Character LCDs

Created by lady ada

https://learn.adafruit.com/character-lcds

Last updated on 2022-12-01 01:49:59 PM EST

Table of Contents

Overview

We sell tons of lovely character LCDs for use with Arduino (), they are extremely common and a fast way to have your project show status messages. This tutorial will show how you can easily connect a character LCD, either 16x2 or 20x4 (http://adafru.it/198).

The LCDs we sell at Adafruit have a low power LED backlight, run on +5v and require only 6 data pins to talk to. You can use any data pins you want!

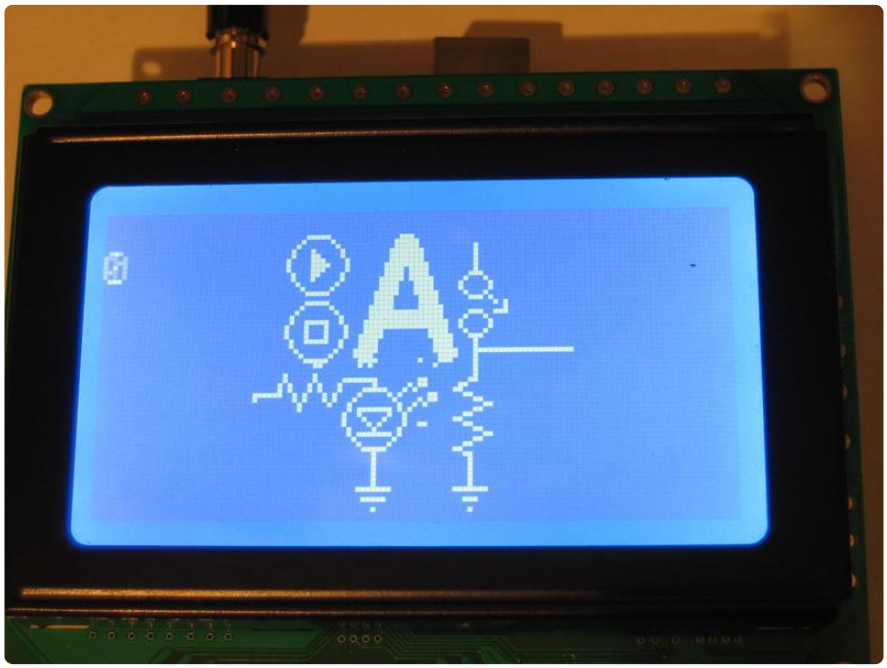
This tutorial will cover character LCDs carried at Adafruit - such as our "standard" blue&white 16x2, RGB 16x2 LCDs, "standard" blue&white 20x4 and RGB 20x4 (). We don't guarantee it will work with any other LCDs. If you need help getting other LCDs to work, please contact the place you purchased it from, they'll be happy to help!

Character vs. Graphical LCDs

There are hundreds of different kinds of LCDs, the ones we'll be covering here are ch aracter LCDs. Character LCDs are ideal for displaying text. They can also be configured to display small icons but the icons must be only 5x7 pixels or so (very small!)

Here is an example of a character LCD, 16 characters by 2 lines:

If you look closely you can see the little rectangles where the characters are displayed. Each rectangle is a grid of pixels. Compare this to a graphical LCD such as the following:

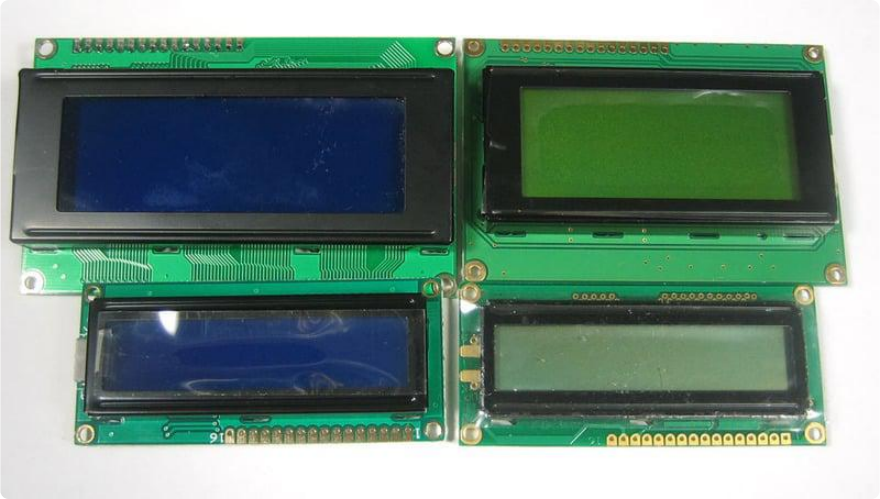


The graphical LCD has one big grid of pixels (in this case 128x64 of them) - It can display text but its best at displaying images. Graphical LCDs tend to be larger, more expensive, difficult to use and need many more pins because of the complexity added.

This tutorial isn't about graphical LCDs. Its only about text/character LCDs!

LCD Varieties

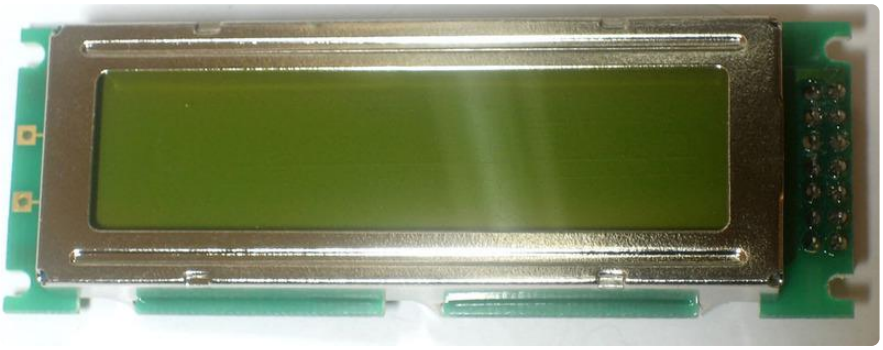
OK now that we're clear about what type of LCD we're talking about, its time to also look at the different shapes they come in.



Although they display only text, they do come in many shapes: from top left we have a 20x4 with white text on blue background, a 16x4 with black text on green, 16x2 with white text on blue and a 16x1 with black text on gray.

The good news is that all of these displays are 'swappable' - if you build your project with one you can unplug it and use another size. Your code may have to adjust to the larger size but at least the wiring is the same!

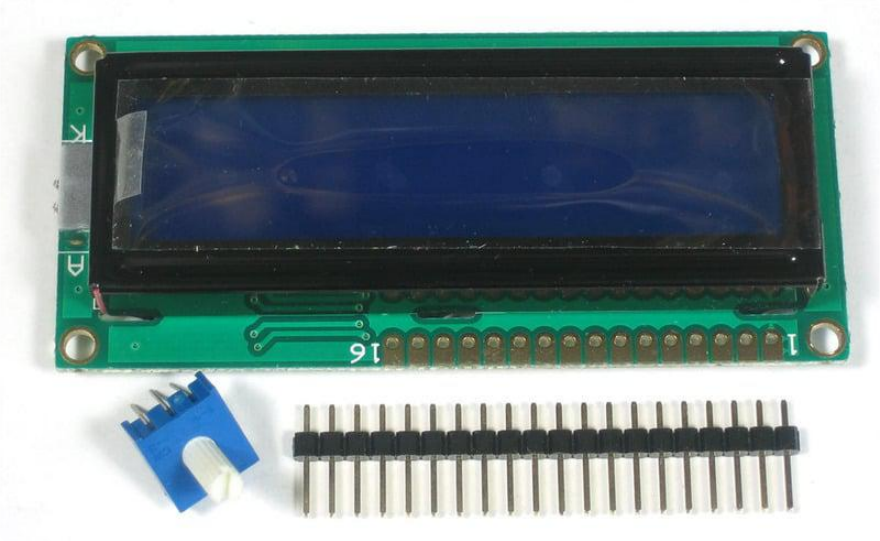
For this part of the tutorial, we'll be using LCDs with a single strip of 16 pins as shown above. There are also some with 2 lines of 8 pins like so:



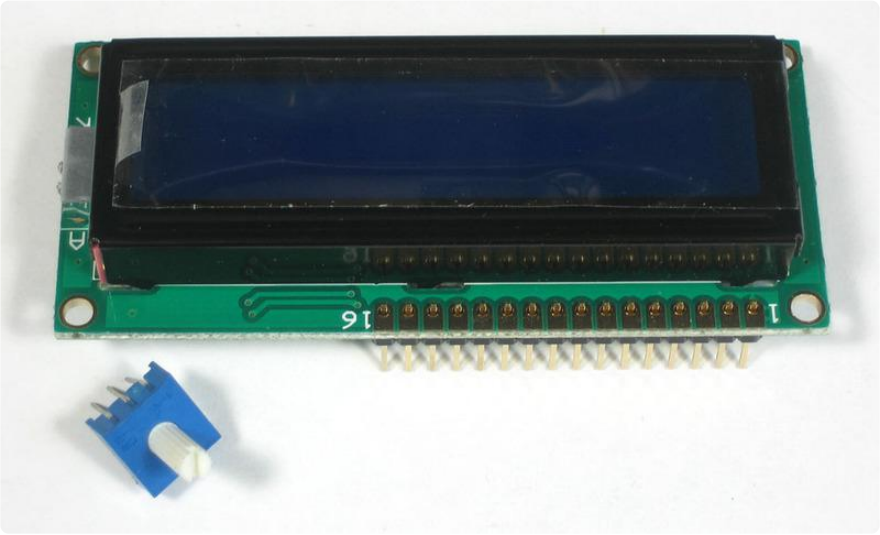
These are much harder to breadboard. If you want some help in wiring these up, check out this page ().

Wiring a Character LCD

Installing the Header Pins



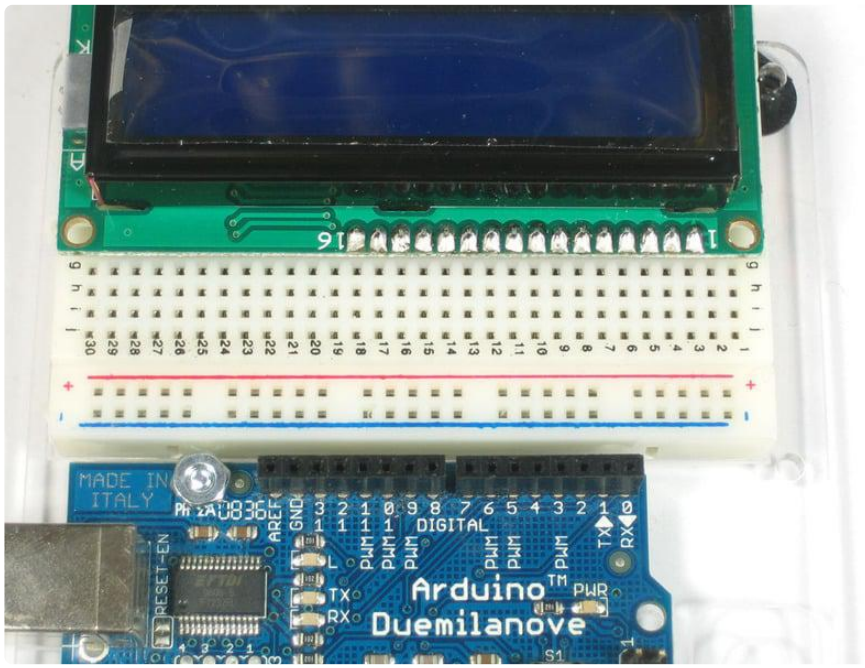
OK now you've got your LCD, you'll also need a couple other things. First is a 10K potentiometer. This will let you adjust the contrast. Each LCD will have slightly different contrast settings so you should try to get some sort of trimmer. You'll also need some 0.1" header - 16 pins long.



If the header is too long, just cut/snap it short!

Next you'll need to solder the header to the LCD.You must do this, it is not OK to just try to 'press fit' the LCD!

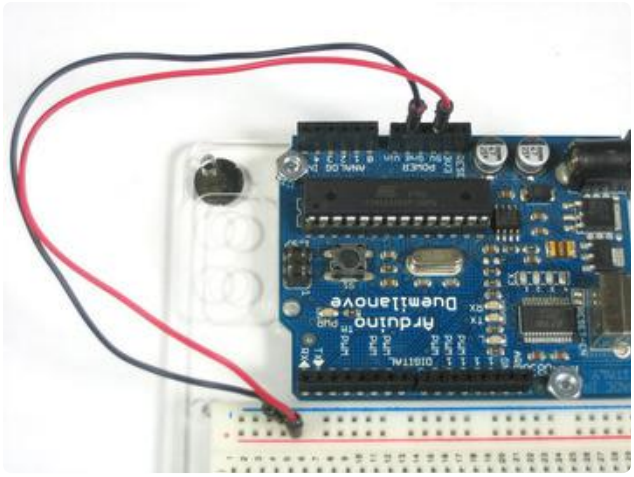
Also watch out not to apply too much heat, or you may melt the underlying breadboard. You can try 'tacking' pin 1 and pin 16 and then removing from the breadboard to finish the remaining solder points



The easiest way we know of doing this is sticking the header into a breadboard and then sitting the LCD on top while soldering. this keeps it steady.

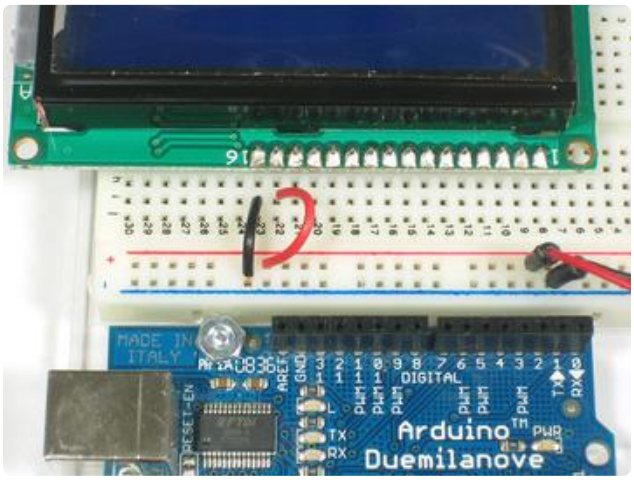
Power and Backlight

K now we're onto the interesting stuff! Get your LCD plugged into the breadboard.

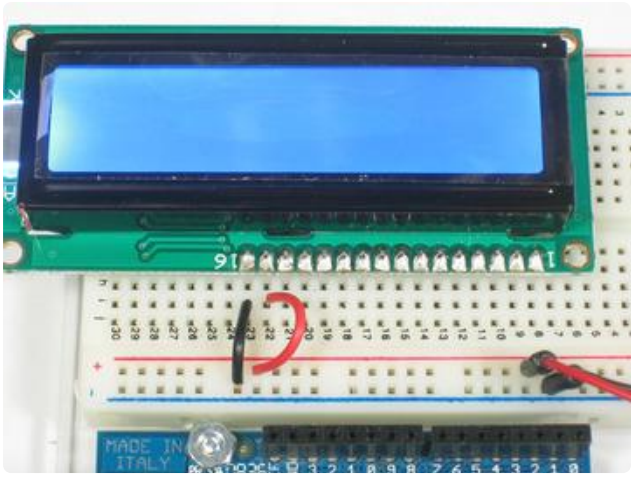


Now we'll provide power to the breadboard. Connect +5V to the red rail, and Ground to the blue rail.

Next we'll connect up the backlight for the LCD. Connect pin 16 to ground and pin 15 to +5V. On the vast majority of LCDs (including ones from Adafruit) the LCD includes a series resistor for the LED backlight.



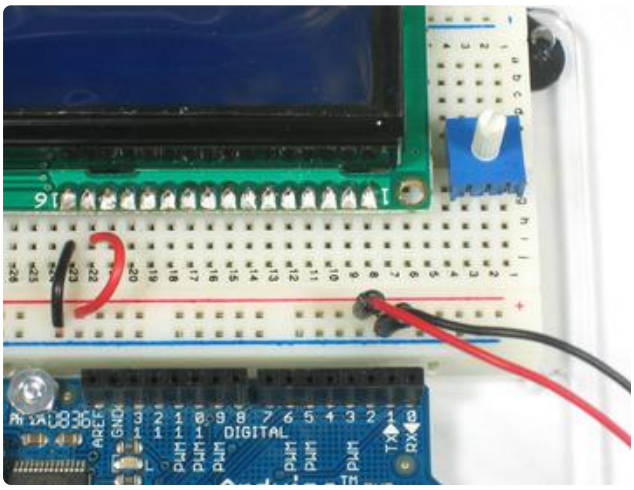
If you happen to have one that does not include a resistor, you'll need to add one between 5V and pin 15. To calculate the value of the series resistor, look up the maximum backlight current and the typical backlight voltage drop from the data sheet. Subtract the voltage drop from 5 volts, then divide by the maximum current, then round up to the next standard resistor value. For example, if the backlight voltage drop is 3.5v typical and the rated current is 16mA, then the resistor should be (5 - 3.5)/ 0.016 = 93.75 ohms, or 100 ohms when rounded up to a standard value. If you can't find the data sheet, then it should be safe to use a 220 ohm resistor, although a value this high may make the backlight rather dim.



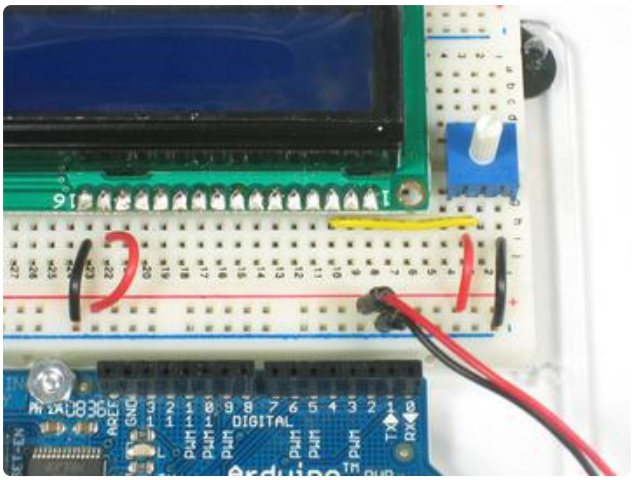
Connect the Arduino up to power, you'll notice the backlight lights up.

Note that some low-cost LCDs dont come with a backlight. Obviously in this case you should just keep going.

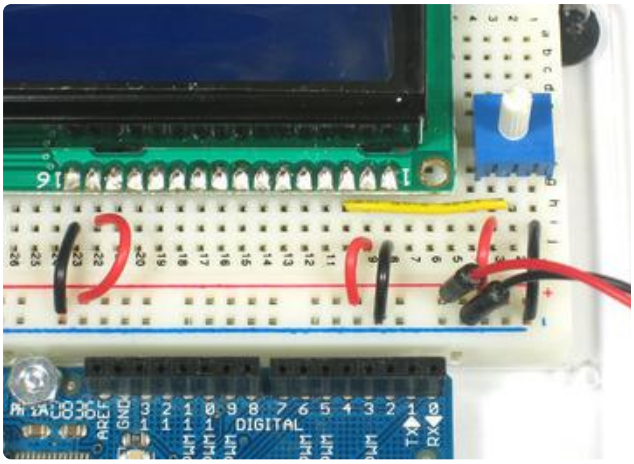
Contrast Circuit



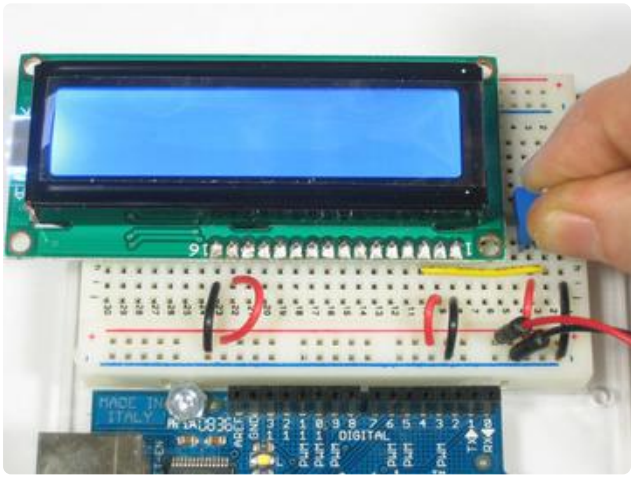
Next, lets place the contrast pot, it goes on the side near pin 1.

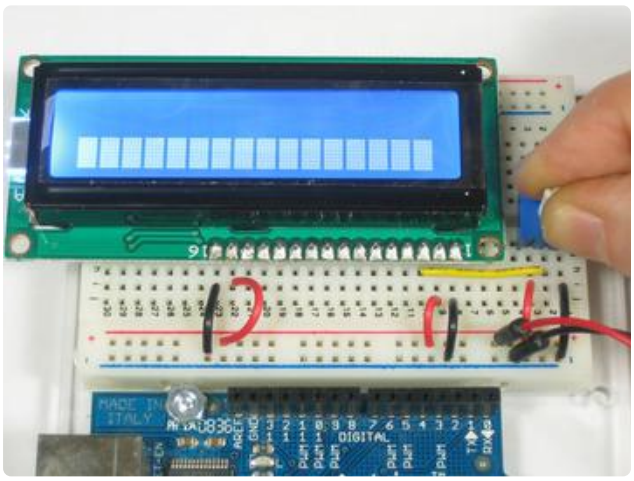


Connect one side of the pot to +5V and the other to Ground (it doesn't matter which goes on what side). The middle of the pot (wiper) connects to pin 3 of the LCD.



Now we'll wire up the logic of the LCD - this is seperate from the backlight! Pin 1 is ground and pin 2 is +5V.





Now turn on the Arduino, you'll see the backlight light up (if there is one), and you can also twist the pot to see the first line of rectangles appear.

This means you've got the logic, backlight and contrast all worked out. Don't keep going unless you've got this figured out!

Bus Wiring

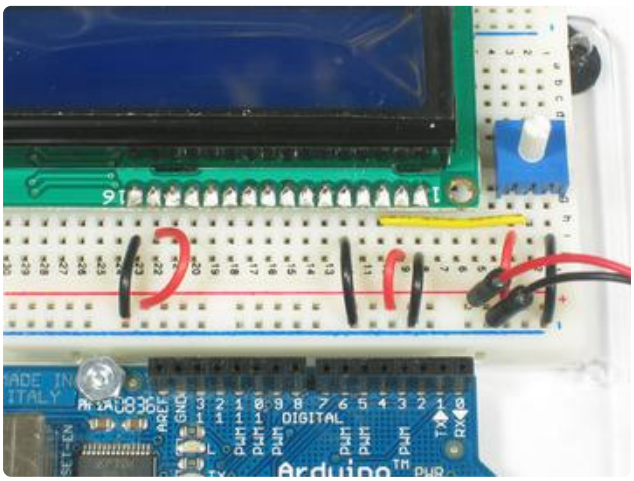
Now we'll finish up the wiring by connecting the data lines. There are 11 bus lines: D0 through D7 (8 data lines) and RS, EN, and RW. D0-D7 are the pins that have the raw data we send to the display. TheRS pin lets the microcontroller tell the LCD whether it wants to display that data (as in, an ASCII character) or whether it is a command byte (like, change posistion of the cursor). The EN pin is the 'enable' line we use this to tell

the LCD when data is ready for reading. The RW pin is used to set the direction - whether we want to write to the display (common) or read from it (less common)

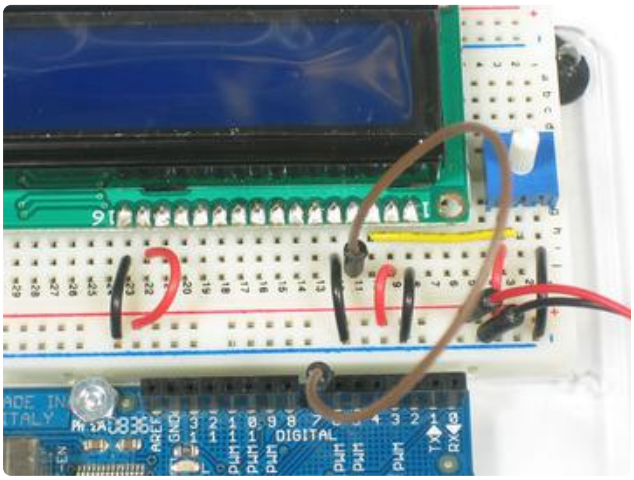
The good news is that not all these pins are necessary for us to connect to the microcontroller (Arduino). RW for example, is not needed if we're only writing to the display (which is the most common thing to do anyways) so we can 'tie' it to ground. There is also a way to talk to the LCD using only 4 data pins instead of 8. This saves us 4 pins! Why would you ever want to use 8 when you could use 4? We're not 100% sure but we think that in some cases its faster to use 8 - it takes twice as long to use 4 - and that speed is important. For us, the speed isn't so important so we'll save some pins!

So to recap, we need 6 pins: RS, EN, D7, D6, D5, and D4 to talk to the LCD.

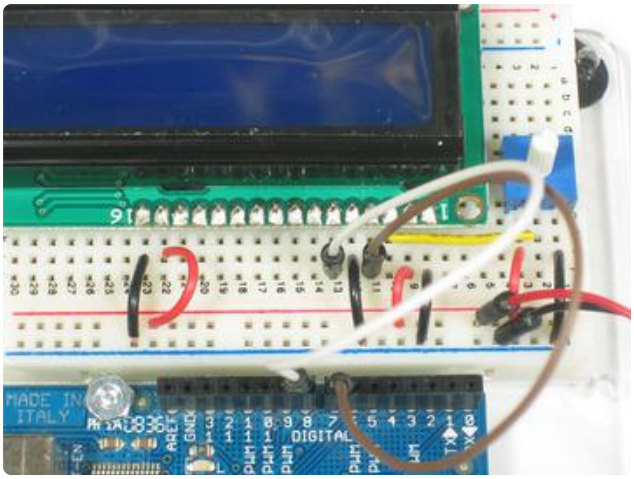
We'll be using the LiquidCrystal library to talk to the LCD so a lot of the annoying work of setting pins and such is taken care of. Another nice thing about this library is that you can use any Arduino pin to connect to the LCD pins. So after you go through this guide, you'll find it easy to swap around the pins if necessary



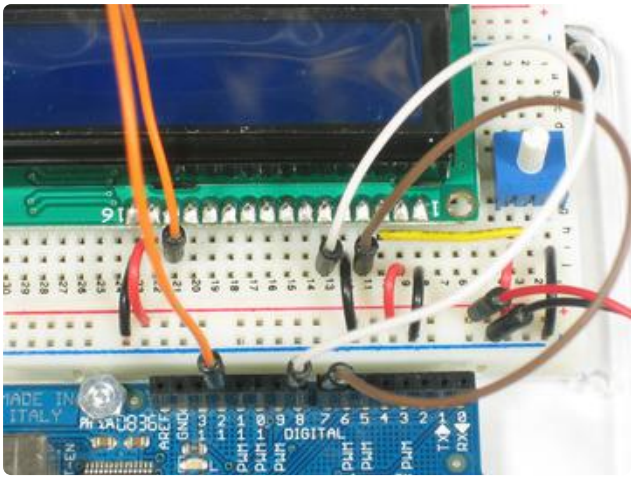
As mentioned, we'll not be using the RW pin, so we can tie it go ground. That's pin 5 as shown here.



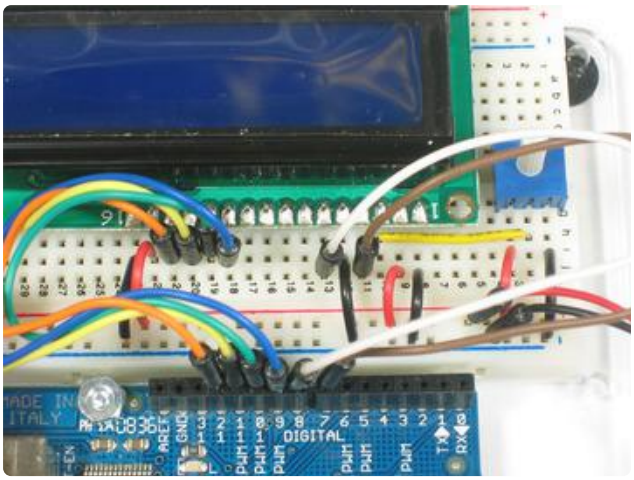
Next is the RS pin #4. We'll use a brown wire to connect it to Arduino's digital pin #7.



Next is the EN pin #6, we'll use a white wire to connect it to Arduino digital #8.

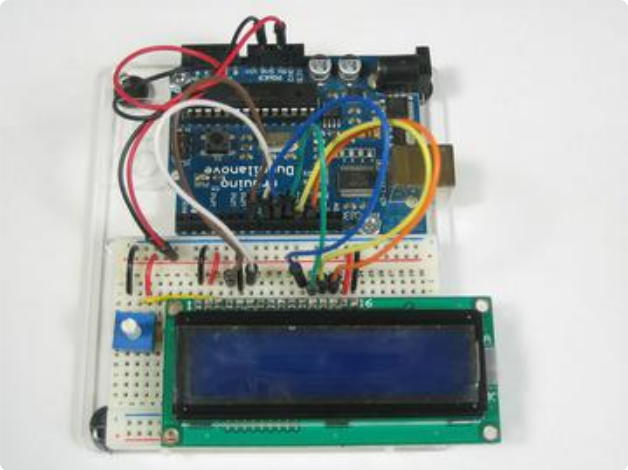


Now we will wire up the data pins. DB7 is pin #14 on the LCD, and it connects with an orange wire to Arduino #12.



Next are the remaining 3 data lines, DB6 (pin #13 yellow) DB5 (pin #12 green) and DB4 (pin #11 blue) which we connect to Arduino #11, 10 and 9.

You should have four 'gap' pins on the LCD between the 4 data bus wires and the control wires.



This is what you'll have on your desk.

Arduino Code

Now we must upload some sketch to the Arduino to talk to the LCD. Luckily the Liquid Crystal library is already built in. So we just need to load one of the examples and modify it for the pins we used.

If you've changed the pins, you'll want to make a handy table so you can update the sketch properly.

LCD pin name	RS	EN	DB4	DB5	DB6	DB7
Arduino pin #	7	8	9	10	11	12

Open up the File→Examples→LiquidCrystal→HelloWorld example sketch

Now we'll need to update the pins. Look for this line:

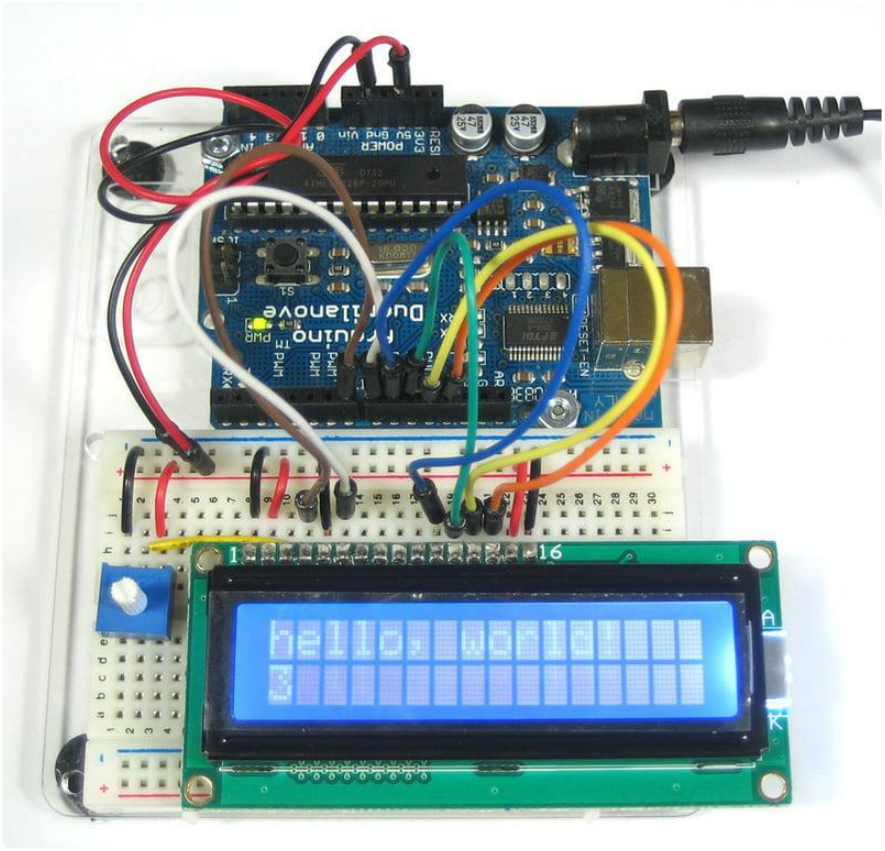
```
LiquidCrystal lcd(12, 11, 5, 4, 3, 2);
```

And change it to:

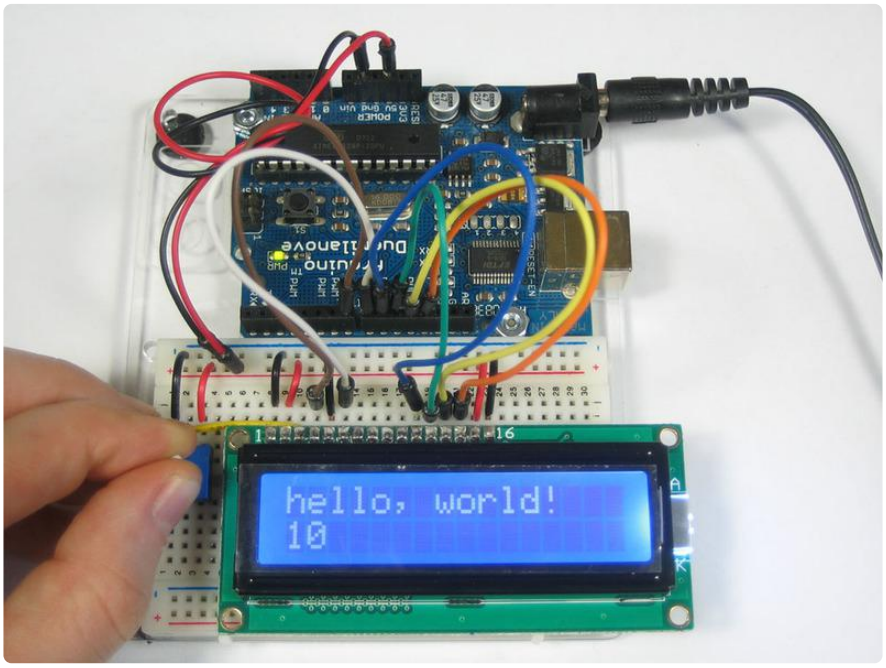
```
LiquidCrystal lcd(7, 8, 9, 10, 11, 12);
```

To match the pin table we just made.

Now you can compile and upload the sketch.

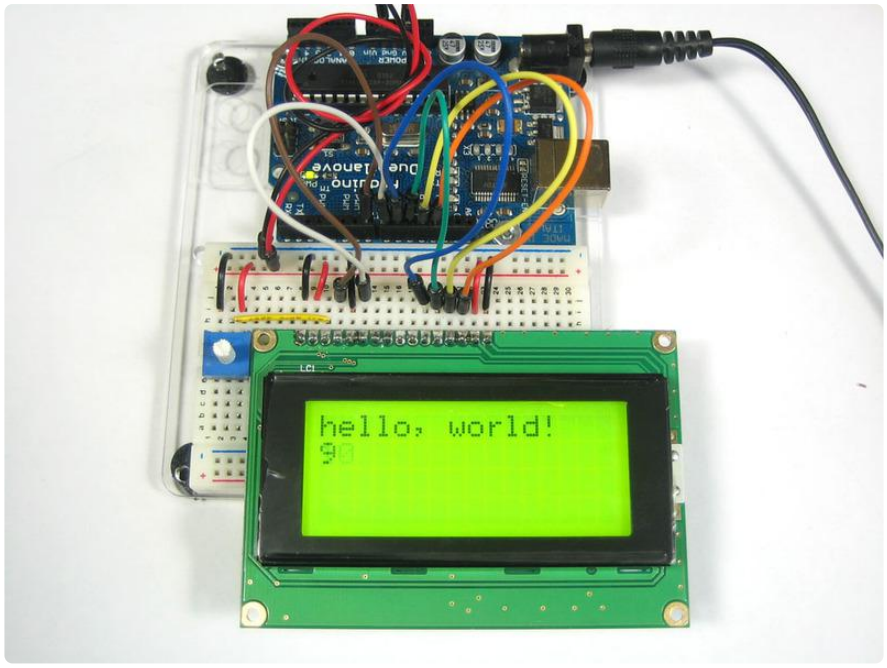


Adjust the contrast if necessary

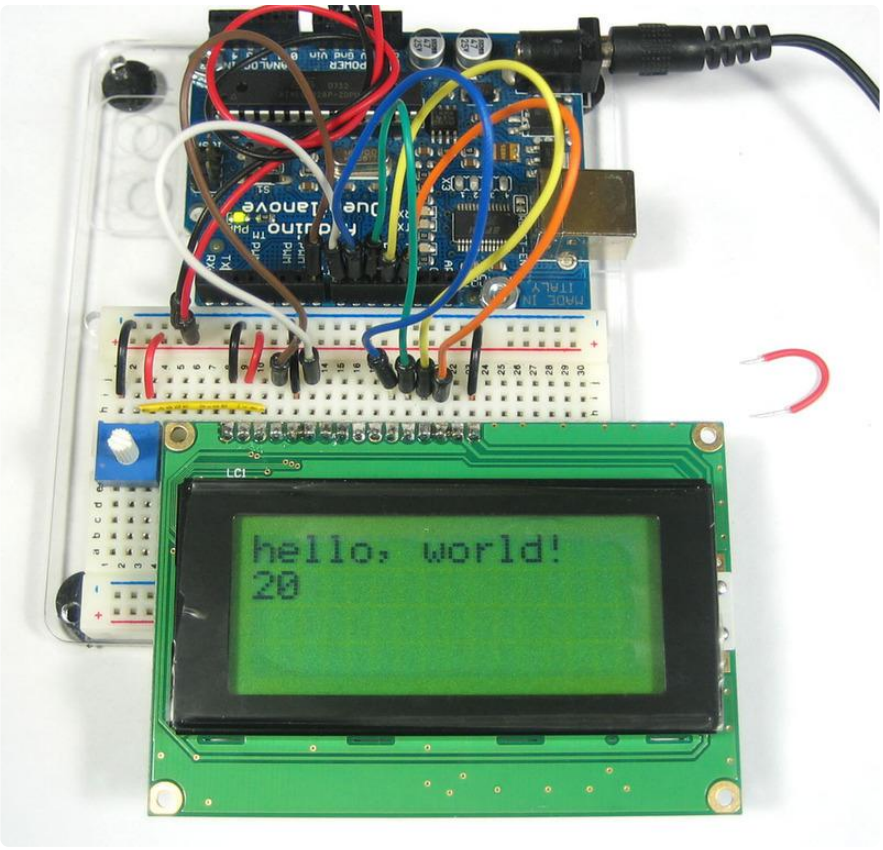


You can of course use any size or color LCD, such as a 20x4 LCD

Or a black on green



The nice thing about the black on green ones is you can remove the backlight. Sometimes they dont come with one!



Multiple Lines

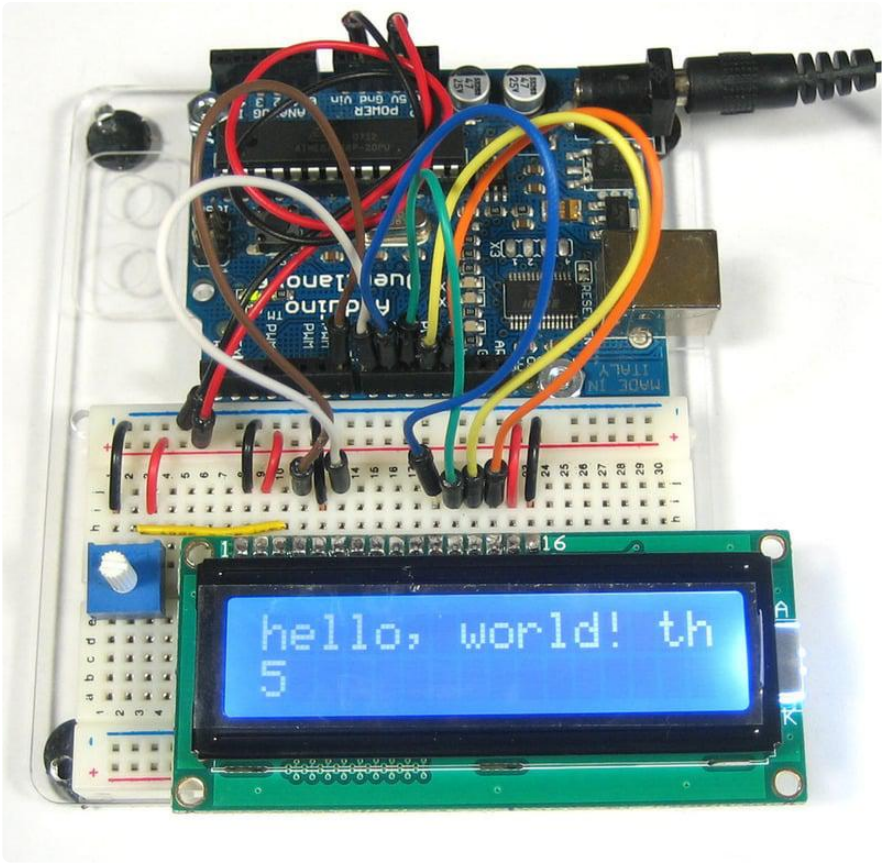
One thing you'll want to watch for is how the LCD handles large messages and multiple lines. For example if you changed this line:

```
lcd.print("hello, world!");
```

To this:

```
lcd.print("hello, world! this is a long long message");
```

The 16x2 LCD will cut off anything past the 16th character:

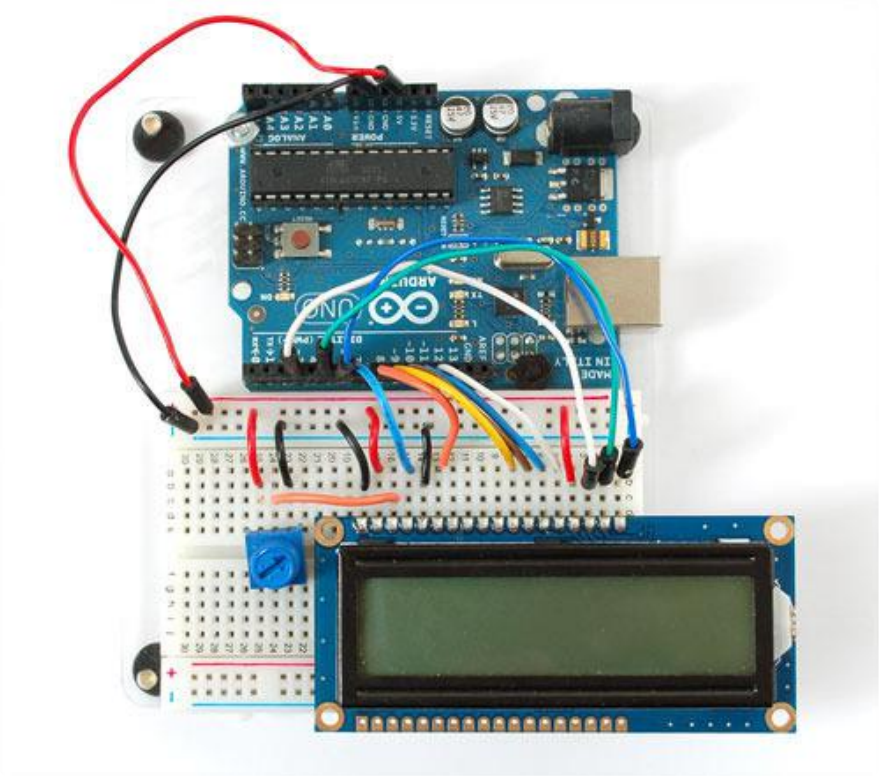


But the 20x4 LCD will 'wrap' the first line to the third line! (Likewise the 2nd line runs into the 4th) This seems really bizarre but its how the LCD memory configured on the inside. This probably should have been done differently but hey that's what we have to live with. Hopefully we'll have a future LCD library that is very smart and wraps lines but for now we are stuck. So when writing long lines to the LCD count your characters and make sure that you dont accidentally overrun the lines!

RGB Backlit LCDs

We now stock a few different RGB backlight LCDs () . These LCDs work just like the normal character type, but the backlight has three LEDS (red/green/blue) so you can generate any color you'd like. Very handy when you want to have some ambient information conveyed.

After you've wired up the LCD and tested it as above, you can connect the LEDs to the PWM analog out pins of the Arduino to precisely set the color. The PWM pins are fixed in hardware and there's 6 of them but three are already used so we'll use the remaining three PWM pins. Connect the red LED (pin 16 of the LCD) to Digital 3, the green LED pin (pin 17 of the LCD) to digital 5 and the blue LED pin (pin 18 of the LCD) to digital 6. You do not need any resistors between the LED pins and the arduino pins because resistors are already soldered onto the character LCD for you!



Now upload this code to your Arduino to see the LCD background light swirl! (Click here to see what it looks like in action ()).

```
// include the library code:
#include &lt;LiquidCrystal.h&gt;
#include &lt;Wire.h&gt;

#define REDLITE 3
#define GREENLITE 5
#define BLUELITE 6

// initialize the library with the numbers of the interface pins
```

```
LiquidCrystal lcd(7, 8, 9, 10, 11, 12);

// you can change the overall brightness by range 0 -&gt; 255
int brightness = 255;

void setup() {
  // set up the LCD's number of rows and columns:
  lcd.begin(16, 2);
  // Print a message to the LCD.
  lcd.print("RGB 16x2 Display  ");
  lcd.setCursor(0,1);
  lcd.print(" Multicolor LCD ");

  pinMode(REDLITE, OUTPUT);
  pinMode(GREENLITE, OUTPUT);
  pinMode(BLUELITE, OUTPUT);

  brightness = 100;
}


void loop() {
  for (int i = 0; i &lt; 255; i++) {
    setBacklight(i, 0, 255-i);
    delay(5);
  }
  for (int i = 0; i &lt; 255; i++) {
    setBacklight(255-i, i, 0);
    delay(5);
  }
  for (int i = 0; i &lt; 255; i++) {
    setBacklight(0, 255-i, i);
    delay(5);
  }
}



void setBacklight(uint8_t r, uint8_t g, uint8_t b) {
  // normalize the red LED - its brighter than the rest!
  r = map(r, 0, 255, 0, 100);
  g = map(g, 0, 255, 0, 150);

  r = map(r, 0, 255, 0, brightness);
  g = map(g, 0, 255, 0, brightness);
  b = map(b, 0, 255, 0, brightness);

  // common anode so invert!
  r = map(r, 0, 255, 255, 0);
  g = map(g, 0, 255, 255, 0);
  b = map(b, 0, 255, 255, 0);
  Serial.print("R = "); Serial.print(r, DEC);
  Serial.print(" G = "); Serial.print(g, DEC);
  Serial.print(" B = "); Serial.println(b, DEC);
  analogWrite(REDLITE, r);
  analogWrite(GREENLITE, g);
  analogWrite(BLUELITE, b);
}
```

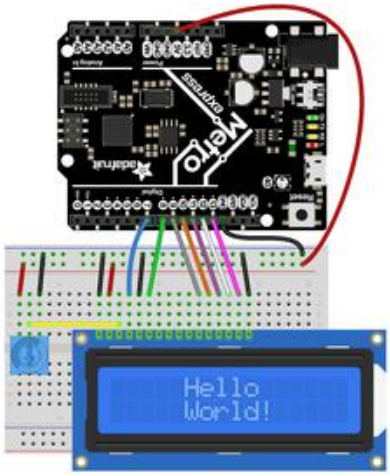
Python & CircuitPython

It's easy to use a character LCD with CircuitPython or Python and the Adafruit CircuitPython CharLCD () module. This module allows you to easily write Python code that controls a character LCD (either single backlight or RGB backlight).

You can use these with any CircuitPython microcontroller board or with a computer that has GPIO and Python thanks to Adafruit_Blinka, our CircuitPython-for-Python compatibility library ().

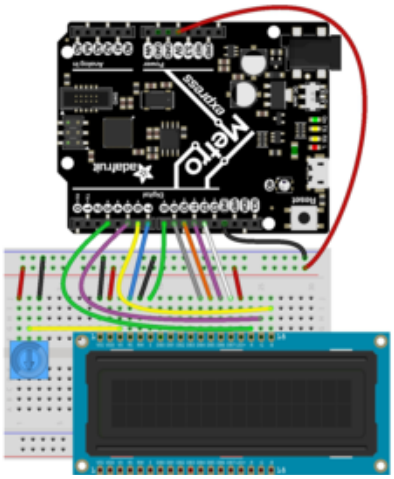
CircuitPython Microcontroller Wiring

First wire up a character LCD to your board exactly as shown on the previous pages for Arduino using the LCD's parallel data bus. Here's an example of wiring a Metro M0 Express to a single color backlight character LCD:



Board 5V to LCD pin 2 and one side of the potentiometer.
Board GND to LCD pin 1, 5, 16, and the opposite side of the potentiometer.
Potentiometer output (middle pin) to LCD pin 3
Board D7 to LCD pin 4
Board D8 to LCD pin 6
Board D9 to LCD pin 11
Board D10 to LCD pin 12
Board D11 to LCD pin 13
Board D12 to LCD pin 14
Board D13 to LCD pin 15
Remember just like the Arduino wiring page mentions there are 4 unused pins on the LCD, pins 7-10.

If you're using a RGB backlight here's an example of wiring it to your board. Remember, if you want to be able to do more than only red, green OR blue, each of the red, green, blue color channels needs to be wired to a PWM-capable output pin on your board (check out this script to find all the PWM capable pins on your board ()). If you'd like to use non-PWM pins, you can, however you'll only be able to turn on red, green OR blue at one time. The following diagram uses PWM capable pins:

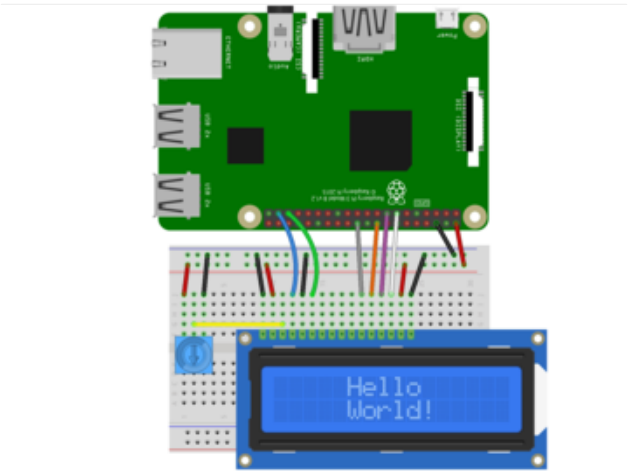


Board 5V to LCD pin 2, 15, and one side of the potentiometer.
Board GND to LCD pin 1, 5, and the opposite side of the potentiometer.
Potentiometer output (middle pin) to LCD pin 3
Board D7 to LCD pin 4
Board D8 to LCD pin 6
Board D9 to LCD pin 11
Board D10 to LCD pin 12
Board D11 to LCD pin 13
Board D12 to LCD pin 14
Board D3 to LCD pin 16 (red backlight)
Board D5 to LCD pin 17 (green backlight)
Board D6 to LCD pin 18 (blue backlight)

Python Computer Wiring

Since there's dozens of Linux computers/boards you can use we will show wiring for Raspberry Pi. For other platforms, please visit the guide for CircuitPython on Linux to see whether your platform is supported ().

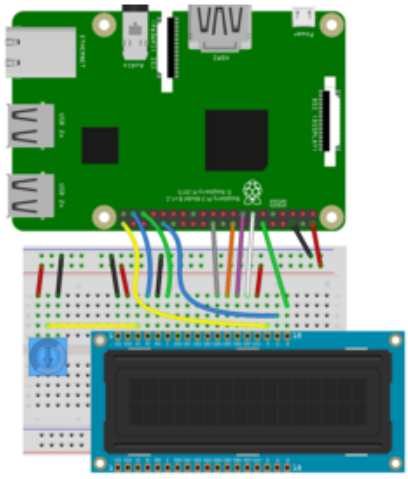
Here's the Raspberry Pi wired to a single color backlight character LCD:



Pi 5V to LCD pin 2, 15, and one side of the potentiometer.
Pi GND to LCD pin 1, 5, 16, and the opposite side of the potentiometer.
Potentiometer output (middle pin) to LCD pin 3
Pi GPIO26 to LCD pin 4
Pi GPIO19 to LCD pin 6
Pi GPIO25 to LCD pin 11
Pi GPIO24 to LCD pin 12
Pi GPIO22 to LCD pin 13
Pi GPIO27 to LCD pin 14

If you're using a RGB backlight character LCD, remember each of the red, green, blue color channels needs to be wired to a PWM-capable output pin on your computer (pins 10, 12, 18, 21 on the Raspberry Pi).

Here's an example of wiring an RGB backlight character LCD to a Raspberry Pi:



Pi 5V to LCD pin 2, 15, and one side of the potentiometer.
Pi GND to LCD pin 1, 5, and the opposite side of the potentiometer.
Potentiometer output (middle pin) to LCD pin 3
Pi GPIO26 to LCD pin 4
Pi GPIO19 to LCD pin 6
Pi GPIO25 to LCD pin 11
Pi GPIO24 to LCD pin 12
Pi GPIO22 to LCD pin 13
Pi GPIO27 to LCD pin 14
PI GPIO21 to LCD pin 16 (red backlight)
Pi GPIO12 to LCD pin 17 (green backlight)
Pi GPIO18 to LCD pin 18 (blue backlight)

CircuitPython Installation of CharLCD Library

You'll need to install the Adafruit CircuitPython CharLCD () library on your CircuitPython board.

First make sure you are running the latest version of Adafruit CircuitPython () for your board.

Next you'll need to install the necessary libraries to use the hardware--carefully follow the steps to find and install these libraries from Adafruit's CircuitPython library bundle (). Our CircuitPython starter guide has a great page on how to install the library bundle ().

For non-express boards like the Trinket M0 or Gemma M0, you'll need to manually install the necessary libraries from the bundle:

- adafruit_character_lcd
- adafruit_mcp230xx
- adafruit_74hc595
- adafruit_bus_device

Before continuing make sure your board's lib folder or root filesystem has the adafruit_character_lcd, adafruit_mcp230xx, adafruit_74hc595 and adafruit_bus_device files and folders copied over.

Next connect to the board's serial REPL () so you are at the CircuitPython >>> prompt.

Python Installation of CharLCD Library

You'll need to install the Adafruit_Blinka library that provides the CircuitPython support in Python. This may also require enabling I2C on your platform and verifying you are running Python 3. Since each platform is a little different, and Linux changes often, please visit the CircuitPython on Linux guide to get your computer ready ()!

Once that's done, from your command line run the following command:

- `sudo pip3 install adafruit-circuitpython-charlcd`

If your default Python is version 3 you may need to run 'pip' instead. Just make sure you aren't trying to use CircuitPython on Python 2.x, it isn't supported!

Python & CircuitPython Usage

To demonstrate the usage of the character LCD we'll initialize it and display text using Python code.

First you need to import the `digitalio` module and define all the pins connected to the LCD. If you followed the Metro M0 wiring for a single color backlight display on this page you'd want to use:

```
import board
import digitalio
lcd_rs = digitalio.DigitalInOut(board.D7)
lcd_en = digitalio.DigitalInOut(board.D8)
lcd_d7 = digitalio.DigitalInOut(board.D12)
lcd_d6 = digitalio.DigitalInOut(board.D11)
lcd_d5 = digitalio.DigitalInOut(board.D10)
lcd_d4 = digitalio.DigitalInOut(board.D9)
```

For use with Raspberry Pi, you need to change the pin assignments to match the Pi pins chosen. If you followed the Raspberry Pi wiring for a single color backlight display on this page, you'd want to use:

```
import board
import digitalio
lcd_rs = digitalio.DigitalInOut(board.D26)
lcd_en = digitalio.DigitalInOut(board.D19)
lcd_d7 = digitalio.DigitalInOut(board.D27)
lcd_d6 = digitalio.DigitalInOut(board.D22)
lcd_d5 = digitalio.DigitalInOut(board.D24)
lcd_d4 = digitalio.DigitalInOut(board.D25)
```

To use a Raspberry Pi with a character LCD with an RGB backlight, check out the example found here ().

For an RGB backlight on the Metro M0 Express, you need to import both the `digitalio` and `pwmio` modules and create the extra PWM lines. For example with the Metro M0 wiring on this page:

```
import board
import digitalio
import pwmio
lcd_rs = digitalio.DigitalInOut(board.D7)
lcd_en = digitalio.DigitalInOut(board.D8)
lcd_d7 = digitalio.DigitalInOut(board.D12)
lcd_d6 = digitalio.DigitalInOut(board.D11)
lcd_d5 = digitalio.DigitalInOut(board.D10)
lcd_d4 = digitalio.DigitalInOut(board.D9)
red = pwmio.PWMOut(board.D3)
green = pwmio.PWMOut(board.D5)
blue = pwmio.PWMOut(board.D6)
```

Now define the size of your character LCD in number of columns and rows, for example for a 16 character wide by 2 row tall display:

```
lcd_columns = 16
lcd_rows = 2
```

Next import the character LCD module and create an instance of the `Character_LCD` or `Character_LCD_RGB` class depending on what type of display you have wired up. For example for the single color backlight display:

```
import adafruit_character_lcd.character_lcd as characterlcd
lcd = characterlcd.Character_LCD_Mono(lcd_rs, lcd_en, lcd_d4, lcd_d5, lcd_d6,
lcd_d7, lcd_columns, lcd_rows)
```

Or for a RGB backlight display:

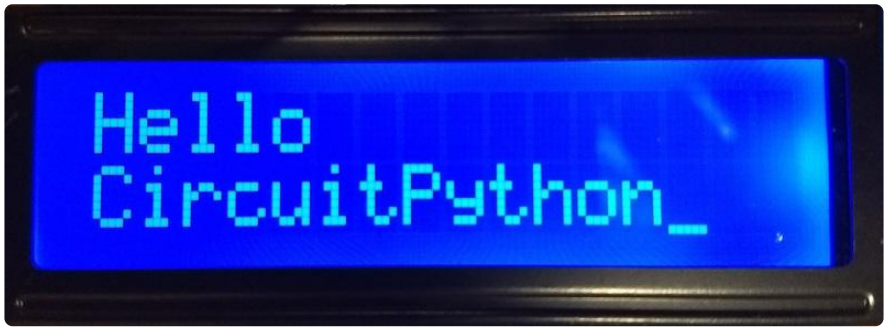
```
import adafruit_character_lcd.character_lcd as characterlcd
lcd = characterlcd.Character_LCD_RGB(lcd_rs, lcd_en, lcd_d4, lcd_d5, lcd_d6,
lcd_d7, lcd_columns, lcd_rows, red, green, blue)
```

Now you can print a message using the `message` property, for example to print on two lines (notice the `\n` line break added to the string in the middle):

```
lcd.message = "Hello\nCircuitPython!"
```

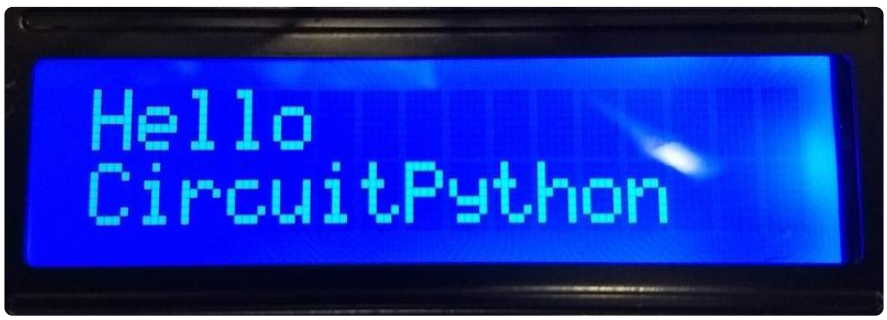
You can turn the cursor on and off using the `cursor` property. Set to True to turn it on and False to turn it off, for example to turn on:

```
lcd.cursor = True
```



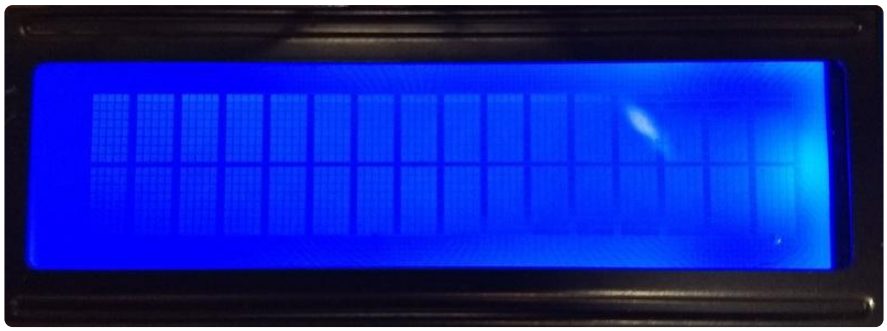
And to turn off:

```
lcd.cursor = False
```



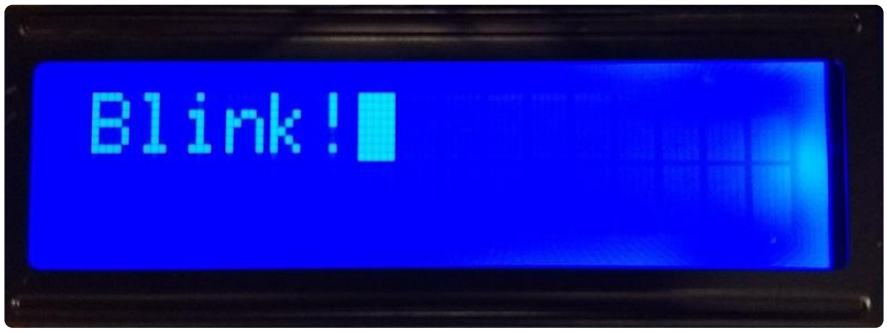
You can clear the entire display using the `clear` function:

```
lcd.clear()
```



You can also blink the cursor by turning it on and then calling the `blink` property and setting it to a boolean value. Set to True will start blinking the cursor, and False will disable blinking. For example to print a message and blink the cursor:

```
lcd.cursor = True
lcd.blink = True
lcd.message = "Blink!"
```

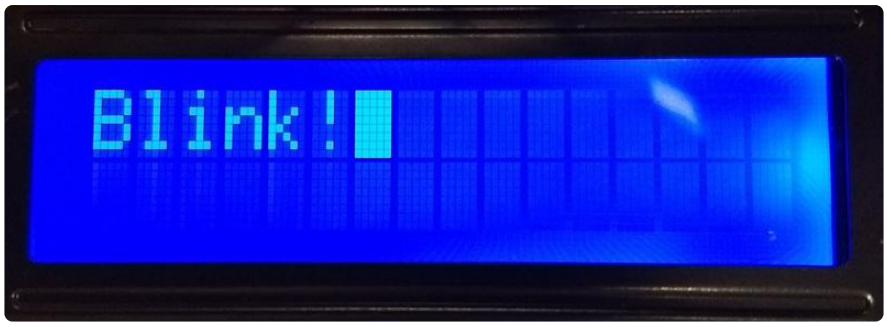


Finally the `move_left` message will move the printed message one character left, like if you wanted to scroll it off the screen. Try it:

```
lcd.move_left()
```

Or call `move_right` to scroll a character back to the right:

```
lcd.move_right()
```

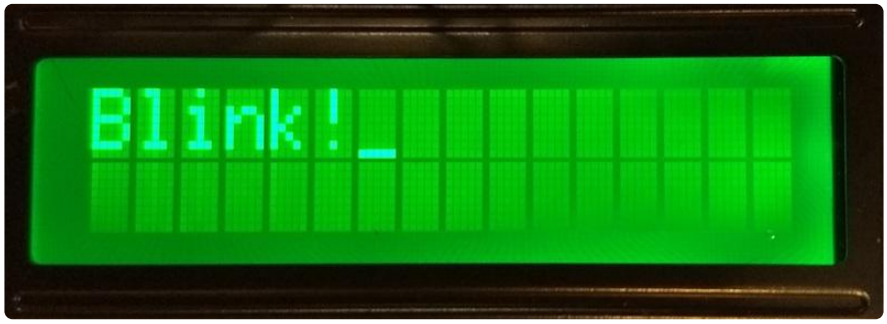


If you're using the RGB backlight display there's one extra function you can use to change the backlight color. Use the `color` function and provide a three-member list of red, green, blue color values, i.e. `[R, G, B]`, that range from 0 to 100. For example to set the color to red:

```
lcd.color = [100, 0, 0]
```

Or to green:

```
lcd.color = [0, 100, 0]
```

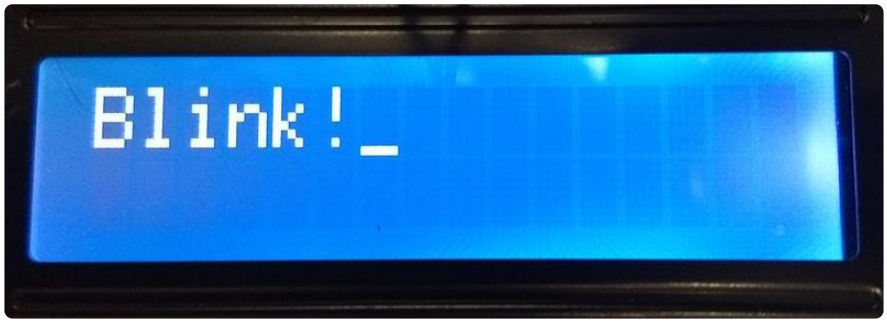


Or to blue:

```
lcd.color = [0, 0, 100]
```

Or any color in between, like a pleasing warm white with full red, nearly full green, and half blue:

```
lcd.color = [100, 80, 50]
```



That's all there is to using a character LCD with CircuitPython! Be sure to see the examples in the character LCD library () too for more fun like creating and printing custom characters.

Full Example Code

Metro M0/M4 simpletest example:

```
# SPDX-FileCopyrightText: 2021 ladyada for Adafruit Industries
# SPDX-License-Identifier: MIT

"""Simple test for monochromatic character LCD"""
import time
import board
import digitalio
import adafruit_character_lcd.character_lcd as characterlcd

# Modify this if you have a different sized character LCD
lcd_columns = 16
lcd_rows = 2

# Metro M0/M4 Pin Config:
lcd_rs = digitalio.DigitalInOut(board.D7)
lcd_en = digitalio.DigitalInOut(board.D8)
lcd_d7 = digitalio.DigitalInOut(board.D12)
lcd_d6 = digitalio.DigitalInOut(board.D11)
lcd_d5 = digitalio.DigitalInOut(board.D10)
lcd_d4 = digitalio.DigitalInOut(board.D9)
```

```
lcd_backlight = digitalio.DigitalInOut(board.D13)

# Initialise the LCD class
lcd = characterlcd.Character_LCD_Mono(
    lcd_rs, lcd_en, lcd_d4, lcd_d5, lcd_d6, lcd_d7, lcd_columns, lcd_rows,
lcd_backlight
)

# Turn backlight on
lcd.backlight = True
# Print a two line message
lcd.message = "Hello\nCircuitPython"
# Wait 5s
time.sleep(5)
lcd.clear()
# Print two line message right to left
lcd.text_direction = lcd.RIGHT_TO_LEFT
lcd.message = "Hello\nCircuitPython"
# Wait 5s
time.sleep(5)
# Return text direction to left to right
lcd.text_direction = lcd.LEFT_TO_RIGHT
# Display cursor
lcd.clear()
lcd.cursor = True
lcd.message = "Cursor! "
# Wait 5s
time.sleep(5)
# Display blinking cursor
lcd.clear()
lcd.blink = True
lcd.message = "Blinky Cursor!"
# Wait 5s
time.sleep(5)
lcd.blink = False
lcd.clear()
# Create message to scroll
scroll_msg = "<-- Scroll"
lcd.message = scroll_msg
# Scroll message to the left
for i in range(len(scroll_msg)):
    time.sleep(0.5)
    lcd.move_left()
lcd.clear()
lcd.message = "Going to sleep\nCya later!"
time.sleep(3)
# Turn backlight off
lcd.backlight = False
time.sleep(2)
```

Raspberry Pi simpletest example:

```
# SPDX-FileCopyrightText: 2021 ladyada for Adafruit Industries
# SPDX-License-Identifier: MIT

"""Simple test for monochromatic character LCD on Raspberry Pi"""
import time
import board
import digitalio
import adafruit_character_lcd.character_lcd as characterlcd

# Modify this if you have a different sized character LCD
lcd_columns = 16
lcd_rows = 2

# Raspberry Pi Pin Config:
```

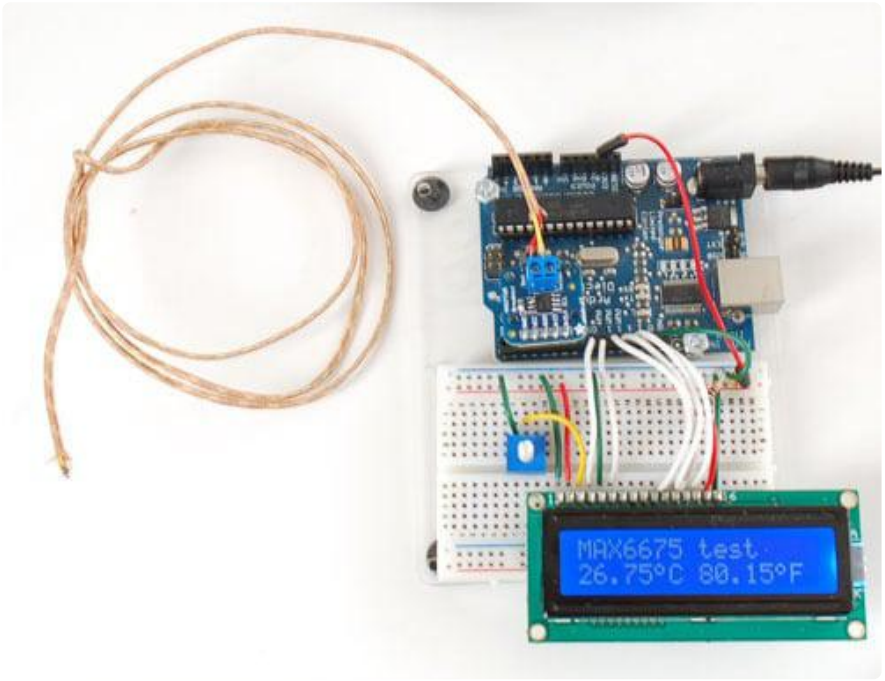
```
lcd_rs = digitalio.DigitalInOut(board.D26)
lcd_en = digitalio.DigitalInOut(board.D19)
lcd_d7 = digitalio.DigitalInOut(board.D27)
lcd_d6 = digitalio.DigitalInOut(board.D22)
lcd_d5 = digitalio.DigitalInOut(board.D24)
lcd_d4 = digitalio.DigitalInOut(board.D25)
lcd_backlight = digitalio.DigitalInOut(board.D4)

# Initialise the lcd class
lcd = characterlcd.Character_LCD_Mono(
    lcd_rs, lcd_en, lcd_d4, lcd_d5, lcd_d6, lcd_d7, lcd_columns, lcd_rows,
lcd_backlight
)

# Turn backlight on
lcd.backlight = True
# Print a two line message
lcd.message = "Hello\nCircuitPython"
# Wait 5s
time.sleep(5)
lcd.clear()
# Print two line message right to left
lcd.text_direction = lcd.RIGHT_TO_LEFT
lcd.message = "Hello\nCircuitPython"
# Wait 5s
time.sleep(5)
# Return text direction to left to right
lcd.text_direction = lcd.LEFT_TO_RIGHT
# Display cursor
lcd.clear()
lcd.cursor = True
lcd.message = "Cursor! "
# Wait 5s
time.sleep(5)
# Display blinking cursor
lcd.clear()
lcd.blink = True
lcd.message = "Blinky Cursor!"
# Wait 5s
time.sleep(5)
lcd.blink = False
lcd.clear()
# Create message to scroll
scroll_msg = "<-- Scroll"
lcd.message = scroll_msg
# Scroll message to the left
for i in range(len(scroll_msg)):
    time.sleep(0.5)
    lcd.move_left()
lcd.clear()
lcd.message = "Going to sleep\nCya later!"
# Turn backlight off
lcd.backlight = False
time.sleep(2)
```

Python Docs

Python Docs ()

The createChar Command

You may want to have special characters, for example in this temperature sensor, we created a 'degree' symbol (°)



You can do that with the createChar command, and to help you out we're going to point you to this really great website that does the hard work for you! ()