EE 299 Lab 1
Introducing the Lab Environment

University of Washington - Department of Electrical Engineering
James K. Peckol

Introduction:

This first part of the lab has three main purposes. The first is to introduce the C language, the Visual Studio development environment, and to explore some of the basic aspects of the language. The second is to introduce the Arduino UNO microprocessor board (gees, what kind of name is that ….do I have to learn to pronounce it – yep, do I have to find out what it means – yep) based upon the Atmel Atmega 328 P microcomputer and the associated Arduino development environment.

The third is to work with the basic Blink program – blinking an LED is fundamental to electrical engineering, make some modifications to it, then interface the microprocessor to two peripheral devices, an LED and a pushbutton, and finally learned how to send output from a simple program to a display device, the Serial Monitor.

In the second part of the lab, we will work with another display device, the LCD – liquid crystal display and learn how send information from one microprocessor, over a very simple network, to a second to control a peripheral device attached to the second microprocessor.

Like a typical embedded system, we have a hardware piece – the Arduino, and a software piece – the software. This quarter, our major focus will be on the software piece; but we need something to run the software on…so, enter the UNO.

You are strongly encouraged to read through the entire project before trying to start designing or exploring. The best way to become familiar with any new piece of hardware is to take a tour of its features; the same holds true for learning a new piece of software or programming language. This is the approach that we will take.

We will begin to learn the C language by starting with a working C program on a conventional PC. We will make several modifications to the program then compile and run our designs on the PC.

As we work with the C language, the Arduino, and its associated language, we'll see that the two languages are very similar, yet, there will be some differences. We will try to point these out. However, our main focus will be on the C language.

As we learn to work with the UNO, we'll begin with the documentation and the data sheet to discover some of the key features of this device. This is exactly how we will do it when we start to work with a new microprocessor in future classes, in our graduate research, or for our company. Then, once again, we will begin with a working application, learn how to build a project, compile it, download it, and run it on the target platform. We will then make some simple modifications to the program and verify that the modified programs work as expected….then, on the second day…

Prerequisites:

No real prerequisites for this class except imagination and a willingness to learn and explore. Oh, no beer until the project is completed. I really mean it.

Background

For us, this is the second time that we've offered the class and the second time that we have worked with the Arduino processors in a class. In preparing for the class, we've learned some things about the tools and the environment. The learning process continues and much remains, however.

At this stage in learning engineering and about the engineering process, playing with the toys often seems to be the most exciting part…the documentation, formal design, and so on, is, well, often seen as rather boring. *Why do I have to go through all this…why can't I just go to the Internet and find something like the design and make a few modifications and be done?*

In the real-world, the documentation and formal design are absolutely critical parts of all phases of the engineering development process. Doing it, doing it right, and doing it well can mean the difference between success or failure of a project, the malfunction of the system following delivery to the customer, or the possible loss of life while it is being used.

It's also very important to recognize and to remember that the answers to most interesting real-world engineering problems originate in our brains, discovered as we use our imaginations and knowledge to creatively apply the underlying theory; they are reached through our persistent hard work and diligence. Solving the most interesting and challenging problems is never easy.

The solutions to challenging problems are not sitting, ready, and waiting for us on the Internet. The Vikings (yep, the Vikings, not that other dude with the good marketing department) didn't discover North America by searching some ancient version of the Internet…they took risks…they explored, they challenged and they worked hard. We didn't find the solution for making the first successful airplane flight, to putting someone on the moon, to making the first soft landing on Mars, or to developing and programming the first microprocessors, or the discovery of the Higgs boson on the Internet…we won't find the answers to many of today's problems or the labs this quarter there either.

We challenge you to explore, to think, and to make those discoveries. The Internet is a very helpful tool – use it…but use it properly. It not the only tool and it should not necessarily be the first tool of choice…that privilege should be assigned to our own meat computer.

Sometimes your instructor or TAs will have the answers and sometimes we won't. As we said in the opening, bear in mind that we're (continuing to work) learning the environment and tools too. This platform and development environment are new to all of us (but we're learning). If we all work together, to identify and solve problems as they occur, everyone benefits and we help to build for the next classes. The answers are there somewhere. Let's all work to find them.

Regardless of the specific platform/environment used, embedded systems development requires at least the following:

1. An understanding of the problem that we are trying to solve.
2. A target platform on which to develop the application.
3. A mechanism for programming the target platform.

Target platform is a term used loosely in industry to mean the actual piece of hardware that performs the computation and/or control – the place where our application (the software) will ultimately reside and run.

A target platform can be a simple a single chip (like the UNO), or as complicated as a feature-rich single board computer (build around a multicore high performance processor and possibly several programmable logic devices – like your cell phone or tablet). Despite the differences in physical characteristics, the target platform's purpose is the same: to execute the software written by the developer. Here, and for the typical microprocessor based application, the software will be written in the C language or a simple variant.

The term *to program* here has two meanings. The first is the more traditional embedded sense and simply means writing software to control a given piece of hardware (in this case the target platform and any peripheral devices that may be connected to it – see above). In the embedded world, *programming* the target also means storing the executable code (we call firmware) into non-volatile memory on the target. As the characteristics of the target platforms can vary greatly, so too can the mechanism used to program the target platform.

Some development environments allow one to develop directly on the target platform (much like developing code on a PC), while others require that code be developed on another *host computer* and then transferred to the target platform. We call this transfer *downloading to the target platform*. Apart from the actual technique(s) used, every embedded system has to provide a mechanism for programming it.

Part 1 – The Laboratory Environment

As we've noted, the laboratory and development environment will be a combination of the traditional PC, Visual Studio, the Arduino UNO microcontroller and peripheral devices, and the Arduino integrated development environment (IDE). The Arduino system is a feature-rich single board computer developed around an Atmel Atmega 328 P microcomputer. We also have a collection of peripheral devices that we will work with over the course of the quarter.

We have run a series of tests with the new environment and believe that all is functioning as it should be. Yet, it is still easy to miss some things. Please let us know if you encounter something that is not working (other than possibly your code).

Working with the Development Environment and Target Platform

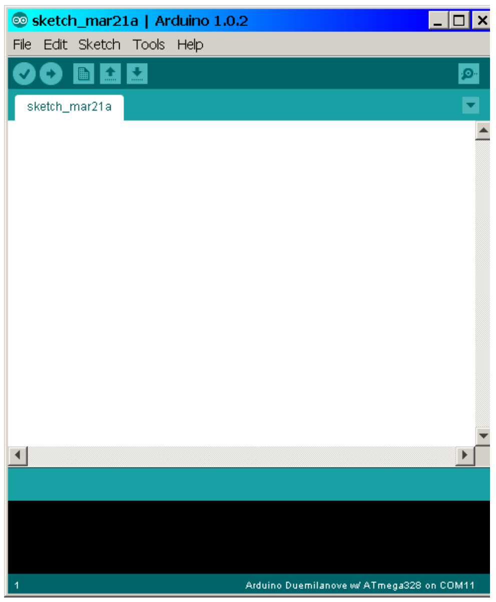
We'll start this project with the Arduino UNO microcontroller, the C language, and Arduino's dialect of C …it's actually a bit like Java.

Note: Every member of your team should do each of these exercises….not just watch. Your team can submit one final report.

First Steps

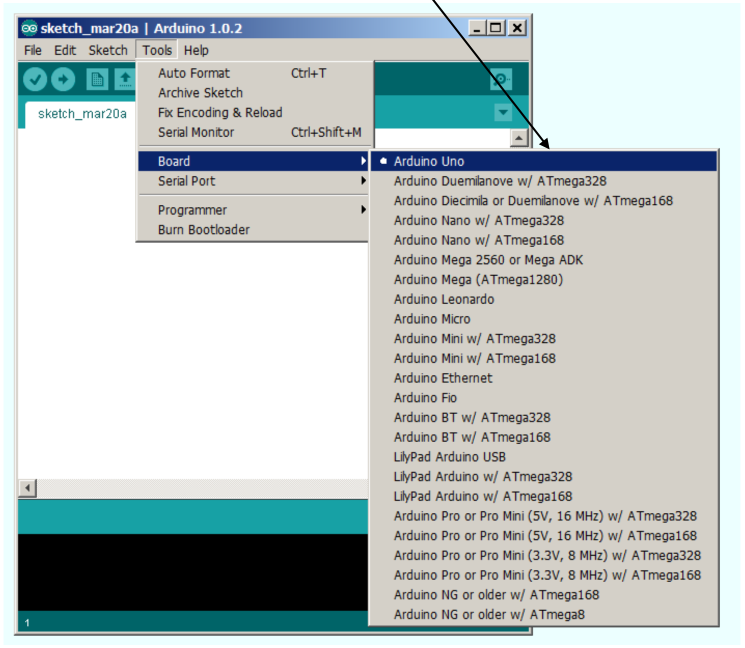
Our first step is to connect the UNO to the PC. Now is as good of a time as any to do that.

For the next step, find the Arduino IDE amongst the installed programs on your PC. Select and open it. You should now see the IDE workspace.


sketch_mar21a | Arduino 1.0.2
File Edit Sketch Tools Help
sketch_mar21a
1
Arduino Duemilanove w/ ATmega328 on COM11

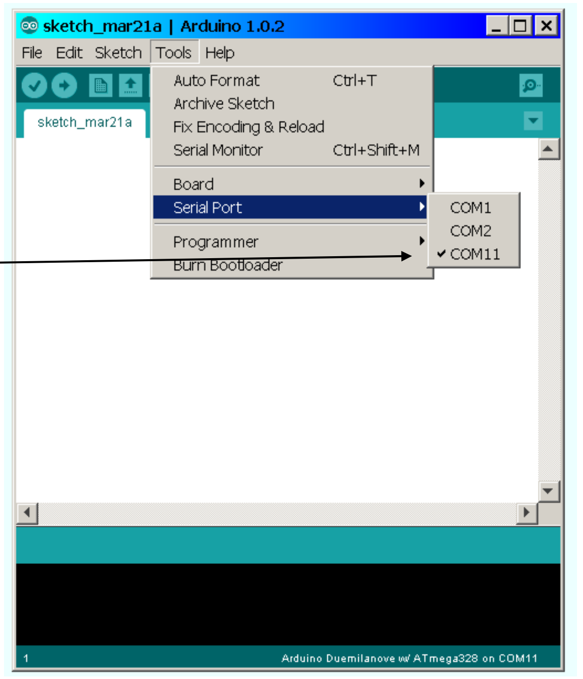
Now, select

- Tools
    - o Board
        - ▪ Arduino UNO w/Atmega328


sketch_mar20a | Arduino 1.0.2
File Edit Sketch Tools Help
sketch_mar20a
Auto Format Ctrl+T
Archive Sketch
Fix Encoding & Reload
Serial Monitor Ctrl+Shift+M
Board
Serial Port
Programmer
Burn Bootloader
Arduino Uno
Arduino Duemilanove w/ ATmega328
Arduino Diecimila or Duemilanove w/ ATmega168
Arduino Nano w/ ATmega328
Arduino Nano w/ ATmega168
Arduino Mega 2560 or Mega ADK
Arduino Mega (ATmega1280)
Arduino Leonardo
Arduino Micro
Arduino Mini w/ ATmega328
Arduino Mini w/ ATmega168
Arduino Ethernet
Arduino Fio
Arduino BT w/ ATmega328
Arduino BT w/ ATmega168
LilyPad Arduino USB
LilyPad Arduino w/ ATmega328
LilyPad Arduino w/ ATmega168
Arduino Pro or Pro Mini (5V, 16 MHz) w/ ATmega328
Arduino Pro or Pro Mini (5V, 16 MHz) w/ ATmega168
Arduino Pro or Pro Mini (3.3V, 8 MHz) w/ ATmega328
Arduino Pro or Pro Mini (3.3V, 8 MHz) w/ ATmega168
Arduino NG or older w/ ATmega168
Arduino NG or older w/ ATmega8
1

We're off to a good start. Cool.

When you connected your Arduino board to the PC, the Arduino IDE created a virtual com port (communications port). This is a connection that functions like a serial port on your PC and it is how your PC talks with the UNO and vice versa.


sketch_mar21a | Arduino 1.0.2
File Edit Sketch Tools Help
sketch_mar21a
Auto Format Ctrl+T
Archive Sketch
Fix Encoding & Reload
Serial Monitor Ctrl+Shift+M
Board
Serial Port
Programmer
Burn Bootloader
COM1
COM2
✓ COM11
1
Arduino Duemilanove w/ ATmega328 on COM11

Select

- Tools
    - o Serial Port
        - ▪ COMXX

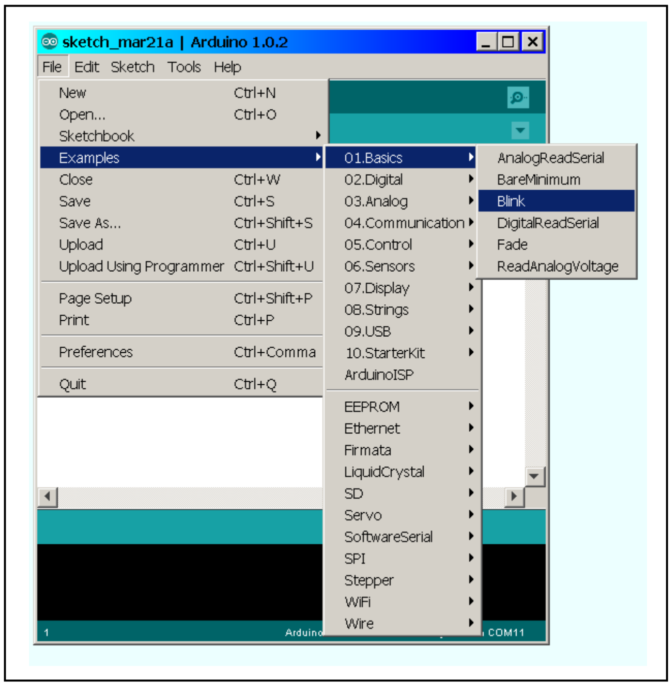
In the graphic above, the serial port that was created is COM11. Yours will probably have a different number. In any case, select that one, not COM1 or COM2 if they appear.

Getting to Work

Now, the ultimate task for all electrical engineers: Getting an LED to flash…this will be our first Arduino program…

Select

- File
    - Examples
        - 01.Basics
            - Blink



Don't get too excited. This is an example program…all your programming assignments will not be this easy or hidden here.

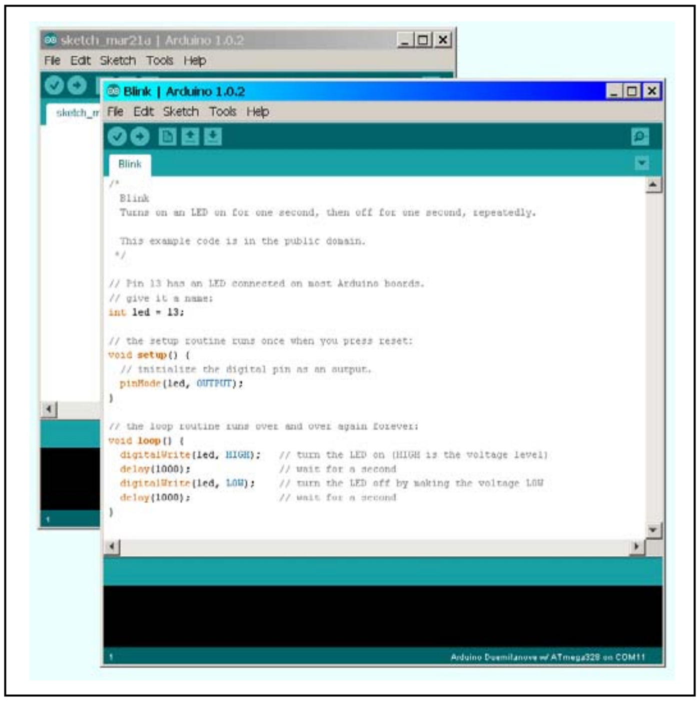
The window that opened contains all of the code for the *Blink* program. The program is well annotated, so read through and understand what each line of code does.

Start by identifying the major functional blocks…you should be able to identify 3.

In Arduino jargon, such a program is called a *sketch*.

An important thing to notice here is that unlike the traditional C program, an Arduino *sketch* does not have a top-level *main()* function. Rather, it has a top-level *loop()* function (delimited by the two {} that we call curly brackets).

Once the *loop()* function is entered, the program will run forever. Well, not if you turn power off, of course or jump up and down on your Arduino board. It's pretty cool, but, it ain't that cool.


sketch_mar21a | Arduino 1.0.2
File Edit Sketch Tools Help
Blink | Arduino 1.0.2
File Edit Sketch Tools Help
Blink
Arduino Duemilanove w/ ATmega328 on COM11

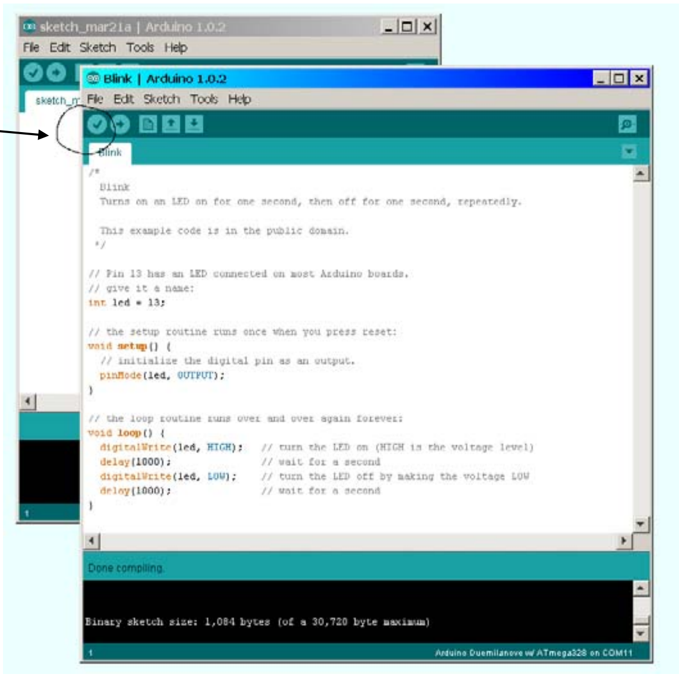
```
/*
  Blink
  Turns on an LED on for one second, then off for one second, repeatedly.
 
  This example code is in the public domain.
 */
 
// Pin 13 has an LED connected on most Arduino boards.
// give it a name:
int led = 13;

// the setup routine runs once when you press reset:
void setup() {
  // initialize the digital pin as an output.
  pinMode(led, OUTPUT);
}

// the loop routine runs over and over again forever:
void loop() {
  digitalWrite(led, HIGH);   // turn the LED on (HIGH is the voltage level)
  delay(1000);               // wait for a second
  digitalWrite(led, LOW);    // turn the LED off by making the voltage LOW
  delay(1000);               // wait for a second
}
```

Our next step is to compile the program into a form that the UNO will understand. It's trilingual – it understands Italian, English, and C.

Select

- √ in the second toolbar

This will appear as *Verify* in the toolbar. Selecting the symbol will compile the program for you. If there are no errors, you will have a notification similar to that shown in the lower pane of the window.

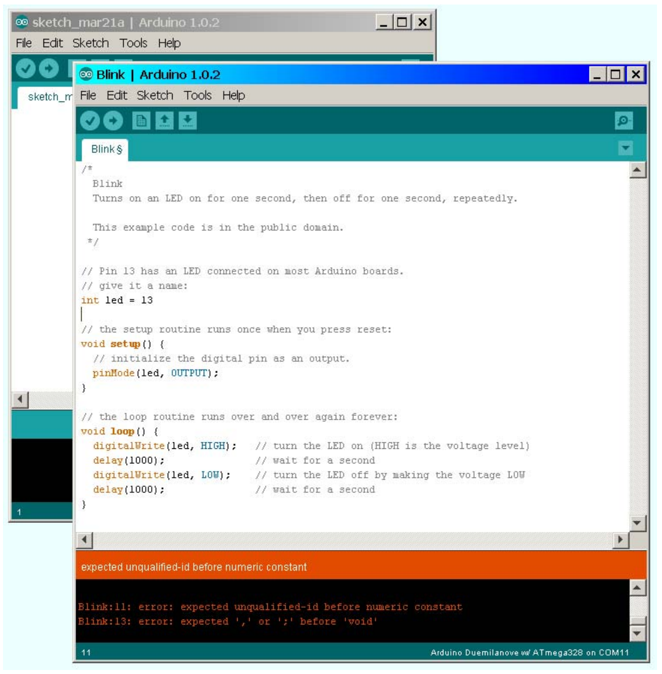

sketch_mar21a | Arduino 1.0.2
File Edit Sketch Tools Help
Blink | Arduino 1.0.2
File Edit Sketch Tools Help
Blink

```
/*
  Blink
  Turns on an LED on for one second, then off for one second, repeatedly.

  This example code is in the public domain.
 */

// Pin 13 has an LED connected on most Arduino boards.
// give it a name:
int led = 13;

// the setup routine runs once when you press reset:
void setup() {
  // initialize the digital pin as an output.
  pinMode(led, OUTPUT);
}

// the loop routine runs over and over again forever:
void loop() {
  digitalWrite(led, HIGH);   // turn the LED on (HIGH is the voltage level)
  delay(1000);               // wait for a second
  digitalWrite(led, LOW);    // turn the LED off by making the voltage LOW
  delay(1000);               // wait for a second
}
```

```
Done compiling.

Binary sketch size: 1,084 bytes (of a 30,720 byte maximum)
```

If there is an error, the bottom pane will contain a message as shown.

Because there is an error, the message is the compiler's best guess as to where it might be; however, its guess as to the location may not be completely accurate.


sketch_mar21a | Arduino 1.0.2
File Edit Sketch Tools Help
Blink | Arduino 1.0.2
File Edit Sketch Tools Help
Blink §

```
/*
  Blink
  Turns on an LED on for one second, then off for one second, repeatedly.

  This example code is in the public domain.
 */

// Pin 13 has an LED connected on most Arduino boards.
// give it a name:
int led = 13

// the setup routine runs once when you press reset:
void setup() {
  // initialize the digital pin as an output.
  pinMode(led, OUTPUT);
}

// the loop routine runs over and over again forever:
void loop() {
  digitalWrite(led, HIGH);   // turn the LED on (HIGH is the voltage level)
  delay(1000);               // wait for a second
  digitalWrite(led, LOW);    // turn the LED off by making the voltage LOW
  delay(1000);               // wait for a second
}
```

```
expected unqualified-id before numeric constant

Blink:11: error: expected unqualified-id before numeric constant
Blink:13: error: expected ',' or ';' before 'void'
```

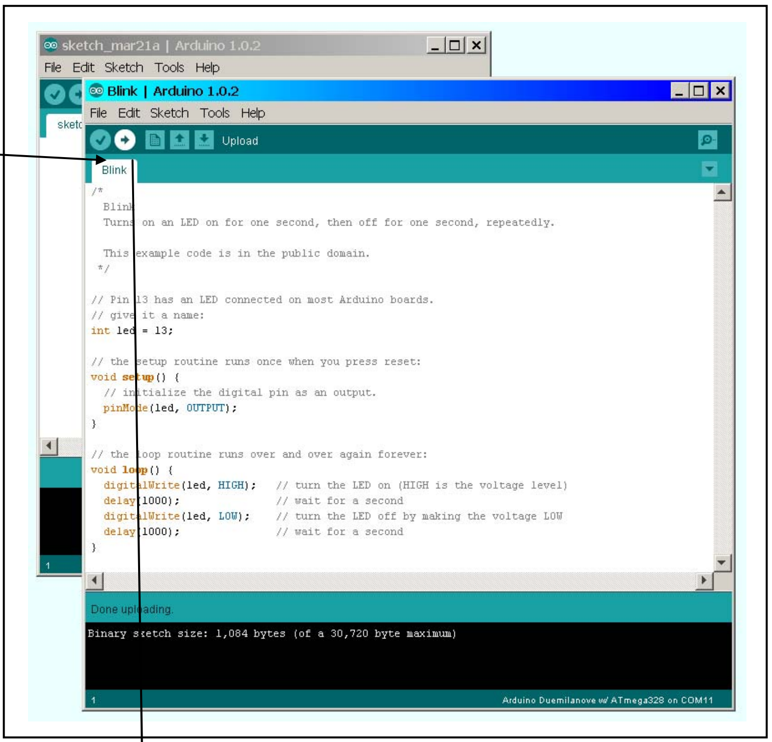
If there are errors, we….strike that, you will have to fix them before proceeding. If there aren't, then, we are ready for the next step. Downloading the code into the microcontroller.

Select

- → in the second toolbar

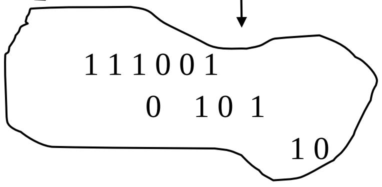
Despite the fact that the symbol indicates *Upload*, we are really *Downloading* to the microcontroller.

If you upload, the 1's and 0's have to get a running start to get up the wire to your processor – this slows them down. On the other hand, by downloading, they are going down hill and will get to your processor much sooner. Just make sure that they don't get going too fast, or they might over run the processor and spill out onto the floor as shown.



```
/*
  Blink
  Turns on an LED on for one second, then off for one second, repeatedly.

  This example code is in the public domain.
 */

// Pin 13 has an LED connected on most Arduino boards.
// give it a name:
int led = 13;

// the setup routine runs once when you press reset:
void setup() {
  // initialize the digital pin as an output.
  pinMode(led, OUTPUT);
}

// the loop routine runs over and over again forever:
void loop() {
  digitalWrite(led, HIGH);   // turn the LED on (HIGH is the voltage level)
  delay(1000);               // wait for a second
  digitalWrite(led, LOW);    // turn the LED off by making the voltage LOW
  delay(1000);               // wait for a second
}
```

```
Done uploading.
Binary sketch size: 1,084 bytes (of a 30,720 byte maximum)
```



When the download is complete, the program will start running on the target processor – the UNO. In this case, an LED on the board will start blinking. Wahoo!!!! An engineer's initiation and success…cool.

Venturing Out – First Steps

Now let's make some modifications to the program.

Program 1: Modify the program so that LED is on for 2 seconds and off for 1 second and repeats.

Program 2: Modify the program so that the LED has two 1 second blinks followed by two 2 second blinks and repeats.

Write these as two separate programs or *sketches*. Be sure to save them.

Venturing Out – Second Steps – Peripheral Devices

Starting with the *Electronic_Bricks_Starter_Kit_Cookbook* under documentation on the class webpage, connect the *LED Brick* to the microcontroller.

Programs 3 and 4: Working from the sample code in the *Cookbook* and the code you developed in the previous exercise, modify Programs 1 and 2 to blink the *LED Brick* rather than the onboard LED. Develop the software, compile, and download it to the target. Verify the proper operation for each program.

Venturing Out – Third Steps – Peripheral Devices

Program 5: Starting with the *Electronic_Bricks_Starter_Kit_Cookbook* under documentation on the class webpage, now connect the *Button Brick* and the *LED Brick* to the microcontroller. Develop the software, compile, and download it to the target. Verify that the *LED Brick* can be controlled by the *Button Brick*.

Venturing Out – More Steps – Processor Output

Program 6: Starting with the working C code, *hw1.ino*, in folder for this project, compile and download the program. When the program has completed downloading,
Select

- Tools
    - Serial Monitor

A small window will open. Verify that the program is properly printing in that window.

Venturing Out – Final Steps (For Now) – Processor Outputting Your Stuff

Program 7: Starting with the working C code, *hw1.ino* and your working C code for homework assignment problem 1, modify the code for *hw1.ino* to incorporate your code for homework problem 1. The main change will be modifying any of your *printf* lines to the proper combinations of,

```
Serial.print("stuff to print");
Serial.println("stuff to print followed by new line");
```

Ok, take a break for a few minutes

Ok, that's long enough…

Part 2:

Working with the Development Environment and Target Platform

In this portion of the lab, we'll continue working with the Arduino UNO microcontroller, the C language, and Arduino's dialect of C.

Note: Every member of your team should do each of these exercises….not just watch. Your team can submit one final report covering Parts 1 and 2 of the lab.

Writing to the LCD

Do not connect the Arudino board to the PC yet. We don't want to be connecting anything to the board with power applied. Doing so may release the smoke demon.

Carefully install the Bus Shield onto the Arduino board. If you have version R3 of the UNO board (look on the back of the board), then make certain that you have the shield positioned properly. There will be several unused pins on the R3 version of the UNO board. Please ask if you are not certain.

Using the 10-pin ribbon cable, connect one end to the LCD and the other to the Bus 1 connector on the Bus Shield.

You can now connect your microprocessor to the PC.

Start with *Course 2 Bus* from the *Electronic Bricks Starter Kit Cookbook*, which is found under **Class Documentation** on the class web page.

In the Arduino IDE, select

- File
    - Examples
        - Liquid Crystal
            - Display

Configure the pin out for the LCD for Bus 1 as specified and shown in the *Electronic Bricks Starter Kit Cookbook.*

Compile and download the program to the Arduino. Verify that the program is executing and properly displaying the *Hello World* message.

To learn about some of the functions you can use to work with the LCD display, go to the LCD library here:

http://arduino.cc/en/Reference/LiquidCrystal

We will now use one of those functions, *setCursor()*. Modify the program to print the first name of each lab partner. Print two names on the first row of the display and one on the second.

Working With the LCD and Peripherals

Connect two of the *Button Bricks* to the *Bus Shield*. Identify these as *button1* and *button2*. Starting with the program from the previous design and working with the LCD library functions, design a program that starts by writing the character 'X' to position row 0 and column 0 on the LCD.

Next, design the code that will read the states of the two buttons. When *button1* is pressed, move the character 'X' to the left by 1 position (column). When *button2* is pressed, move the character 'X' to the right by 1 position (column).

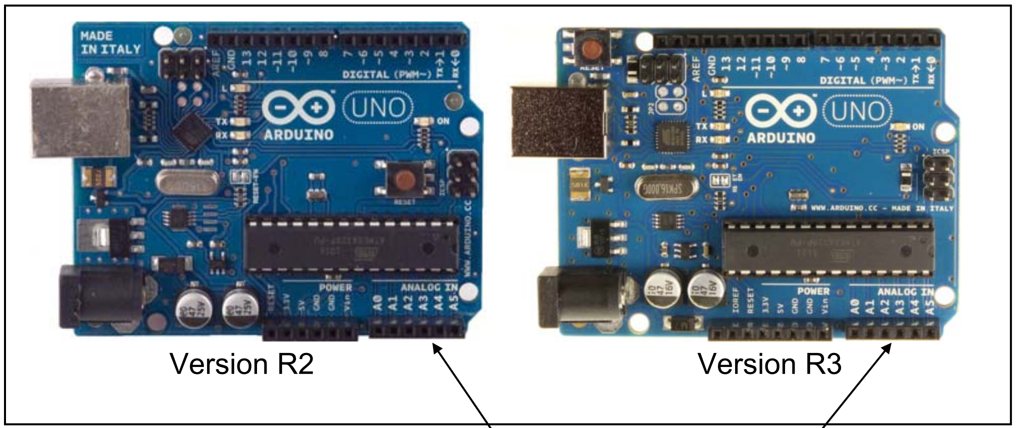
Decide how you want to handle the end cases, i.e. when the ‘X’ is in the 0th or last column. What should you or do you want to do if both buttons are pressed?

Communicating Between Two Microprocessors

For this project, we will work with a very simple local area network (LAN) called the I^2C (Inter Integrated Circuit) or TWI (Two Wire Interface) Bus.

Disconnect the Arudino board from the PC. We don’t want to be connecting or disconnecting anything to or from the board with power applied. Doing so may release the smoke demon.

Start by carefully removing the Bus Shield from the Arduino board.
The following figure gives a top view of the two revisions of the UNO board. Revision R2 is shown on the left, and R3, is shown on the right.



Cut and strip about 1/4 inch of insulation from the ends of two pieces of wire, approximately 6 to 7 inches long. For clarity and to avoid mistakes, it is best to use two different colors.

Observe that both versions of the board show the six *Analog In* pins on the connector on the lower right hand corner of the board.

Let’s now build the I^2C, two wire bus. Start with two Arduino boards and the two wires we just cut. The specific version of each board does not matter. Actually, the version of the wires doesn’t matter too much either. Connect one of your 6-inch wires between pins A4 on each of the two boards and the second between pins A5 on each of the two boards. We now have a two-wire bus connecting the two microprocessors. Hey, that’s our first network. That’s wasn’t too difficult now was it? Sorry, it’s not a social network, though.

We designate one of the boards as the *master* and one as the *slave*. This is a common configuration in many systems. For the USB, for example, the root hub on your PC is the master and the myriad peripheral devices are the slaves.

From the folder *I2C0* in the Lab1 directory on the class web page, bring down the two files: *i2cm.ino* and *i2cs.ino*. The first is the code we'll use for the *master* and the second is the code that we'll use for the *slave*.

Open an Arduino IDE and connect the board that you designated as the *slave* to the PC. Configure the board as the *Arduino Uno* and note the virtual *serial port* that was created. Open the file *i2cs.ino* then compile it and download to the *slave*.

Next, open a second Arduino IDE, and connect the board that you designated as the *master* to the PC.

Note: This is a second IDE, NOT a second file on the first IDE

You should now have *two* instances of the Arduino IDE open. Using the second IDE, configure the board as the *Arduino Uno* and note the virtual *serial port* that was created. This should be different from the port that was created for the *slave* board. In the second IDE, open the file *i2cm.ino* then compile it and download to the *master*.

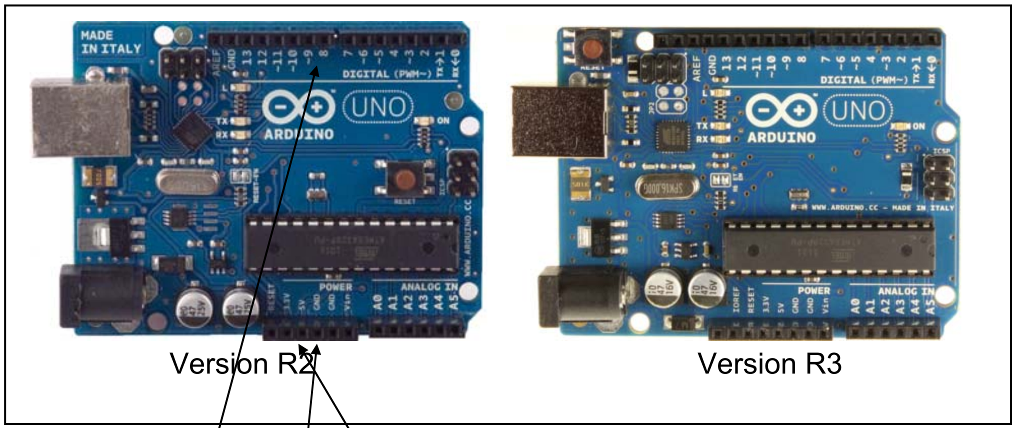
On the *master*, open the serial monitor. You should see the data that is being sent to the *slave* displayed. On the *slave*, open the serial monitor. You should see the data that is being received from the *master* displayed.

Working With the Two Processors and Peripherals

For this final project, we will use the I^2C (Inter Integrated Circuit) or TWI (Two Wire Interface) Bus to send data from one microprocessor to a second to control a peripheral device connected to the second processor.

Modify the code from the previous exercise to send a command from the *master* to the *slave* telling the *slave* to blink the LED ON or OFF. At this point, we do not need the actual LED. As we did in the previous exercise, test your design first by using the *Serial Monitor* to confirm that your commands are correctly being sent by the *master* and correctly received by the *slave*.

Let's now complete the design, integration, and test of our system. Disconnect the Arudino boards from the PC. We don't want to be connecting or disconnecting anything to or from the boards with power applied. Doing so may release the smoke demon.


Version R2 Version R3

Connect one end of one of the three-pin cables to the LED brick. Then, using three wires approximately 6-7 inches long, connect the *white* wire on the open end of the three-pin cable to *Digital Pin 8* on the *slave* Arduino board. Then connect the *black* wire on the open end of the three-pin cable to *ground* on the Arduino board and finally connect the *red* wire on the open end of the three-pin cable to *+5 volts* on the Arduino board.

You have now connected the LED peripheral device to the microprocessor. Reconnect the UNO boards to the PC using the same procedure as in the previous program. Then compile your programs, download them, and test your design. You should have a blinking light on the *slave* Arduino board. Cool!!!

Deliverables

A lab report for the first and second parts of the lab containing,

1. The annotated source code for all your *programs* / *sketches* for the UNO.
2. Representative screen shots showing the results of executing the applications on the PC.
3. Anything that we haven't thought of.