Character LCDs

Wiring up a character LCD to an Arduino

April 27, 2012 11:17

Introduction

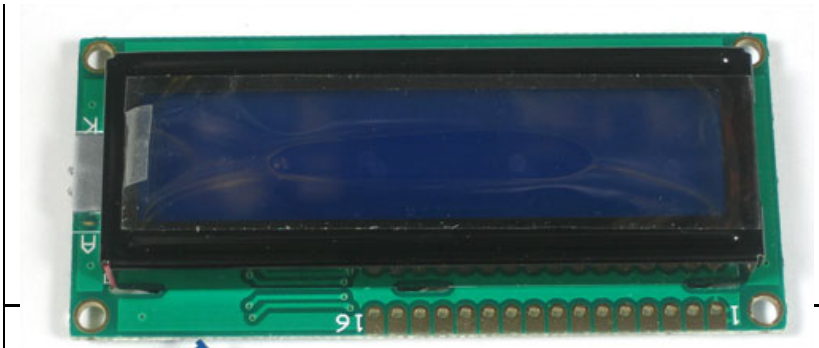
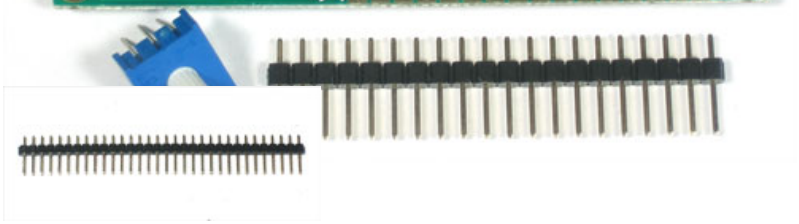
We sell tons of lovely character LCDs for use with Arduino , they are extremely common and a fast way to have your project show status messages. This tutorial will show how you can easily connect a character LCD, either 16x2 or 20x4 .

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!

What you'll need

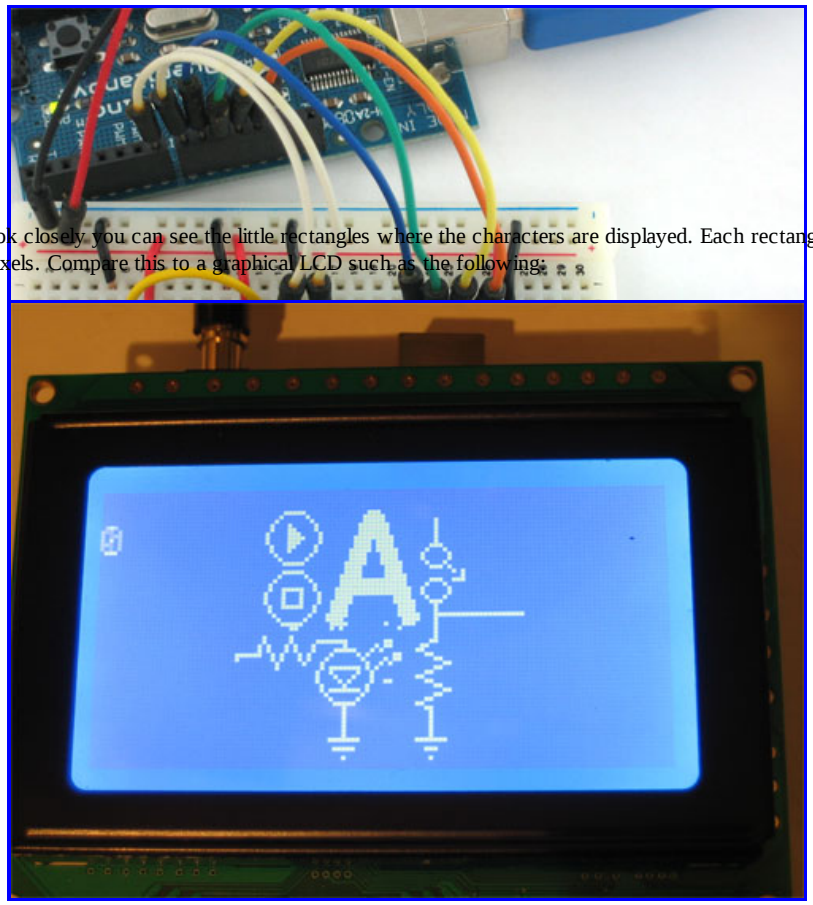
	Assembled Arduino board, preferrably a Duemilanove, or Diecimila (or whatever the latest version is) but NG is OK too	Adafruit $30
	USB Cable. Standard A-B cable is required. Any length is OK.	Adafruit $4
	Character LCD with parallel	

	interface The one from Adafruit comes with extra parts below	Adafruit $10
	Strip of 0.1" header - at least 16 pins long This comes with the Adafruit LCD's but if you got some elsewhere you'll want to buy some	Adafruit male header pack
	10K linear potentiometer The one that comes with the Adafruit kit is perfect, but you can use any 10K potentiometer or trimmer	Adafruit
	Hookup Wire Make sure its *not* stranded wire!	Adafruit

Character v. Graphical LCDs

There are hundreds of different kinds of LCDs, the ones we'll be covering here are **character** 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:



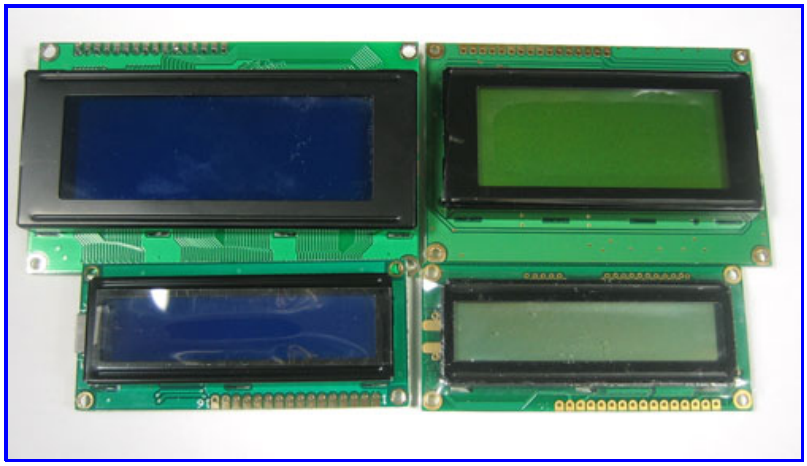
If you look closely you can see the little rectangles where the characters are displayed. Each rectangle is a grid of pixels. Compare this to a graphical LCD such as the following:

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!

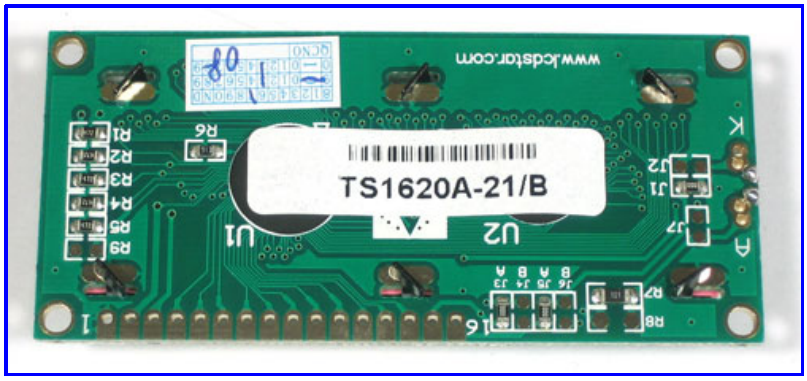
All kinds!

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

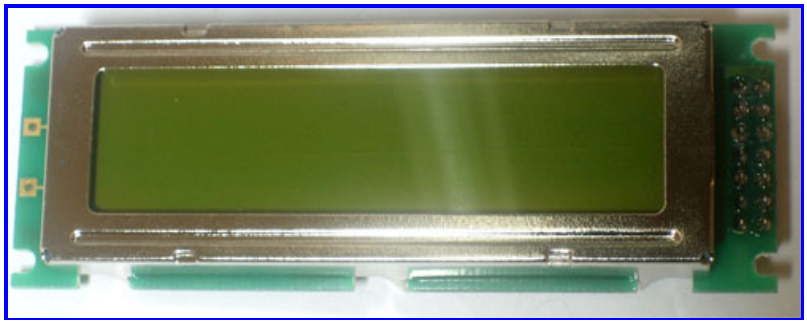


Although they display only text, they do some in many shapes: from top left we have a 20x4 with white text on blue background, a 16x4 with black text on green, 16x2 with white text on blue and a 16x1 with black text on gray.

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!



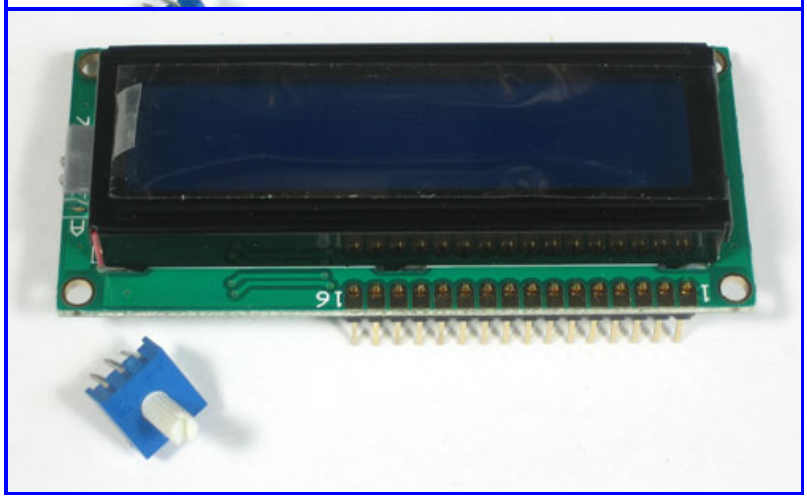
For this part of the tutorial, we'll be using LCDs with a single strip of 16 pins as shown above. There are *also* some with 2 lines of 8 pins like so:



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

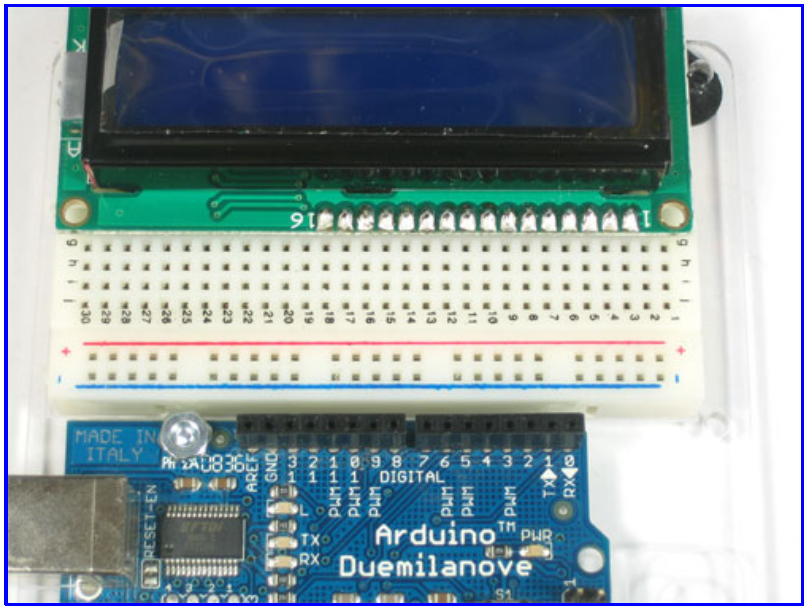
Get ready!

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!

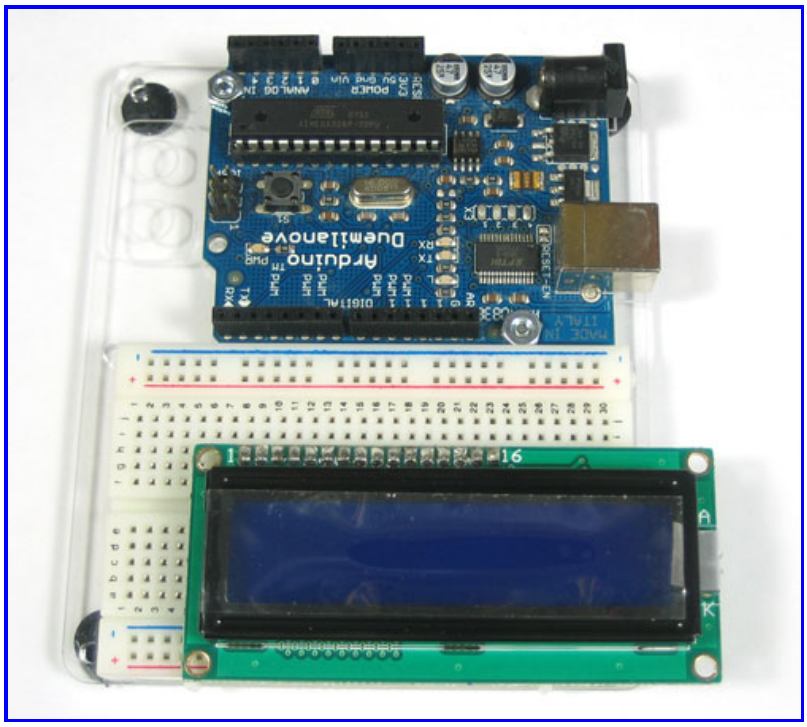
Next you'll need to solder the header to the LCD.**You must do this, it is not OK to just try to 'press fit' the LCD!**



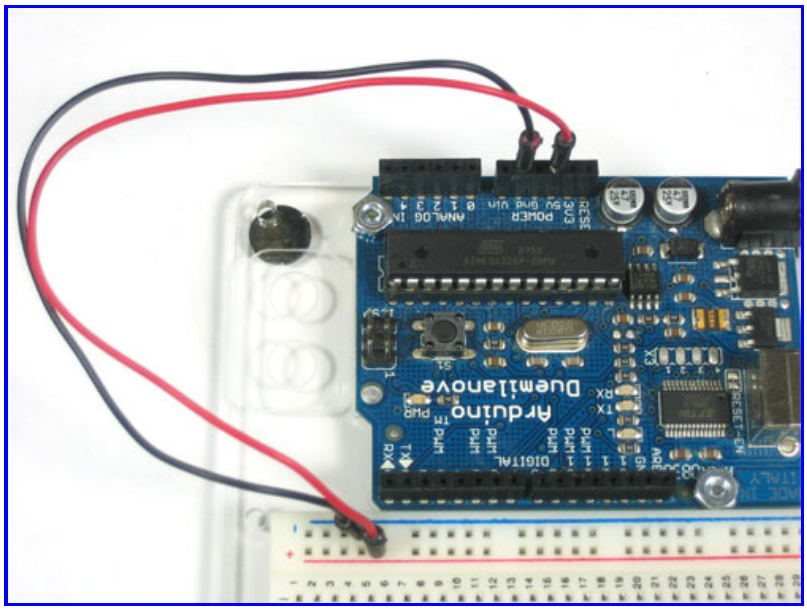
The easiest way we know of doing this is sticking the header into a breadboard and then sitting the LCD on top while soldering. this keeps it steady.

Power and backlight

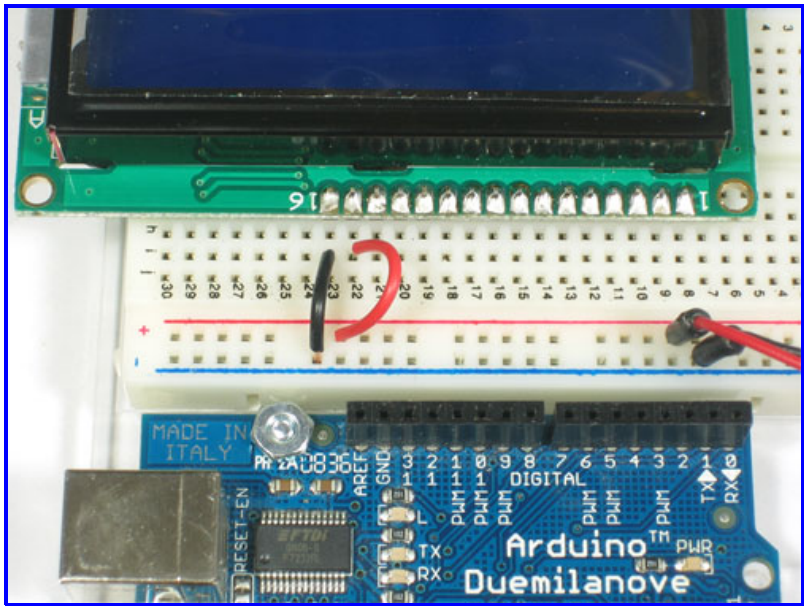
OK now we're onto the interesting stuff! Get your LCD plugged into the breadboard



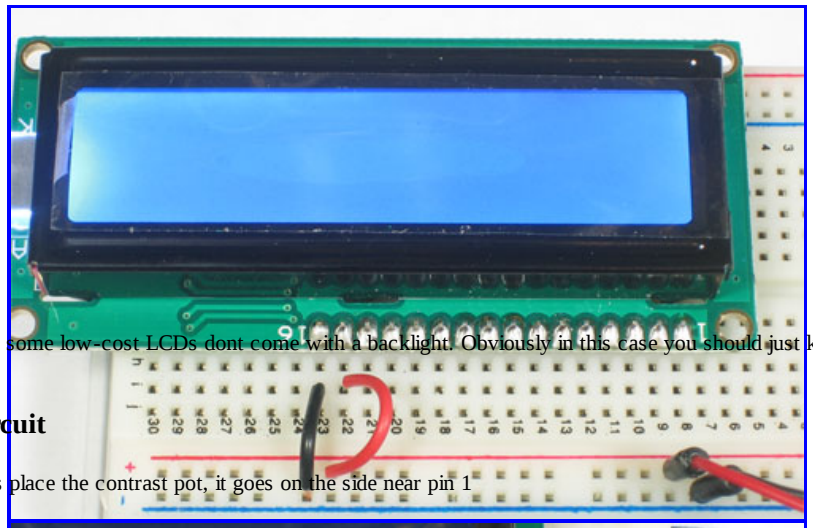
Now we'll provide power to the breadboard. Connect +5V to the red rail, and Ground to the blue rail.



Next we'll connect up the backlight for the LCD. Connect pin 16 to ground and pin 15 to +5V through a series resistor. 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.



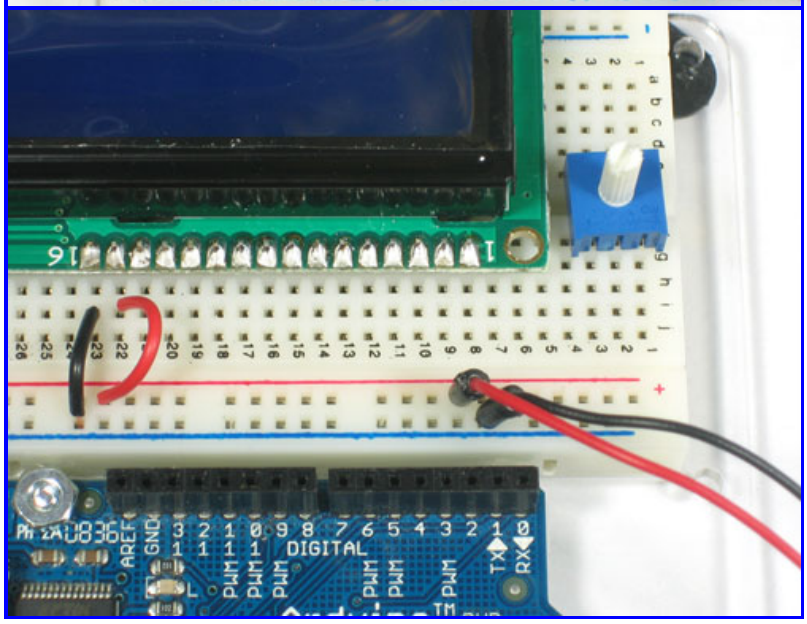
Connect the Arduino up to power, you'll notice the backlight lights up



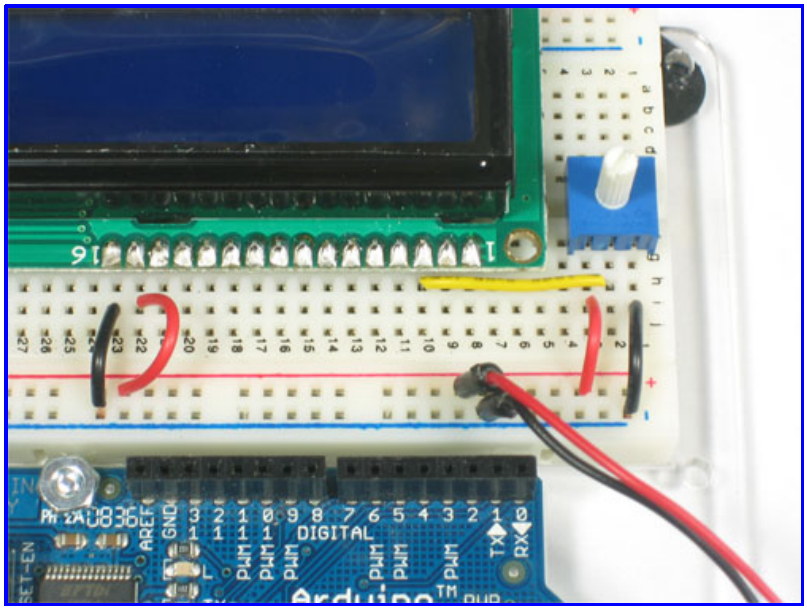
Note that some low-cost LCDs dont come with a backlight. Obviously in this case you should just keep going.

Contrast circuit

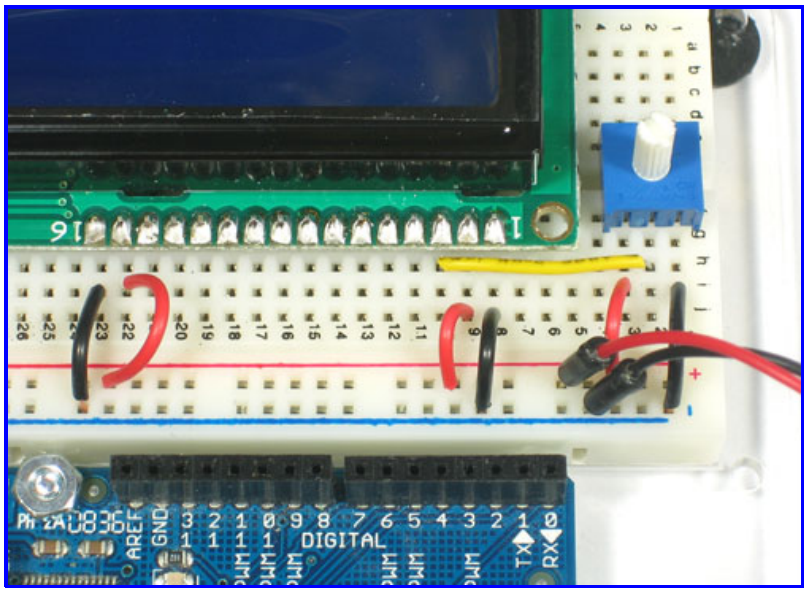
Next, lets place the contrast pot, it goes on the side near pin 1



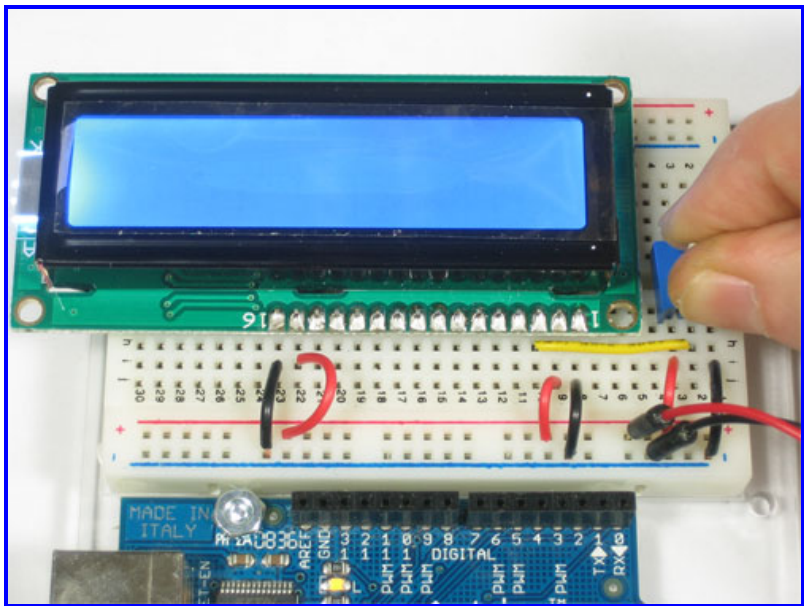
Connect one side of the pot to +5V and the other to Ground (it doesn't matter which goes on what side). The middle of the pot (wiper) connects to pin 3

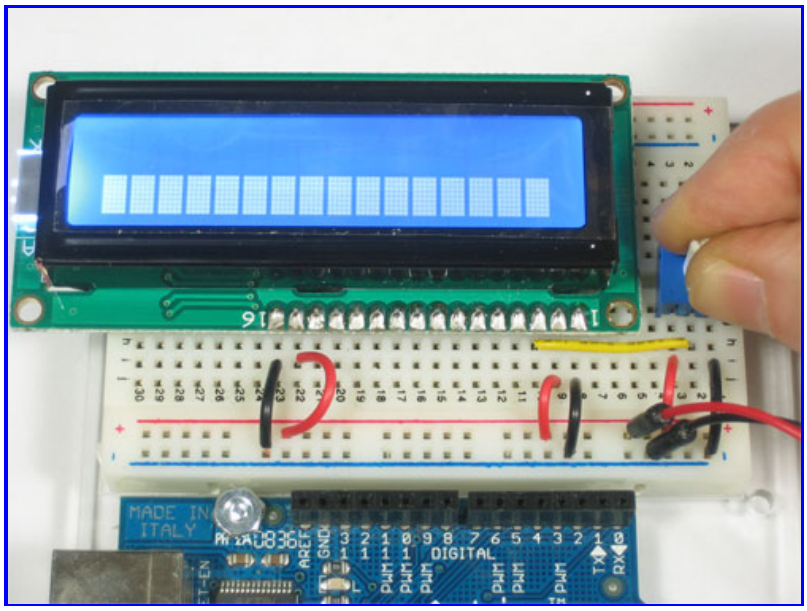


Now we'll wire up the logic of the LCD - this is seperate from the backlight! Pin 1 is ground and pin 2 is +5V



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

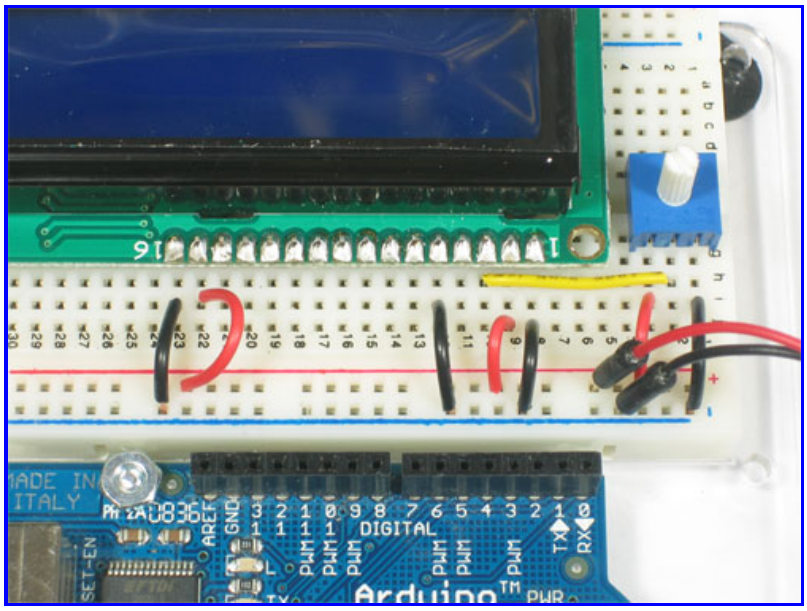
Now we'll finish up the wiring by connecting the data lines. There are 11 bus lines: **D0** through **D7** (8 data lines) and **RS**, **EN**, and **RW**. D0-D7 are the pins that have the raw data we send to the display. The **RS** pin lets the microcontroller tell the LCD whether it wants to display 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!

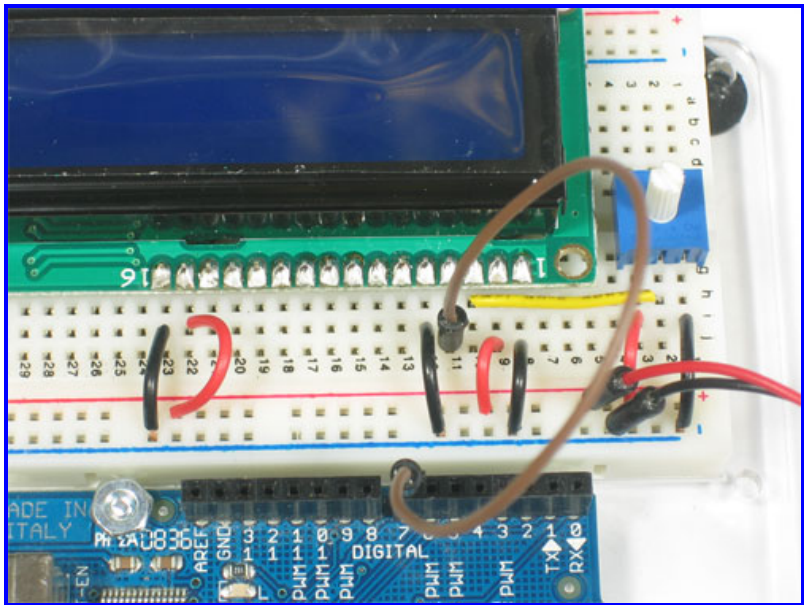
So to recap, we need 6 pins: **RS, EN, D7, D6, D5,** and **D4** to talk to the LCD.

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

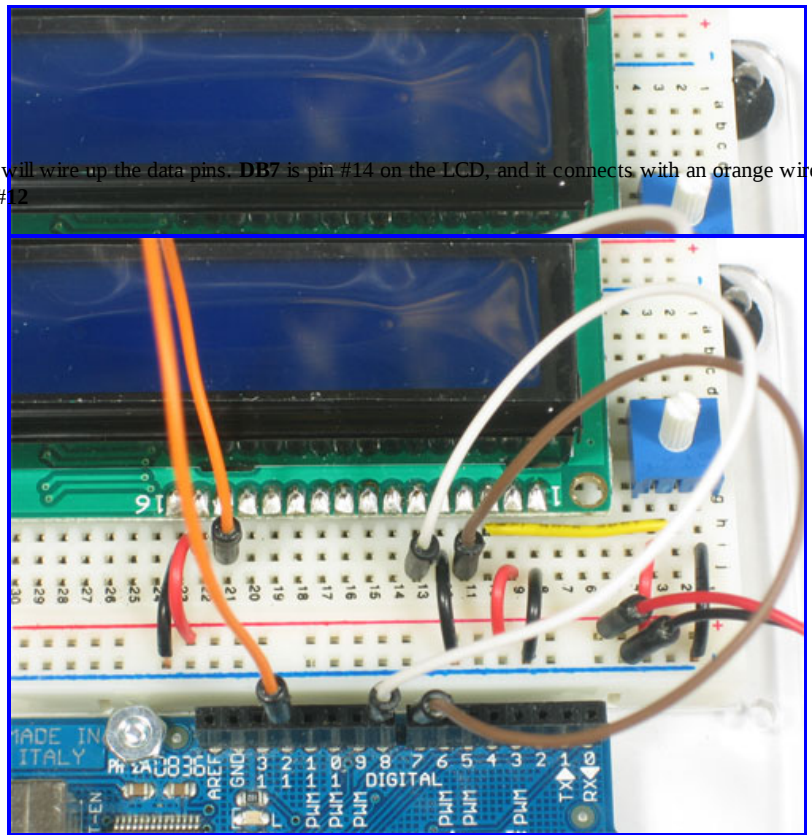
As mentioned, we'll not be using the **RW** pin, so we can tie it go ground. That's pin 5 as shown bere:



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

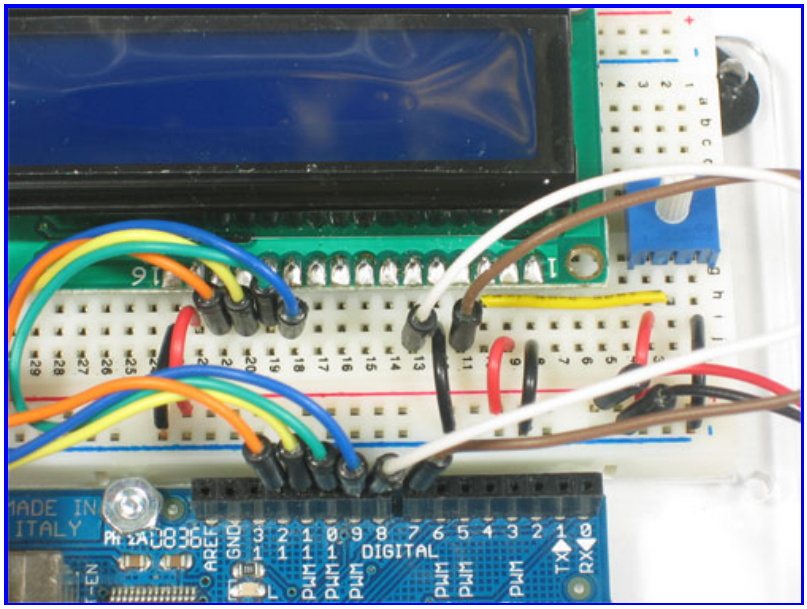


Next is the **EN** pin #6, we'll use a white wire to connect it to Arduino digital #**8**

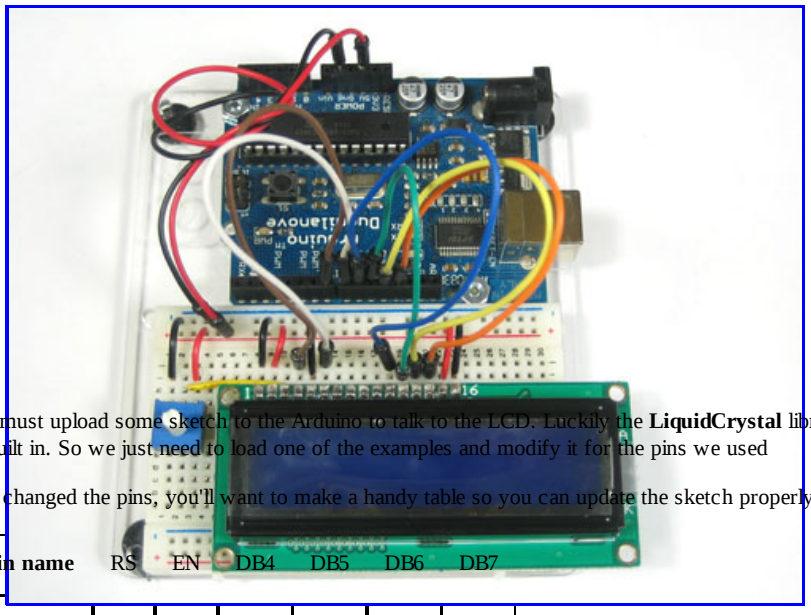


Now we will wire up the data pins. **DB7** is pin #14 on the LCD, and it connects with an orange wire to Arduino #12

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



This is what you'll have on your desk:



Sketch time

Now we must upload some sketch to the Arduino to talk to the LCD. Luckily the **LiquidCrystal** 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:

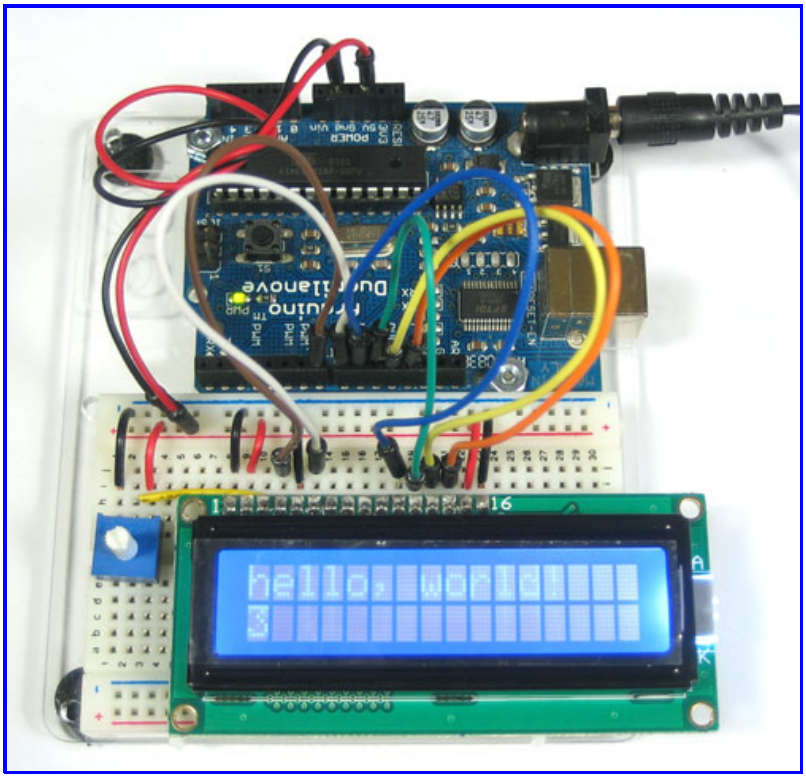
```
LiquidCrystal lcd(12, 11, 5, 4, 3, 2);
```

And change it to:

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

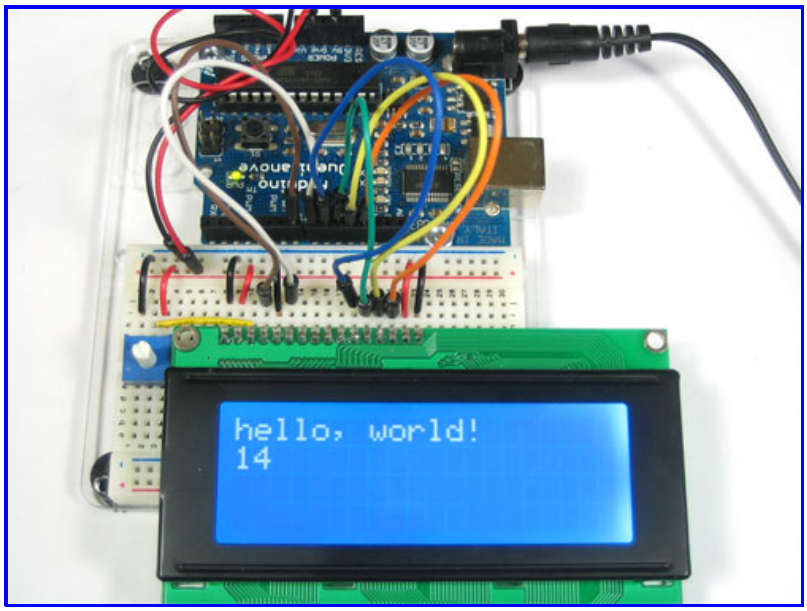
To match the pin table we just made.

Now you can compile and upload the sketch

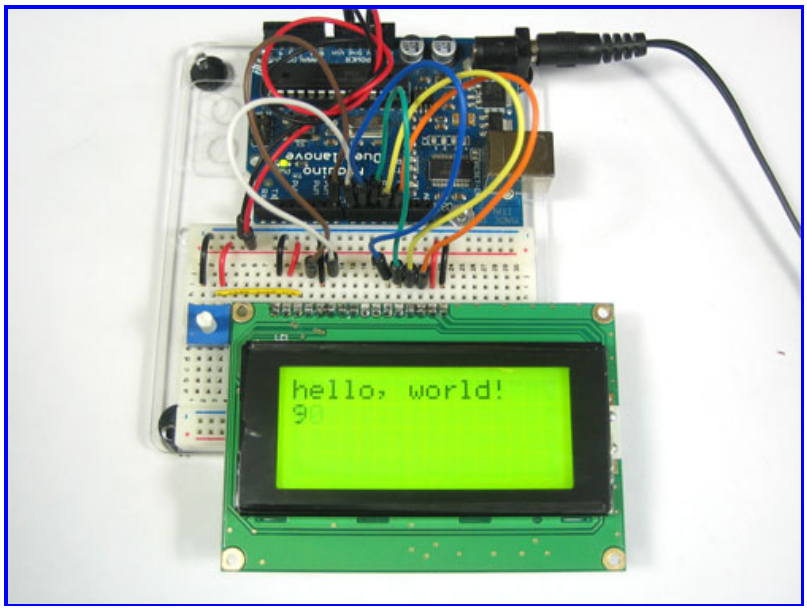


Adjust the contrast if necessary

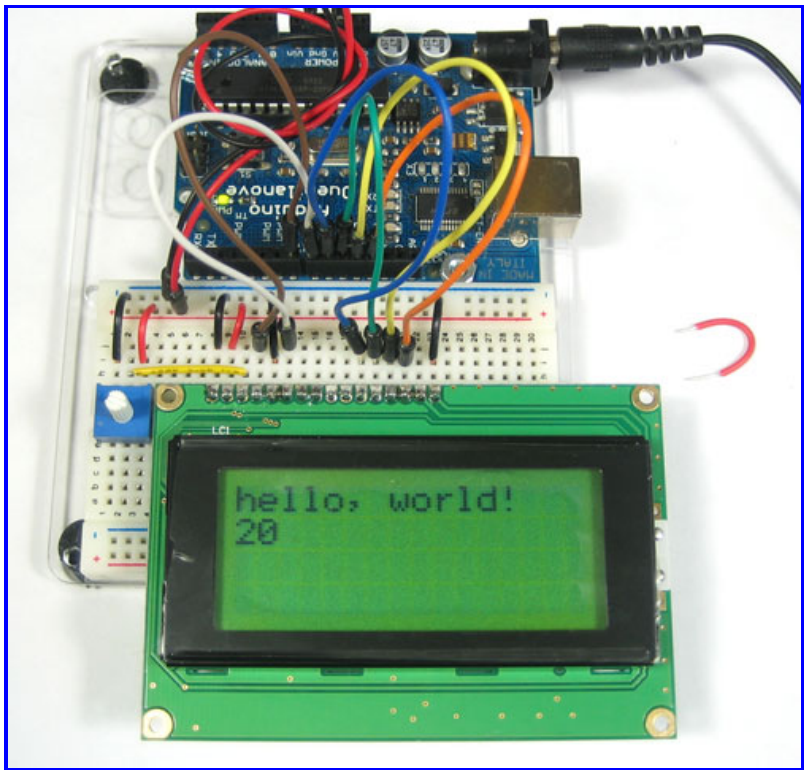
You can of course use any size or color LCD, such as a 20x4 LCD:



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

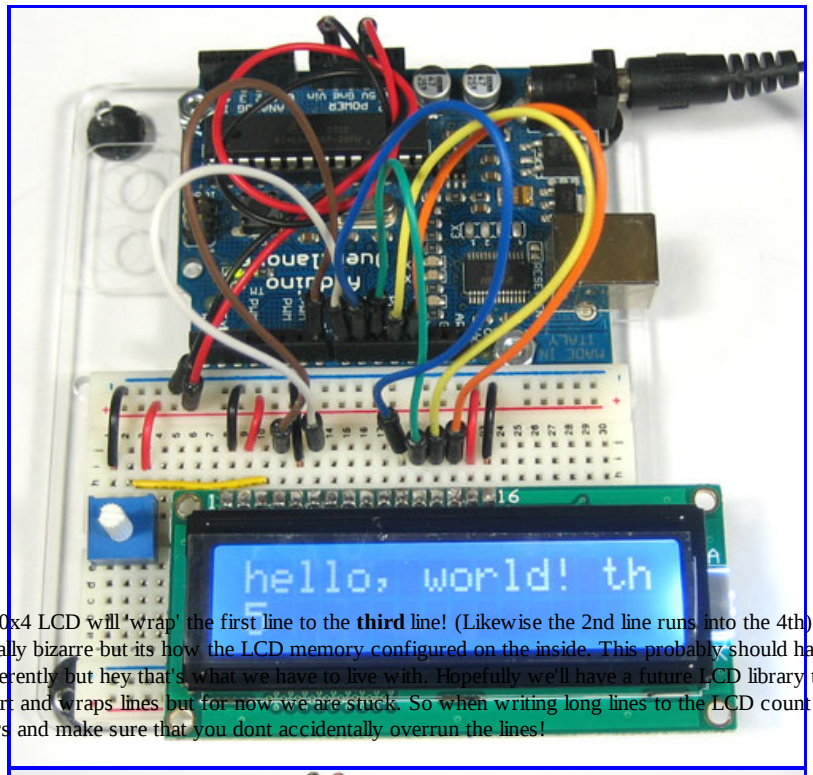
One thing you'll want to watch for is how the LCD handles large messages and multiple lines. For example if you changed this line

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

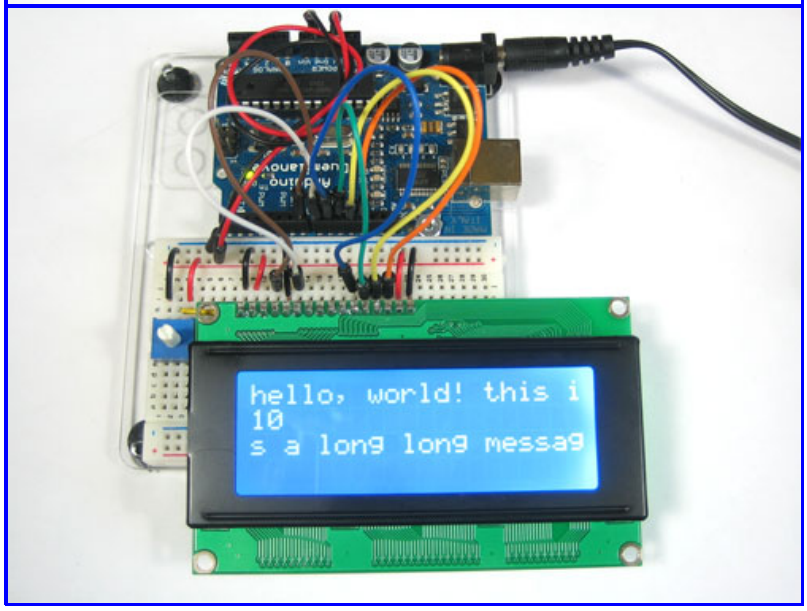
To this:

```
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!

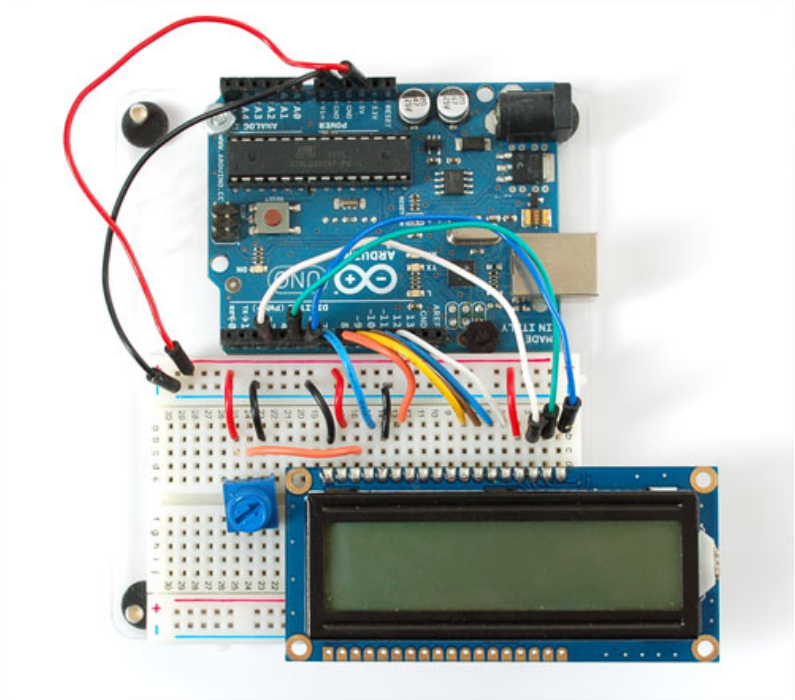


RGB backlight LCDs

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 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!



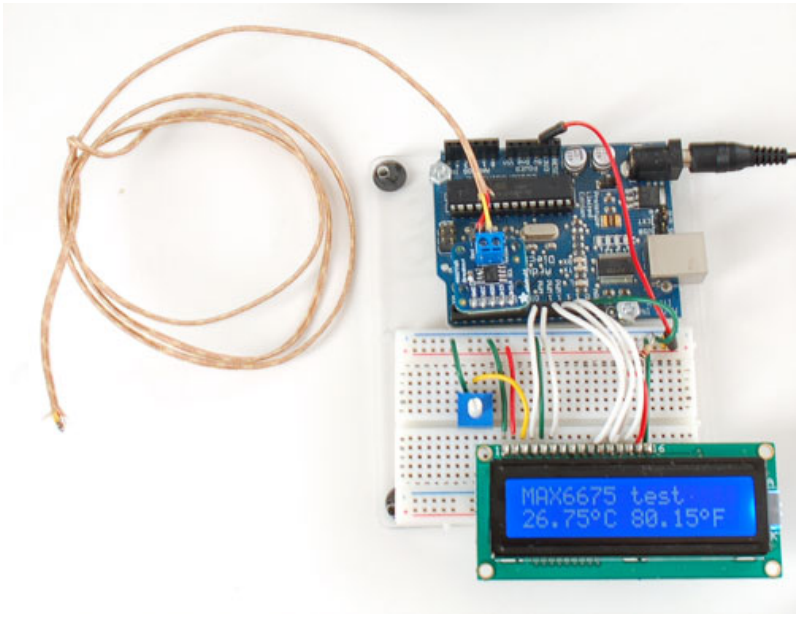
Now upload this code to your Arduino to see the LCD background light swirl! (Click here to see what it looks like in action)

```
// include the library code:
#include <LiquidCrystal.h>
#include <Wire.h>

#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 -> 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 ");

  brightness = 100;
}


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

BONUS! making your OWN character

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!

This page was autogenerated from http://www.ladyada.net/wiki//tutorials/learn/lcd/charlcd.html
Please edit the wiki to contribute any updates or corrections.