW,INPUT);
}

void loop() {

  if (digitalRead(SW) == LOW) {
    delay(300);          // Delay 300ms for debouncing
    tog = !tog;
    digitalWrite(LED2,tog);
  }
}
```

GPIO – General Purpose Input & Output

Analog Input

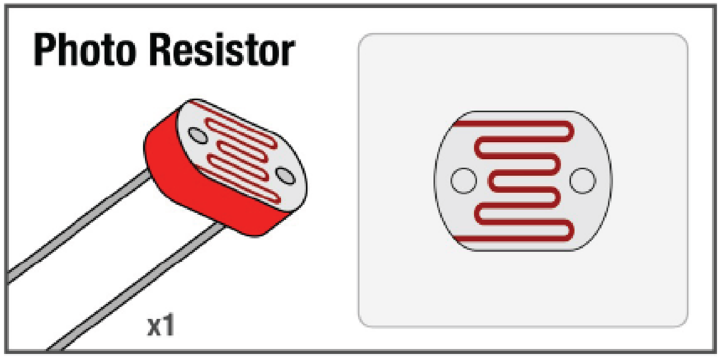
Analog Inputs of Arduino UNO are **A0**, **A1**, **A2**, **A3**, **A4**, and **A5**.

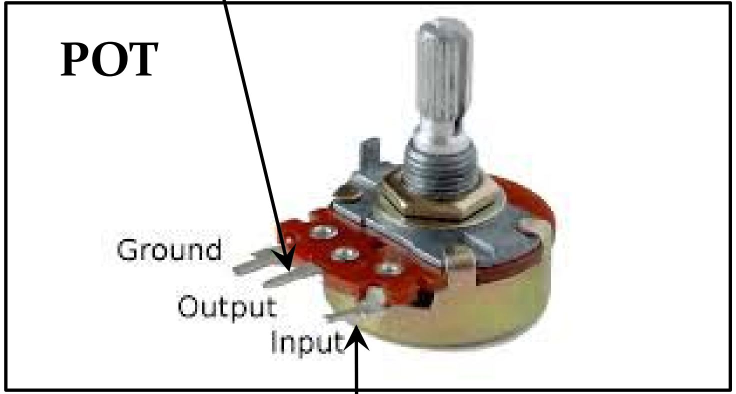


GPIO – General Purpose Input & Output

Parts

Connect to a 10k Resistor





Connect to 5V

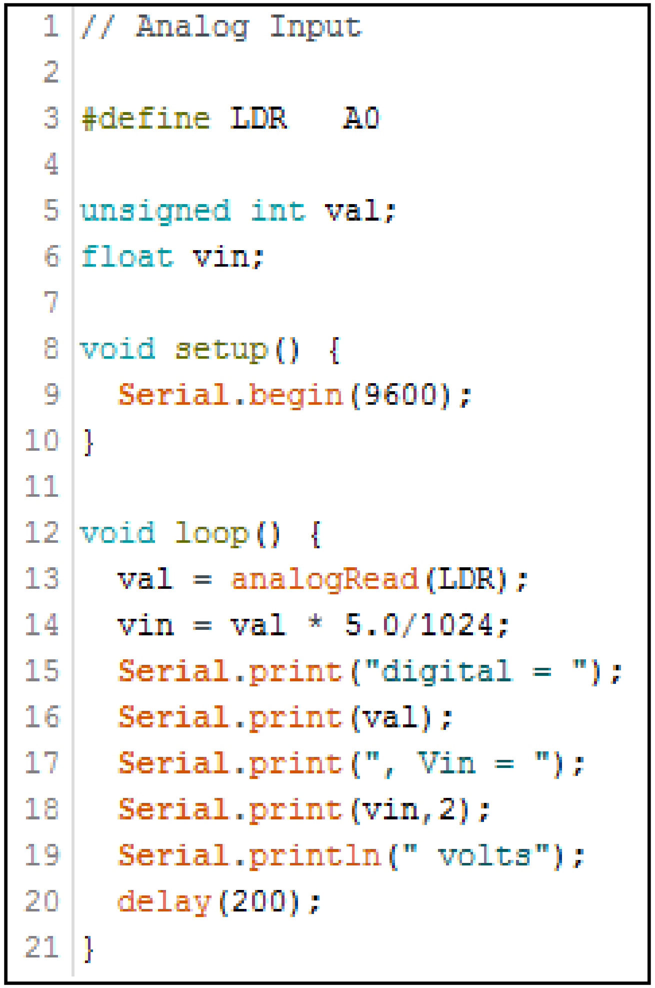
analogRead(pin)

Reads the value (0-1023) from the specified analog pin.

pin: the number of the analog input pin.

GPIO

Exercise 6: Analog Input

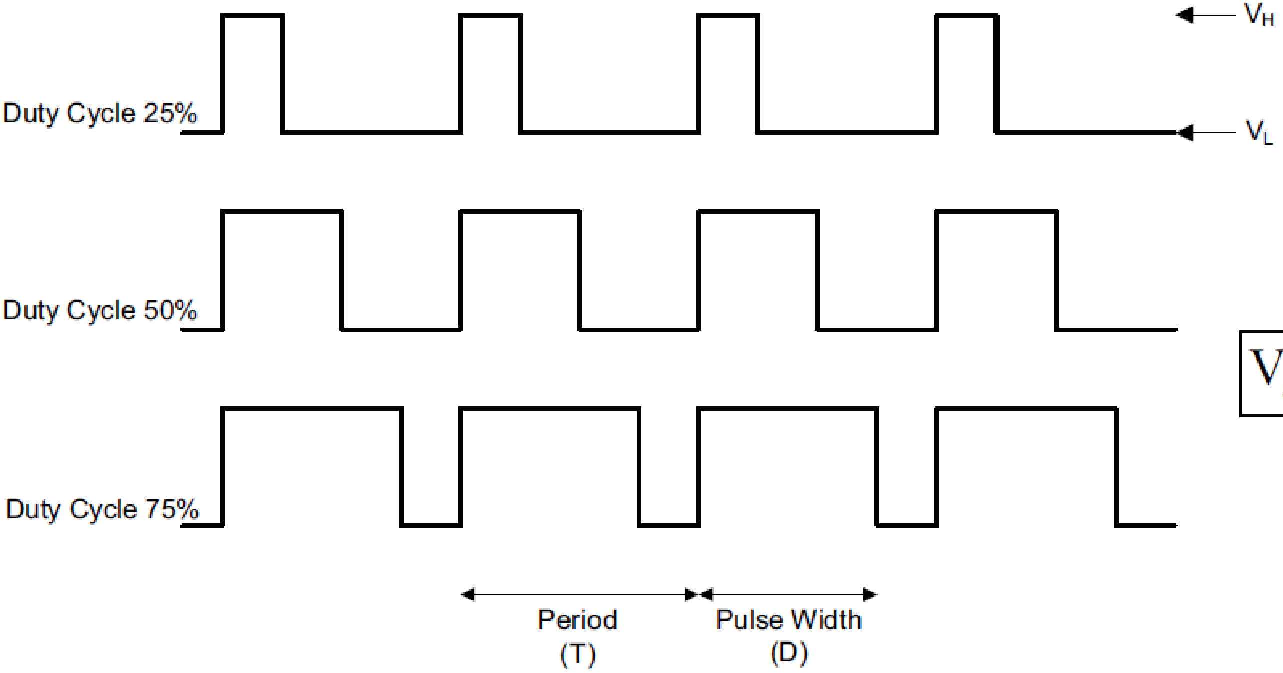


```
// Analog Input

#define LDR   A0

unsigned int val;
float vin;

void setup() {
  Serial.begin(9600);
}

void loop() {
  val = analogRead(LDR);
  vin = val * 5.0/1024;
  Serial.print("digital = ");
  Serial.print(val);
  Serial.print(", Vin = ");
  Serial.print(vin,2);
  Serial.println(" volts");
  delay(200);
}
```

GPIO – General Purpose Input & Output

Pulse Width Modulation (PWM)

$$\boxed{\text{Duty Cycle}\,(\text{D}) = \frac{\text{Pulse Width}}{\text{Period}} \times 100 \quad \%}$$



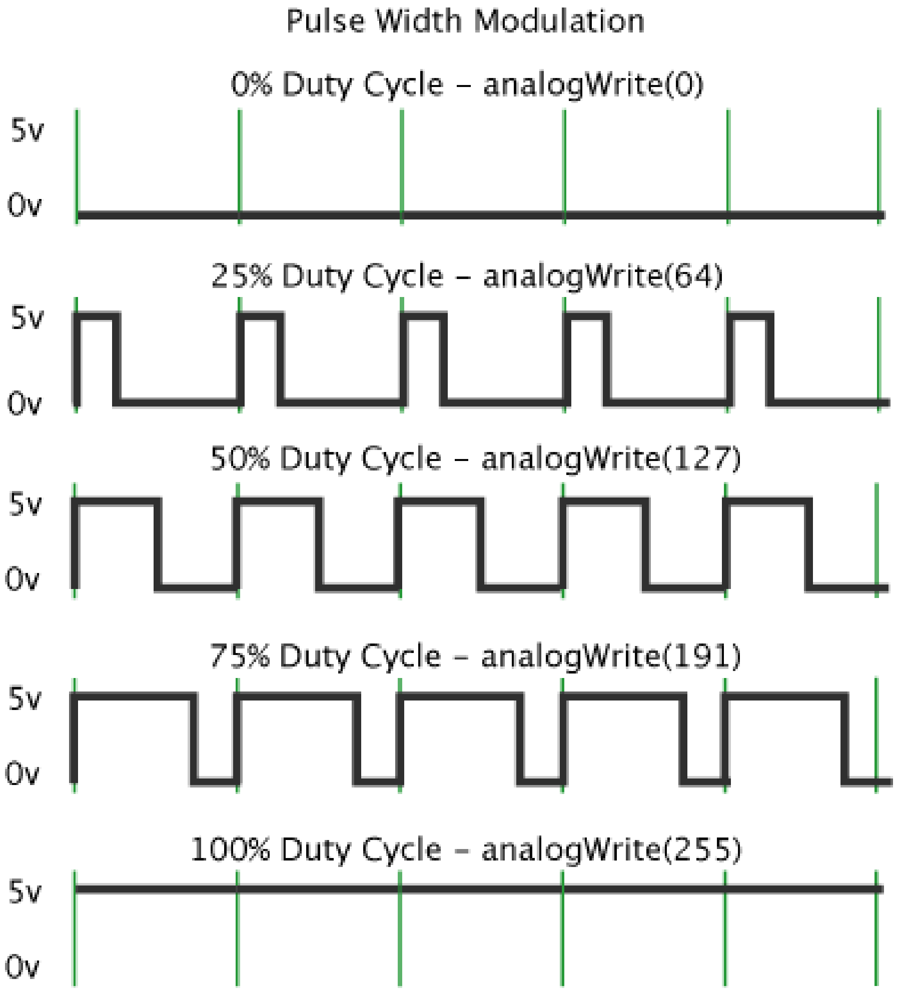
$$\boxed{V_{avg} = D \cdot V_H + (1 - D) \cdot V_L}$$

GPIO – General Purpose Input & Output

analogWrite(pin,dutyCycle)

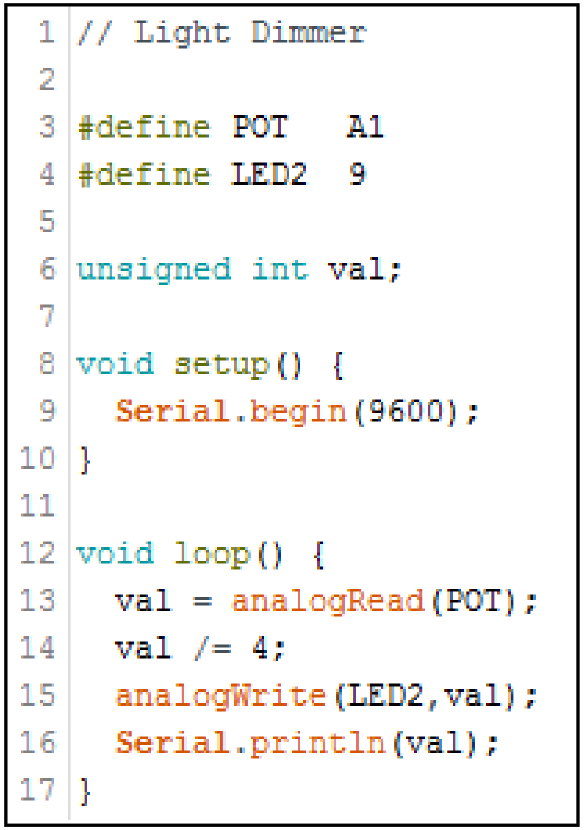
- **dutyCycle** is a value from 0 to 255.
- **pin** is one of the PWM pins (3, 5, 6, 9, 10, or 11).

Most pins have a frequency of about 490 Hz. On the Uno and similar boards, pins 5 and 6 have a frequency of about 980 Hz.


Pulse Width Modulation
0% Duty Cycle – analogWrite(0)
5v
0v
25% Duty Cycle – analogWrite(64)
5v
0v
50% Duty Cycle – analogWrite(127)
5v
0v
75% Duty Cycle – analogWrite(191)
5v
0v
100% Duty Cycle – analogWrite(255)
5v
0v

GPIO – General Purpose Input & Output

Exercise 7: Dimming an LED



```
// Light Dimmer

#define POT   A1
#define LED2  9

unsigned int val;

void setup() {
  Serial.begin(9600);
}

void loop() {
  val = analogRead(POT);
  val /= 4;
  analogWrite(LED2,val);
  Serial.println(val);
}
```

How to add a library

Method 1: Manual

- Unzip the library zip file.
- Move the unzipped folder to your arduino folder, e.g., ...\Arduino\libraries
- Done!

Method 2: Import

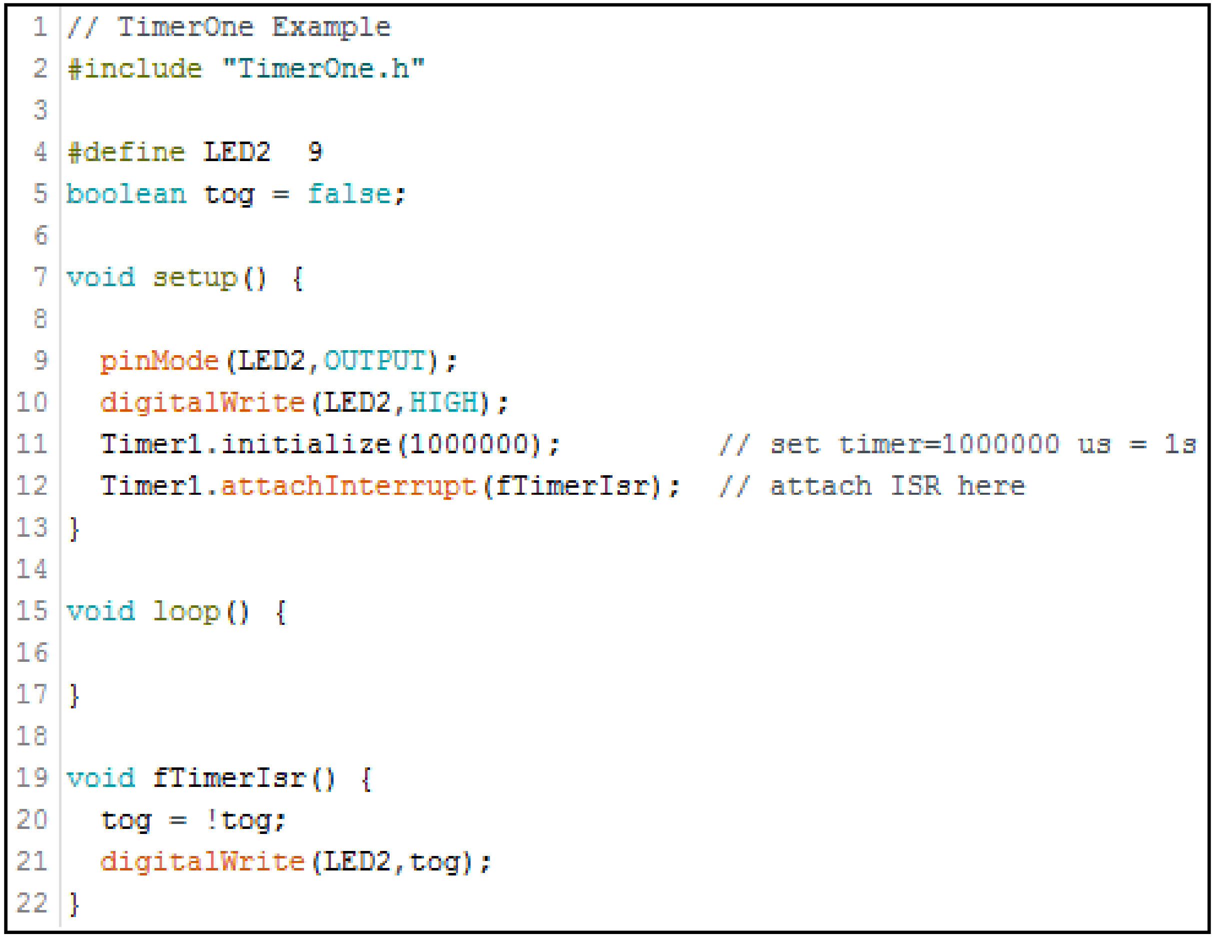
- Open your Arduino IDE.
- Click on Sketch tab >> Include Library >> Add .ZIP Library
- Then locate your library zip file and click enter.
- Done!

To remove a library, just delete its folder from its physical location.

Timer

Example 1:

Please add TimerOne library (TimerOne-master.zip) to your Arduino IDE’s library and try this code.



```
// TimerOne Example
#include "TimerOne.h"

#define LED2  9
boolean tog = false;

void setup() {

  pinMode(LED2,OUTPUT);
  digitalWrite(LED2,HIGH);
  Timer1.initialize(1000000);         // set timer=1000000 us = 1s
  Timer1.attachInterrupt(fTimerIsr);  // attach ISR here
}

void loop() {

}

void fTimerIsr() {
  tog = !tog;
  digitalWrite(LED2,tog);
}
```

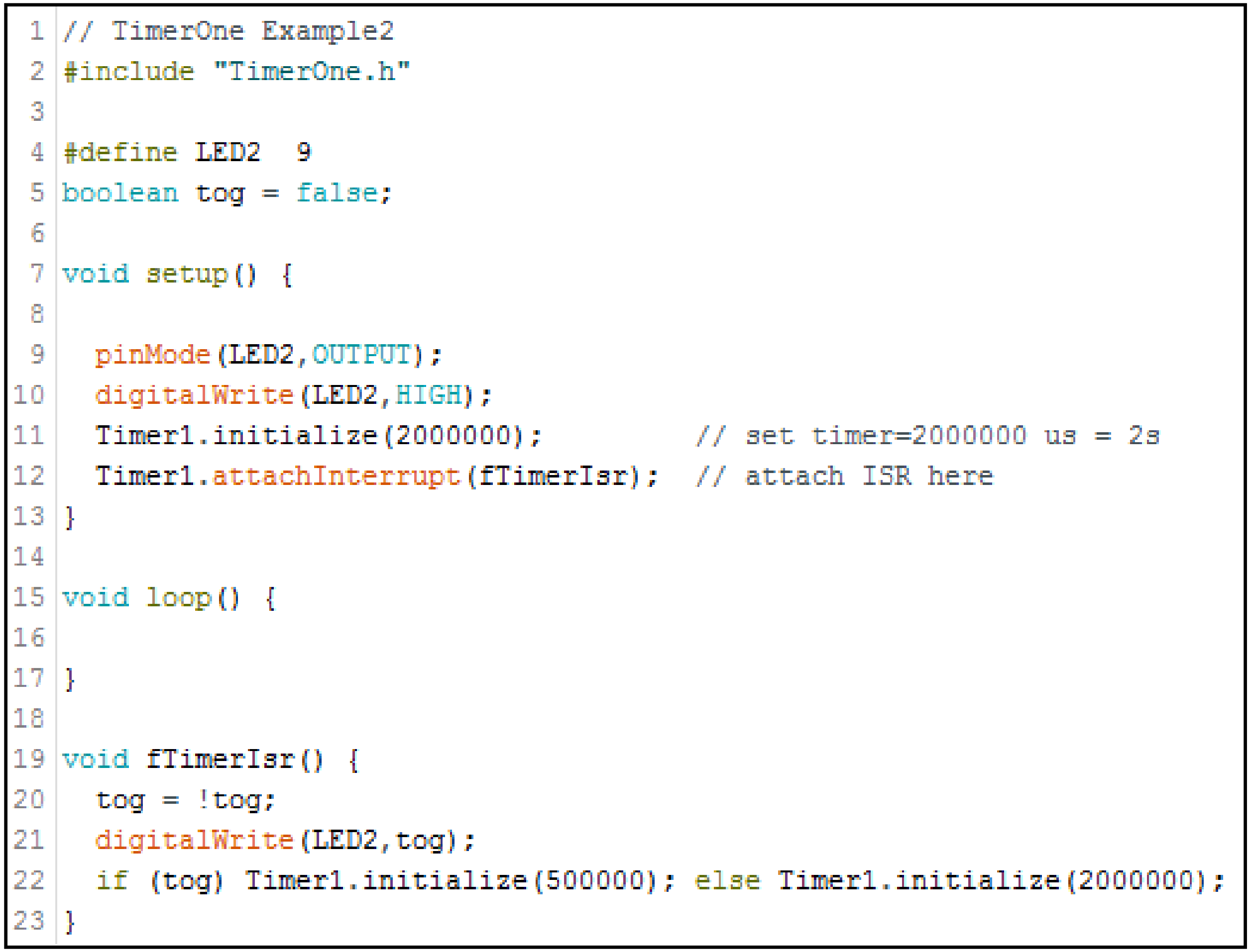
Timer

Example 2:

Modify the previous example like this>>

What can you see?

Try other values.



```
// TimerOne Example2
#include "TimerOne.h"

#define LED2  9
boolean tog = false;

void setup() {

  pinMode(LED2,OUTPUT);
  digitalWrite(LED2,HIGH);
  Timer1.initialize(2000000);         // set timer=2000000 us = 2s
  Timer1.attachInterrupt(fTimerIsr);  // attach ISR here
}

void loop() {

}

void fTimerIsr() {
  tog = !tog;
  digitalWrite(LED2,tog);
  if (tog) Timer1.initialize(500000); else Timer1.initialize(2000000);
}
```

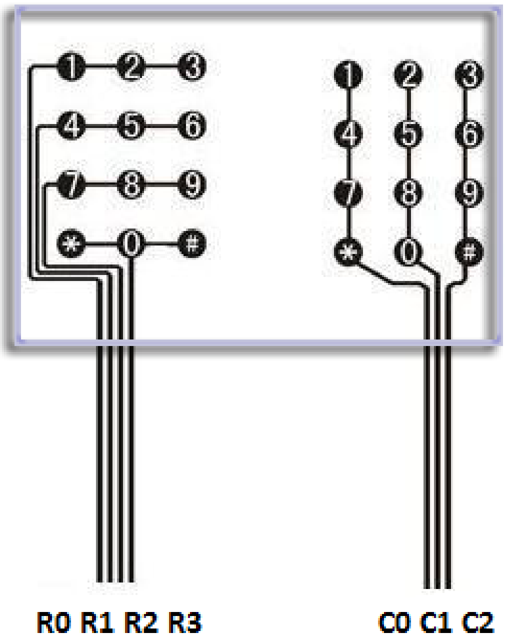
Keypad

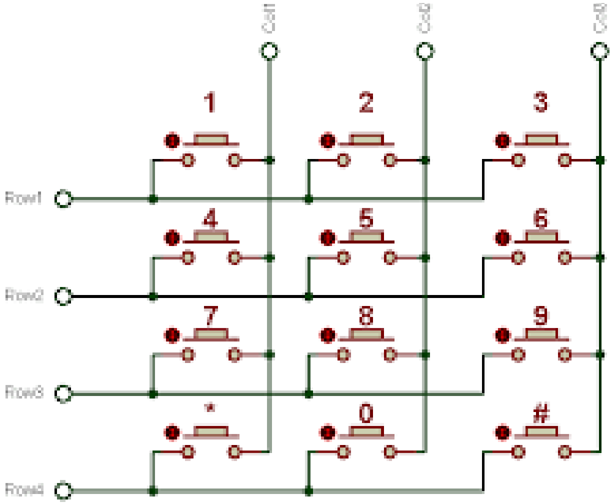
Pinout and Connections to Arduino Uno:

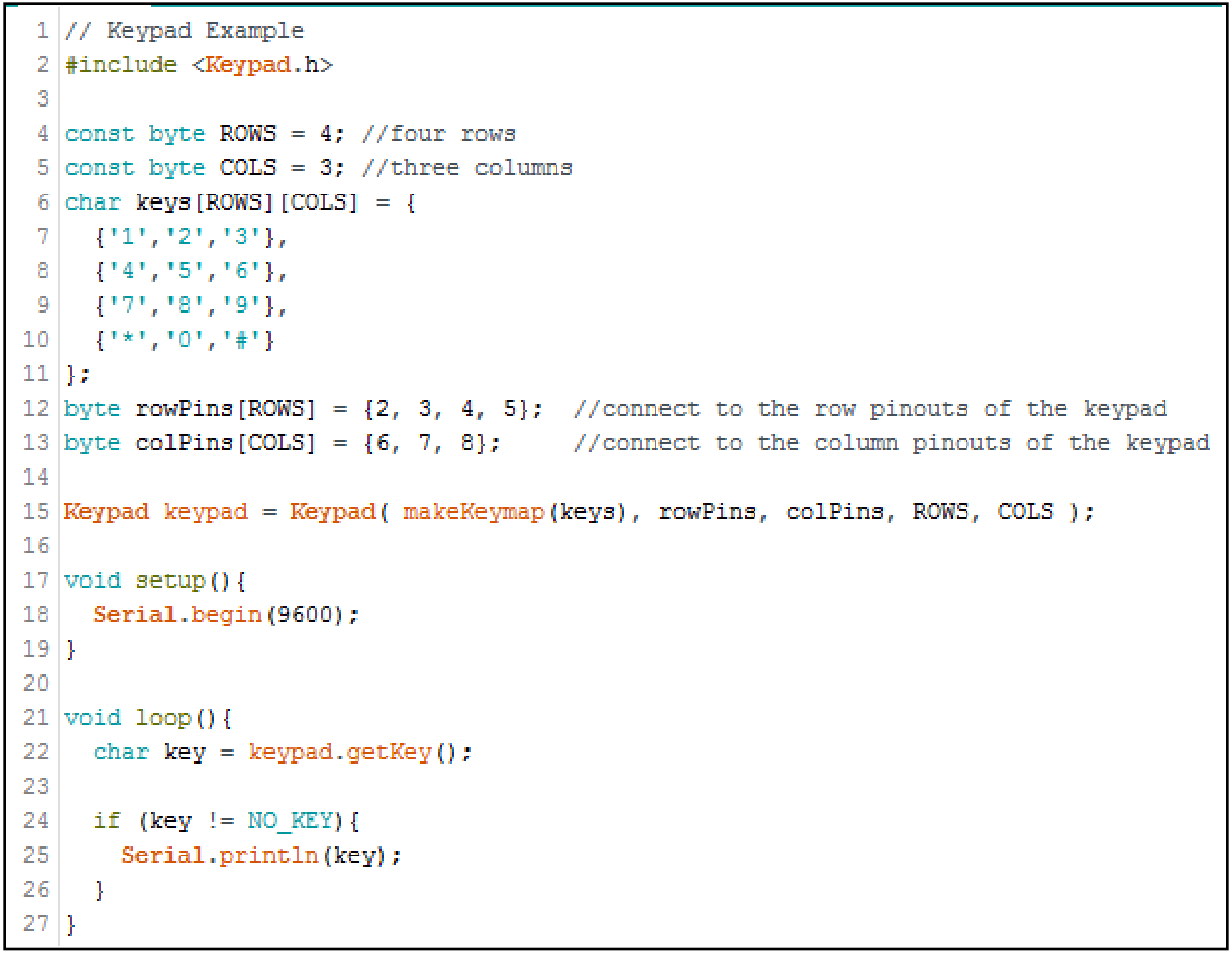
R0 = Row0 = D2
R1 = Row1 = D3
R2 = Row2 = D4
R3 = Row3 = D5

C0 = Column0 = D6
C1 = Column1 = D7
C2 = Column2 = D8

Add Keypad library and see next page.

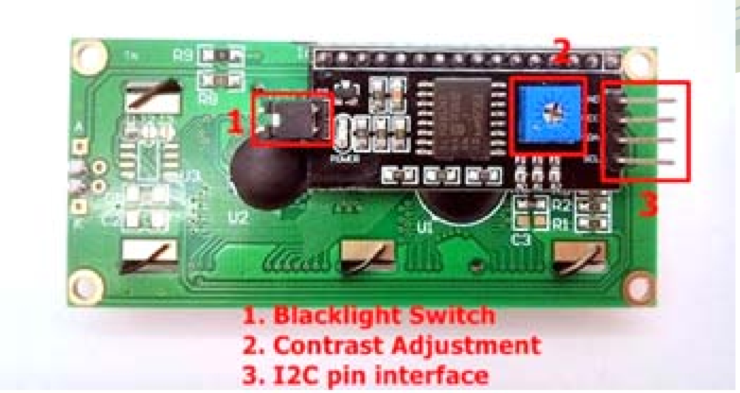






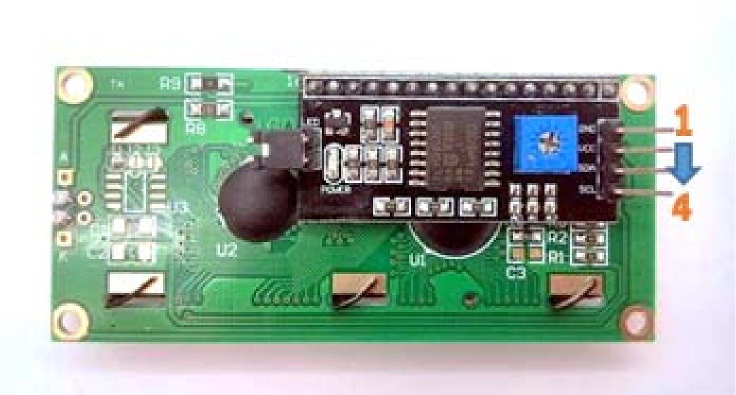
```cpp
// Keypad Example
#include <Keypad.h>

const byte ROWS = 4; //four rows
const byte COLS = 3; //three columns
char keys[ROWS][COLS] = {
  {'1','2','3'},
  {'4','5','6'},
  {'7','8','9'},
  {'*','0','#'}
};
byte rowPins[ROWS] = {2, 3, 4, 5};  //connect to the row pinouts of the keypad
byte colPins[COLS] = {6, 7, 8};     //connect to the column pinouts of the keypad

Keypad keypad = Keypad( makeKeymap(keys), rowPins, colPins, ROWS, COLS );

void setup(){
  Serial.begin(9600);
}

void loop(){
  char key = keypad.getKey();

  if (key != NO_KEY){
    Serial.println(key);
  }
}
```

LCD via I2C



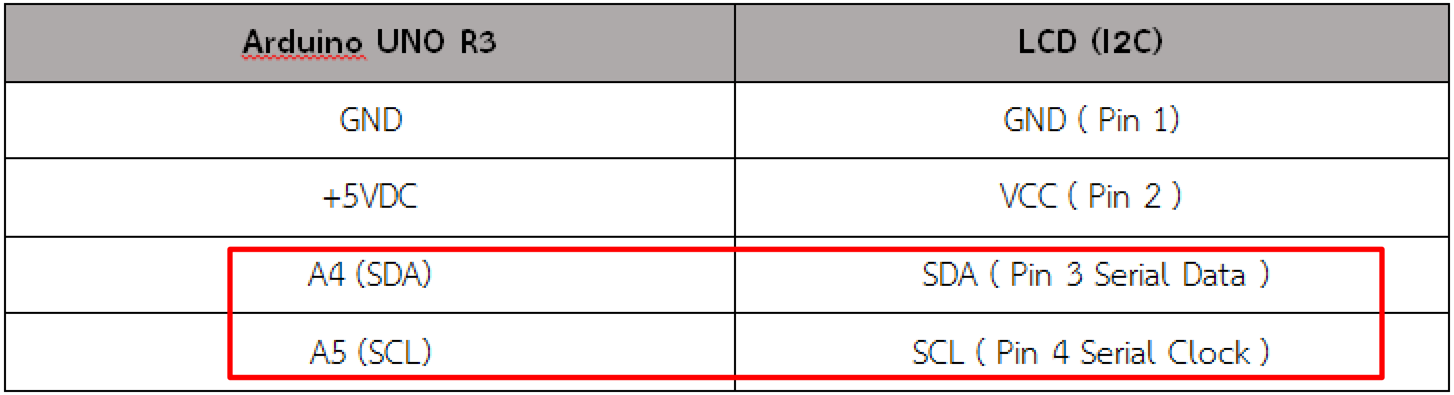
I2C Interface:

- **SDA** – Serial Data
- **SCL** – Serial Clock



Be careful !

Pinout and Connections:



Arduino UNO R3	LCD (I2C)
GND	GND (Pin 1)
+5VDC	VCC (Pin 2)
A4 (SDA)	SDA (Pin 3 Serial Data)
A5 (SCL)	SCL (Pin 4 Serial Clock)

LCD via I2C

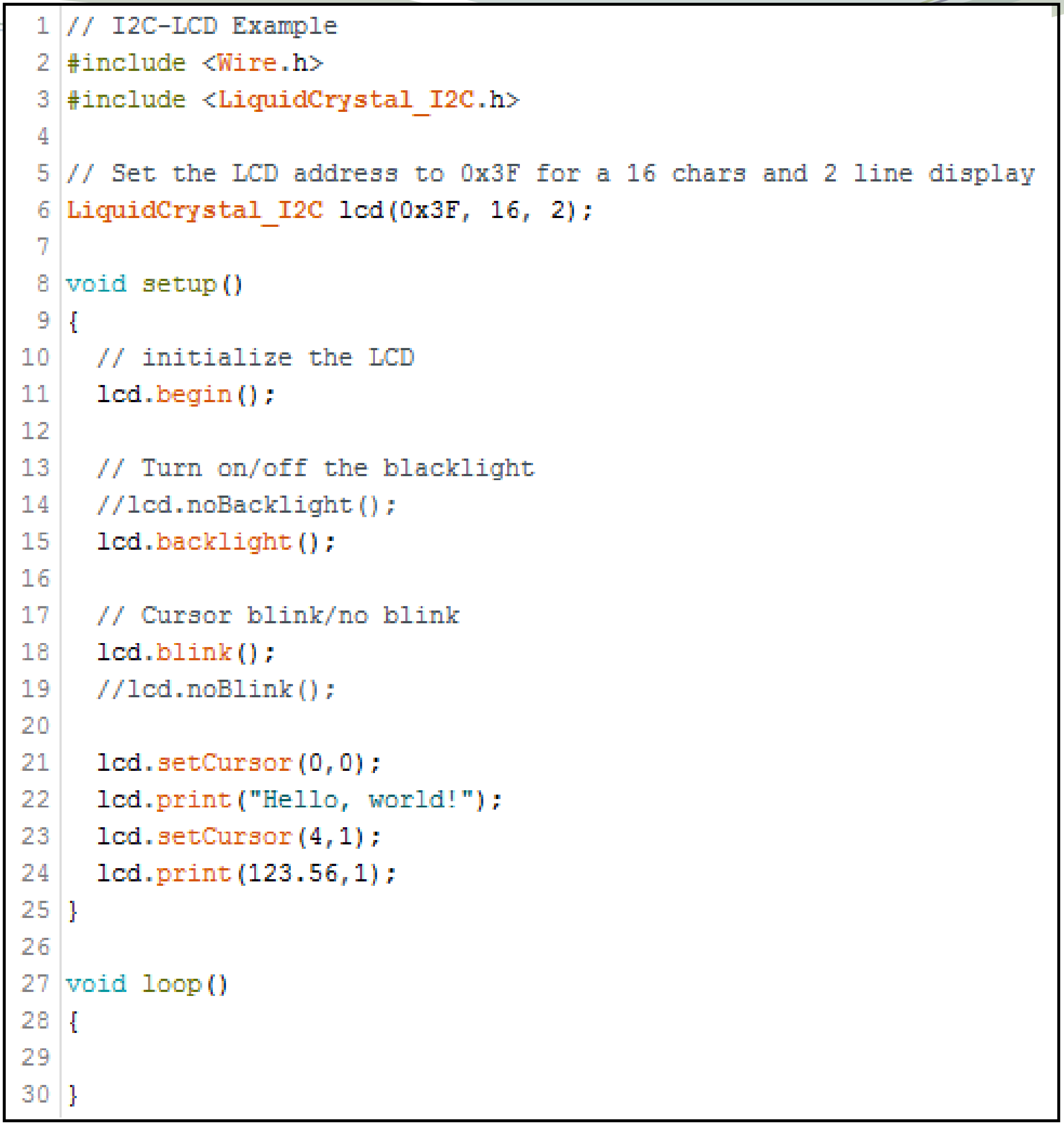
Add and use “LiquidCrystal_I2C” library. Useful commands are given below:

```
LiquidCrystal_I2C lcd(0x3F, 16, 2);       /* Set the LCD address to 0x3F for a 16 chars
                                             and 2 line display */
.begin();         // Set LCD to the begin state, must be called before any others.
.clear();         // Remove all the characters currently shown.
.home();          // Next print/write operation will start from the first position of LCD.
.noDisplay();  // Turn off display
.display();       // Turn on display, used only after noDisplay has been used.
.noBlink();       //  No cursor blink.
.blink();         // Cursor blink
.noCursor();   // No cursor
.cursor();        // Show cursor
.setCursor(column,line);        // Set position of cursor
```

LCD via I2C

Example:

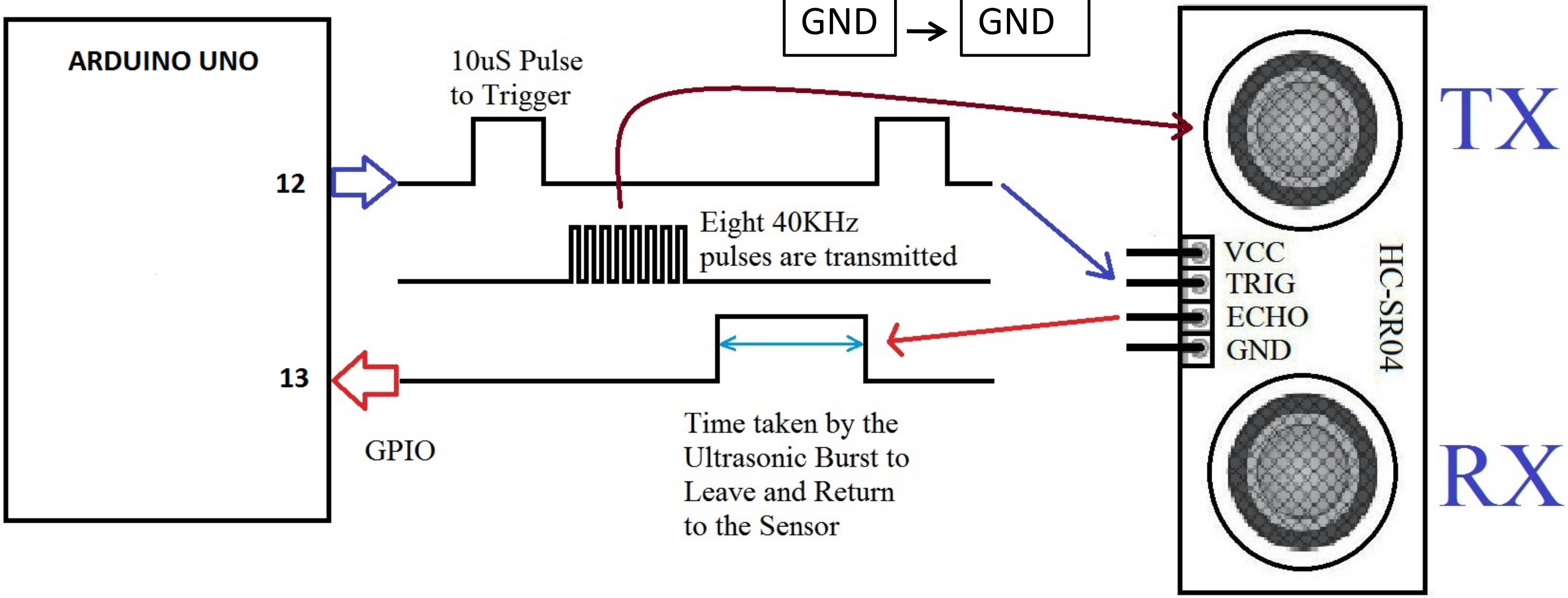
You may try to display something!



```
// I2C-LCD Example
#include <Wire.h>
#include <LiquidCrystal_I2C.h>

// Set the LCD address to 0x3F for a 16 chars and 2 line display
LiquidCrystal_I2C lcd(0x3F, 16, 2);

void setup()
{
  // initialize the LCD
  lcd.begin();

  // Turn on/off the blacklight
  //lcd.noBacklight();
  lcd.backlight();

  // Cursor blink/no blink
  lcd.blink();
  //lcd.noBlink();

  lcd.setCursor(0,0);
  lcd.print("Hello, world!");
  lcd.setCursor(4,1);
  lcd.print(123.56,1);
}

void loop()
{

}
```

Ultrasonic Range Finder (HC-SR04)

Arduino		HC-SR04
5V	→	VCC
12	→	TRIG
13	→	ECHO
GND	→	GND

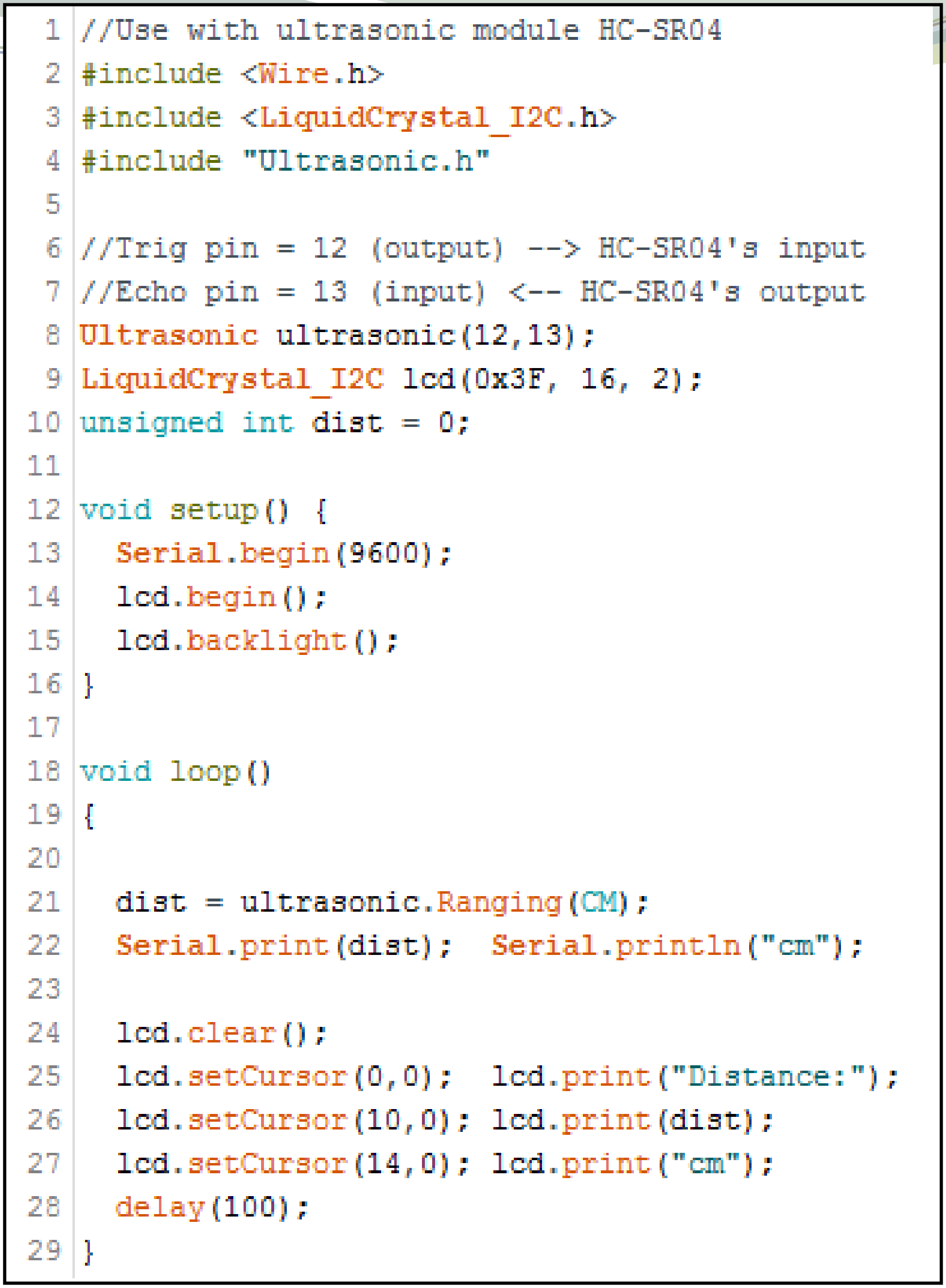


Use “**Ultrasonic**” library.

Example:

This code is to measure distances.

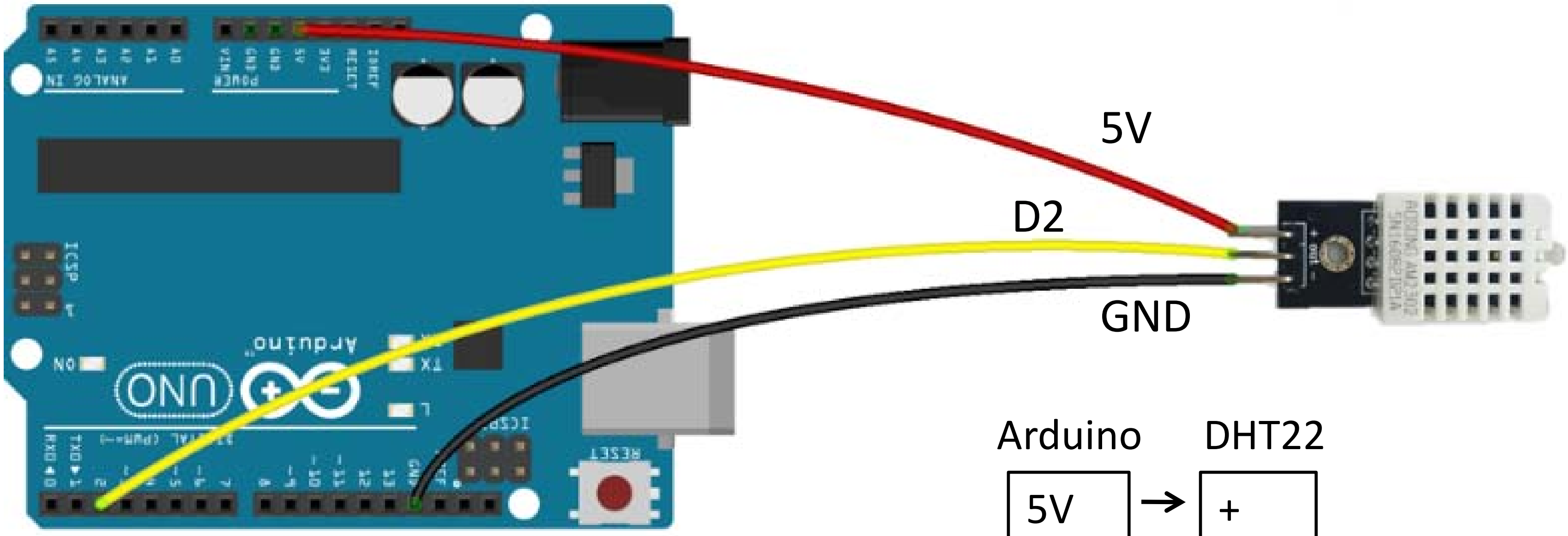
Can you convert the unit **cm** into **m**?



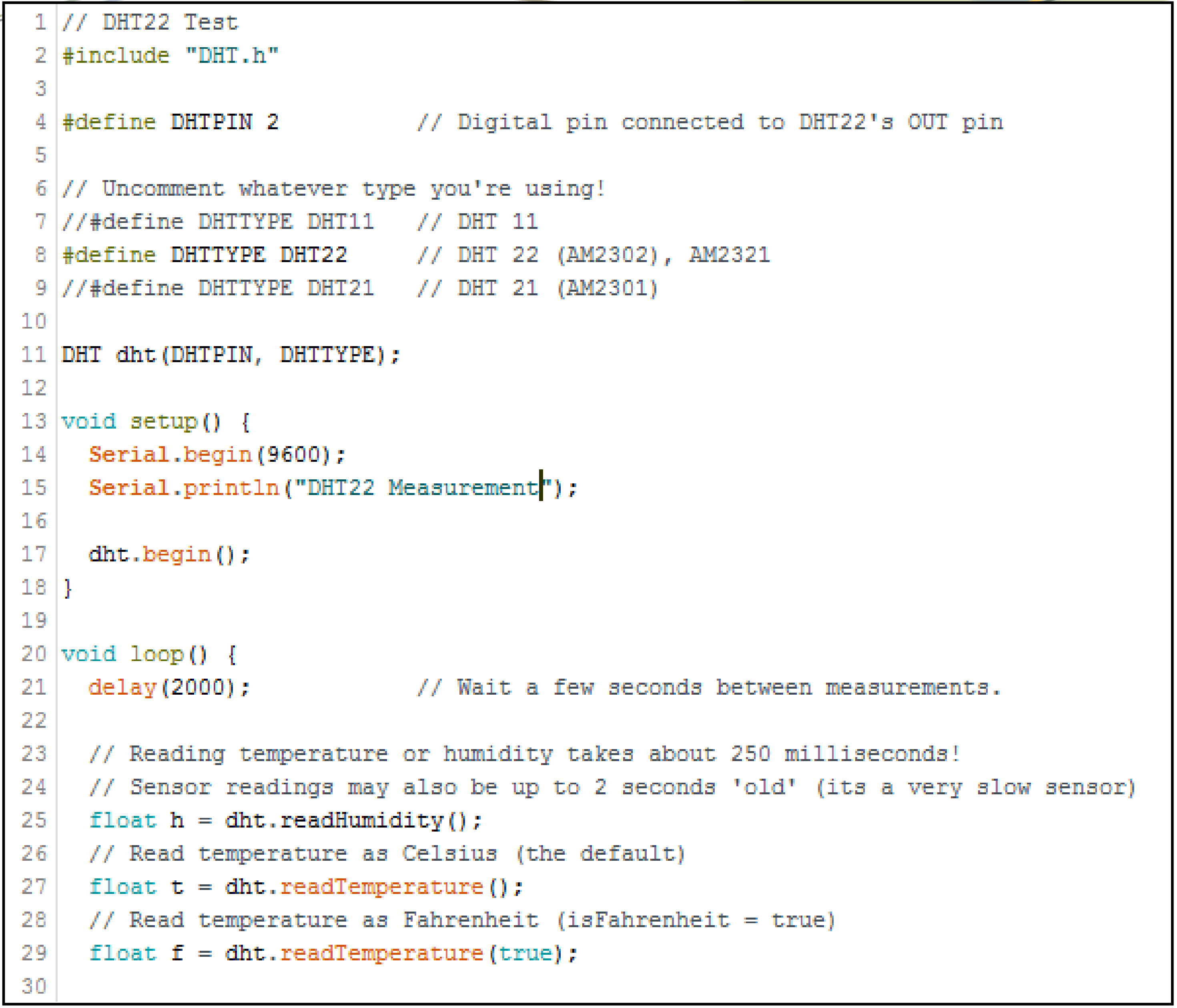
```
//Use with ultrasonic module HC-SR04
#include <Wire.h>
#include <LiquidCrystal_I2C.h>
#include "Ultrasonic.h"

//Trig pin = 12 (output) --> HC-SR04's input
//Echo pin = 13 (input) <-- HC-SR04's output
Ultrasonic ultrasonic(12,13);
LiquidCrystal_I2C lcd(0x3F, 16, 2);
unsigned int dist = 0;

void setup() {
  Serial.begin(9600);
  lcd.begin();
  lcd.backlight();
}

void loop()
{

  dist = ultrasonic.Ranging(CM);
  Serial.print(dist);  Serial.println("cm");

  lcd.clear();
  lcd.setCursor(0,0);  lcd.print("Distance:");
  lcd.setCursor(10,0); lcd.print(dist);
  lcd.setCursor(14,0); lcd.print("cm");
  delay(100);
}
```

Humid & Temp Sensor (DHT22)

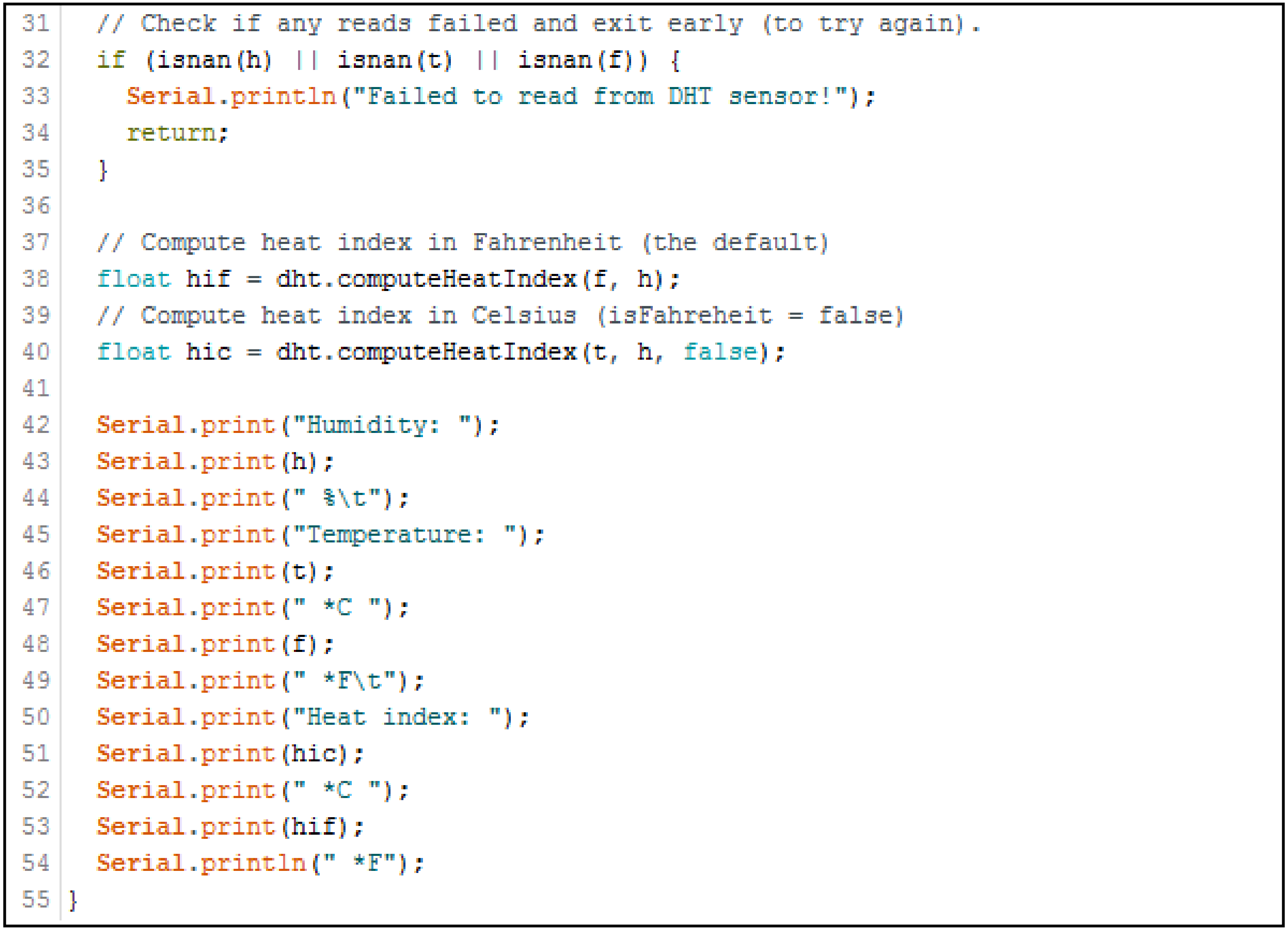
Make hardware connections like the below picture and add **DHT** library into your Arduino IDE.



Arduino		DHT22
5V	→	+
D2	→	OUT
GND	→	-



```
// DHT22 Test
#include "DHT.h"

#define DHTPIN 2          // Digital pin connected to DHT22's OUT pin

// Uncomment whatever type you're using!
//#define DHTTYPE DHT11   // DHT 11
#define DHTTYPE DHT22     // DHT 22 (AM2302), AM2321
//#define DHTTYPE DHT21   // DHT 21 (AM2301)

DHT dht(DHTPIN, DHTTYPE);

void setup() {
  Serial.begin(9600);
  Serial.println("DHT22 Measurement");

  dht.begin();
}

void loop() {
  delay(2000);            // Wait a few seconds between measurements.

  // Reading temperature or humidity takes about 250 milliseconds!
  // Sensor readings may also be up to 2 seconds 'old' (its a very slow sensor)
  float h = dht.readHumidity();
  // Read temperature as Celsius (the default)
  float t = dht.readTemperature();
  // Read temperature as Fahrenheit (isFahrenheit = true)
  float f = dht.readTemperature(true);

```

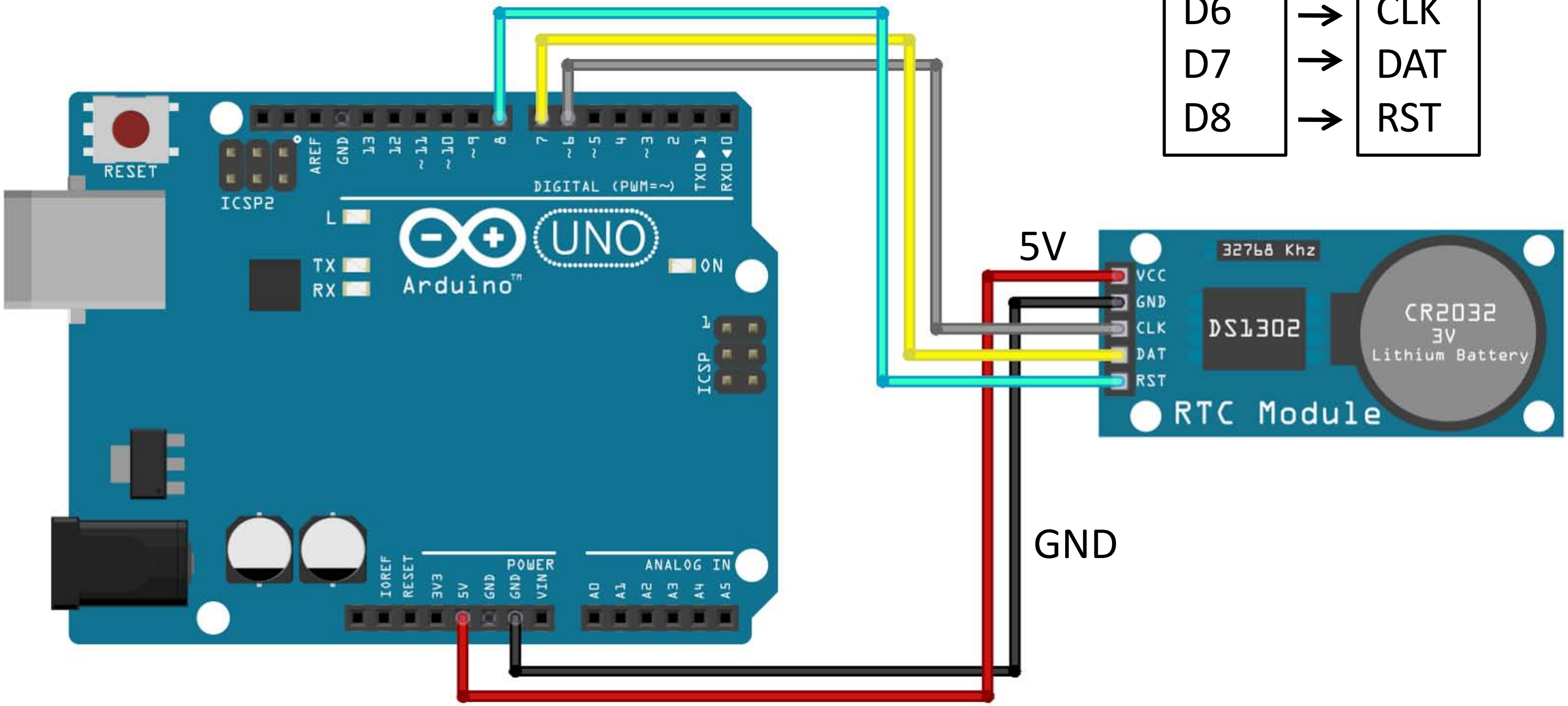


```
  // Check if any reads failed and exit early (to try again).
  if (isnan(h) || isnan(t) || isnan(f)) {
    Serial.println("Failed to read from DHT sensor!");
    return;
  }

  // Compute heat index in Fahrenheit (the default)
  float hif = dht.computeHeatIndex(f, h);
  // Compute heat index in Celsius (isFahreheit = false)
  float hic = dht.computeHeatIndex(t, h, false);

  Serial.print("Humidity: ");
  Serial.print(h);
  Serial.print(" %\t");
  Serial.print("Temperature: ");
  Serial.print(t);
  Serial.print(" *C ");
  Serial.print(f);
  Serial.print(" *F\t");
  Serial.print("Heat index: ");
  Serial.print(hic);
  Serial.print(" *C ");
  Serial.print(hif);
  Serial.println(" *F");
}
```

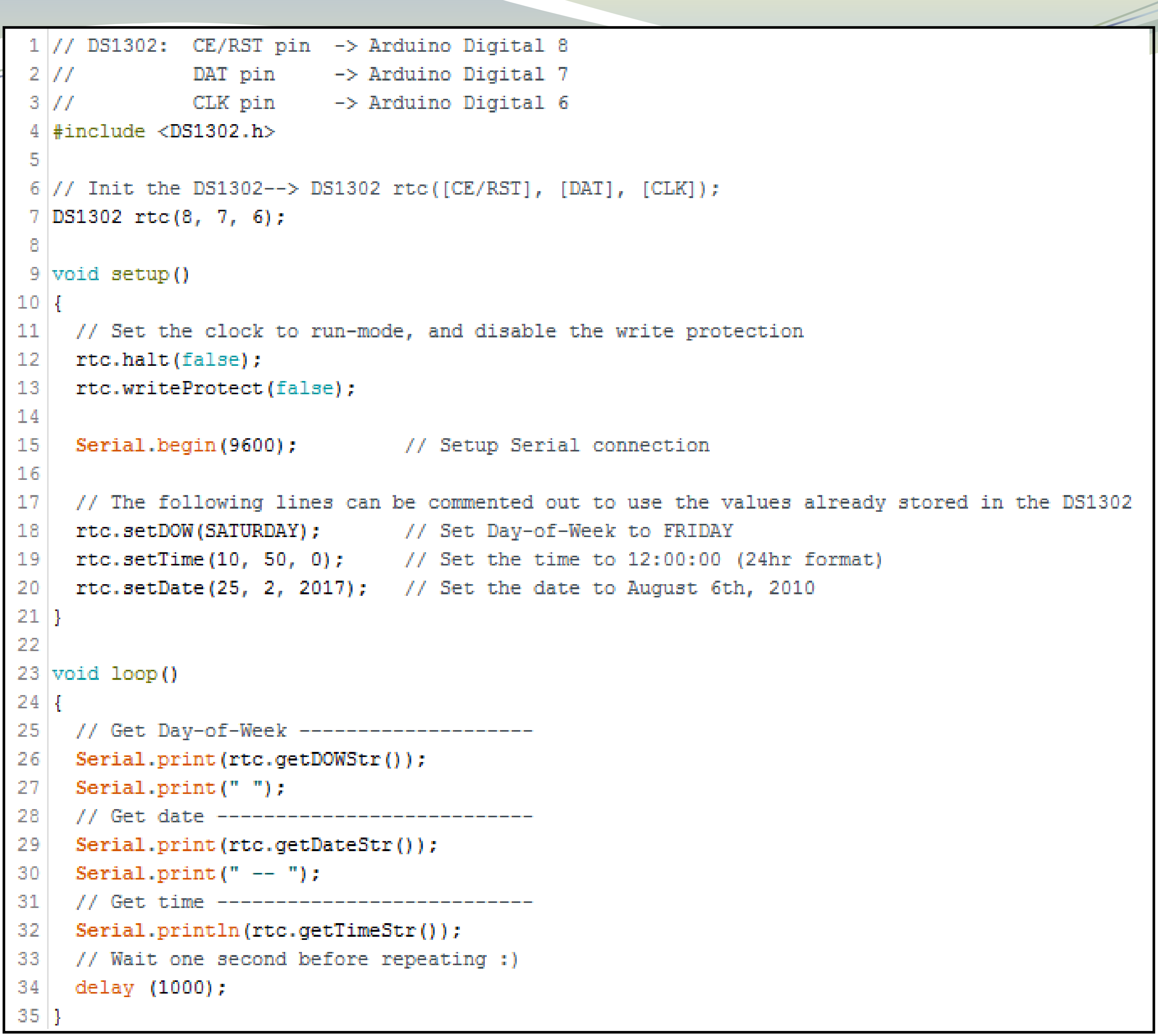
Real Time Clock (DS1302)

It performs as a clock as well as a calendar!

Arduino		DS1302
5V	→	VCC
GND	→	GND
D6	→	CLK
D7	→	DAT
D8	→	RST



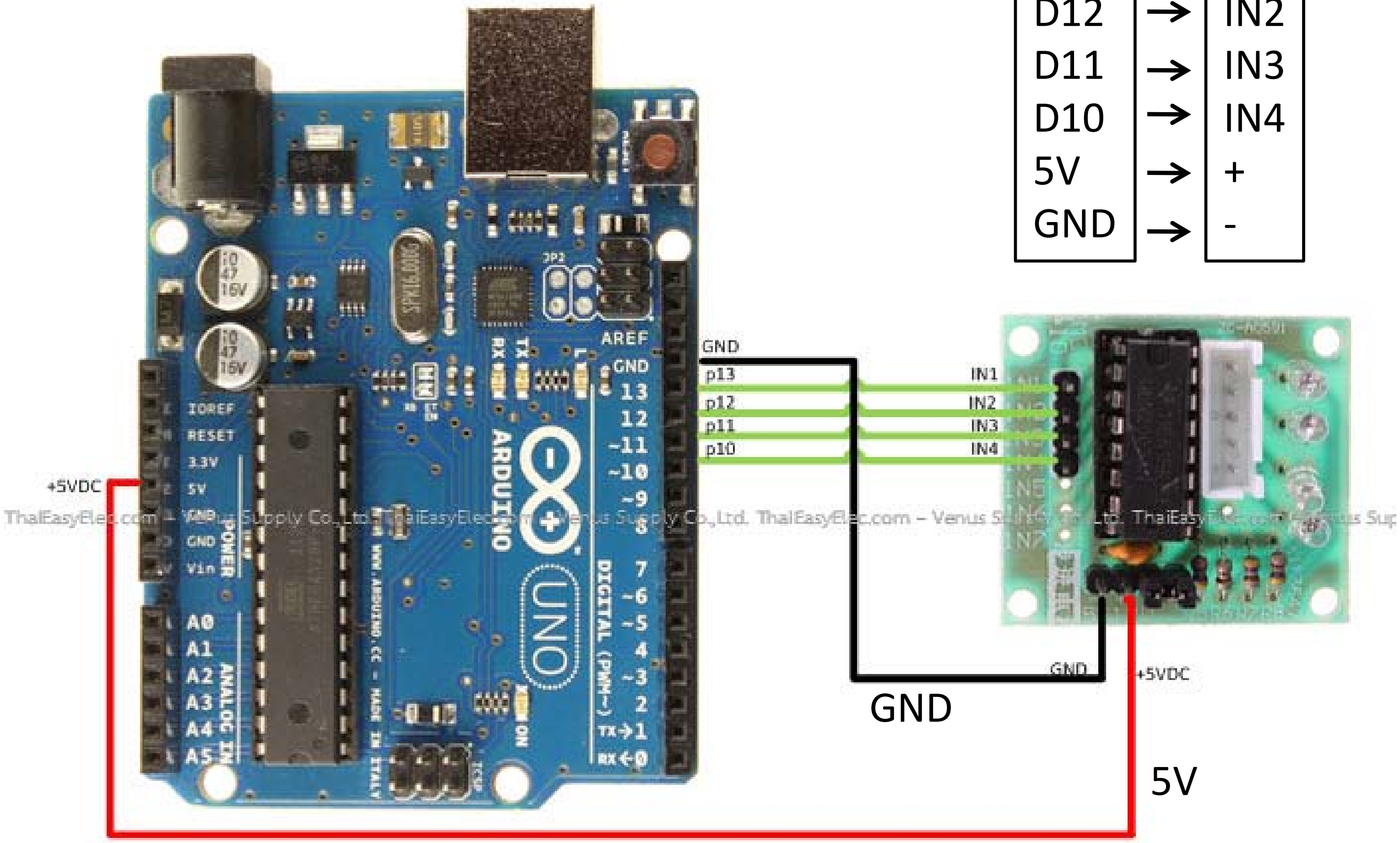
RTC



```
// DS1302:  CE/RST pin  -> Arduino Digital 8
//          DAT pin     -> Arduino Digital 7
//          CLK pin     -> Arduino Digital 6
#include <DS1302.h>

// Init the DS1302--> DS1302 rtc([CE/RST], [DAT], [CLK]);
DS1302 rtc(8, 7, 6);

void setup()
{
  // Set the clock to run-mode, and disable the write protection
  rtc.halt(false);
  rtc.writeProtect(false);

  Serial.begin(9600);         // Setup Serial connection

  // The following lines can be commented out to use the values already stored in the DS1302
  rtc.setDOW(SATURDAY);       // Set Day-of-Week to FRIDAY
  rtc.setTime(10, 50, 0);     // Set the time to 12:00:00 (24hr format)
  rtc.setDate(25, 2, 2017);   // Set the date to August 6th, 2010
}

void loop()
{
  // Get Day-of-Week --------------------
  Serial.print(rtc.getDOWStr());
  Serial.print(" ");
  // Get date ---------------------------
  Serial.print(rtc.getDateStr());
  Serial.print(" -- ");
  // Get time ---------------------------
  Serial.println(rtc.getTimeStr());
  // Wait one second before repeating :)
  delay (1000);
}
```

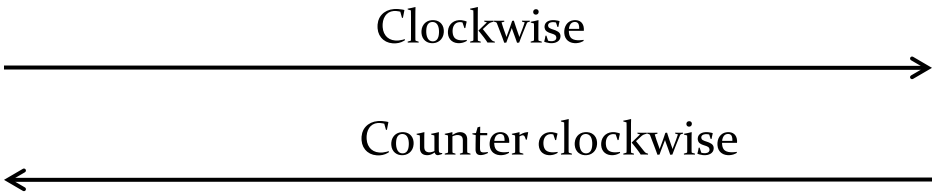
Stepper Motor

Arduino		Stepper Board
D13	→	IN1
D12	→	IN2
D11	→	IN3
D10	→	IN4
5V	→	+
GND	→	-



Stepper Motor

Phase	Step							
	0	1	2	3	4	5	6	7
0	L	L	L	L	L	H	H	H
1	L	L	L	H	H	H	L	L
2	L	H	H	H	L	L	L	L
3	H	H	L	L	L	L	L	H



Open “StepMotor_Ex1.ino” to see how it works!

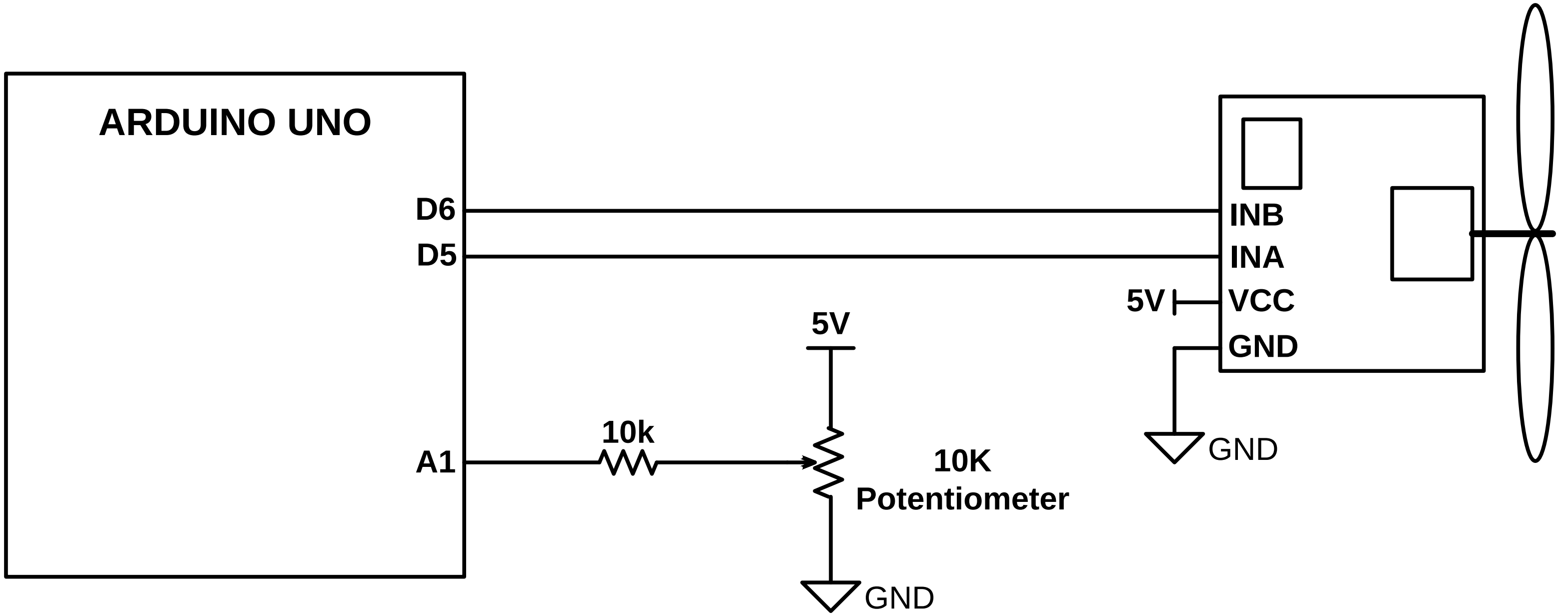
NOTE: the variable “dir” is to set the direction.

Brushed DC Motor

Use PWM to control its spinning speed! How about direction?

Counterclockwise:
INA → PWM
INB → LOW

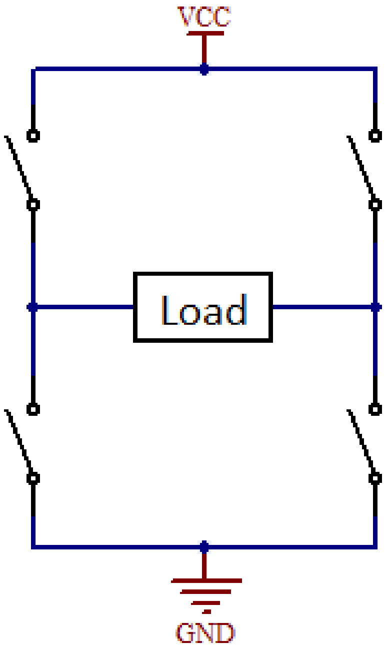
Clockwise:
INA → LOW
INB → PWM



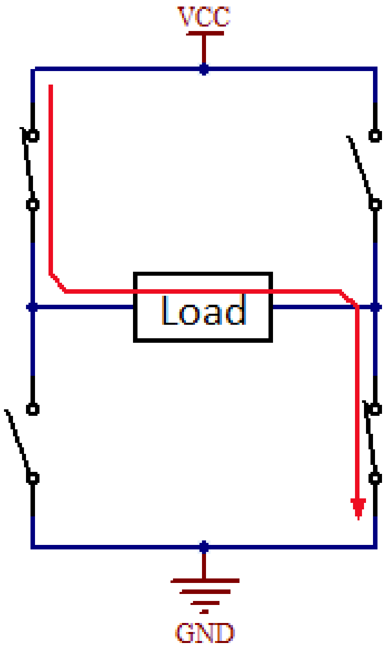
Brushed DC Motor

H-Bridge

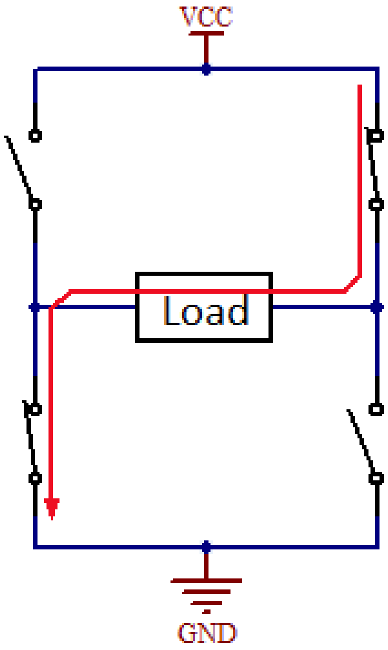
H bridge topology



Connecting the load
in one direction



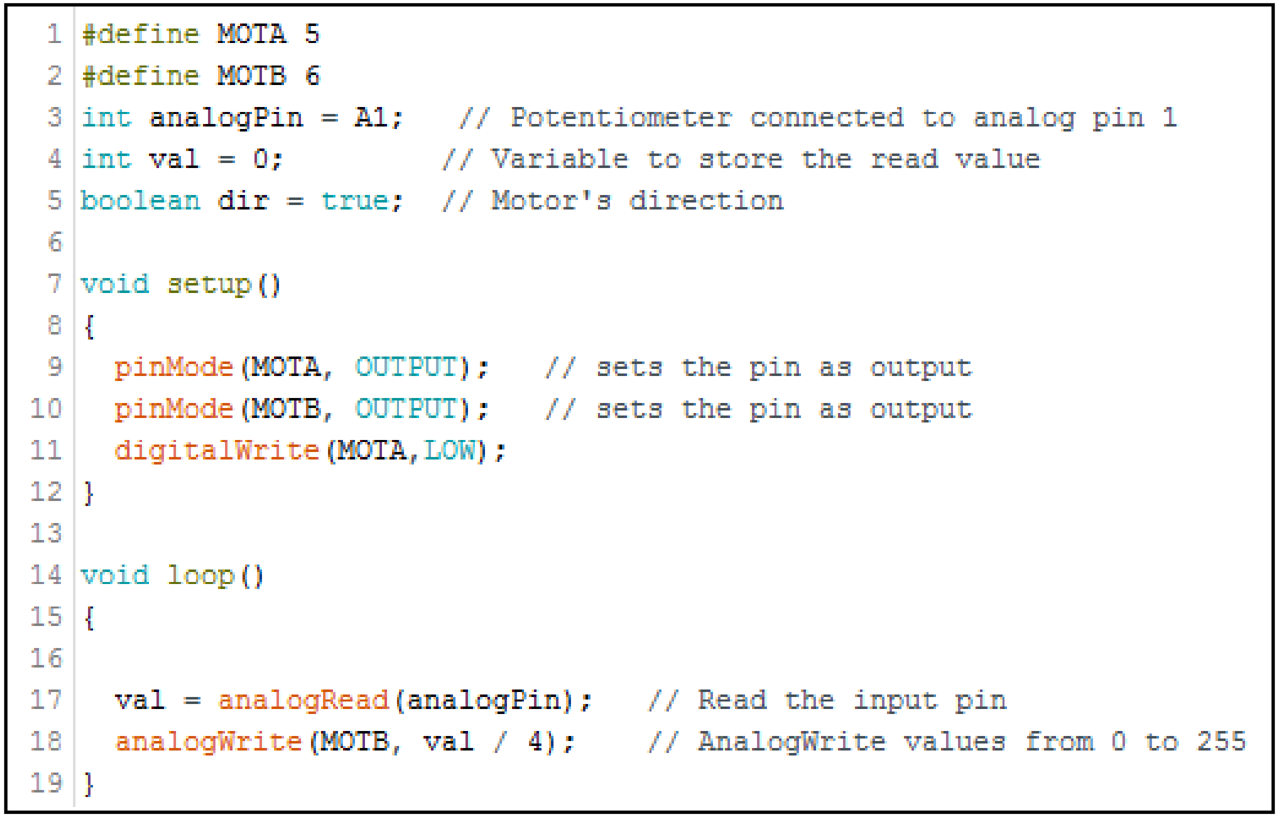
Connecting the load
in the other direction



Brushed DC Motor

Example:

Control the speed by turning the POT.

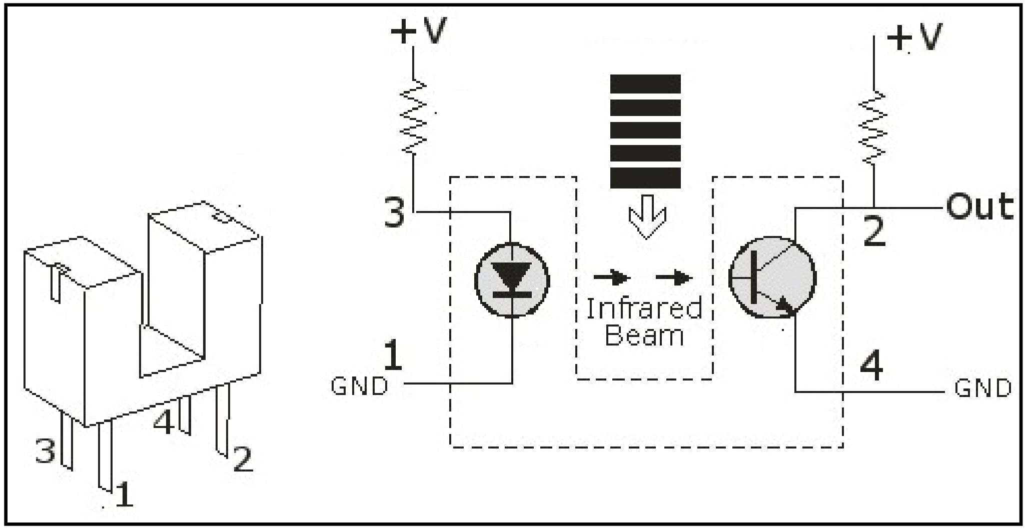


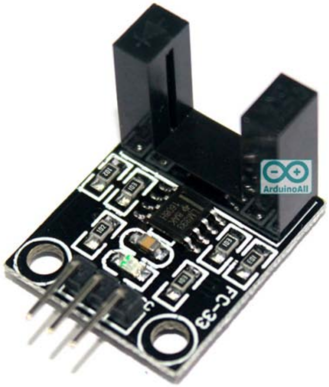
```
#define MOTA 5
#define MOTB 6
int analogPin = A1;   // Potentiometer connected to analog pin 1
int val = 0;         // Variable to store the read value
boolean dir = true;  // Motor's direction

void setup()
{
  pinMode(MOTA, OUTPUT);  // sets the pin as output
  pinMode(MOTB, OUTPUT);  // sets the pin as output
  digitalWrite(MOTA,LOW);
}

void loop()
{

  val = analogRead(analogPin);   // Read the input pin
  analogWrite(MOTB, val / 4);    // AnalogWrite values from 0 to 255
}
```

Photo Interrupter

Applications:

- Tape-end sensors
- Timing sensors
- Edge sensors
- copiers





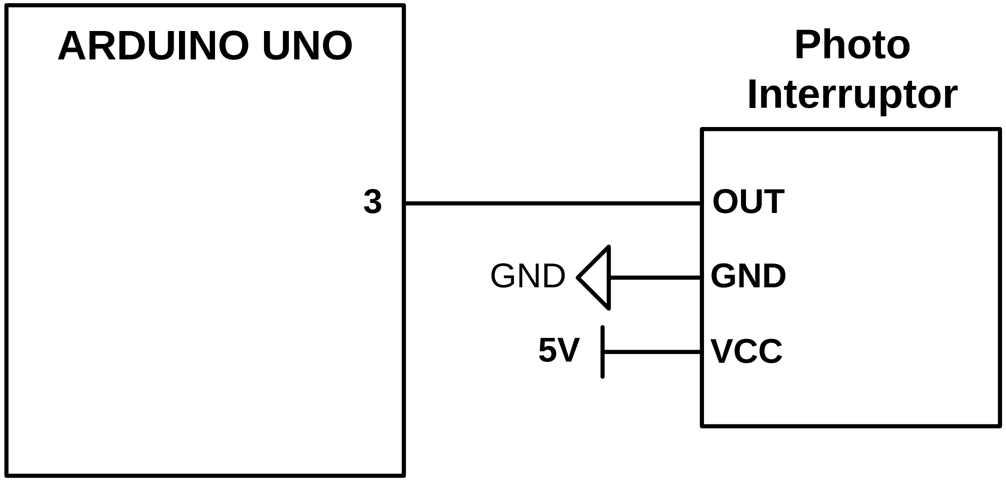
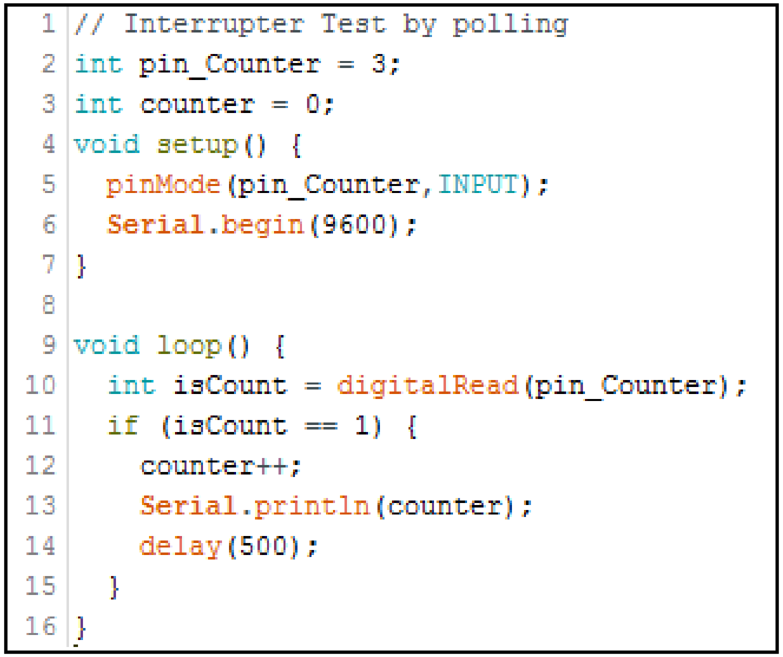


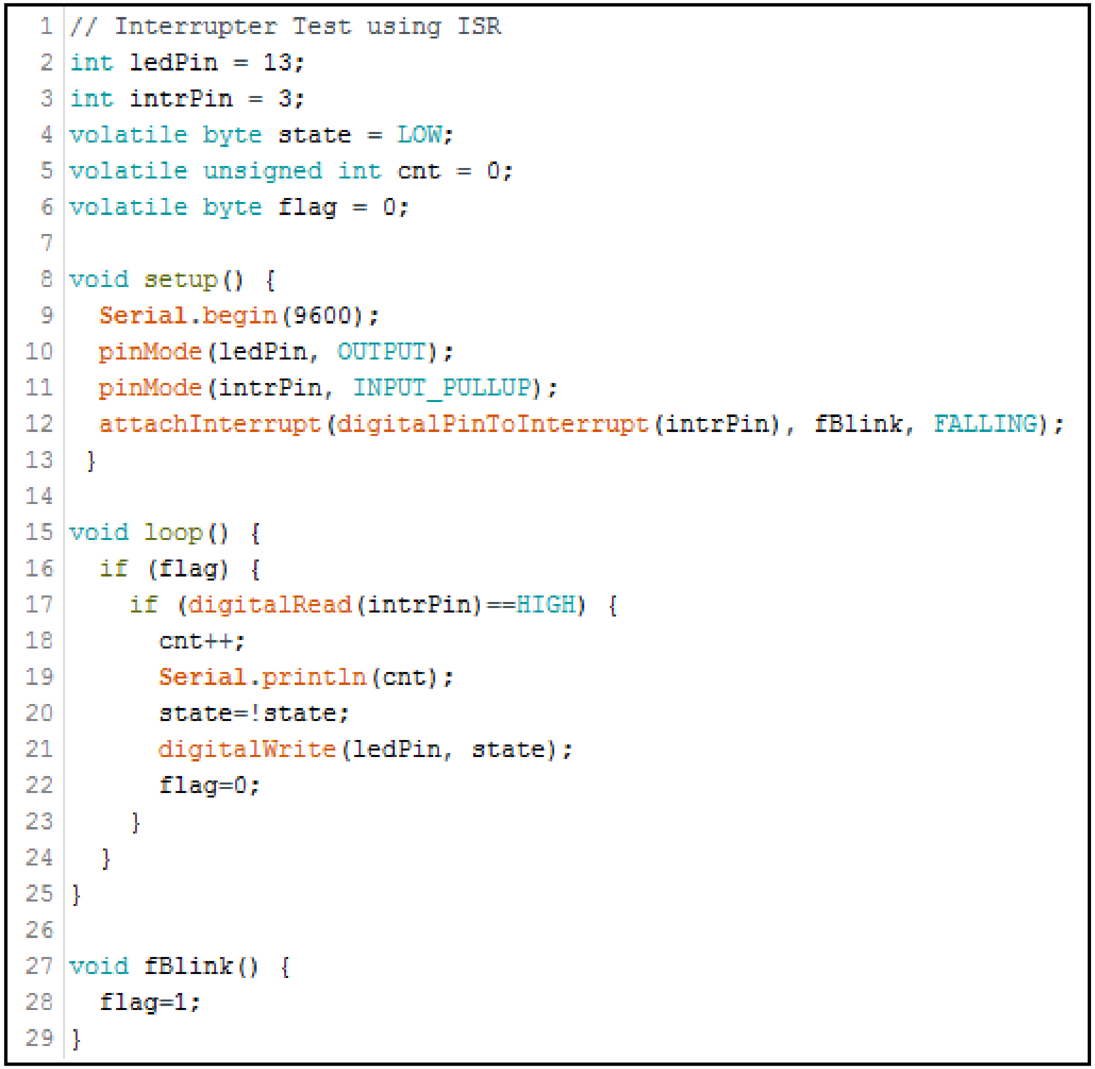
Photo Interrupter

Example 1



```
// Interrupter Test by polling
int pin_Counter = 3;
int counter = 0;
void setup() {
  pinMode(pin_Counter,INPUT);
  Serial.begin(9600);
}

void loop() {
  int isCount = digitalRead(pin_Counter);
  if (isCount == 1) {
    counter++;
    Serial.println(counter);
    delay(500);
  }
}
```

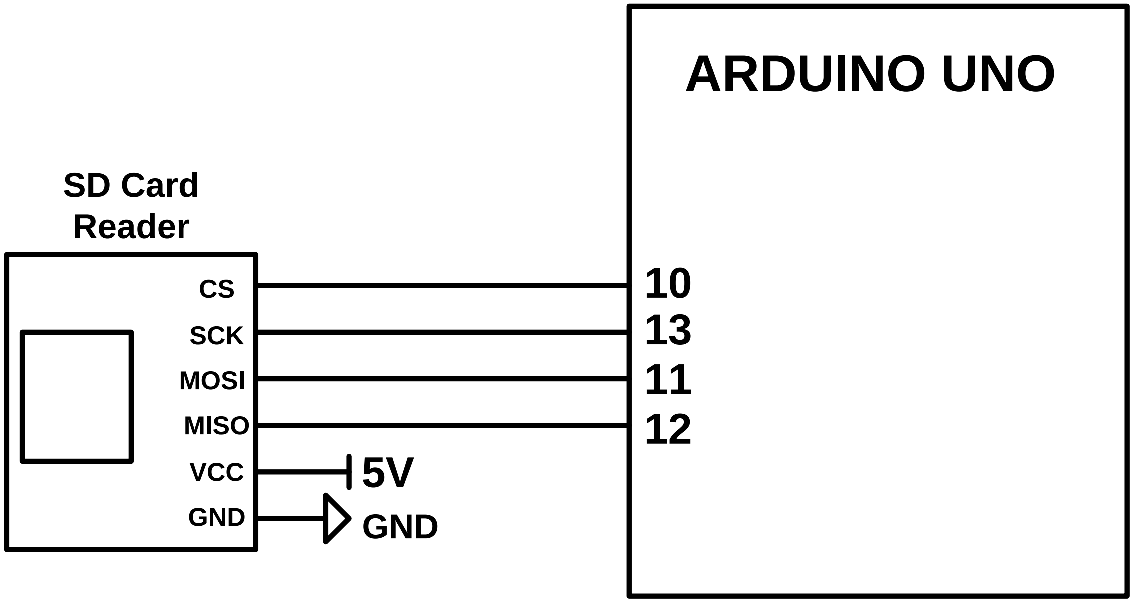
Example 2



```
// Interrupter Test using ISR
int ledPin = 13;
int intrPin = 3;
volatile byte state = LOW;
volatile unsigned int cnt = 0;
volatile byte flag = 0;

void setup() {
  Serial.begin(9600);
  pinMode(ledPin, OUTPUT);
  pinMode(intrPin, INPUT_PULLUP);
  attachInterrupt(digitalPinToInterrupt(intrPin), fBlink, FALLING);
 }

void loop() {
  if (flag) {
    if (digitalRead(intrPin)==HIGH) {
      cnt++;
      Serial.println(cnt);
      state=!state;
      digitalWrite(ledPin, state);
      flag=0;
    }
  }
}

void fBlink() {
  flag=1;
}
```

SD Card Reader

Make hardware connection as below. You also need to insert a micro SD card into the reader. Open and program SD_Ex1.ino into your Arduino board.

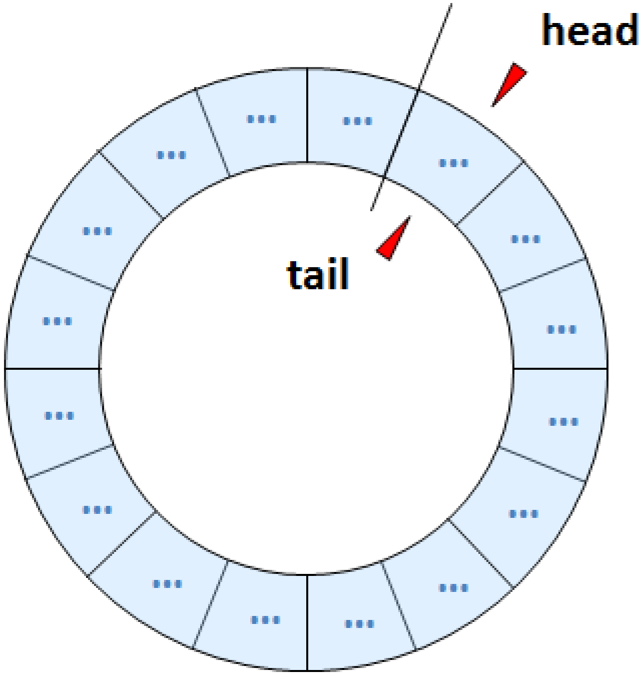


Ring Buffer

- Ring Buffer (or Circular Buffer)
  - Ring Buffer is a FIFO buffer.
  - The buffer has no real end and it can loop around the buffer. However, its memory is not physically a ring.
  - The ring buffer usually has two indices to the elements within the buffer. The distance between the indices can range from zero to the total number of elements within the buffer.
    - The use of the dual indices means the queue length can be from zero (empty) to the total number of elements (full).

Ring Buffer

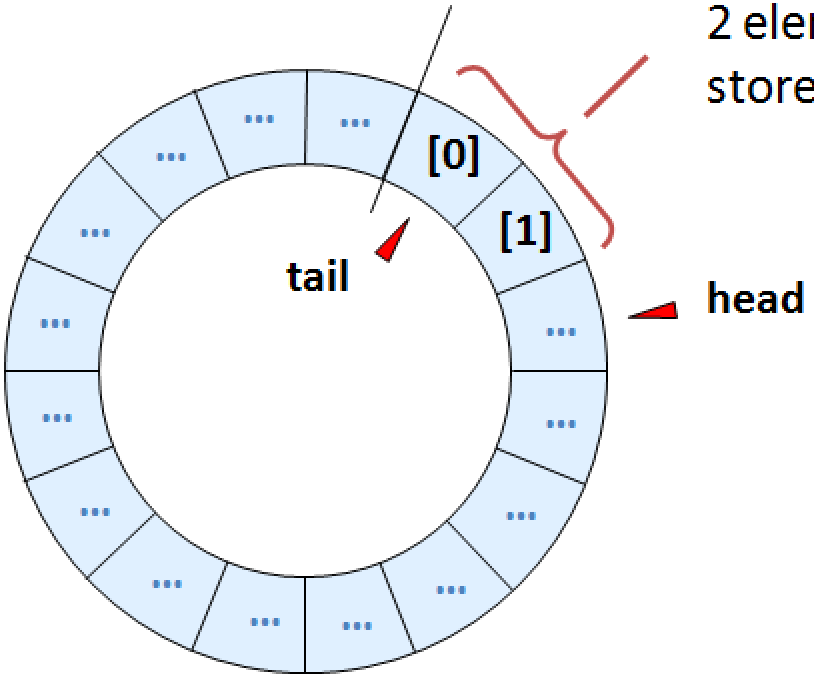
- Example:
  - 16 elements, empty.



The buffer is empty when:
tail = head

Ring Buffer

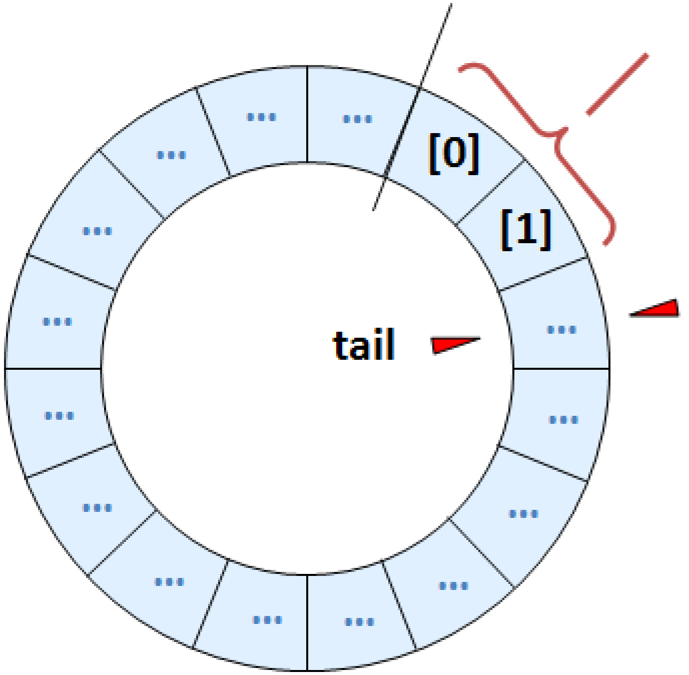
- Example:
    - After two write operations.



2 elements have been stored in the buffer.

Ring Buffer

- Example:
    - After two read operations.

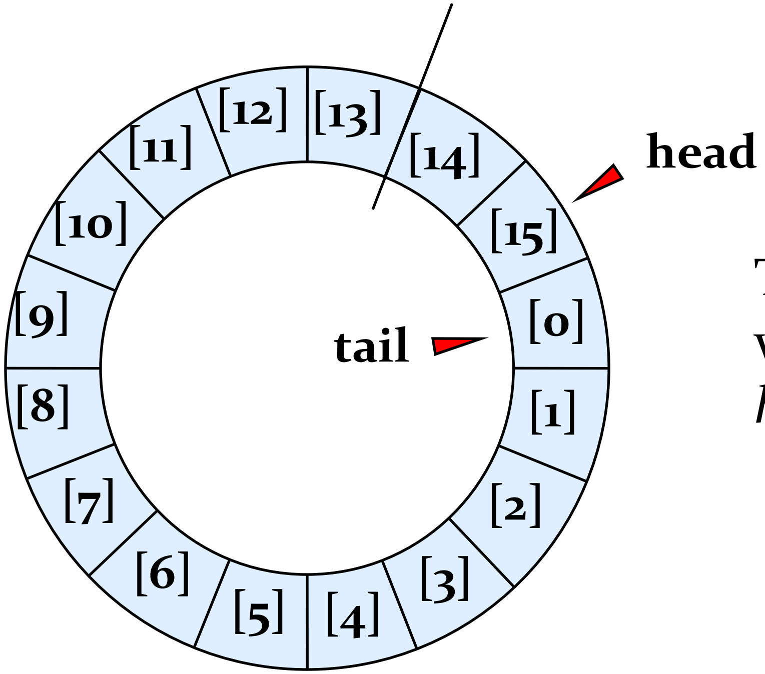


Read elements remain in the buffer. After that they can be overwritten.

(buffer empty)

Ring Buffer

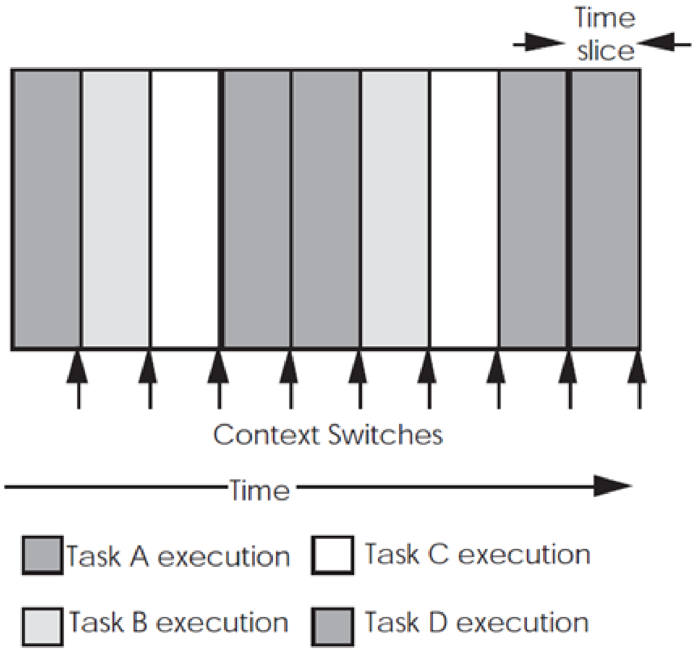
- Example:
  - After 16 write operations.



The buffer is full when:
head = tail – 1

Multitasking

- It is to run multiple tasks simultaneously.
- It works by dividing processor's time into <u>discrete time slots</u> → each application or task requires a certain number of time slots to complete its execution.
- Scheduler decides which task can have the next time slot.

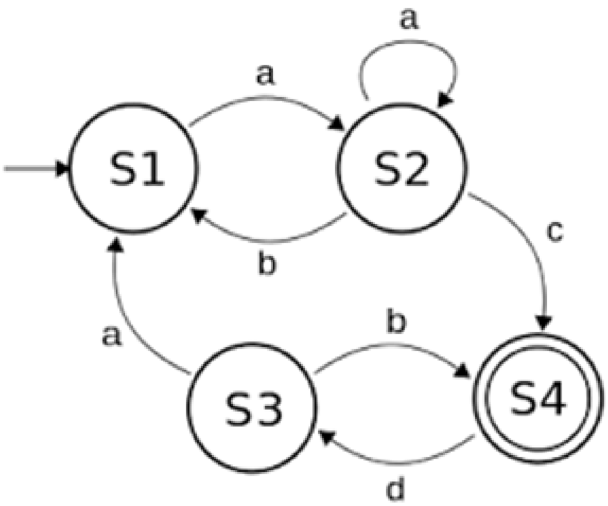


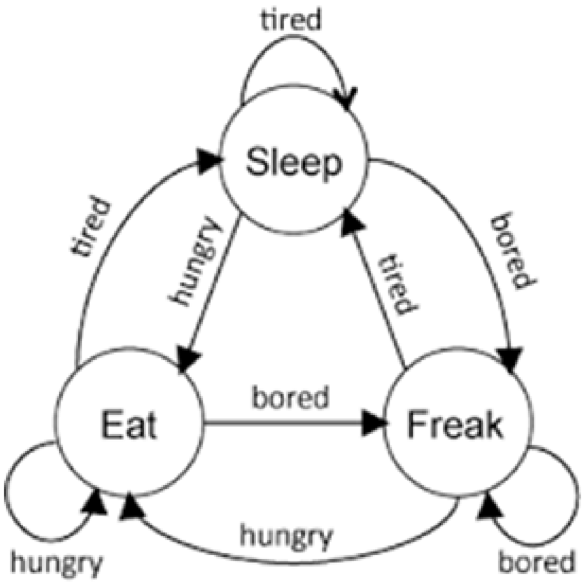
State Machines

- Finite State Machines
  - A finite state machine has a finite number of states.
  - It is a mathematical model of computation.
  - It is in only one state at a time → Current State.
  - It can change from one state to another when initiated by a trigger event/condition.
  - It changes state in such a way that the next state depends only on the current state and input.
  - Advanced study: automata, Markov models, hidden Markov models, etc. → very useful, can you believe this!

State Machines

Examples of Finite State Machines:



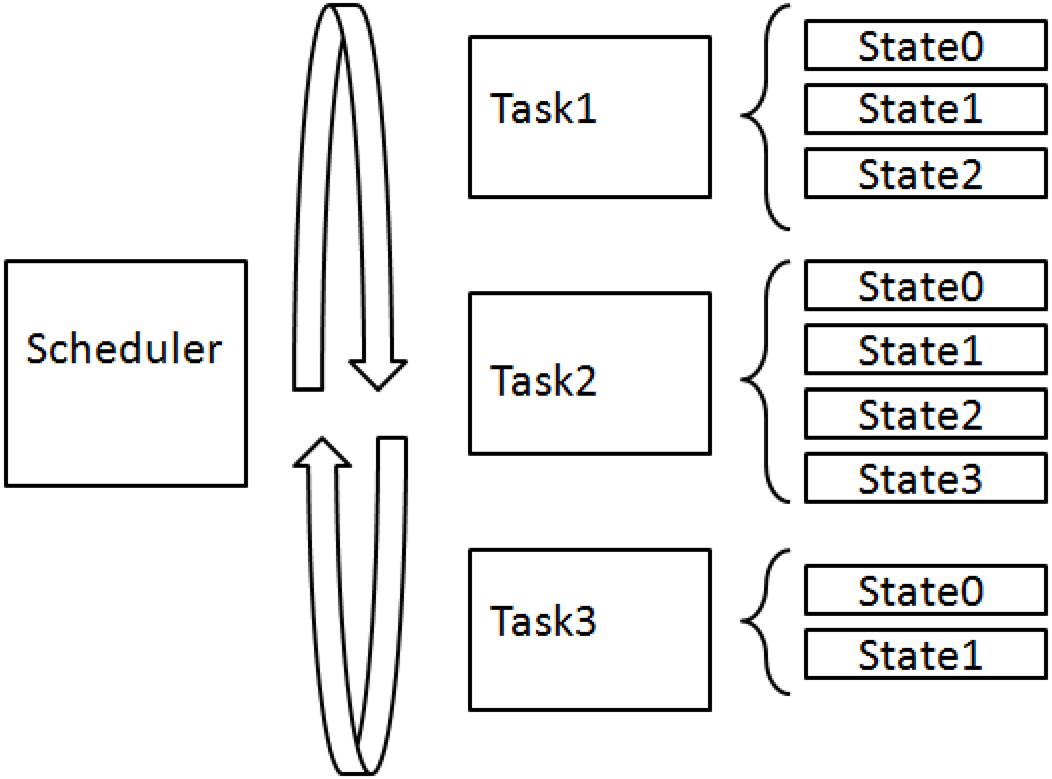


Scheduling

- A Simple Multitasking System
  - No priority.
  - The Scheduler:
    - Timer Interrupt → precise time slots, why?...................
    - Loop → variable time slots, why?..................................
  - Each task is divided into state machines according to the defined time slot.

Scheduling

- A Simple Non-Priority Scheduler



Scheduling

```
void setup() {
        fTask1Open();
        fTask2Open();
        fTask3Open();
}
```

```
void loop() {
        //- Scheduling --------
        fTask1Run();
        fTask2Run();
        fTask3Run();
}
```

```
void fTask1Run(void) {
        fTask1Entry(&m_task1);
}
```

```
static void fTask1Entry(tTask1 *m) {
        switch (m->state) {
                case 0 : fTask1State0(m);break;
                case 1 : fTask1State1(m);break;
                case 2 : fTask1State2(m);break;
                default : break;
        }
}
```

Scheduling

Header file “Mpublic.h”

```
#define tByte   unsigned char     // 8  Bits
#define tWord   unsigned int      // 16 Bits
#define tDWord  unsigned long     // 32 Bits

#define tFloat  float             // Real number

#define bNull   0xFF
#define wNull   0xFFFF
#define dNull   0xFFFFFFFF
#define pNull   NULL

boolean schZone;      /* Dividing 2 scheduling spaces
                      true = normal scheduling
                      false = timer scheduling */

//== Switch States =====================
#define sON    1      // Switch state=ON
#define sOFF   0      // Switch state=OFF
#define sIDL   bNull  // Switch state=IDLE
```

Scheduling

```
tByte fTask1State0(tTask1 *m) {
            *LED = On;
            m->cnt=0;
            m->led=On;
            m->state=1;
            return 0;
}

tByte fTask1State1(tTask1 *m) {
            m->cnt++;
            if (m->cnt>10000) {
                  if (m->led=On) m->state=2;
                  else m->state->=0;
            return 0;
}

tByte fTask1State2(tTask1 *m) {
            *LED = Off;
            m->cnt=0;
            m->led=Off;
            m->state=1;
            return 0;
}
```

```
typedef struct {
      tByte     state;

      tWord      cnt;
      tByte      led;
} tTask1;
```

```
tByte fTask1Open(void) {
        m_task1.cnt=0;
        m_task1.led=off;
        m_task1.state=0;
}
```

Two Schedulers

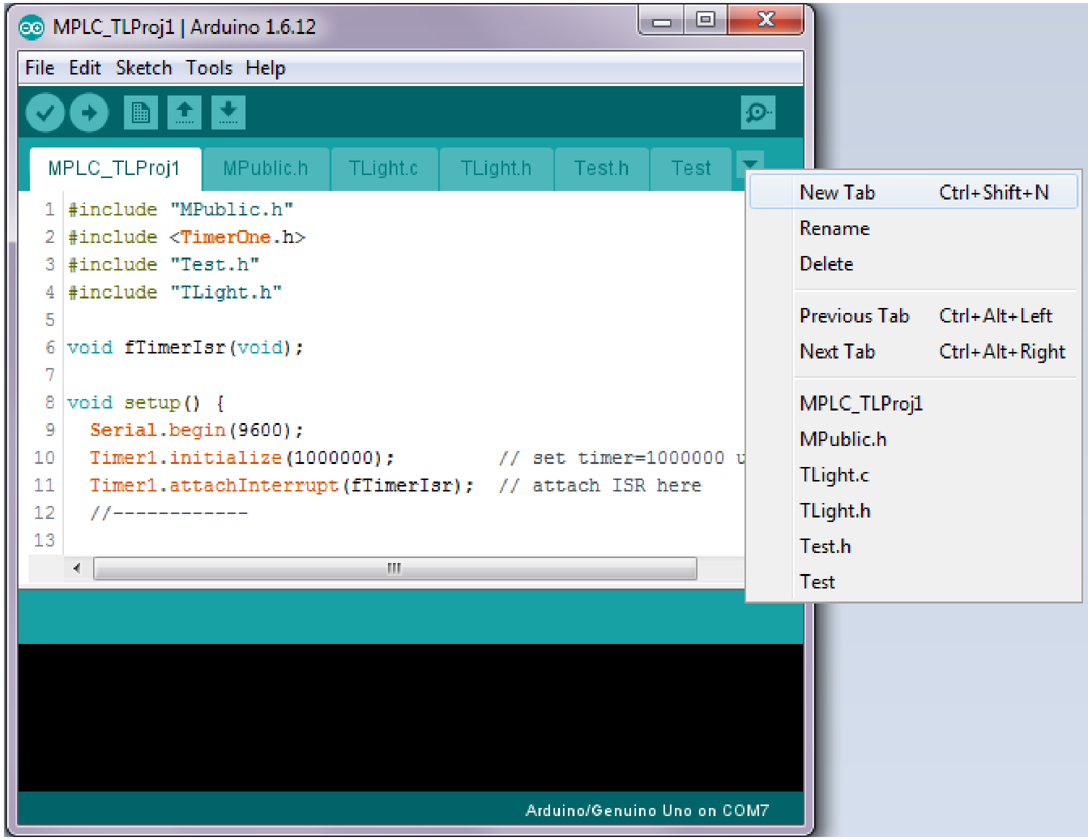
- In this workshop, you will try two schedulers:
  - Normal scheduling running at the CPU clock.
  - Timer scheduling running at a preset-time clock.

Traffic Light Project 1

- You’re going to build a traffic light system using state machines!
- Open PROJ_Temp.ino which is a template sketch for scheduling-based programs in this workshop.
- In this sketch, you can see 3 different files attached:
  - PROJ_Temp.ino which is the main code.
  - MPublic.h which is a header file for public declarations.
  - Test.c and Test.h are template files for creating new state machine-based codes.
    - Test.c is the code file.
    - Test.h is the header file.

Traffic Light Project 1

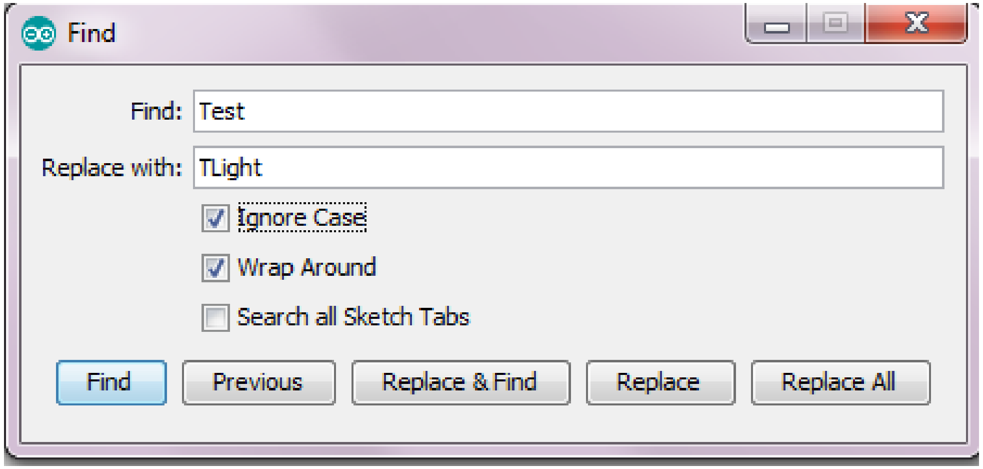
- Save PROJ_Temp.ino as PROJ_TLight1.ino
- Create a new tab by clicking on the right top little arrow shown below:



```
#include "MPublic.h"
#include <TimerOne.h>
#include "Test.h"
#include "TLight.h"

void fTimerIsr(void);

void setup() {
  Serial.begin(9600);
  Timer1.initialize(1000000);         // set timer=1000000 u
  Timer1.attachInterrupt(fTimerIsr);  // attach ISR here
  //------------

```

Traffic Light Project 1

- Type TLight in order to name the new file.
- Create a new tab again and name it as TLight.h which is a header file.
- Copy codes from Test.ino and Test.h into TLight.c and TLight.h, respectively.
- Now, for both TLight.ino and TLight.h, replace the word Test with TLight throughout the files.



Find: Test

Replace with: TLight

☑ Ignore Case

☑ Wrap Around

□ Search all Sketch Tabs

Traffic Light Project 1

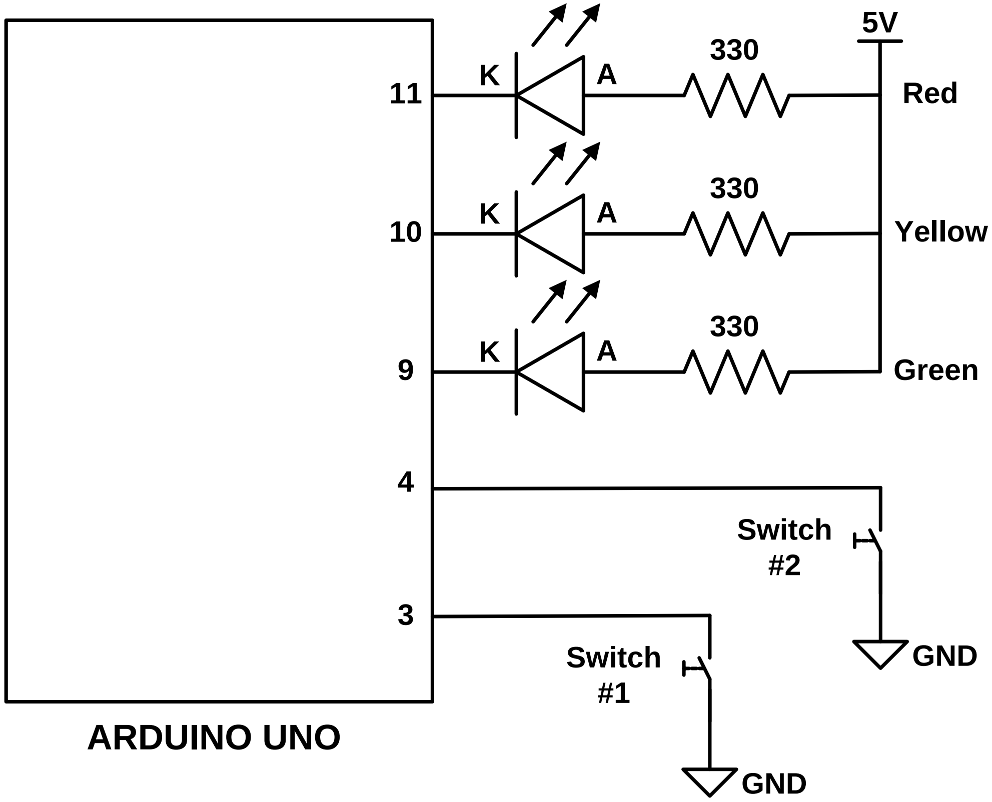
- In PROJ_TLight.ino,
  - include TLight.h by typing #include “TLight.h”.
  - Open the module by calling fTLightOpen(); in the setup() function.
  - Run the module by calling fTLightRun(); in the loop() function and/or fTimerISR() function.

NOTE:

- We need to include XXX.h in order to let the system get to know the new module.
- fXXXOpen() is to setup/initialize the module.
- fXXXRun() is to run tasks of that module in multitasking fashion, by the scheduler.

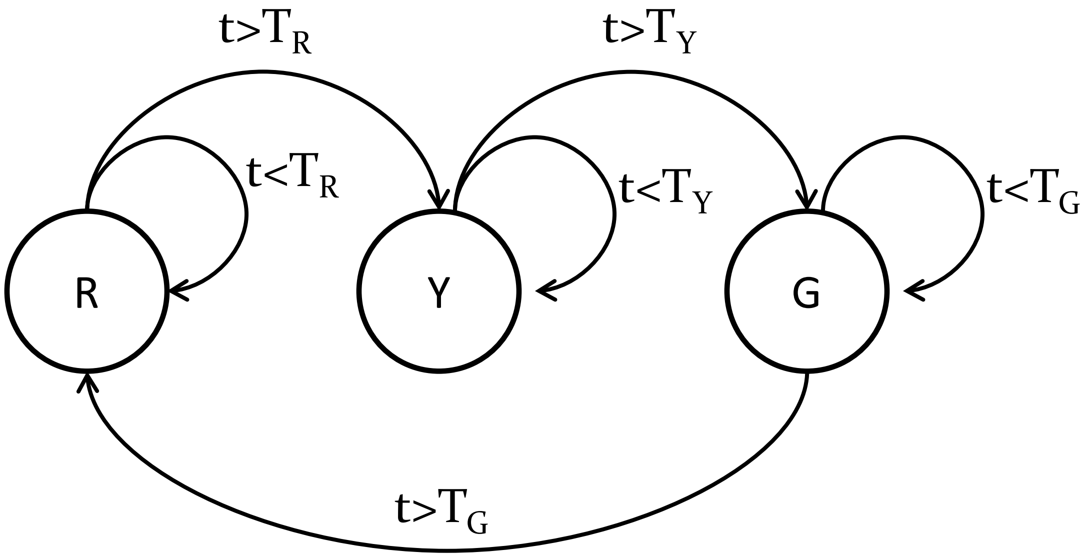
Traffic Light Project 1

- You’re going to build a traffic light system!
- Make circuit connections as shown below.



Traffic Light Project 1

- **DESIGN**: 3 State Machines



We do not use function delay(). Why?

Open MPLC_TLProj1.ino and upload it to your Arduino.

Traffic Light Project 1

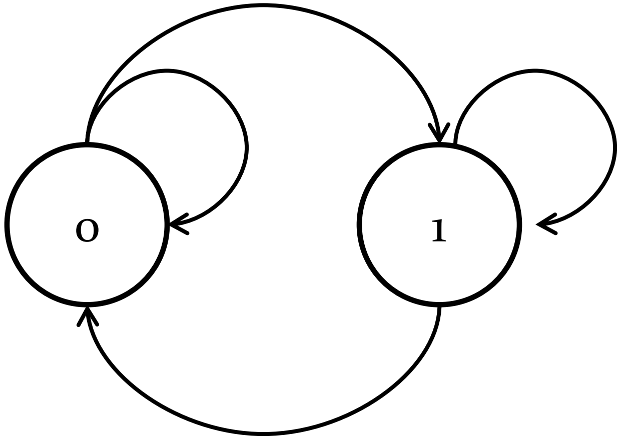
ASSIGNMENT:

Please modify the given code such that:

- The red LED’s bright duration is 5 seconds,
- The yellow LED’s bright duration is 2 seconds, and
- The green LED’s bright duration is 8 seconds.

Traffic Light Project 2

- Now it is time to learn how to add two switch tasks into your main program.
- You still use the same circuit as Traffic Light Project1.
- Of course, we are still working on multitasking processes.
- We may design 2 state machines for each switch.

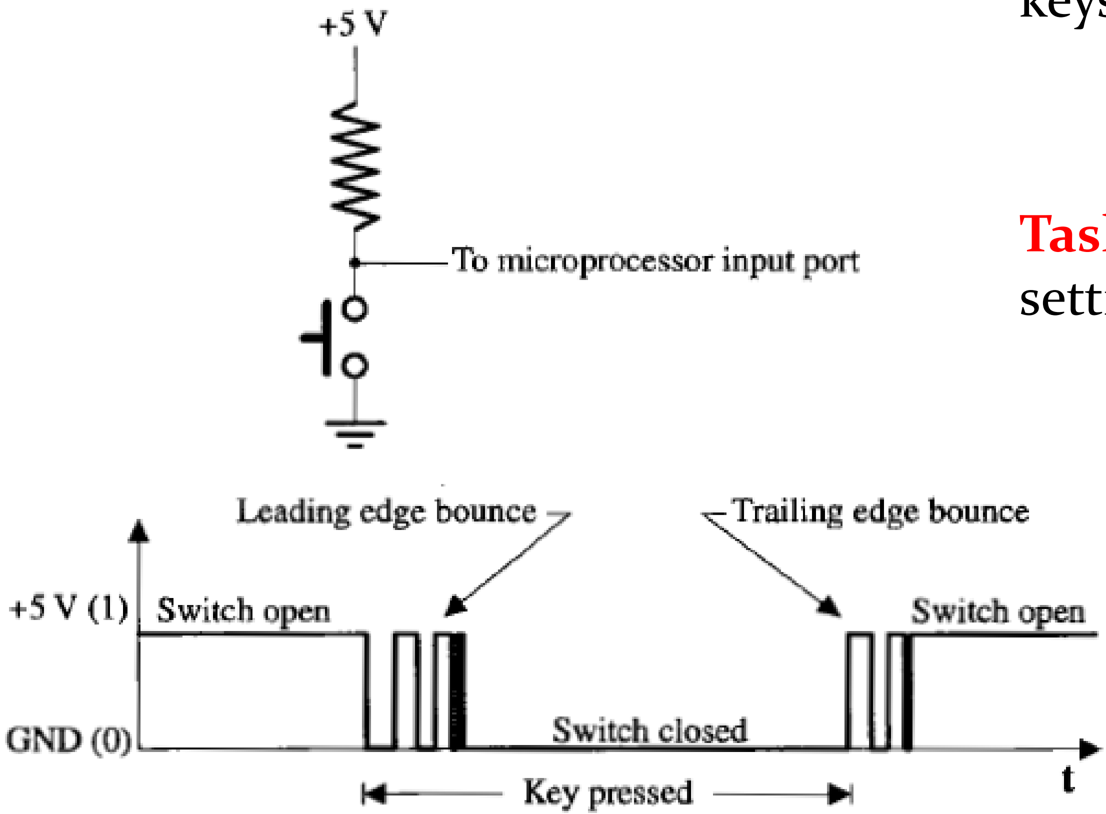


Task "Key":

State 0 = Low

State 1 = High

Traffic Light Project 2



Task Key: Read all switch keys.

- Key.ino
- Key.h

Task KCon: Key control for setting delay times.

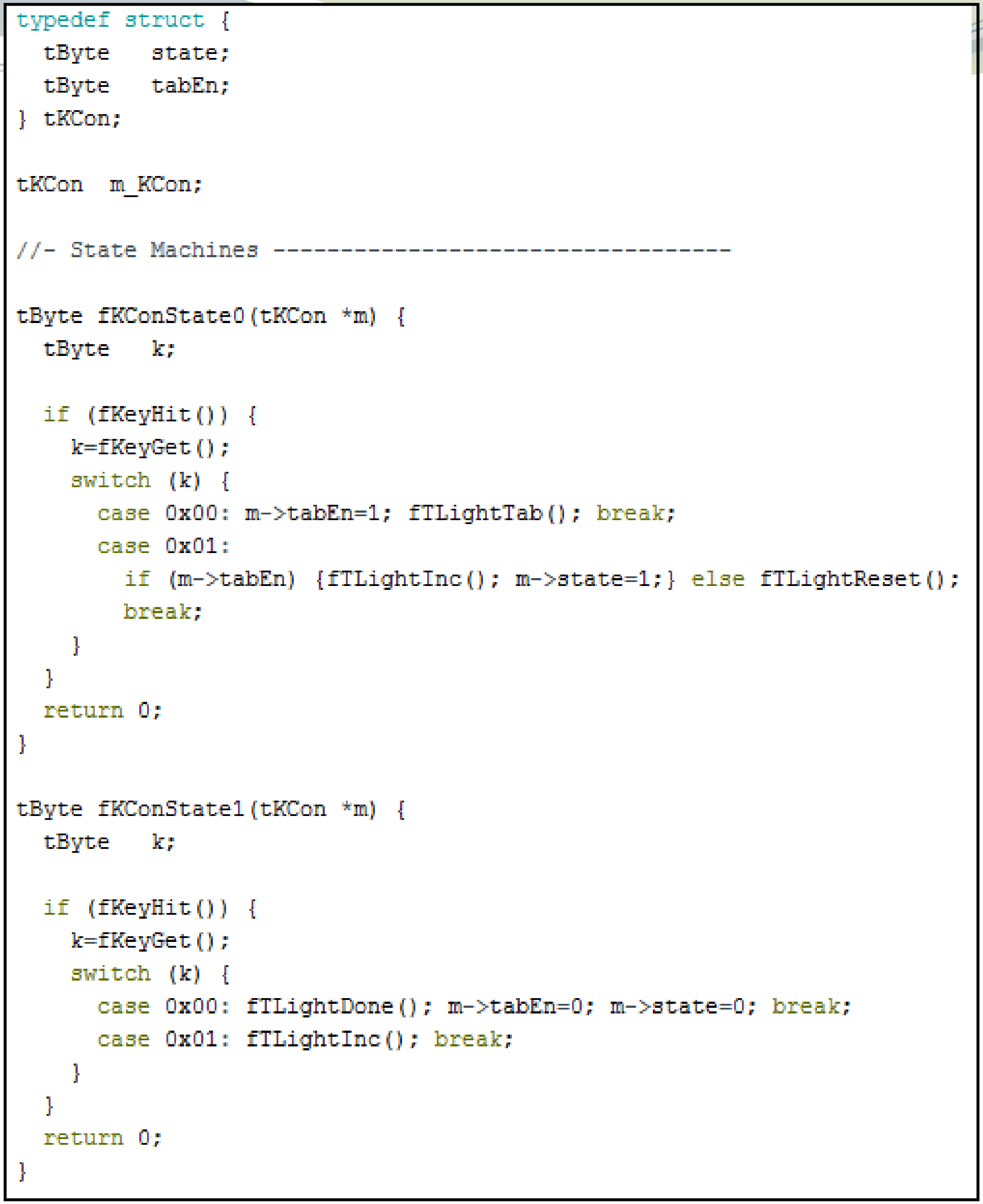
- KCon.ino
- KCon.h

Traffic Light Project 2

- **SW1** is to select which LED you need to set its timer.
  - Each time SW1 is pressed, a LED will be blinking which represents the current one that you may set its timer. Once you press SW1 again the next light will be blinking. This happens to all three lights in a round robin fashion, e.g., RED, YELLOW, GREEN, RED, YELLOW, GREEN, …
- **SW2** is to set a LED's timer. After selecting a LED, you need to change its timer, you may press SW2 to change its timer time. Each time you press SW2 the time will be increased by one with a maximum value of 10, beyond that the time will be rolled back to 1, e.g., 1→ 2 → 3 → 4 → 5 → 6 → 7 → 8 → 9 → 10 → 1 → 2 → 3 → ….

Traffic Light Project 2

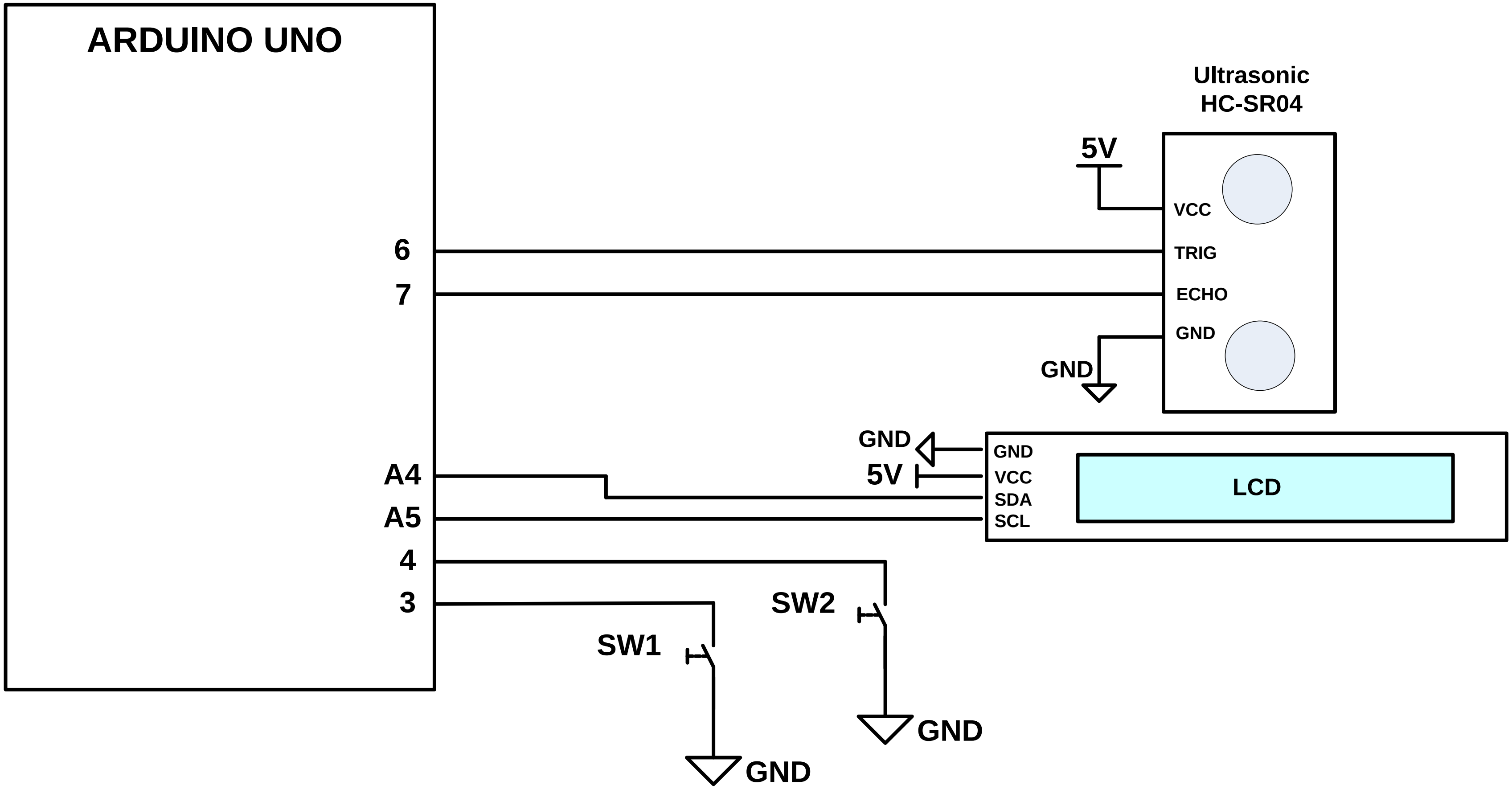
How to read two switches



```
typedef struct {
  tByte   state;
  tByte   tabEn;
} tKCon;

tKCon  m_KCon;

//- State Machines -----------------------------------

tByte fKConState0(tKCon *m) {
  tByte   k;

  if (fKeyHit()) {
    k=fKeyGet();
    switch (k) {
      case 0x00: m->tabEn=1; fTLightTab(); break;
      case 0x01:
        if (m->tabEn) {fTLightInc(); m->state=1;} else fTLightReset();
        break;
    }
  }
  return 0;
}

tByte fKConState1(tKCon *m) {
  tByte   k;

  if (fKeyHit()) {
    k=fKeyGet();
    switch (k) {
      case 0x00: fTLightDone(); m->tabEn=0; m->state=0; break;
      case 0x01: fTLightInc(); break;
    }
  }
  return 0;
}
```

Range Finder Project 1

Make circuit connections as below. Then open PROJ_RFind1.ino

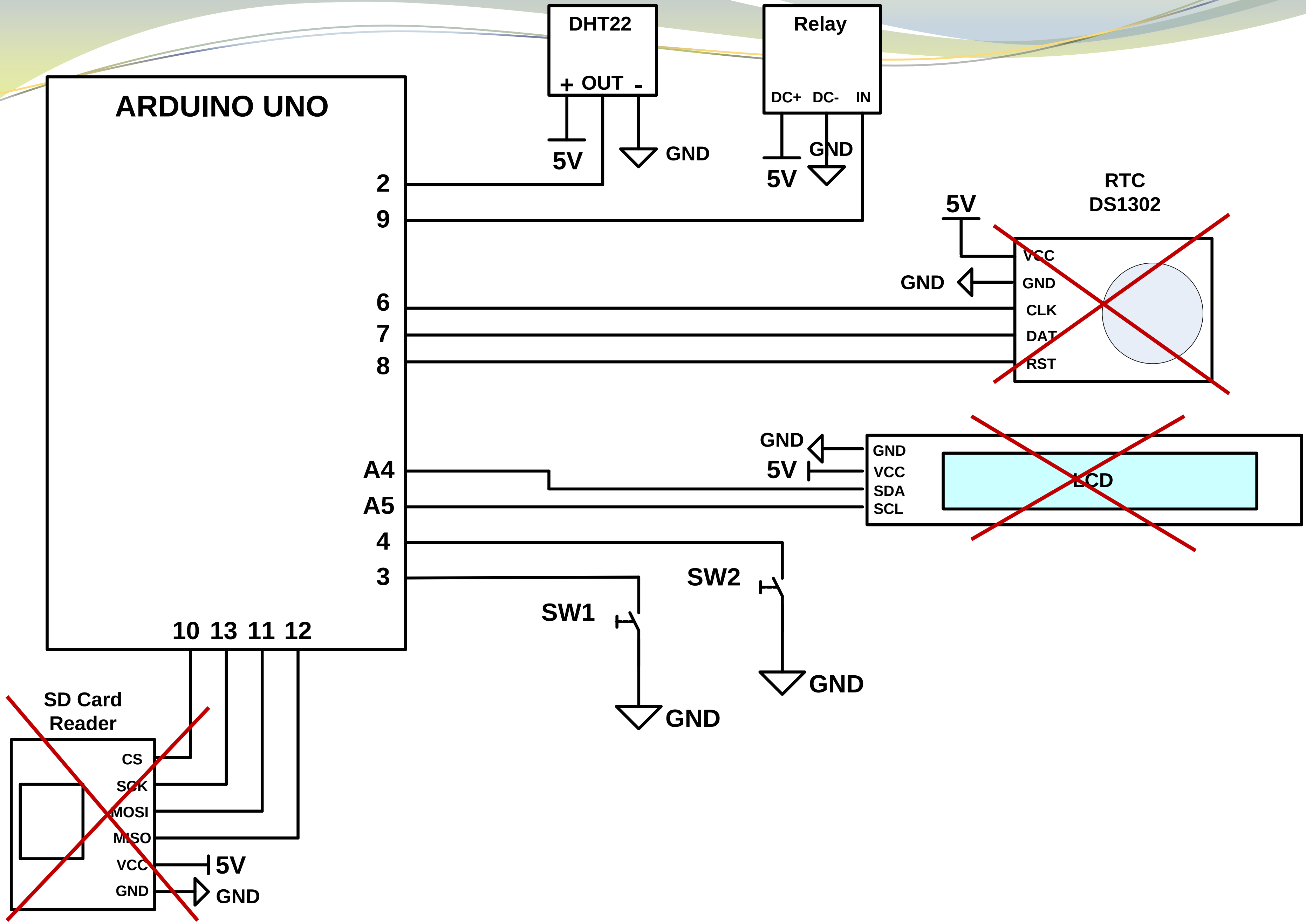


Range Finder Project 2

- Now, you are assigned to implement a simple range finder with the following features:
    - Nothing shown if you don’t press SW1.
    - Once SW1 is pressed, an ultrasonic sensor will be working and consecutively display the current measured distance.
    - When SW1 is released, only the last measured distance value will be shown on the LCD until SW1 is pressed again to start over.
    - SW2 is to clear the display.
- Use the same circuit as Range Finder Project1.
- You may start now!

Little PLC Project 1

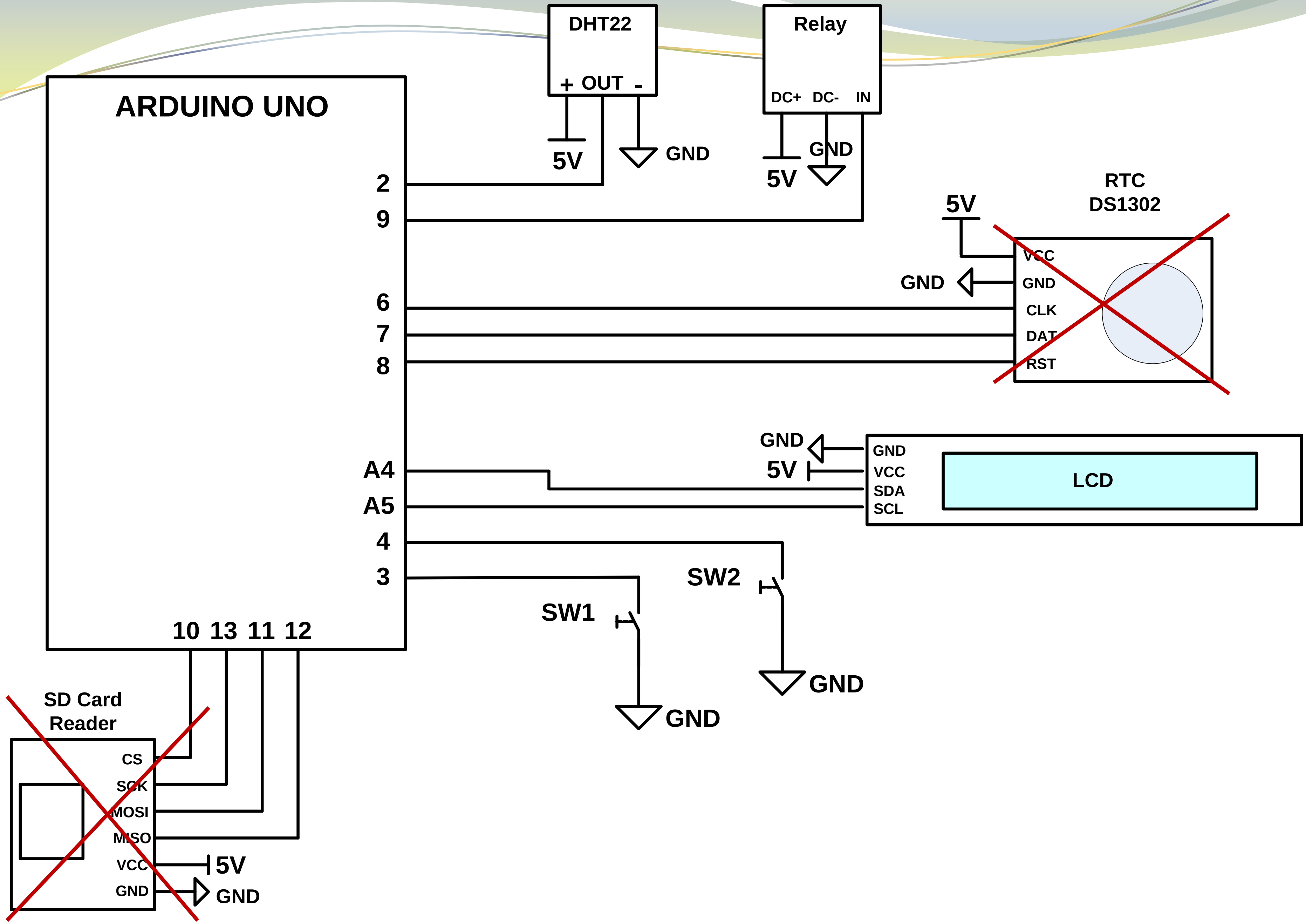
- In this project, you will learn how to implement self-defined instructions for controlling some devices:
  - Get the current temperature → GETT
  - Get the current humidity → GETH
  - Turn On a relay → RYON
  - Turn Off the relay → RYOF
  - Description of all instructions → H
- See the next slide and make circuit connections like that.
- Open PROJ_LittlePLC1.ino and program it into your board.
- Open Serial Monitor. Type H and click Enter.


ARDUINO UNO
2
9
6
7
8
A4
A5
4
3
10 13 11 12
DHT22
+ OUT -
5V
GND
Relay
DC+ DC- IN
GND
5V
5V
RTC
DS1302
VCC
GND
CLK
DAT
RST
GND
GND
5V
GND
VCC
SDA
SCL
LCD
SW2
SW1
GND
GND
SD Card
Reader
CS
SCK
MOSI
MISO
VCC
GND
5V
GND

Little PLC Project 2

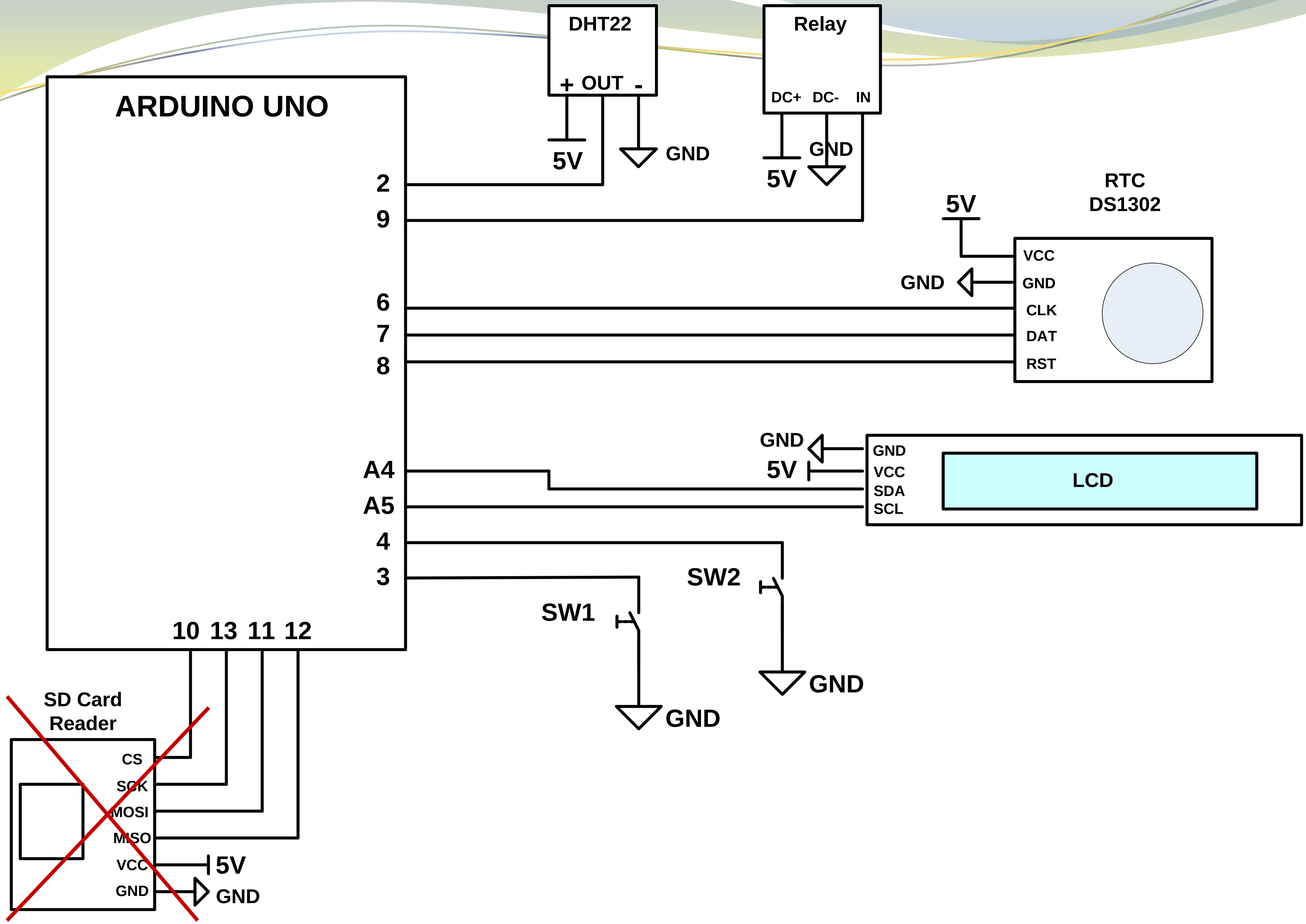
- Add an LCD module like the picture in the next slide.
- ASSIGNMENT:

  Modify the code in Little PLC Project 1 such that the LCD can display any received instruction along with its result.
- If you give up, open PROJ_LittlePLC2.ino and try it!


ARDUINO UNO
2
9
6
7
8
A4
A5
4
3
10 13 11 12
DHT22
+ OUT -
5V
GND
Relay
DC+ DC- IN
GND
5V
RTC
DS1302
5V
VCC
GND
CLK
DAT
RST
GND
LCD
GND
VCC
SDA
SCL
GND
5V
SW1
SW2
GND
GND
SD Card
Reader
CS
SCK
MOSI
MISO
VCC
GND
5V
GND

Little PLC Project 3

- Add a RTC module like the picture in the next slide.
- In this project, you will add two more self-defined instructions as follows:
  - GETDATE → to get the current date
  - GETTIME → to get the current time
- Open PROJ_LittlePLC3.ino and try it!

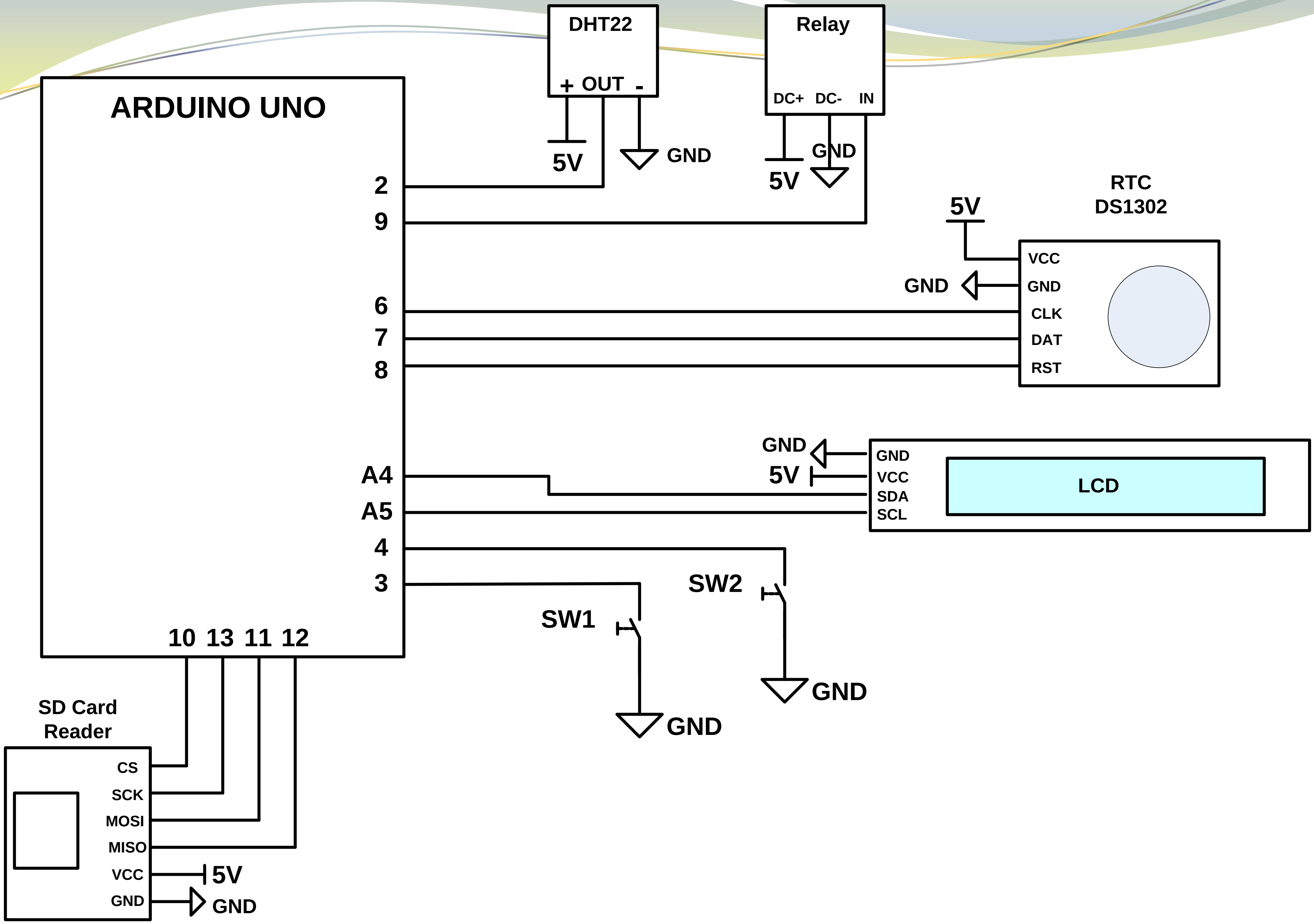

ARDUINO UNO
2
9
6
7
8
A4
A5
4
3
10 13 11 12
DHT22
+ OUT -
5V
GND
Relay
DC+ DC- IN
GND
5V
5V
RTC
DS1302
VCC
GND
CLK
DAT
RST
GND
GND
5V
GND
VCC
SDA
SCL
LCD
SW2
SW1
GND
GND
SD Card
Reader
CS
SCK
MOSI
MISO
VCC
GND
5V
GND

Little PLC Project 4

- In this project, you will add two more self-defined instructions as follows:
  - RYSET → to let the relay switch on and off periodically.
  - RYSTP → to stop RLYSET.
- Moreover, configure SW1 to turn on/off the LCD’s backlight and SW2 to turn on/off the relay.
- You don’t need to add any hardware component.
- Open PROJ_LittlePLC4.ino and try it!

Little PLC Project 5

- Add a micro SD card reader like the picture in the next slide.
- In this project, you will add one more self-defined instruction as below:
  - SAVTH → to save the current temperature and humidity values with the current time stamp.
- Open PROJ_LittlePLC5.ino and try it!


ARDUINO UNO
2
9
6
7
8
A4
A5
4
3
10 13 11 12
DHT22
+ OUT -
5V
GND
Relay
DC+ DC- IN
GND
5V
5V
RTC
DS1302
VCC
GND
CLK
DAT
RST
GND
GND
5V
GND
VCC
SDA
SCL
LCD
SW2
SW1
GND
GND
SD Card
Reader
CS
SCK
MOSI
MISO
VCC
GND
5V
GND

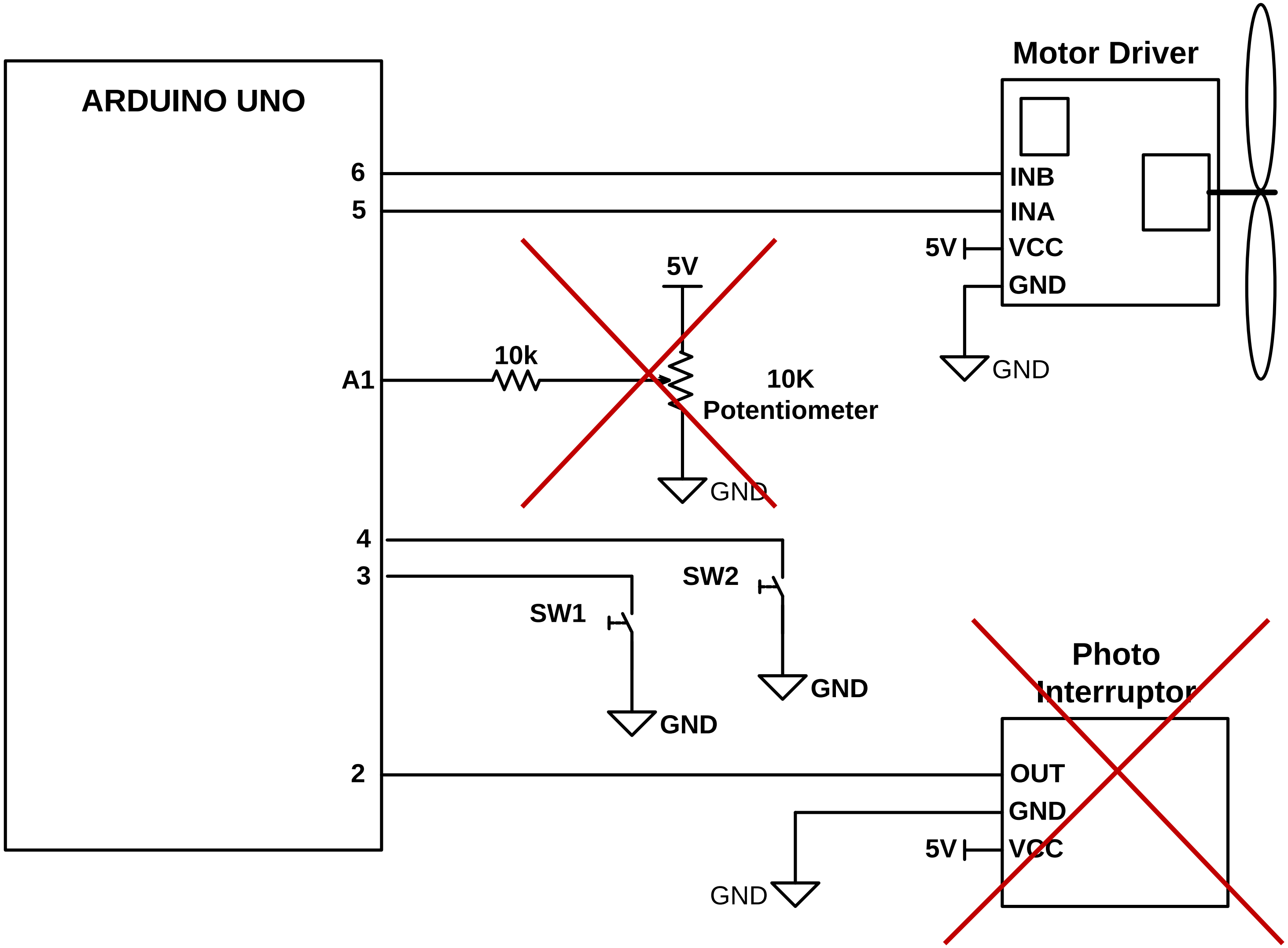
Little PLC Project - Assignment

- Use **Little PLC Project 2** to add your own instructions to control something, whatever you need.
- Please do not forget to save Little PLC Project 2 as another name.
- Let us see your creativity.

Control Project 1

- You are assigned to build a fan the speed of which can be controlled by two buttons:
  - SW1 : low speed.
  - SW2 : high speed.
- Make circuit connections as shown in the next slide.
- If you give up, find the solution!
- ASSIGNMENT:
  - Change the low speed value and also
  - Change the high speed value.

Control Project 1



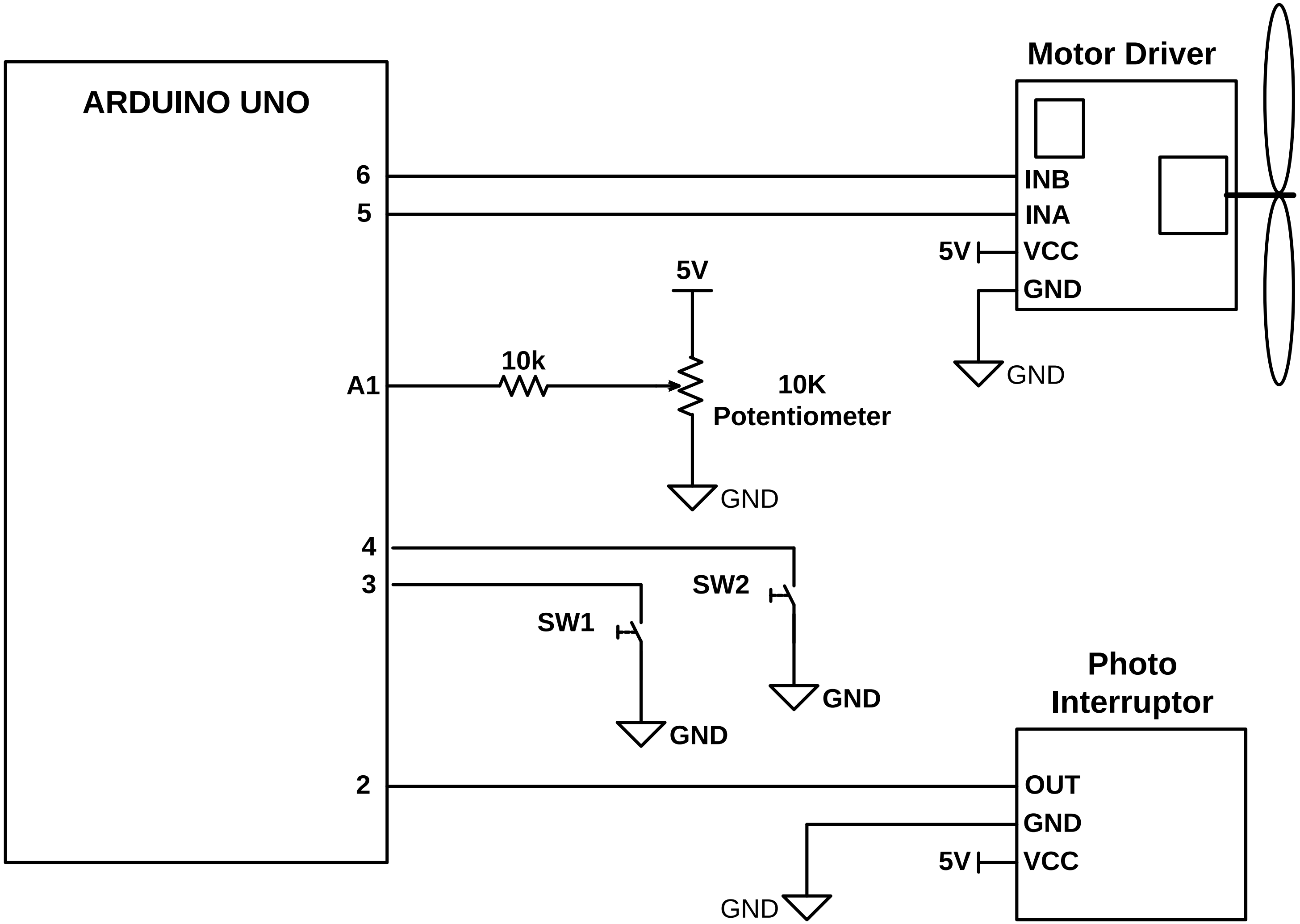
Control Project 2

- You are assigned to build a fan the speed of which can be controlled by two buttons:
  - SW1 is to turn on/off the fan.
  - SW2 is to change the speed:
    - First press → low speed
    - Next press → middle speed
    - Next press → high speed
    - Next press → low speed
- You still use the same hardware as Control Project 1.

Control Project 3

- You are assigned to build a fan the speed of which can be controlled by a potentiometer and measured through the use of a photo interrupter.
- Make hardware like that shown in the next slide.
- You can monitor your motor’s speed through Serial Monitor.
- Open PROJ_Contr3.ino and program it into your board.

Control Project 3



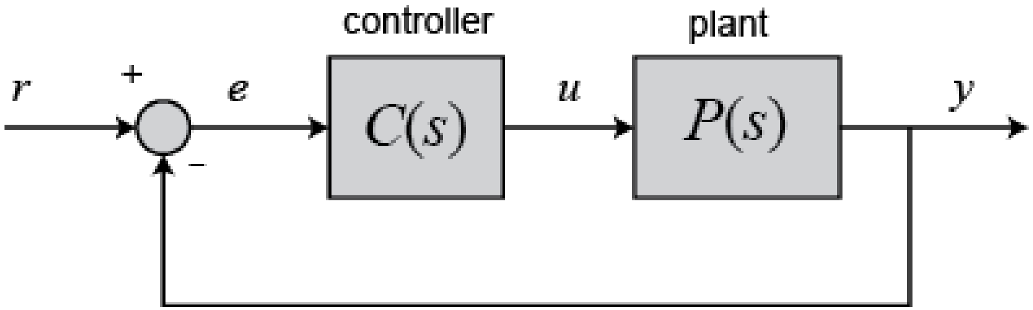
Control Project 4

- This project is to show how PID control can work for a motor's speed control. Your motor can be self-adjusted to a specified speed automatically!
- Use the same hardware as before.
- You need to add the PID library which can be found from: http://playground.arduino.cc/Code/PIDLibrary
- Open PROJ_Contr4.ino and program it to your board.

In the code, you will see

- **Input** = The current motor speed.
- **Output** = PWM output (0-255).
- **SetPoint** = The motor speed you need to see.

Control Project 4



$$u(t) = K_p e(t) + K_i \int e(t)dt + K_p \frac{de}{dt}$$

K_p: Determines how aggressively the PID reacts to the current amount of error (Proportional) (double >=0)
K_i: Determines how aggressively the PID reacts to error over time (Integral) (double>=0)
K_d: Determines how aggressively the PID reacts to the change in error (Derivative) (double>=0)

Thank You!

Dusadee Treeumnuk, PhD
National Electronics and Computer Technology Center
112 Phahon Yothin Rd., Klong 1, Klong Luang, Pathumthani 12120
Email: dusadee.treeumnuk@gmail.com