**Resources:**

Lecture and Lab class notes, PC with internet connection, teensy microcontroller, prototyping plug board, plug board connection wires.

**By the end of this lab you should be:**

* Output string and char data to a serial port monitor using sprintf().
* Use formatting codes to print Hex, Binary and decimal values.
* Read analog voltages from analog input pins.
* Scale raw input data using integer and floating point techniques.

**Background:**

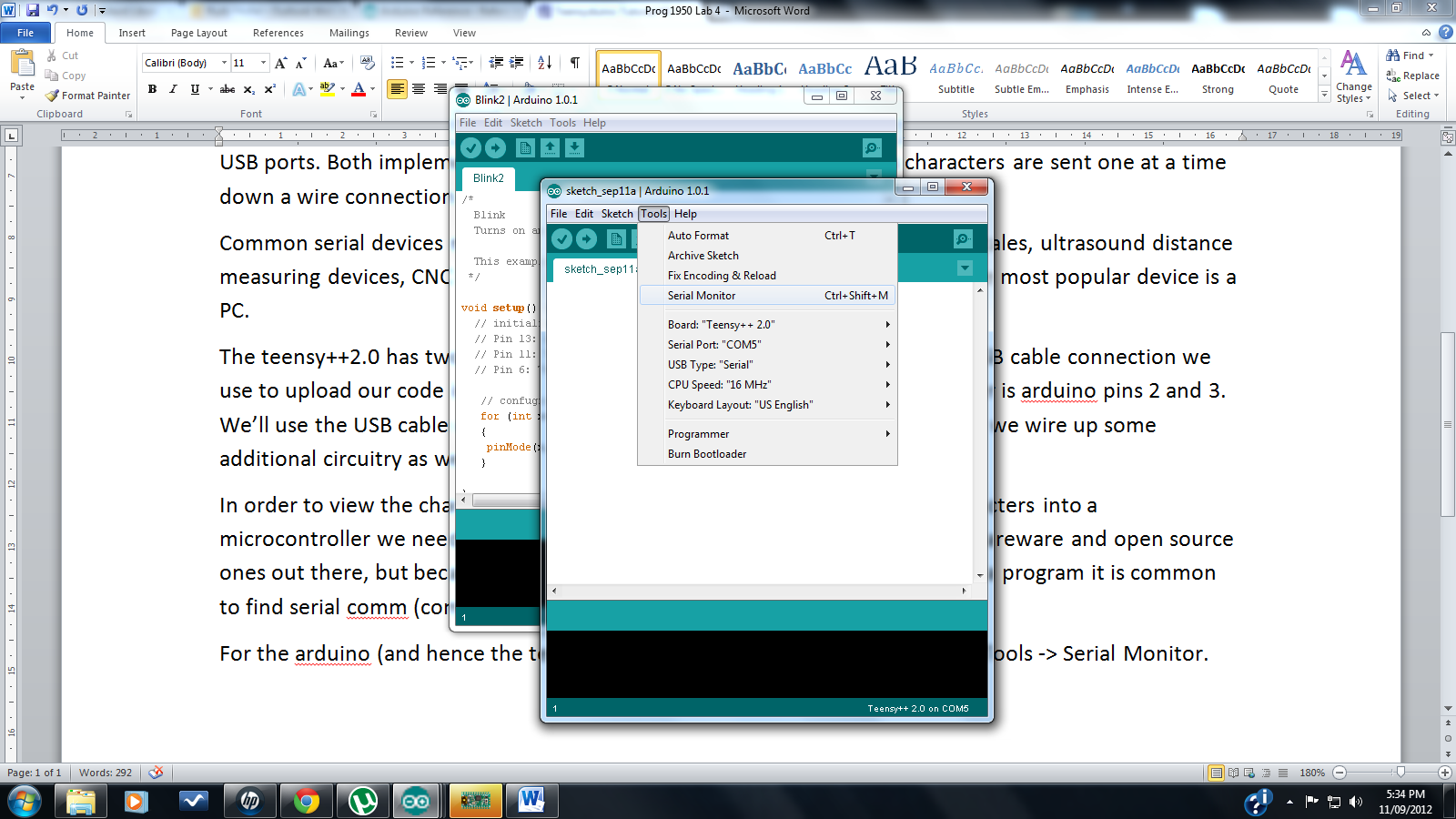
Almost every microcontroller you will encounter will have a serial port output. Older microcontrollers adhere to the RS-232 standard. (Recommended Standard no. 232) and recent microcontrollers use USB ports. Both implementations send or receive data serially. That is characters are sent one at a time down a wire connection between microcontroller and other device.

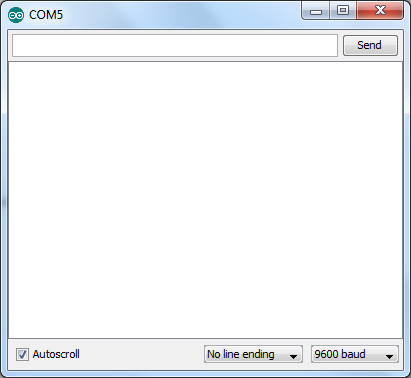
Some common serial devices microcontrollers connect to would be GPS’s, weigh scales, ultrasound distance measuring devices, CNC machines, bar code scanners, LCD screens, and probably the most popular device is a PC.

The teensy3.1 has four serial connections to the outside world. As well as the USB cable connection that can be usee to upload our code or as serial port.

Sometimes the acronym USART is used to denote circuitry that can transmit serial information synchronously or asynchronously (Universal Synchronous Asynchronous Receiver transmitter). Since the serial ports usually connect using the RS232 communications standard, they are sometimes called rs232 ports. However, serial port is more correct as the RS232 standard describes an electrical interface that uses +/- 12V signals – our teensy only outputs 3.3V.

We’ved used the USB cable connection for now as using other pins requires that we wire up some additional circuitry (as well as needing an RS232 port on our computer).

In order to view the characters output from a microcontroller or to input characters into a microcontroller we need a serial communications program. There are many shareware and open source ones out there, but because it is so useful to be able to print while developing a program it is common to find serial comm (communications) windows implemented in the IDE. For the arduino (and hence the teensy) we access the comm window through Tools -> Serial Monitor. (or simpler still by clicking on the magnifying glass on the right top corner of the IDE)

And if you were to press that now, you’d open up the comm window.

Serial output from the teensy will show up here.

Serial input to the teensy is entered here.

**What to do in the Lab:**

Got to pjrc.com and go to Teensy -> teensyduino -> tutorial -> USBserial. (or click on this link: <http://www.pjrc.com/teensy/td_serial.html>). Review the section on USB Serial communication paying close attention to the Serial object methods Serial.begin() and Serial.println() and the example program.

Don’t pay a lot of attention to the Teensy USB Serial Extensions. We won’t be implementing any of these extensions just yet.

You already know that to print, the Serial object along with its print or println method is used:

Serial.println("Hello World...");

The println function is written as an object oriented C++ function where Serial is known as an object and println as the method that the object can use. When you want to use the println() method it’s almost always attached to the Serial object. (If not, then you’d see it attached to the Uart object, in which case println would print a line using the Uart and TX and RX pins on the teensy).

print() and println() as described in the pjrc web site, and can print either a string eg:

Serial.println("Hello World...");

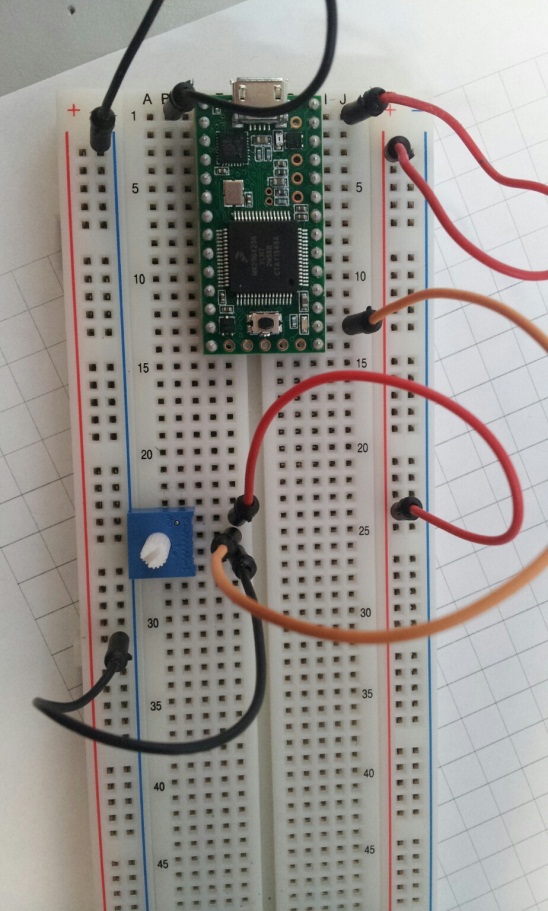
or a number, eg:

int num1 = 1234;

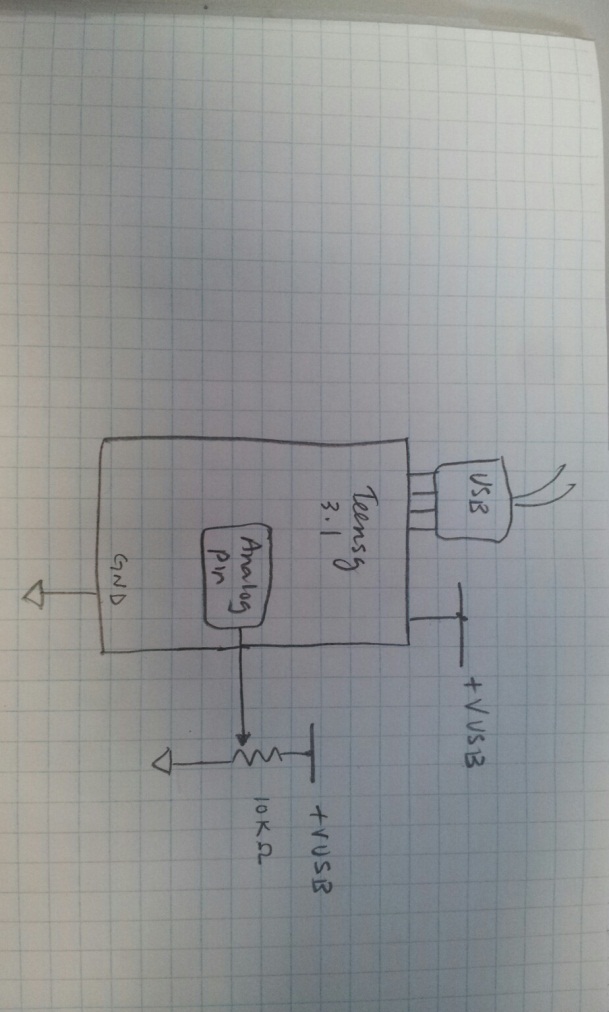
float num2 = 3.1415926;

Serial.println(1234);

Serial.println(num2);

**Task 1**

Wire a 10k trim-pot in your parts kit to one of the analog input pins as shown in the schematic and photo below.



Goto the arduino reference section on analogRead() and read the information provided for this function. We will be using the analogRead() to read in the voltage present at one of the analog inputs to the teensy. Note that analogRead() returns a 10 bit int value from 0 – 1023 corresponding to 0V – 3.3V.

Write code that will read the value of the voltage input to the arduino pin. Use the println() method and the Serial object to display the value measured every 500ms.

Call me over when this task is complete so I can check you off and as always, print off this program and place in your binder afterwards and e-mail a copy of the code to prog1950lab@hotmail.com.

**Task 2**

As is often the case, the nature or appearance of our data is not always scaled to the units needed. For example we might need the voltage read to be a number between 0 and 100% or we might need the number to reflect a value from 0 to 360 degrees, or in the case of our trim pot from 20 to 340 degrees. It could be that the pot will be used to set a thermostat from -5.0 to 30.0 degree C.

Scaling for the first case described (converting 0 – 1023 to a value from 0 – 100) is fairly simple. The formula we’d use would look like:

= the raw unprocessed digital data the varies from 0 - 1023

= a value that varies from 0 to 100

= the scaling factor

Rearranging the formula and solving for x yields a scaling factor of = 0.097751711.

And substituting 100 for P (the maximum of the new scale) and 1023 for R (the maximum for the raw data) yields,

This means that our code will need a float variable to hold the scaling factor of 0.097751711. It also introduces the need to typecast as you can’t multiply unlike data types. That is you can multiply an int by an int and a float x float is ok, but int x float number is not allowed. Makes sense if you think about, an int variable holds 2 bytes, a float holds 4 bytes. A float has an entirely different structure than an int – it has a mantissa and an exponent component, an int is a signed 16 bit number.

For this code you’ll want to use typecasting to temporarily convert datatypes to other datatypes.

**What’s typecasting?**

From: <http://www.cprogramming.com/tutorial/c/lesson11.html>

“Typecasting is a way to make a variable of one type, such as an int, act like another type, such as a char, for one single operation. To typecast something, simply put the type of variable you want the actual variable to act as inside parentheses in front of the actual variable. (char)a will make 'a' function as a char.”

So our code that scales 0 – 1023 to 0 – 100 would have somewhere in its lines something like the following:

.

.

.

int trimPotVoltage;

float trimPotVoltage0\_100;

const float x = 0.097751711;

.

.

.

Void loop(){

trimPotVoltage = analogRead(trimPotPin);// 0 - 1024

trimPotVoltage0\_100 = x\*(float)trimPotVoltage; //scale 0-100

Serial.println(trimPotVoltage0\_100);

.

.

.

**What to code for task 2**

Write code that scales the raw data from the voltage input to the arduino to values that vary from 0 to 100. Use the println() method to print values to the comm. monitor every 500ms.

Call me over when this task is complete so I can check you off and as always, print off this program and place in your binder afterwards and e-mail a copy of the code to prog1950lab@hotmail.com.

**Task 3**

You’ll notice that task 2 displays the percentage of rotation with two decimal places. That’s the default setting for the println() and print() methods. You may remember reading that on the PJRC web site earlier. There is another print function available in C called sprintf(). sprintf() stands for **s**tring **print** with **f**ormating hence its name. It is not described in the arduino reference section however it is a common C stdio library function and googling sprintf() will result in many good references.

sprintf() allows the printing of text strings and the inclusion of numeric data within a string and the numeric data can be formatted to display in a number of ways.

Eg:

.

.

.

int trimPotVoltage;

char strBuffer[];

.

.

Void loop(){

trimPotVoltage = analogRead(trimPotPin);// 0 - 1024

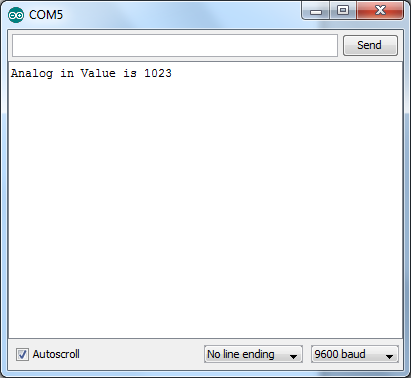
sprintf(strBuffer,“Analog in value is: %d”,trimPotVoltage);

Serial.println(strBuffer);

.

.

.

 Would print the line:

The numeric variable to print in the % position .

Let’s dissect the line of code:

sprintf(strBuffer,“Analog in value is: %d”,trimPotVoltage);

A text string

a char buffer that you will be “printing” to

A format “tag” or “code” for a number. The % tells the function that the number that follows should be printed here and the d describes how the number should look.

Name of the function:  
**s**tring **print** with **f**ormatted output

More than one variable can be printed, just use more than one %sign and format code. Eg:

.

.

int rT = 20;

int cT = 30;

int oT = 22;

int strBuffer[];

.

.

Void loop(){

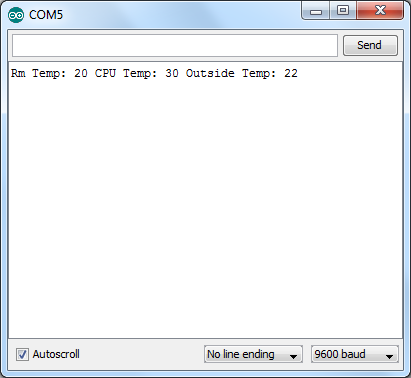
trimPotVoltage = analogRead(trimPotPin);// 0 - 1024

sprintf(strBuffer,“Rm Temp: %d CPU Temp: %d Outside T %d”,rT,cT,oT);

Serial.println(strBuffer);

.

.

 Would result in:

First variable is placed at the first instance of % .

Second variable is placed at the second instance of %.

Third variable is placed at the third instance of % .

sprintf(strBuffer,“Rm Temp: %d CPU Temp: %d Outside T %d”,rT,cT,oT);

d code tells sprintf() to format the numeric value as a decimal integer.

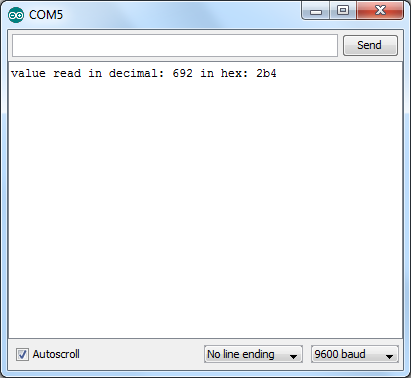
Google sprintf() and you will find reference information describing the various format codes. You should notice that there are optional flag, width, precision and modifier parameters you can specify before the final format type. For now focus on the final letter used to describe format type.

**Note:**

After experimentation I noticed that the sprintf() function implemented by the arduino supports only the d and x type, i.e: %d and %x work for displaying integers as decimals or in lower case hex. Sadly, Floating point number formats are not supported.

**Code to write for task 3**

Write code that uses the sprint() function and prints the raw data read from the analog voltage input to the comm window as shown.

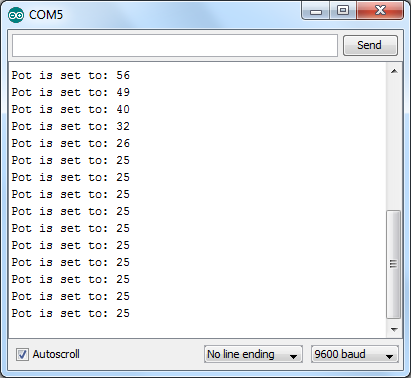


Call me over when this task is complete so I can check you off and as always, print off this program and place in your binder afterwards and e-mail a copy of the code to prog1950lab@hotmail.com.

**Task 4**

As noted above, floating point numbers are not supported by the arduino. There are however workarounds. Say we wanted to display our output from task 4 so that only the whole number is displayed all we need do it typecast our float variable to an int.

Modify task 2 code to display the following:



Use the sprintf() function with a %d format tag and typecast the float variable to an int

Call me over when this task is complete so I can check you off and as always, print off this program and place in your binder afterwards and e-mail a copy of the code to prog1950lab@hotmail.com.

**Task 5**

Say we needed to display the percent setting with one decimal point of precision. Ie: from 0.0 to 100.0. Normally we’d use the floating point type and width and .precision and modifiers that are part of sprintf(). But we can’t because the arduino sprintf() doesn’t support floating point numbers.

We could however: (Thank you Ig Kolenko for this technique and code.)

1 . Take floating point value and multiply by 10.0 … ie: 55.913 \* 10.0 = 559.13

2. Convert this to an integer i.e: 559

3. Take the whole number 559 and divide by 10 i.e: (in integer math) = 55

4. And lastly, perform a modulo (remainder) division by 10 i.e: 559 % 10 (in integer math) = 9

So basic code would be:

float f = 55.913;

int val;

val = (int)(f \* 10.0); // typecast (technically automatic, but nice to emphasize it)

sprintf (buffer, “value is %d.%d\n”, val / 10, val % 10);

if you want 2 digits, change the constant 10 above to 100, and so forth … in fact, the generalization is:

float f = 55.913;

int val;

int precision = 100;

val = (int)(f \* (float)precision); // typecast (technically automatic, but nice to emphasize it)

sprintf (buffer, “value is %d.%d\n”, val / precision, val % precision);

**Code to write for Task 5**

Display the voltage input scaled between 0.0 and 100.0.

**Task 6**

Write code that will display the 0-5V input voltage from the pot to 0.00 to 5.00 and display as 0.00V to 5.00V

**Bonus Task**

Write code that will scale the 0-5V input voltage from the pot from -5.0 to 30.0 and display as -5.0°C to 30.0°C.