CSIS 10C Lab 4B Electronic Mood Ring

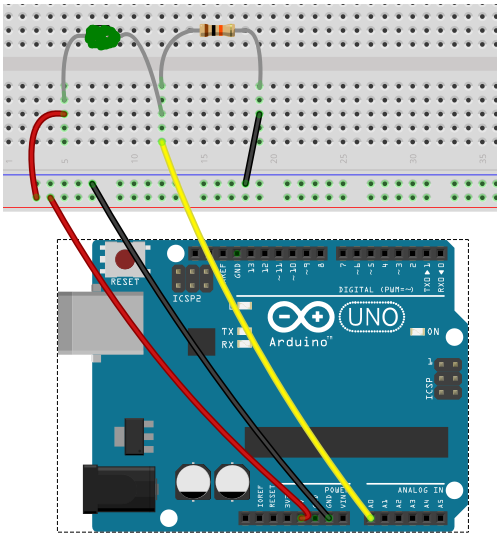
Introduction

Are you "hot"? As we learned last week, many electronic sensors are built around resistors that are sensitive to something in the environment – light, heat, humidity, force, etc. In today's lab we will explore how to measure temperature using a thermistor– a temperature-sensitive resistor, two examples of which are shown at right, one green, one gold-red-yellow. *In the diagrams below, a green blob will indicate the thermistor*.

We will build a circuit with a thermistor arranged in a voltage divider, then we will add an RGB LED to the circuit and write software to use the temperature reading from the thermistor to control the color shown by the LED. This will give us an effect similar to the "mood ring" craze of the 1970s. You can see the effect we will be looking for at the video demo here.

Part 1: Resistor-Thermistor Divider

1. Start with folder 4B in the Labs folder.
2. Build the circuit at right, which consists of a resistor divider with a 10kΩ resistor in the R2 position (one end connected to ground) and the thermistor in the R1 position (one end connected to 5V).
   (10kΩ = Brn-Blk-Orng for 4 band resistors,
   or Brn-Blk-Blk-Red for 5 band resistors)



3. For this exercise, use the existing file *main.c* in folder *1_GetReading* and use "*make flash"* to compile/link/load the file onto the Uno.

4. Record the maximum and minimum numeric readings you get on the display when the thermistor reads different temperature levels. Try making your hands hot by doing a lot of pushups or something before the last reading.

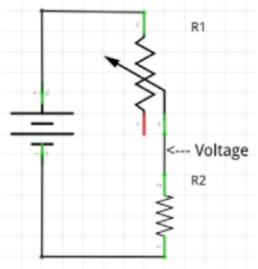
   Room temperature reading _____

   Normal Finger temperature reading _____

   After physical activity reading _____

   (your numbers should be around 500 – 600 if you have the correct resistor in the R2 location)

5. From last week we learned that the resistor divider formula allows us to get a voltage that moves up or down depending on the value of a variable resistor. If the thermistor resistance (R1) goes down when the temperature goes up, which way will the voltage at the midpoint of the divider go?



eVoltage = Voltage (Battery) x R2 / (R1 + R2)

6. Switch to folder *2_getVoltage* and open *main.c*. Add the code necessary to calculate and print the voltage

7. We are connecting the voltage of the resistor divider to pin `A0` of the Arduino, which will give us a number when we call the `analogRead(A0)` function. Let's call this number `N0`, remembering that `N0` is a digital number between 0 and 1023, representing a voltage between 0 and +5V. A typical value of `N0` will be about 500 or so given you have the correct resistor in the R2 place.

   To begin, rename the variable `sensorValue` in your current version to `N0`.

   The conversion of this number N0, read from pin A0 of the Arduino, to an actual temperature in Celsius, has many steps. The following formulae seems to work but may not be accurate since it is for a different model thermistor than the one we are using in this lab.

   Here are the steps needed to compute temperature in degrees Celsius from the N0 value read from the Arduino:

$$R = \left(\frac{1024}{N_0} - 1\right)$$

$$D = log\ (R)$$

$$T_K = \frac{1}{(0.003354016 + 0.0002569850\,D + 0.000002620131\,D^2 + 0.00000006383091\,D^3}$$

$$T_C = T_K - 273.15$$

8. Write a function that calculates and returns TC from N0 using the above steps. Call this function Therm_to_degC. It should have one parameter (an int for N0) and return a double. Make sure you use floating point arithmetic for all the steps above. You can use the log( ) function to compute the natural logarithm. And when you want to square or cube a variable you can try using the pow function, such as pow(D,2), or if that doesn't work, just multiple D by itself.

9. Call your function from loop, and use the return value to print the estimate of the temperature of the thermistor at room temperature. Compare the results to the thermometer – are they reasonably close?

   If you are wildly off, check your calculations. You should get a room temperature reading near 20°C. If not, you may want to do step 8 below to help get a better feel of different variables involved in your computation.

Part 2: Support Functions

In this section we will build a number of additional functions that will make our code easier to read and write as we go forward in this lab.

10. **<u>Printing multiple variables on one line:</u>** If you are trying to debug your code it can be very helpful to show more than one number on a line at a time, yet the Serial.println( ) function doesn't let us concatenate values like in Java.

    To make it easier, let's make our own print functions. We'll make three, one that takes a single parameter, one that takes two parameters and a third that takes three parameters. Each version will have the same name (let's call it print), and each one will print all of its parameters using Serial.print or Serial.println.

    So, for example, if you want to print one item, you will call print(N0);

    On the other hand, if you want to print two items, you will call print(N0, T_C); and so on for three items.

    Due to *function overloading*, even though all three functions have the same name, the correct function will be invoked based on matching the number of parameters in the function call.

    Write these functions now. Let's make the parameter type double so we can pass an int or a double variable to it. When you want to print more than one item on a line, make repeated calls to Serial.print, but for the last item print using Serial.println so that you go to a new line.

    For example, this code will print variables x and y on the same line, separated by a comma:

    ```
    Serial.print(x);
    Serial.print(", ");
    Serial.println(y);
    ```

    When you have these written, test your functions out by calling them on two or three variables (like N0 and T_C) you're using in the loop function.

11. **<u>Writing our own map function (myMap):</u>** We will be using the map function again to convert our temperature readings into values for the analogWrite command to an RGB LED, covering a range of 0 to 255. In other words, we will use the command

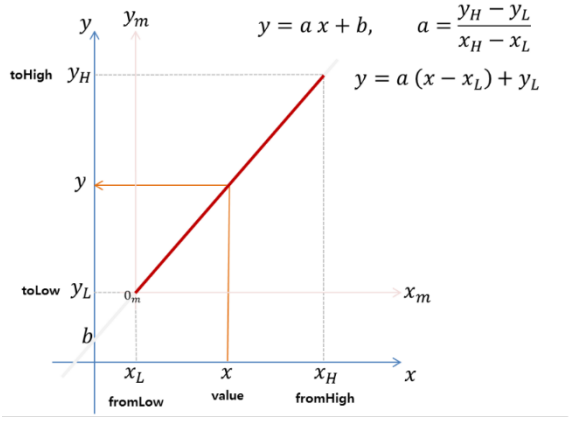
    ```
    int red = map(T_C, 25, 35, 0, 255)   // change 25 and 35 to your temperature min/max temps
    ```

    to map our temperature value to a number range that will affect the brightness of red light we shine from the RGB LED. A red value of 255 is a fully activated red LED, while 0 means no energy at all.

    Although you might be comfortable using the map command, this function is so useful that it may be necessary for you to use it in situations that don't involve the Arduino microcontroller, which is why we will be writing our own version of it now.

    What the map function does is take as input a value x which can extend over some domain xL to xH and returns a value y which extends over some range yL to yH. Mathematically, this is often called a linear transformation. The idea is expressed graphically by the image at right.

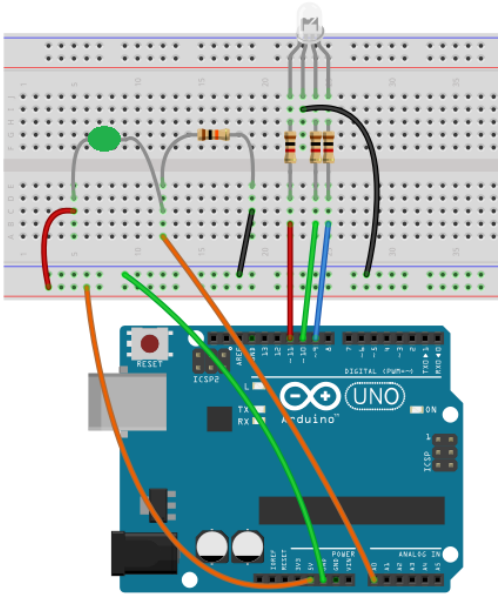
    Note that what we are computing is the equation of a line. The variable a at right is the slope of the line. The y-offset is yL – a(xL).



12. Write your myMap function now. It should accept five double variables and return an int. Try out your myMap and compare it to the map function by doing identical calls of each and printing the two results out to the console using your print function.

Part 3: Adding the RGB LED

We can now build the rest of the circuit by adding the RGB LED and adding code to turn on the red and blue LEDs along a linear range indicating how warm the temperature sensed by thermistor is.



13. Add three resistors and the RGB LED to your circuit as shown at right. The black wire must connect to the *longest wire* from the RGB LED. The example is using 1kΩ resistors (Brn-Blk-Rd), but you can use smaller values) *If you are using the AdaFruit kit, the black wire should go to +5V, and you can use the Grn-Blu-Brn resistors going to the LED pins.*

14. Add this function to your program so that you can set the red and blue LEDs.

```
void setLED(int blue, int red){
   analogWrite(BLUE_PIN, blue);
   analogWrite(RED_PIN, red);
}
```

15. You will also need to define the constants BLUE_PIN and RED_PIN in your program. If you wired up like the figure at right, these will be 9 and 11. These statements should go at the top of your program.

    int RED_PIN = 11;
    int GREEN_PIN = 10;
    int BLUE_PIN = 9;

16. Finally, in your loop function, add statements to compute the blue LED value, which should be 255 minus the red value so that these two colors are in a kind of opposition (when one is strong the other is weak):

    blue = 255 – red;

    When you now call the setLED function, you should see the RGB LED light up in some color that is between full blue and full red. *If you are using the AdaFruit kit, turn off the green LED by setting pin 10 to high using digitalWrite(10, HIGH); Also, to make it turn red when it gets warmer, swap the definitions of blue and red, so that blue gets the map from the T_C variable, and red gets 255-blue.*

    If you'd like an extra challenge, make a function that rapidly flashes the light 3x in pure red. Add statements to your loop function that calls this method if a preset high temperature is reached.

So if the person can raise their temperature to the target, this flashing behavior will indicate success.

When you are finished, save your code and upload it to Canvas.

Developed from: https://create.arduino.cc/projecthub/ben/temperature-controlled-rgb-led-6c8cdf