Introduction to robotics

7th lab

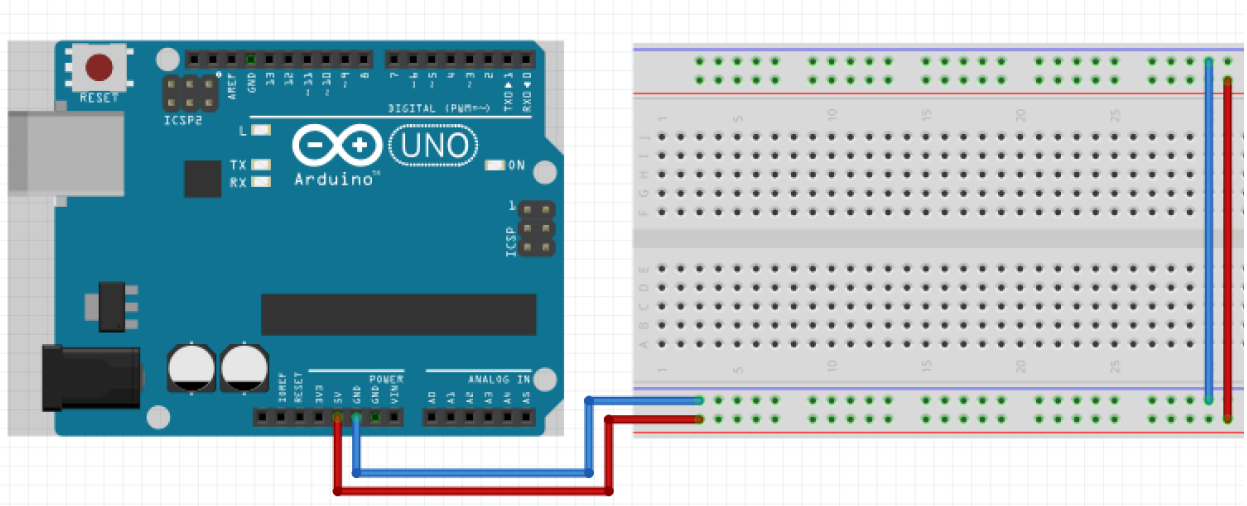
Remember, when possible, choose the wire color accordingly:

- **BLACK** (or dark colors) for **GND**
- **RED** (or colored) for **POWER (3.3V / 5V / VIN)**
- **Remember** than when you use digitalWrite or analogWrite, you actually send power over the PIN, so you can use the same color as for **POWER**
- **Bright Colored** for read signal
- We know it is not always possible to respect this due to lack of wires, but the first rule is **NOT USE BLACK FOR POWER OR RED FOR GND!**

Now, let's pick it up where we left off…

Pull out your Arduino and breadboard and connect them like in the schematic. This is to "power up" the breadboard so we can easily have access to **5V** and **GND**.

Attention! Remember how the breadboard works. Use correct wire colors.



1. LCD Display

1.1 Introduction

Liquid Crystal Display(LCDs) provide a cost effective way to put a text output unit for a microcontroller. As we have seen in the previous tutorial, LEDs or 7 Segments do not have the flexibility to display informative messages. This display has 2 lines and can display 16 characters on each line. Nonetheless, when it is interfaced with the microcontroller, we can scroll the messages with software to display information which is more than 16 characters in length.

The LCD is a simple device to use but the internal details are complex. Most of the 16x2 LCDs use a **Hitachi HD44780** or a compatible controller. Yes, a microcontroller is present inside a Liquid crystal display as shown in figure 2.

The Display Controller takes commands and data from an external microcontroller and drives the LCD panel **(LCDP)**. It takes an ASCII value as input and generates a pattern for the dot matrix. E.g., to display letter 'A', it takes its value **0X42(hex)** or **66(dec)** decodes it into a dot matrix of 5x7 as shown in figure 1.

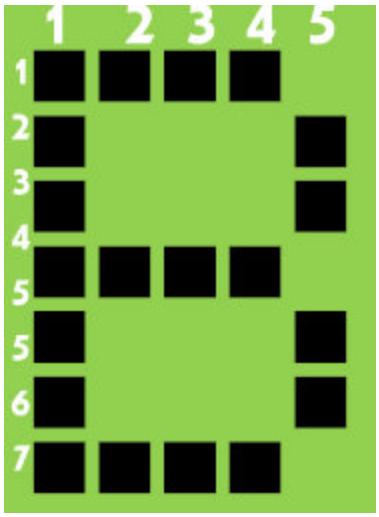
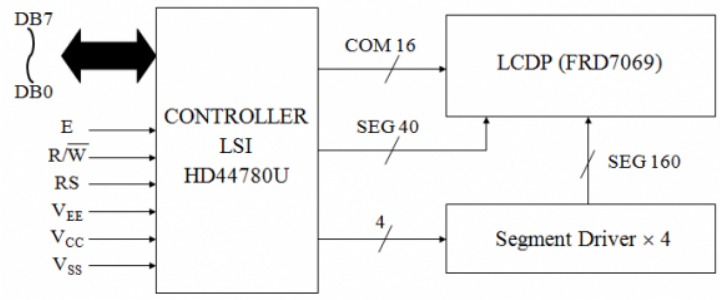


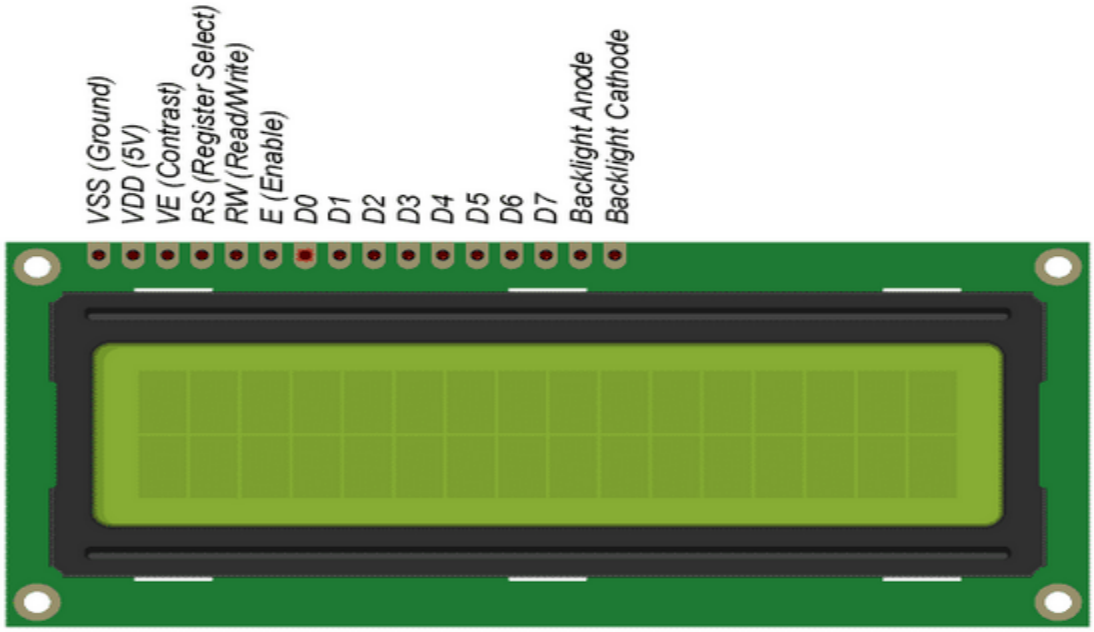
Fig 1: LCD Char 5x7 Matrix
(Each “cell” of the display)

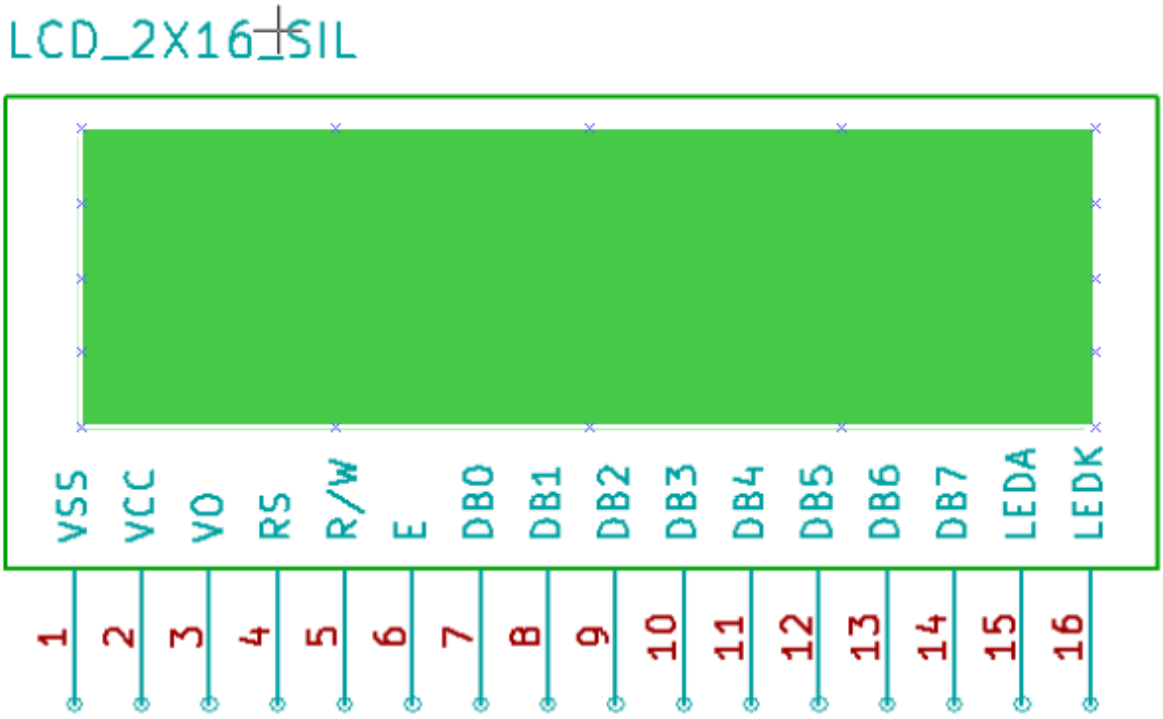
(source:https://exploreembedded.com/wiki/LCD_16_x_2_Basics)



as

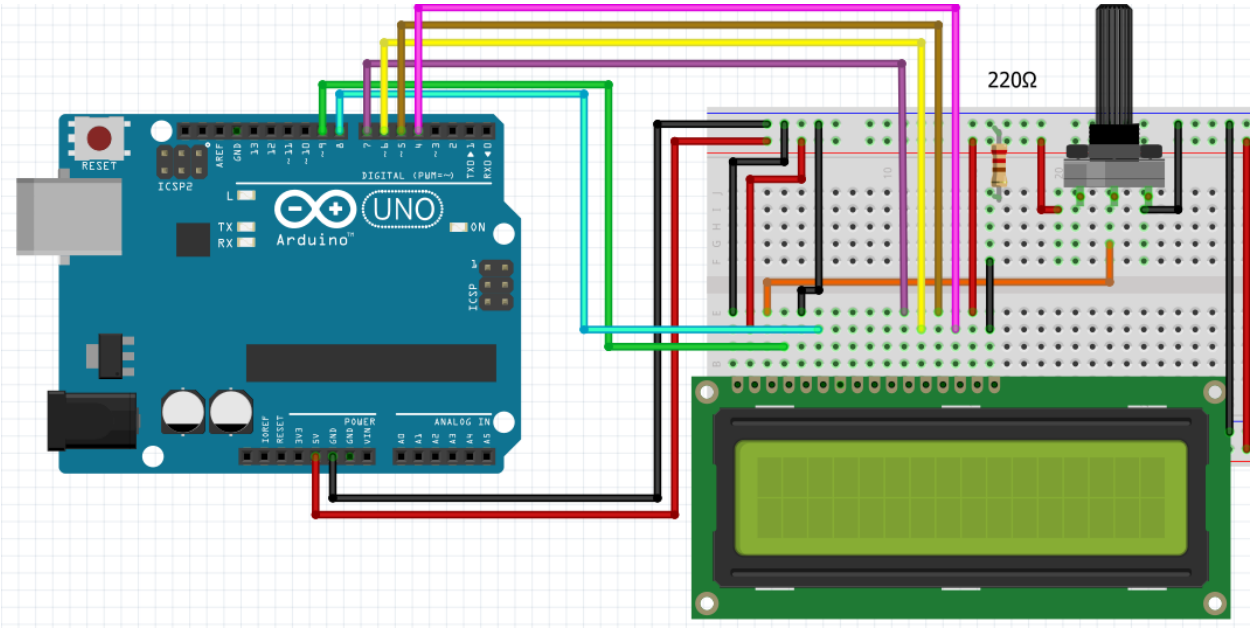
Fig 2: LCD Block diagram





All the pins are identically to the LCD internal controller discussed above

Display PIN	FUNCTION	Arduino PIN
VSS (1)	Ground	GND
VDD (2)	5V	5V
V0 (3)	Contrast adjustment	Potentiometer (analog). PWD after
RS (4)	Register Select. RS=0: Command, RS=1: Data	9
RW (5)	Read/Write (R/W). R/W=0: Write, R/W=1: Read	GND
E (6)	Clock (Enable). Falling edge triggered	8
D0 (7)	Bit 0 (Not used in 4-bit operation)	-
D1 (8)	Bit 1 (Not used in 4-bit operation)	-
D2 (9)	Bit 2 (Not used in 4-bit operation)	-
D3 (10)	Bit 3 (Not used in 4-bit operation)	-
D4 (11)	Bit 4	7
D5 (12)	Bit 5	6
D6 (13)	Bit 6	5
D7 (14)	Bit 7	4
A (15)	Back-light Anode(+)	5V (with 220+ ohm resistor) (for direct control to pin 3)
K (16)	Back-Light Cathode(-)	GND



Default setup. Resistor is about 220ohms, just like with a regular LED

LCD can be interfaced with the microcontroller in two modes, 8 bit and 4 bit. Today we'll interface with the 4bit. The tradeoff is speed, as we send half as much data in one go, but we gain a reduced number of wires.

Data Lines: In this mode, all of the 8 datalines DB0 to DB7 are connected from the microcontroller to an LCD module as shown in the schematic.
Control Lines:' The RS, RW and E are control lines, as discussed earlier.
Power & contrast: Apart from that the LCD should be powered with 5V between PIN 2(VCC) and PIN 1(gnd). PIN 3 is the contrast pin and is the output of the center terminal of potentiometer(voltage divider) which varies voltage between 0 to 5v to vary the contrast.
Back-light: The PIN 15 and 16 are used as backlights. The led backlight can be powered through a simple current limiting resistor as we do with normal leds.

1.2 Code Examples

Most of them are from https://docs.arduino.cc/learn/electronics/lcd-displays (be mindful of different wiring)

1.2.1 Hello world!

```
#include <LiquidCrystal.h>
const byte RS = 9;
const byte enable = 8;
const byte d4 = 7;
const byte d5 = 6;
const byte d6 = 5;
const byte d7 = 4;

LiquidCrystal lcd(RS, enable, d4, d5, d6, d7);

void setup() {
  // set up the LCD's number of columns and rows:
  lcd.begin(16, 2);
  // Print a message to the LCD.
  lcd.print("hello, world!");
}

void loop() {
  // set the cursor to column 0, line 1
  // (note: line 1 is the second row, since counting begins with 0):
  lcd.setCursor(0, 1);
  // print the number of seconds since reset:
  lcd.print(millis() / 1000);
}
```

1.2.1 Try all the LCD functions in the guide

1. Go through all the functions in this guide: https://docs.arduino.cc/learn/electronics/lcd-displays
2. For each one, think of how you can use it in your homework
3. What could custom characters be used for?

2. Serial.read()

2.1 Reading data from serial

Let's learn how to read data from the keyboard. Serial monitor can be used to both send and receive data. Let's try the following code:

```
int incomingByte = 0; // for incoming serial data

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

void loop() {
  // send data only when you receive data:
  if (Serial.available() > 0) {
      // read the incoming byte:
      incomingByte = Serial.read();

      // say what you got:
      Serial.print("I received: ");
      Serial.println(incomingByte);
  }
}
```

First, we check if there is a

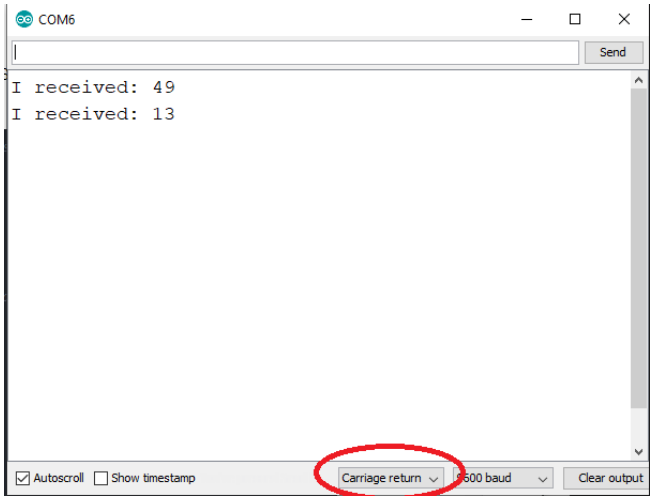
Serial.available()

- Get the number of bytes (characters) available for reading from the serial port. This is data that's already arrived and stored in the serial receive buffer (which holds 64 bytes).

Serial.read() - Reads incoming serial data.

Questions:

1. **Try writing in Serial Monitor the number "1". What is the output?**
    a. **A:** First, the ascii value of what they entered. 2ndly, depending if they have newline / carriage return they also have the value for that
2. **If you have multiple values, why is that?**
    a. **A:** Because beside the initial byte, Serial Monitor also adds a newline / carriage return. Switch to no line ending.



A carriage return, sometimes known as a cartridge return and often shortened to CR, <CR> or return, is a control character or mechanism used to reset a device's position to the beginning of a line of text. It is closely associated with the line feed and newline concepts, although it can be considered separately in its own right.

(source:https://en.wikipedia.org/wiki/Carriage_return)

Play a bit more with newlines etc.

2.2 Serial to Display Example

Let's read data from the Serial and write it to the display

```
#include <LiquidCrystal.h>

// initialize the library by associating any needed LCD interface pin
// with the arduino pin number it is connected to
const byte RS = 9;
const byte enable = 8;
const byte d4 = 7;
const byte d5 = 6;
const byte d6 = 5;
const byte d7 = 4;
LiquidCrystal lcd(rs, en, d4, d5, d6, d7);

void setup() {
  // set up the LCD's number of columns and rows:
  lcd.begin(16, 2);
  // initialize the serial communications:
  Serial.begin(9600);
```

```
}

void loop() {
  // when characters arrive over the serial port...
  if (Serial.available()) {
    // wait a bit for the entire message to arrive
    delay(100);
    // clear the screen
    lcd.clear();
    // read all the available characters
    while (Serial.available() > 0) {
      // display each character to the LCD
      lcd.write(Serial.read());
    }
  }
}
```

2.3 (Optional) Using the power of Serial.print(ln)

The received data is printed byte is printed by default in DEC (the ASCII value) but we can cast to a different type directly or using the power of serial print.

```
int incomingByte = 0; // for incoming serial data

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

void loop() {
  // send data only when you receive data:
  if (Serial.available() > 0) {
      // read the incoming byte:
      incomingByte = Serial.read();

      // say what you got:
      Serial.print("I received (DEC): ");
      Serial.println(incomingByte, DEC);

      Serial.print("I received (HEX): ");
```

```
      Serial.println(incomingByte, HEX);

      Serial.print("I received (OCT): ");
      Serial.println(incomingByte, OCT);

      Serial.print("I received (BIN): ");
      Serial.println(incomingByte, BIN);
  }
}
```

2.4 (Optional) Reading a char and/or a string

So, let's read the char instead of the ASCII value.

```
char incomingByte = 0; // for incoming serial data

String inputString = "";      // a String to hold incoming data
bool stringComplete = false;  // whether the string is complete

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

void loop() {
  // send data only when you receive data:
  if (Serial.available() > 0) {
      // read the incoming byte:
      incomingByte = (char)Serial.read();
      inputString += incomingByte;
      Serial.print("I received: ");
      Serial.println(incomingByte);
      // if the incoming character is a newline, set a flag so the main
loop can
      // do something about it:
      if (incomingByte == '\n') {
      stringComplete = true;
      Serial.println(inputString);
      }
  }
}
```

3. EEPROM

https://docs.arduino.cc/learn/programming/eeprom-guide

3.1 Introduction

The microcontroller on the Arduino board (ATMEGA328 in case of Arduino UNO) has EEPROM (Electrically Erasable Programmable Read-Only Memory). This is a small space that can store byte variables. The variables stored in the EEPROM are kept there, even when you reset or power off the Arduino. Simply, the EEPROM is permanent storage similar to a hard drive in computers.

The EEPROM can be read, erased and rewritten electronically. In Arduino, you can read and write from the EEPROM easily using the EEPROM library.

The microcontroller on the Arduino Uno board has 1024 bytes (1kB) of EEPROM: memory whose values are kept when the board is turned off (like a tiny hard drive).

Functions in the EEPROM class are automatically included with the platform for your board, meaning you do not need to install any external libraries.

The EEPROM has a finite life. In Arduino, the EEPROM is specified to handle 100 000 write/erase cycles for each position. However, reads are unlimited. This means you can read from the EEPROM as many times as you want without compromising its life expectancy. That is why, when possible, we use update() instead of write()!

You can easily read and write into the EEPROM using the EEPROM library.
To include the EEPROM library:

```
#include <EEPROM.h>
```

3.2 EEPROM Read

To read a byte from the EEPROM, you use the EEPROM.read() function. This function takes the address of the byte as an argument.

```
EEPROM.read(address);
```

For example, to read the byte stored previously in address 0.:

```
EEPROM.read(0);
```

This would return 9, which is the value stored in that location.

```
#include <EEPROM.h>

int brightness;         //Create an empty variable

void setup() {
  Serial.begin(9600);
  while (!Serial) {
    // wait for user to open serial monitor
  }
}

void loop() {
  // read a byte from the address "0" of the EEPROM
  brightness = EEPROM.read(0);
  Serial.println(brightness);
  delay(500);
}
```

3.3 EEPROM Write (always use update() instead!)

To write data into the EEPROM, you use the EEPROM.write() function that takes in two arguments. The first one is the EEPROM location or address where you want to save the data, and the second is the value we want to save:

```
EEPROM.write(address, value); // DO NOT ACTUALLY USE
```

For example, to write 9 on address 0, you'll have:

```
EEPROM.write(0, 9);
```

```
#include <EEPROM.h> //include the library

int brightness;         //Create an empty integer variable

void setup() {
  Serial.begin(9600);   //Start serial monitor
  EEPROM.write(0, 25);  // write value 25 on address 0
  //Value must be between 0 and 255
}

void loop() {
  // read a byte from the address "0" of the EEPROM
  brightness = EEPROM.read(0);
  Serial.println(brightness); //Print it to the monitor
  delay(500);
}
```

3.4 EEPROM Update

The EEPROM.update() function is particularly useful. It only writes on the EEPROM if the value written is different from the one already saved.

As the EEPROM has limited life expectancy due to limited write/erase cycles, using the EEPROM.update() function instead of the EEPROM.write() saves cycles.

You use the EEPROM.update() function as follows:

```
EEPROM.update(address, value);
```

At the moment, we have 9 stored in the address 0. So, if we call:

```
EEPROM.update(0, 9);
```

It won’t write on the EEPROM again, as the value currently saved is the same we want to write.

```
#include <EEPROM.h> //include the library

int brightness;         //Create an empty integer variable

void setup() {
  Serial.begin(9600);   //Start serial monitor
  EEPROM.update(0, 25);  // write value 25 on address 0
```

```
  //Value must be between 0 and 255
}

void loop() {
  // read a byte from the address "0" of the EEPROM
  brightness = EEPROM.read(0);
  Serial.println(brightness); //Print it to the monitor
  delay(500);
}
```

3.5 EEPROM GET

Read any data type or object from the EEPROM.

```
EEPROM.get(address, data)
```

Parameters
address: the location to read from, starting from 0 (int)
data: the data to read, can be a primitive type (eg. float) or a custom struct

3.5.1 Read (get) an int

```
#include <EEPROM.h>

int brightness;         //Create an empty variable, type: integer

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

void loop() {
  // read an integer starting on address "0" and store the value on
"brightness"
  EEPROM.get(0, brightness);    //EEPROM.get(start address, variable);
  Serial.println(brightness);
  delay(500);
}
```

3.5.2 Read (get) a float

```
#include <EEPROM.h>

float brightness;         //Create an empty variable, type: float

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

void loop() {
  // read a float starting on address "0" and store the value on
"brightness"
  EEPROM.get(0, brightness);    //EEPROM.get(start address, variable);
  Serial.println(brightness);
  delay(500);
}
```

3.6 EEPROM Put

The purpose of this example is to show the EEPROM.put() method that writes data on EEPROM using also the EEPROM.update() that writes data only if it is different from the previous content of the locations to be written. The number of bytes written is related to the data type or custom structure of the variable to be written.

```
EEPROM.put(address, data)
```

Parameters
address: the location to write to, starting from 0 (int)
data: the data to write, can be a primitive type (eg. float) or a custom struct

3.6.1 Write (put) a float

```
#include <EEPROM.h>

float brightness;                  //Create an empty variable, type: float

void setup() {
  Serial.begin(9600);
  float sensor_read = 70000;     //Define a variable as float
  EEPROM.put(0, sensor_read);    //Write that value starting on address 0
}
```

```
void loop() {
  // read a float starting on address "0" and store the value on
"brightness"
  EEPROM.get(0, brightness);
  Serial.println(brightness);
  delay(500);
}
```

3.6.2 Write (put) a float with decimal point

```
#include <EEPROM.h>
float brightness;                    //Create an empty variable, type: float

void setup() {
  Serial.begin(9600);
  float sensor_read = 56.7;        //Define a variable as float
  EEPROM.put(0, sensor_read );     //Write that value starting on address 0
}

void loop() {
  // read a float starting on address "0" and store the value on
"brightness"
  EEPROM.get(0, brightness);
  Serial.println(brightness);
  delay(500);
}
```

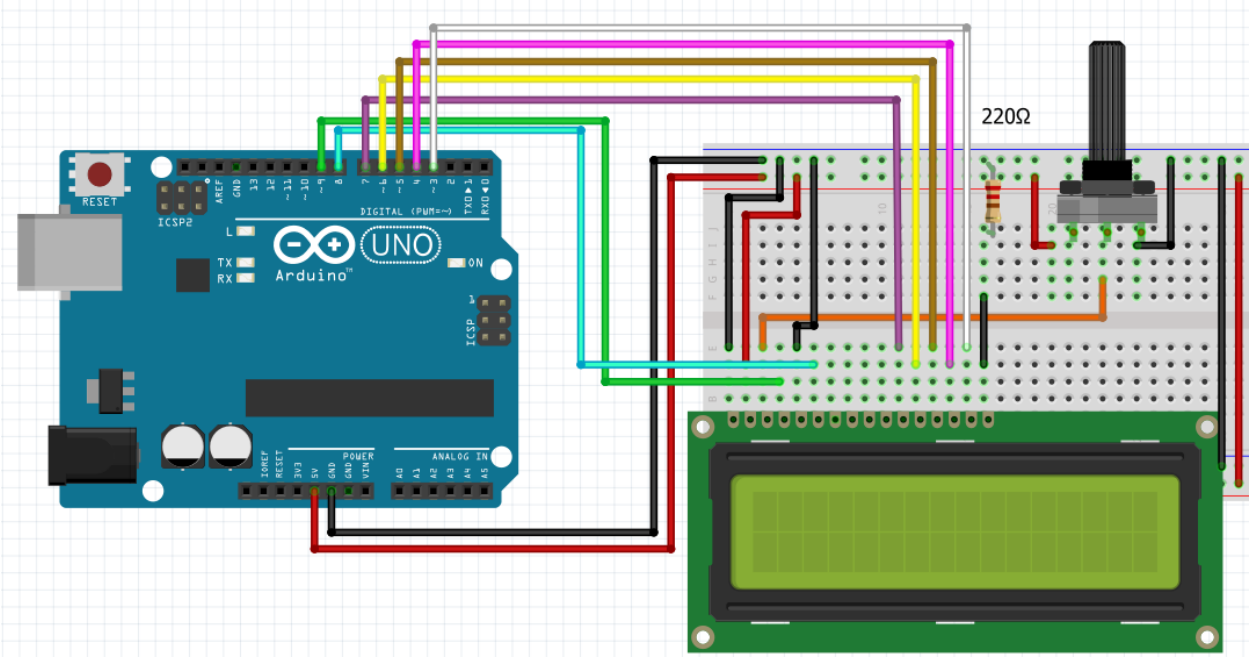
3.7 !important

- Don’t write multiple values on the same address, otherwise you will lose the previously written number (unless that’s what you want to do)
- A memory location can only store one byte of data. So, for numbers between 0 and 255, that’s fine, but for other numbers you’ll have to split the number in several bytes, and store each byte separately. For example, a double value in Arduino Uno takes 4 bytes. Also, that means that you can only store 1024/4 = 256 double values in the EEPROM memory.
- Don’t write a value to the EEPROM inside an infinite loop without any delay or check for user input. Remember, you only have about 100 000 write cycles available per address. If you just write to EEPROM in the loop() function with no other code, you might destroy your EEPROM storage pretty fast.

4. Lab exercises (time permitting)

4.1 Controlling the LED intensity

1. Connect the LED Anode to Pin 3 and control it using analogWrite.
2. Read data from serial and use it to change the intensity of the LED
3. Save the value to eeprom



Setup with LCD LED connected to Arduino PWM pin

4.2 Controlling the contrast intensity

1. Remove the potentiometer and replace it with a PWM pin (do you have any left?)
2. Change some wires to free up a PWM (Analog Inputs can be used as normal digital pins, that do not have PWM)
3. Read data from serial and use it to change the contrast Value
4. Find the right value that works
5. Save it to eeprom and load it at start

4.3 Create a custom character of your choosing and scroll it around

1. Create a custom character of choice
2. Scroll it on the display, playing with the settings

Resources:

1. https://docs.arduino.cc/learn/electronics/lcd-displays
2. https://docs.arduino.cc/learn/programming/eeprom-guide
3. https://www.arduino.cc/reference/en/language/functions/communication/serial/
4. https://docs.arduino.cc/learn/programming/memory-guide
5. https://omerk.github.io/lcdchargen/
6. https://electronoobs.com/eng_arduino_tut167.php