Week2 – Let's recap week 1

- Week 1
  - Learned what a microcontroller is
    - And Arduino v PIC v Raspberry Pi
  - Installed the IDE
    - Despite problems with drivers
    - MAC needs signed drivers
  - Introduced Arduino Program structure
    - header/set-up/loop
  - Wrote some simple sequencer code

Common Mistakes from the homework

- pinMode - this defaults to INPUT
  - so must be explicitly set to OUTPUT if you want to use that pin as OUTPUT
- syntax - the language is VERY PRECISE, any typing, wrong case, missing or erroneous punctuation will result in an error.
  - Always check back with example programs or the superb online documentation to make sure you have the correct syntax.
- error messages - at first there are pretty indecipherable aren't they.
  - Don't worry experienced programmers become experts at understanding what arcane error messages mean because they cause them so often!
- Failing to communicate with the Arduino
  - USB-serial is a pain - quadruply so on Windows - COM ports change or disappear
  - always check the IDE Menu/Tools/Port to check you are connecting to the Arduino
    - the port with the Arduino on should say 'Arduino' which helps
  - MAC needs signed drivers so a few hoops to jump through for clone Arduinos

We can do digital output – what else?

- We have Digital Input – using a pushbutton
  - Logically we should cover this next, but…
  - There's quite a few considerations and I don't want to do too much theory in one chunk
  - So we'll cover it later
- There's also Analog Input and Analog Output
  - Analog Input reads a voltage
  - Analog Output uses Pulse Width Modulation to simulate an Analog Voltage
  - We'll cover this next

Before we think about Analog I/O

- Analog values won't be integer so let's look at the Arduino's numeric types:
  - int – an integer in the range -32,768 to 32,767
  - unsigned int – an integer in the range 0 to 65,535
  - long – an integer in the range -2,147,483,648 to 2,147,483,647
  - unsigned long – an integer in the range 0 to 4,294,967,295
  - float – a decimal number with max 3.4028235E+38 and as low as -3.4028235E+38
    - But with only 6 or 7digits of precision
  - You may also see short (same as int on an Arduino) and long (same as float on an Arduino)
- Some considerations
  - long variables take up twice as much space as int variables
    - But may be needed for big numbers (such as the milliseconds clock)
  - Floating point maths is slow compared to integer maths
  - Integer maths always yields integer results
    - e.g. 3/2 = 1 (not 1.5)
  - To get decimal results
    - use a decimal somewhere in the equation - e.g 3/2.0 = 1.5
    - Or 'cast' an int to a long for the equation – e.g. int mvar = 2; Serial.print(3/(float)myvar));
    - Casting temporarily changes the variable type
    - so can also cast a float to an int (which rounds it down to the lower int)

Reading input – how do we see it?

- One problem with microcontroller programming is 'seeing' what is going on
- The Arduino has a Serial monitor that allows viewing of program variables
  - Initialise it with Serial.begin(9600); in setup()
    - 9600 is the baud rate which can be changed if really needed
  - Then, at anytime, use Serial.print() or Serial.println() to write to it
  - Serial.println() will add a line return after the data, Serial.print() won't
    - So if you want to build up a line of several values then use Serial.print() and finish with a Serial.println()
    - For example, if you have a variable myvar (current value 99) then you could display it with
      - `Serial.print(“Myvar variable is currently = “);`
      - `Serial.println(myvar);`
    - This would print the following to the Serial monitor and then start a new line
      - Myvar variable is currently = 99
- To see what is written to the Serial monitor
  - Select Serial Monitor from the Tools section of the IDE Menu
  - It will automatically connect and start displaying

At last, reading an analog voltage

- The Uno and Leonardo have a 6 channel 10bit ADC
  - i.e. 6 pins (A0-A5) will provide analog to digital conversion and the result will be an integer in the range 0-1023
  - By default 0-1023 maps to the range 0-5v
    - The range can be changed (see analogReference)
    - But this is outside the scope of this course
  - To read a pin we use
    - analogRead(pin)
    - This returns an int in the range 0-1023
  - To convert this to a voltage we just multiply this by 5/1023
    - But since we will get a float voltage we should use 5.0/1023.0

analogRead – to convert to voltage or not?

- analogRead gives an int value in the range 0-1023
- This can easily be converted to a float:

```
void setup() {
    Serial.begin(9600);
}
void loop() {
  int sensorValue = analogRead(A0); //read analog input on A0
  float voltage = sensorValue * (5.0 / 1023.0); //convert to real voltage
  Serial.println(voltage);  //and print it out
}
```

Do we always need to do this – look at this code:

```
int fwd_voltage = analogRead(A0);
int ref_voltage = analogRead(A1);
float vswr = (float)(fwd_voltage+ref_voltage)
             /(float)(fwd_voltage-ref_voltage)
```

- Here we can just use the raw uncalibrated values
  - The actual voltages are of little interest
  - We can just as well do the arithmetic with the digital values.

Analog output – well PWM anyway

- The Arduino is a digital device and doesn't have a DAC to produce real analog voltages
- However it can use Pulse Width Modulation (PWM) to simulate an analog voltage
  - This is good enough to dim lights or LEDs and control motor speeds
- Only certain output pins support PWM
  - On the UNO and Leonardo these are:
    - pins 3, 5, 6, 9, 10, and 11
    - Look for a ~ symbol on the board next to the pin
  - Normally the PWM frequency is 490Hz
  - 980Hz for pins 5 & 6 (and pins 3 & 11 on a Leonardo)
  - Pins 5 & 6 may not always behave well close to 0 PWM
    - So not a good choice if you want a hard off

Using analogWrite

- analogWrite(digiPin, value);
  - Where value is an int in the range 0-255
  - e.g.
    - analogWrite(9,255); turns pin 9 on 100%
    - analogWrite(9,0); turns it full off
    - analogWrite(9,128) is 50% duty cycle
  - But make sure the pin you've picked supports PWM!

- How about having a pot controlling an LED brightness?
  - Can use analogRead to get the pot voltage
  - And analogWrite to dim the LED
  - But analogRead gives a value 0 – 1023 and analogWrite wants a value 0-255
  - So we could the analogRead value divided by 4 (what I would do)
  - But there is also the map function
    - This is used by the Example so let's cover it next

Map Function

- The Arduino provides a handy-dandy function to convert from one range to another – map
- Easy to use
  - You need the val, the start & end of the original range and the start & end of the desired range
  - So to map an analogRead value (0-1023) to an analogWrite value (0-255) you would use:
    - int newval = map (val, 0, 1023, 0,255);
    - Dividing by 4 also works :-)
      - but map is more flexible for other cases
  - If you wanted to invert an analogRead value
    - i.e. you want 0 if you read 1023 and 1023 if you read 0
      - int invval = map(val,0,1023,1023,0);

OK – let's try this – test out Examples/03.Analog/AnalogInOutSerial

Expanding the Arduino

- This is all good stuff but not terribly exciting
- The real power of the Arduino comes from all of the other devices it can interact with and control
- These modules or shields cover a huge range of function and are generally surprisingly simple to use
- The secret is to use devices that already have a library written for them
  - All the code required to drive them will be written and all you need do is simple function calls
  - We've already seen this in part with Serial
    - to set up a Serial monitor you just needed to call the Serial.begin() function
    - And to write to it you used the Serial.println() function
    - Without those functions you would have had to establish handshaking and managed the flow of 0s and 1s yourself – difficult and tedious!
    - The functions for Serial are included as standard but for other libraries we will need to add them to our program

Let's add an LCD but first we need to talk about libraries in a bit more detail

Arduino Libraries

- The Arduino IDE already comes with a wide range of libraries
  - Take a look at Sketch/Include Library
  - Good library writes also include good Examples
    - Take a look at Examples/LiquidCrystal/HelloWorld
  - It also has a good library management system
    - You can add new libraries
    - It will tell you if updates are available for libraries
- To add a library to your program you need a #include statement
  - Remember last week we learned that all variables etc. must be defined before they can by used
    - The #include header file contains all such definitions
  - No need to code this just click on the library from Sketch/Include Library
    - And the IDE will add it automatically for you
- New libraries can be found
  - In the library manager – Sketch/Include Library/Manage Libraries
  - On the playground section of the Arduino website
  - Or just by searching the web
  - The are often several libraries supporting the same function – spend some time finding the best

Check Examples/LiquidCrystal/HelloWorld and note the #include <LiquidCrystal.h> statement

Adding a new library

- There are 3 ways to add a library:
  - If the library is in the library manager
    - Simply click on the more info link and then the Install button
  - If the library is found on the playground or web and is a zip file
    - Simply use Sketch/Include Library/Add .ZIP library
  - If the library isn't a zip file or won't install using the Add .ZIP library
    - Firstly consider is this the best library for the job
    - The Arduino saves all your programs to a folder it refers to as a sketchbook (it calls programs 'sketches')
    - Find the location of your sketchbook using File/Preferences/Settings
    - Navigate to your sketchbook folder
    - If you previously installed libraries you will have a 'libraries' subfolder
      - If not you will need to create an empty one
    - Now copy or unzip your library here – it should be in its own subfolder

We're almost ready to use the LCD

- If we want to attach a standard (HD44780) LCD display we can just use the standard LiquidCrystal library
- Our LCD uses a two-wire i2c adapter
  - Means we need 2 pins rather than 7
  - But does mean we need to install a special library module
  - We will use the improved New LiquidCrystal library
    - From https://bitbucket.org/fmalpartida/new-liquidcrystal/wiki/Home
    - This supports both i2c LCDs and standard HD44780 ones
    - We can install it using Add .ZIP library

So let's install the New Liquid Crystal library and look at its examples

Using an i2c LCD module

- Problem: they come in 3 variants
  - Fortunately easily identified here
    - https://arduino-info.wikispaces.com/LCD-Blue-I2C
  - Ours Is version 1
    - YwRobot Arduino LCM1602 IIC V1
- So we need to set up the interface using
  - LiquidCrystal_I2C lcd(0x27, 2, 1, 0, 4, 5, 6, 7, 3, POSITIVE);
- So we just need to connect GND, VCC, SDA & SCL
  - SDA & SCL are the I2C pins
  - These have pre-defined pin mappings on Arduinos
  - On an Uno (or Uno clone)
    - SDA = pin A4 and SCL = pin A5
  - On a Leonardo
    - SDA = pin 2 and SCL = pin 3 (note these are digital not analog pins)

G0UKB's i2c LCD “Hello World”

```
// include the i2c and LiquidCrystal Libraries
#include <Wire.h>
#include <LiquidCrystal_I2C.h>

// Set up the I2C address and I2C-HD44780 pin mapping
LiquidCrystal_I2C lcd(0x27, 2, 1, 0, 4, 5, 6, 7, 3, POSITIVE);

void setup() {
lcd.begin(16,2);                  // send initialisation sequence to the LCD
lcd.print("Hello from G0UKB");  // and write a message
}

void loop() {
  // We'll just loop around doing nothing – the message will remain
}
```

Try it – then make the obvious change to see your callsign in lights!

It's pretty much the same for all modules

- Firstly find and install the library
  - Remember to #include it
- Then set up pin mapping
  - We know what pins map to the module
  - But we need to define this in the code
  - This does not do anything with the module
    - It just sets up some definitions for our code
- Then in setup() we initialise the module
- Finally, we just use the module by calling its library functions

- Note in the previous example we didn't need to define pins for SDA and SCL
  - They are fixed pins

LCDs – what else can we do?

- There are lots of other functions was well as the simple print.
- The most useful ones are
  - clear() - clears the LCD
  - home() - move the cursor to the 1st char, 1st line
  - setCursor(col,row) moves the cursor
  - scrollLeft/scrollRight() scrolls the display 1 character
  - cursor/noCursor() – shows/hides the cursor
  - display/noDisplay() – shows or hides text on the display
    - clear() clears the display and loses all the text
    - noDisplay() just hides the text and display() shows it again
  - CreateChar() creates bespoke characters to display
  - And others ...

- How do you find all the available functions for any module?
  - Read the documentation and look at the examples

Displaying variables on an LCD

- Just do it!
- e.g.

```
int digivolts = analogRead(A0); //get volts as 0-1023
float volts = digivolts*(5.0/1023.0); //convert to 0-5.0v
LCD.setCursor(0,1);        // go to start of 2nd line of LCD
LCD.print("Voltage = ");   // and write out …
LCD.print(volts);           // the analog voltage
```

- Easy to change from Serial.print to LCD
- Remember in programming numbers start at zero
  - So set setCursor(0,0) is the first char of the first line
  - SetCursor(7,1) is the 8th character of the 2nd line etc.
- Print (whether LCD.print or Serial.print) prints floats with 2 decimal places
  - To change this add a 2nd parameter specifying the number of decimal places
  - e.g LCD.print(volts,4); would print 5.0000

Hold on – we can almost build our voltage protector!!!

- We know how to read voltage and how to display it, how to turn LEDs on and off
- Some more programming syntax – if statements

```
if (volts>4.0) {
    Some lines of code
    ...
} else {
   Some other lines of code
   ...
}
```

- The else clause is optional
- Note – there is no ; after a }
- It makes code easier to read if you indent the conditional lines of code
- NOTE testing for equality ALWAYS use '=='
  - e.g. if (digivolts==0) { …
  - If you use single '=' you will probably get a hard to spot bug – DO NOT DO IT!!!!!

One very last piece of programming

- Boolean variables
  - Can have just one of two values – true or false
  - Are defined using the keyword boolean
  - e.g. boolean max_exceeded = false;
  - Are often set to indicate some condition has been met

```
if (volts>4.0) {
    max_exceeded = true;
}
```

  - Can be tested

```
if (max_exceeded) {
    Serial.println(“At some stage we went over-voltage”);
}
```

  - The not case can be tested too (use a '!')

```
if (!max_exceeded) {
    Serial.println(“Never been over-voltage”);
}
```

  - To toggle the value (true → false) or (false → true) we could use if statements but it's simpler to use not

```
max_exceeded=!max_exceeded;
```

You knew it was coming - Homework

- Build a voltage protector
  - It should have a startup 'Welcome' screen that displays for 5 seconds
    - Just to appear friendly
  - It should display the digital reading on the 1st line of the LCD
  - It should display the real analog voltage on the 2nd line
  - It should light an amber LED if the voltage goes above 4v and a red one above 4.5v
  - It should light an amber LED if the voltage falls below 2v and a red one if the voltage falls below 1.5v
  - It should light a green LED if the voltage is in the range 2-4v
- If you want to make it more challenging
  - Option 1: red LEDs should flash
  - Option 2: if the voltage EVER falls out of range a red LED should light and stay on even if the voltage is adjusted back in range