Making an Arduino Temperature and Humidity Sensor

Inside and outside Model

Knowledge is a process of piling up facts; wisdom lies in their simplification.

Martin H Fischer

Modification Record

Issue	Date	Author	Changes (Including the change authority)
A	05 April 2021	ZizWiz	Original from various notes
B	01 June 2021	ZizWiz	v2 includes LCD and pressure
C	04 June 2021	ZizWiz	v3 includes data logging.

Contents

1. Introduction

This project attaches 2 of DH22 Temperature/Humidity sensors to the Arduino Nano allowing us to measure the temperature inside and outside at the same time. We will add code to the Arduino to display the data on the LCD screen.

We also write a Windows app that will allow us to bring back the data to a PC

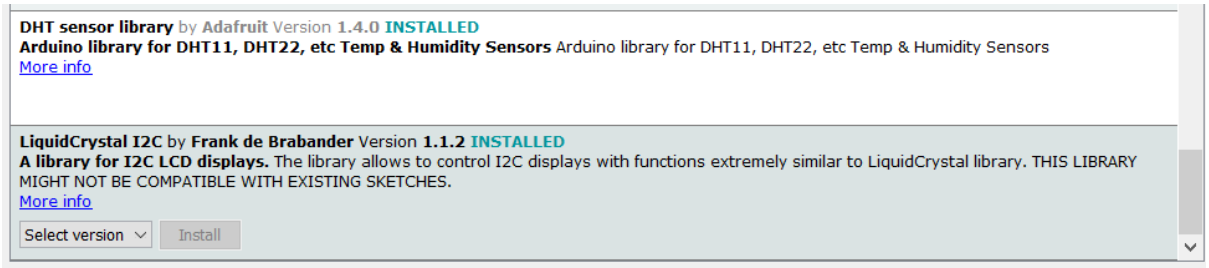
2. Parts list

For this project we will use:

- Arduino Uno or Nano
  - 16MHz
  - 14 Digital I/O
  - 6 Pulse Width
  - 6 Analogue I/O
  - 32kb Flash
  - 2kb RAM
- 1x DHT22 Temp/Hum module
- I^2C 4x20 LCD screen

3. Libraries

To make this work a few libraries are needed. Using them makes this a lot simpler.



4. Find Modules Addresses

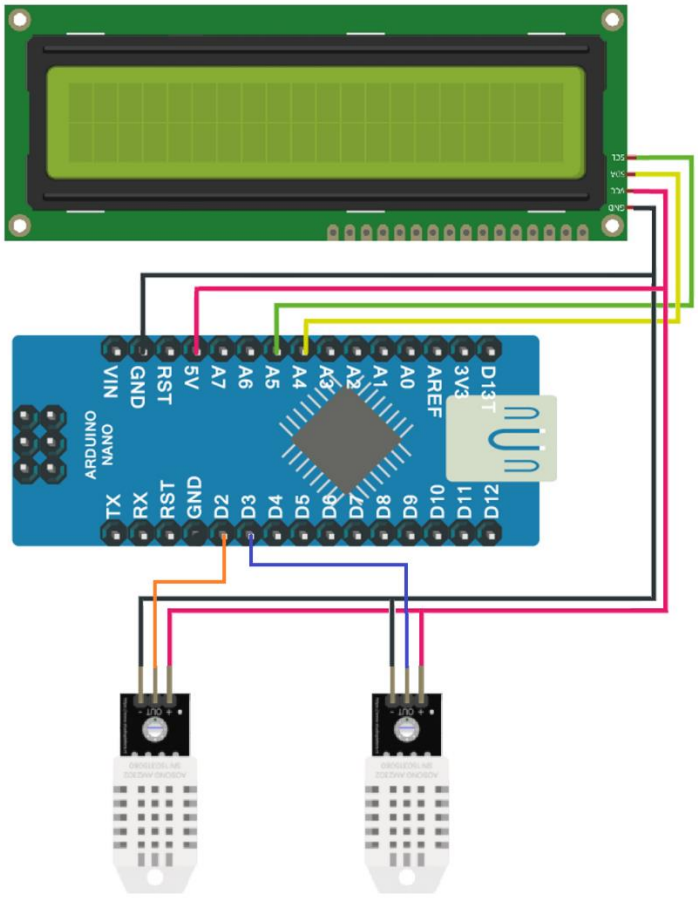
If you want you can find the address of each correctly wired I^2C device on your circuit.

You will find a code sample in Appendix A that you can use to scan for all the I^2C devices on your circuit.

5. v1 - DHT22/ AM2302 Temparature and Humidity Module

The unit I am building will have an Arduino Nano, a 2 line of 16 char LCD and two sensors DHT22. You can use DHT11 sensors, but they are less accurate than the DHT22.

5.1. Circuit Diagram



5.1.1. The DHT22 Spec

- 3 to 5V power and I/O
- 2.5mA max current use during conversion (while requesting data)
- Good for 0-100% humidity readings with 2-5% accuracy
- Good for -40 to 80°C temperature readings ±0.5°C accuracy
- No more than 0.5 Hz sampling rate (once every 2 seconds)

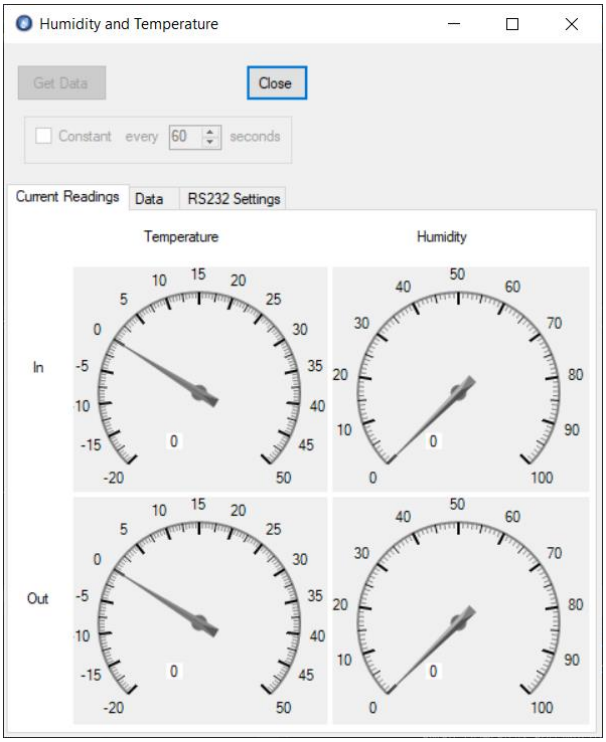
For this example, we put code onto the Arduino to display the time on a 2-line LCD. We can also use a Windows App to bring back the data from the sensors to display in Windows.

The code to run this module is show in Appendix B. The code is commented to explain some finer points of what we do. The Windows app is also available in the GitHub Folder.

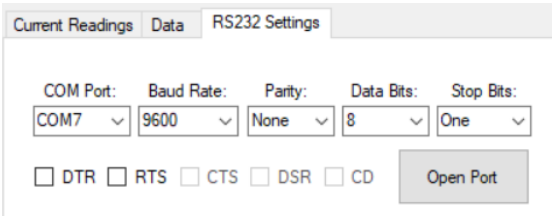
5.2. Humidity and Temperature Windows App

The aim of this app is to allow you to get the Inside and Outside humidity and temperature and bring it back to your PC/Laptop from where you can display it and/or save it to file. This is only a small app to show you how I did it. There are many ways to do it and the app can be updated to save the data or show it in graphs.

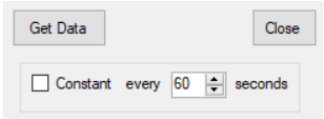
While doing this App I wanted to look at using some gauges that I had seen. I used this in the App and did modify it to how I wanted it to look. If you want the original code or to know more about it then I got it from http://www.ucancode.net/CSharp_Tutorial_GDI+_Gauge_Source_Code.htm. Thanks to the person who originally wrote it.



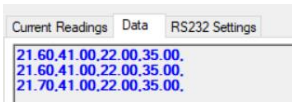
First to open the RS232 port to the Arduino. Do this in the RS232 tab



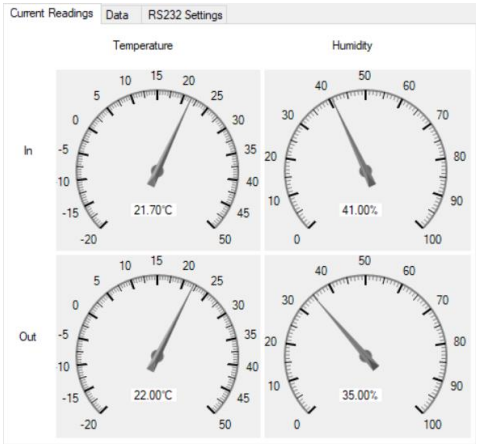
After this you will see that some of the buttons are now visible. Here you can get the data when you press the button or set it to run constantly if you tick the box and set the time in the box.



You will now see the data that is coming from the Arduino in the Data tab

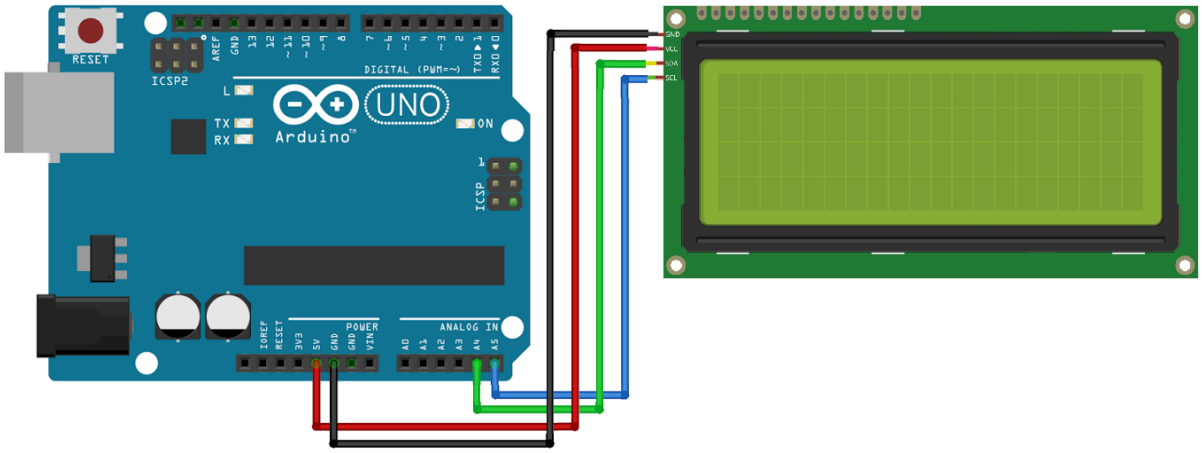


The data will also be show in the gauges and written in the box below the Gauges.



The C# .Net solutions code is available should you need to adjust it to your needs.

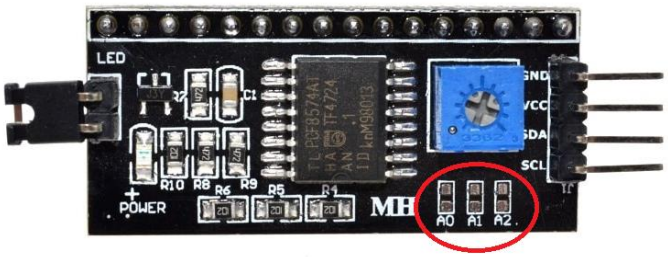
6. I²C LCD



The image is of a 4 line LCD. In my example I used a 2 line one.

6.1. I²C Address

Some I²C LCD module have an address selector solder pad.



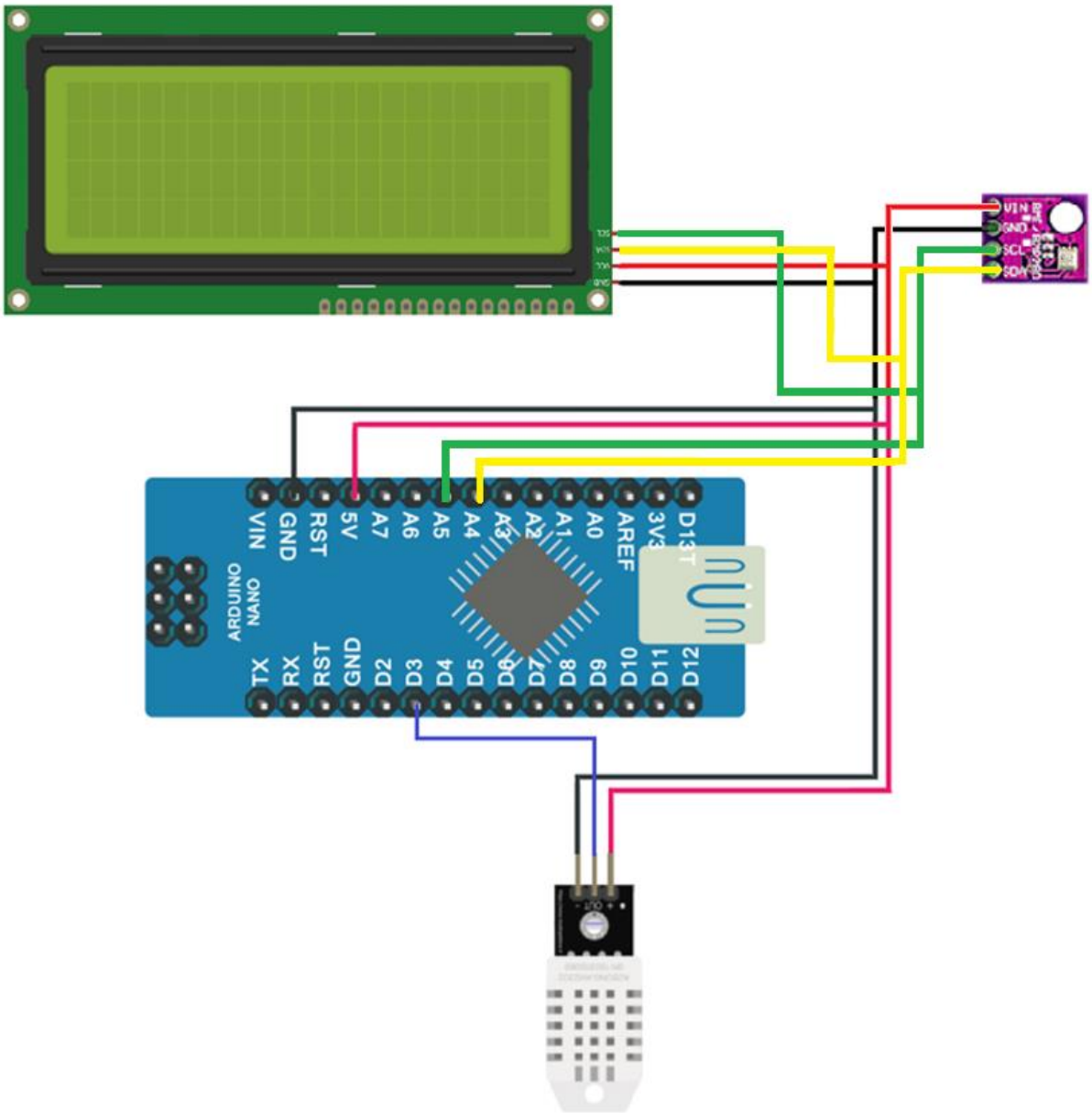
They are usually labelled with A0, A1, and A2. The following table show you how to interpret the selector. “1” = Not Connected, “0” = Connected.

A0	A1	A2	HEX Address
1	1	1	27
0	1	1	26
1	0	1	25
0	0	1	24
1	1	0	23
0	1	0	22
1	0	0	21
0	0	0	20

7. v2 – We add Pressure and simple graphics.

In this example we replace one of the DHT22 sensors with a BME280 sensor. This BME280 sensor will enable us to obtain the air pressure as well as temperature and humidity.

Warning: I found out most of the sensors which you buy as BME280 are in actual fact BMP280 sensors which do not have Humidity. Check the actual chip on the sensor PCB if it does not look Square then it is likely that it is not a BME280.



The readout has been modified to use a 4 line of 20 characters. Some basic graphics is added as show by the blue and red arrow below.

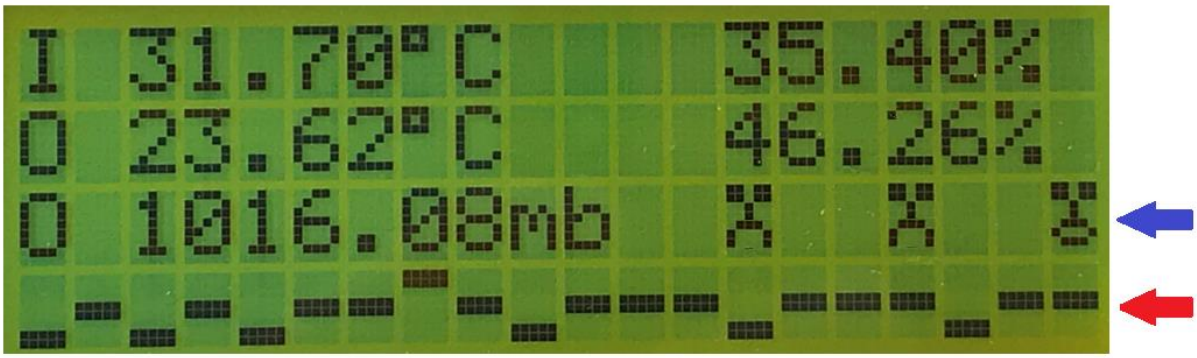
Blue Arrow = This shows 3 smileys. Happy = smiling, Ambivalent = circular mouth and Sad = dropping mouth.

- On the right we have a smiley that shows the difference in the last hour.
- Middle smiley is the difference between 10 hours ago and now.
- Left Smiley is difference between 20 Hours ago and now.

If smileys are smiling then we should have fine weather.

Red Arrow = Pressure graph. This runs for 20 hours then the oldest is replaced by the newest, so it rolls to the left. Youngest data is on right. The line is in one of three positions. Top means the pressure has risen in the last hour and bottom means it has fallen. This is at the same time one hour apart and does not take into account what happens between the hour. The middle line says the pressure has remained constant during the last hour.

- The top line prefixed by I is the internal temperature and humidity.
- The second line is the outside temperature and humidity.
- The third line on the left is the pressure. This is measured outside simply because that is the sensor with the correct chip in to read pressure. The pressure inside and outside should all be the same.



8. v3 – Add SD card Storage and Real Time clock.

In this version we will add a Real time clock and an SD card reader/writer. Once we do this, we can log the data we get and in later versions we could display it on the PC or on a Graphics screen. In this example the data is stored on the SD card in the root as a file called `Datalog.csv`.

8.1. SD card Reader

This will need interfacing using SPI mode. There are other modes but they are not open source so we cannot use them. The spec of the module says it will be Ok to use a

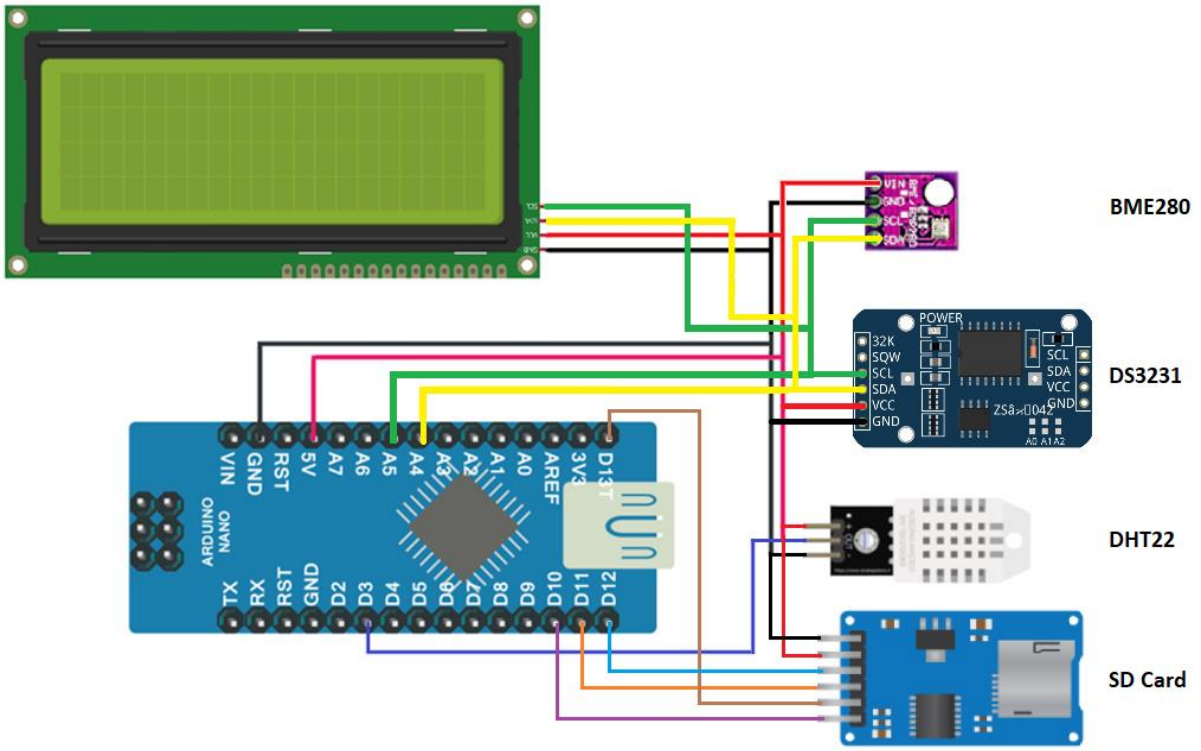
- MicroSD card up to 2GB in size. If you cannot find one that small them, you can always use a larger one but only allocate 2GB of space and format that pace only leaving the rest of the card unallocated and unformatted.
- MicroSDHC card up to 32GB in size.
- All cards should be formatted FAT16.
- All files on card should be named 8.3

8.2. Real Time Clock

I am using a module set up in a previous project. Please refer to that project @ https://github.com/zizwiz/Arduino_Projects/tree/main/RealTime%20Clock to fin dout the details.

8.3. Wiring diagram

The diagram used is shown below. Please adapt it to your needs.



8.4. Things to Note

We are using an SD Flash card. When you write to a card it copies out a block, adds the new info and then copies back the block. Each block is 512Bytes large. This means if you have lots of other items using your Ram then you may not have enough and when you compile you will get an Alert telling you the system may now be unstable and likely to have issues when it is running.

To get use less memory you can use less variables (May not be possible). Write some of your variables like strings to Flash memory rather than to RAM.

If we write the following to RAM

```
Serial.println("Card failed, or not present");
```

We can see it uses 28 bytes counting all the characters including spaces which are also characters.

If we save this same string to FLASH

```
Serial.println(F("Card failed, or not present"));
```

We now save 28 bytes more of RAM to use

In the code we use in this example we end with just over the minimum amount of memory free for writing to the SD card. Even though this is more than the 512bytes needed for writing to an SD card the system would not run very stably.

```
Sketch uses 29046 bytes (90%) of program storage space. Maximum is 32256 bytes.
Global variables use 1513 bytes (73%) of dynamic memory, leaving 535 bytes for local variables. Maximum is 2048 bytes.
```

After removing some items from the code mostly the “Smileys” and changing some others like making variables local to each function rather than Global, I end up with the following

```
Sketch uses 28368 bytes (87%) of program storage space. Maximum is 32256 bytes.
Global variables use 1393 bytes (68%) of dynamic memory, leaving 655 bytes for local variables. Maximum is 2048 bytes.
```

This figure seems to be a lot better and allows the system to run. If I make any future versions, I may need to look at the architecture and maybe rewrite the code in a different way but for now it is working.

The display on this version will not use smileys for reasons described above.

8.5. Details of SD card used

File System:	FAT16
Sectors per Cluster:	32 (16384 bytes)
Number of Fats:	2
Reserved Sectors:	1
Sectors per FAT:	243
Root Entries Limit:	512

9. Appendix A – Find I²C addresses

```
#include <Wire.h>

void setup() {
  Serial.begin (115200);

  // wait for serial port to connect
  while (!Serial)
    {
    }

  Serial.println ();
  Serial.println ("I2C scanner. Scanning ...");
  byte count = 0;

  Wire.begin();
  for (byte i = 8; i < 120; i++)
  {
    Wire.beginTransmission (i);
    if (Wire.endTransmission () == 0)
      {
      Serial.print ("Found address: ");
      Serial.print (i, DEC);
      Serial.print (" (0x");
      Serial.print (i, HEX);
      Serial.println (")");
      count++;
      delay (1);  // maybe unneeded?
      } // end of good response
  } // end of for loop
  Serial.println ("Done.");
  Serial.print ("Found ");
  Serial.print (count, DEC);
  Serial.println (" device(s).");
}  // end of setup

void loop() {}
```

11. Appendix B – v1 Arduino Code

```
#include <DHT.h>;
#include <LiquidCrystal_I2C.h>

#define DHTPIN1 2 //define the pins we will use to get the data from.
#define DHTPIN2 3

LiquidCrystal_I2C lcd(0x27,16,2);  // set the LCD address to 0x27 for a 16 chars and 2 line display

//create an array of sensors
DHT dht[] = {
  {DHTPIN1, DHT22},
  {DHTPIN2, DHT22},
};

//create arrays to put the data into
float humidity[2];
float temperature[2];

char SerialInput = '0';

void setup()
{
          lcd.init();  // initialise the class
    lcd.backlight(); //switch on the backlight

  Serial.begin(9600);  //open the RS232 port at 9600 baud
  for (auto& sensor : dht) {
    sensor.begin();
  }

//Write the non-changing data
   lcd.setCursor(0,0);
   lcd.print("I");
   lcd.setCursor(0,1);
   lcd.print("O");

}

void loop()
{


  //Read the sensor data and put into array
  for (int i = 0; i < 2; i++) {
    temperature[i] = dht[i].readTemperature();
    humidity[i] = dht[i].readHumidity();
```

```
  }

  int line = 0;
  String output = "";

// write the data to the LCD screen
  for (int i = 0; i < 2; i++) {

    lcd.setCursor(1,line);
    lcd.print(" ");
    lcd.print(temperature[i]);
    lcd.print((char)223);
    lcd.print("C");
    lcd.setCursor(10,line);
    lcd.print(humidity[i]);
    lcd.print("%");

    // construct the data to send out the RS232 if required
    output += String(temperature[i]) + "," + String(humidity[i]) + ",";


    line = 1;
    }

    //read from serial port and send to Windows App

    if (Serial)
    {
    SerialInput = Serial.read();
    if (SerialInput=='1')
    {
      Serial.println(output); //send output to the App
    }
    SerialInput = '0';
    }


  delay(5000);
}
```

12. 2004 LCD character set

https://maxpromer.github.io/LCD-Character-Creator/


Upper 4 Bits
Lower 4 Bits
0000 0001 0010 0011 0100 0101 0110 0111 1000 1001 1010 1011 1100 1101 1110 1111
xxxx0000 CG RAM (1)
xxxx0001 (2)
xxxx0010 (3)
xxxx0011 (4)
xxxx0100 (5)
xxxx0101 (6)
xxxx0110 (7)
xxxx0111 (8)
xxxx1000 (1)
xxxx1001 (2)
xxxx1010 (3)
xxxx1011 (4)
xxxx1100 (5)
xxxx1101 (6)
xxxx1110 (7)
xxxx1111 (8)