CSE 102 – Studio 1

Introduction to Studio

The studio experience is quite similar to what you have previously experienced in CSE 131.

Tell Us About Yourself

Individually, please complete this survey asking about skills topics that we plan to incorporate into CSE 102. What do you already know about topics we are considering, and what additional topics would you like us to include?

Also, please individually tell us when your schedule is sufficiently open that if we schedule help sessions then you could make it to them.

Next, on to the good stuff (do the following activities in your studio group):

Getting Started with the Arduino Development Environment

Many of you will be familiar with the Eclipse development environment. You know how to check out a local copy of your workspace from a repository, how to author programs in Java, and execute those programs on the computer that is running Eclipse. Here, you are being introduced to another development environment, targeting the Arduino platform.

The Arduino is a (very) small computer that has dramatically fewer capabilities than the desktop or laptop machines that you have used in the past to run Java programs. For example, it doesn't have a keyboard, it doesn't have a screen, its processor is over 100 times slower, and you might even be wondering, "What is the point?" The point is that small computers like the Arduino are priced relative to their capabilities. Want a computer for under $10? If so, the Arduino is a great choice! Over the course of the semester, we'll discover all that the Arduino can do, even though it starts out as humbly as it does.

Because of the limited input and output capabilities of the Arduino, we will use a desktop or laptop computer to assist in writing and executing programs on the Arduino. The Arduino development environment actually runs on your desktop machine. As with Eclipse, you have a local workspace on the desktop machine, you checkout the workspace from a repository, and author programs for the Arduino. Not in Java, though, the language we will use for most of our Arduino development is C.

The other distinction between Java development in Eclipse and the Arduino development environment is that to execute a program on the Arduino processor, the program to be executed must be loaded into the Arduino's memory so the processor on the board can execute it. This is accomplished via the use of a USB connection (which both supports two-way communication between the Arduino and the desktop machine as well as provides power to the Arduino board).

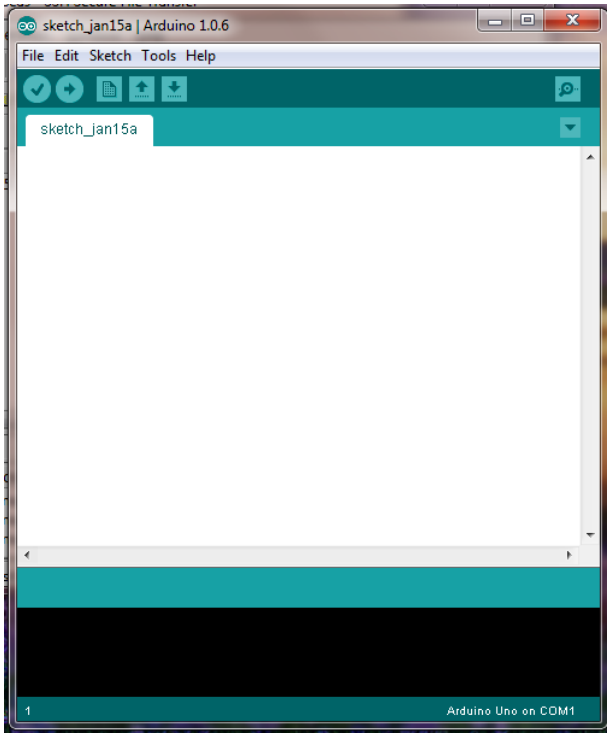
Install Development Environment

If you wish to use your own laptop for development, the Arduino development environment can be downloaded from http://arduino.cc/en/main/software and installed on Windows, Linux, or Mac systems. Download and install version 1.0.6 of the development environment.

The development environment is already installed on the lab machines, so if you have difficulty with the installation process, continue with the studio exercises below and then ask for help with the software installation.

Authoring Programs

Start up the Arduino development software and you will be presented with a text editor window:



For all of your software development on the Arduino platform, do not hesitate to access the Language Reference documentation on-line.

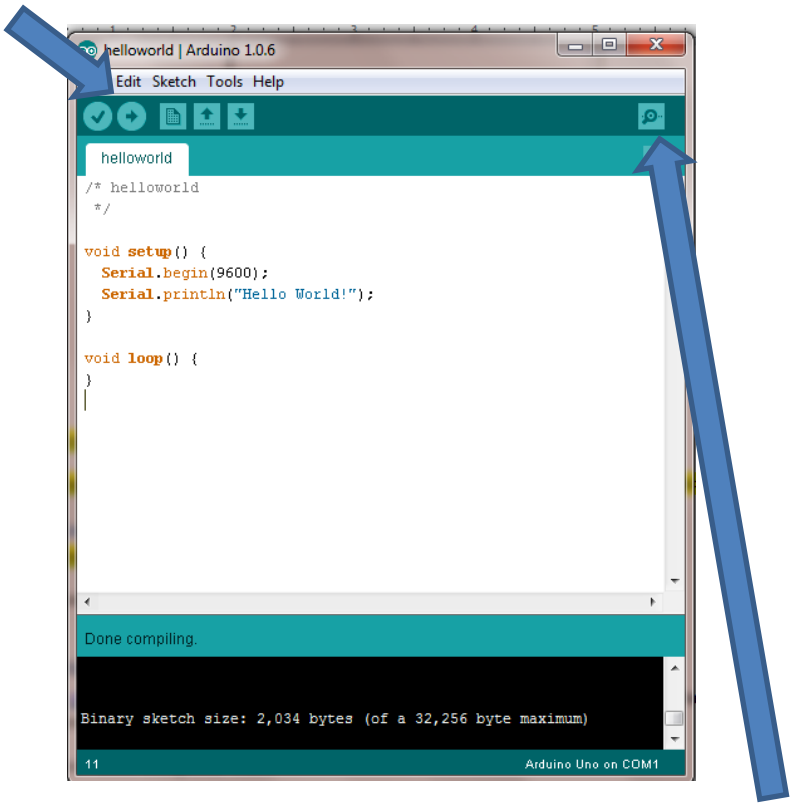
Copy and paste the following code into the text editor:

```
/* helloworld
 */

void setup() {
  Serial.begin(9600);
  Serial.println("Hello World!");
}

void loop() {
}
```

Save the program (`File -> Save As` ...) under the name `helloworld`.

Compiling and Executing Programs

Using the Upload button, compile `helloworld` and send it to the Arduino for execution.



The window that displays the output can be opened using the Serial Monitor button. You might need to set which COM port via `Tools -> Serial Port` before this will work. Which COM port (e.g., COM3, COM4) will depend on which USB port you are using on the PC (it is unlikely to be COM1 or COM2).

Authoring a Program

Using the `delay()` library function, author a heartbeat program (named `heartbeat1`) that, once per second, prints

```
<n> sec elapsed time
```

Where <n> is the number of seconds that has elapsed since the program was reset.

Execute your program on the Arduino and verify (via an external time reference) that it is operating at approximately the correct rate.

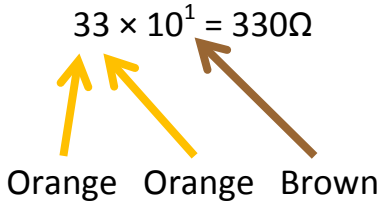
Also, author a program in Java, to execute on your desktop computer, which does the same thing. It should print a message to the console once per second. Use the `Thread.sleep()` method to measure the passage of time, as described in class.

Startup both programs together, and observe how long it takes for them to start drifting in time relative to one another. Add text output to `heartbeat1` that displays the return value from the `millis()` function. What additional does that tell you about the accuracy of the time using `delay()`?

Digital Outputs

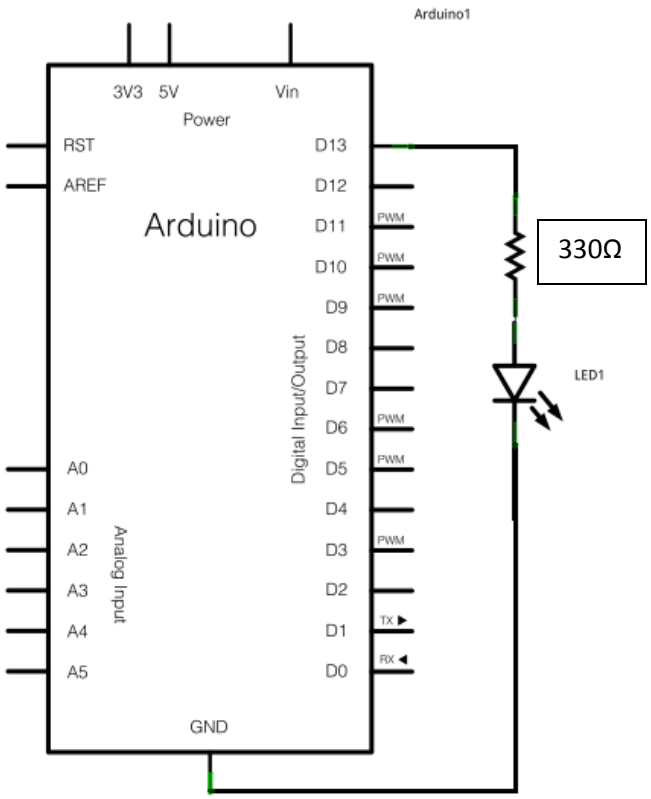
So far, we have exclusively used the desktop computer's display to serve as the output device for our programs that execute on the Arduino processor. The next task is to add a digital output capability to the Arduino itself. You will start by building one LED output, and then later extend this to 6 LED outputs.

Construct the circuit below, connecting a 330Ω resister in series with a red LED from connector D13 to ground (GND). You can recognize the resistor's value as 330Ω by the color code on the part. The first two colors are significant digits, and the 3rd color is the value multiplier (thing scientific notation where the decimal point follows the two significant digits. Orange is a 3 when encoding significant digits, so the first two digits are 33. Brown is 10^1 when encoding the multiplier, so we multiply the 33 by 10 to get 330. (Note: Brown encodes a 1, the exponent of the multiplier.)



The table of values is as follows:

0	1	2	3	4	5	6	7	8	9
black	brown	red	orange	yellow	green	blue	violet	grey	white



Program Heartbeat1

Once you have constructed your digital output, let's alter your heartbeat program to output directly to the LED. Once per second, using the `digitalWrite()` function, toggle the output to the LED (either LOW to HIGH or HIGH to LOW). This should give you a blinking LED that is flashing at 0.5 Hz (one on-off cycle every 2 seconds). It is fine to have the program continue to send the "elapsed time" message to the desktop computer's screen.

Show your running `heartbeat1` to an instructor or TA.

Program Heartbeat2

Copy the program `heartbeat1` into a new file named `heartbeat2`. Next, alter the program `heartbeat2` so that it flashes at 1 Hz (one on-off cycle every second) and the amount of time that the LED is on is specified via

```
const int LEDOnTime_ms = <n>;
```

where the on-time <n> is specified in milliseconds. Instead of using the `delay()` function, this time use the `millis()` function to access the free-running timer. Play with the on-time parameter until the LED flash is a reasonable length (i.e., it is easy to see, but doesn't stay on too long). Again, run your PC-based heartbeat and Arduino `heartbeat2` together, and see how long it takes for them to start drifting in time relative to one another.

Show your running `heartbeat2` to an instructor or TA.

Analog Output

Read the tutorial on pulse width modulation (PWM). Open the program `Fade`

```
File->Examples->01.Basics->Fade
```

and move your red LED output circuit to pin 9 (or add a second red LED output circuit at pin 9). Don't forget to include the resistor!

Alter the delay() call at the bottom of the program (or use some other program alteration) so that the analog output value stays stable for a long enough time so that you can observe the output voltage on the oscilloscope for several different analog output values. Ask an instructor or TA how to operate the oscilloscope if you are not yet familiar with its use.

Metronome

For the last program using a single LED output, author a metronome application (call it `metronome`). Continue to use the `LEDOnTime_ms` parameter to specify the amount of time that the LED is on each time it flashes. Add a new parameter

```
const int metronomeRate_bpm = <b>;
```

which specifies the rate of the metronome (in beats per minute). Again, use `millis()` for timing purposes.

When `metronomeRate_bpm` is 60, program `metronome` will work just like program `heartbeat2`. Explore how fast your metronome can work. What is limiting the effective rate as it gets higher and higher?

Run your metronome for a while to see how accurate it is by comparing it to an alternative time reference (e.g., the clock/stopwatch on your phone). You might want to use an output similar to the "elapsed time" from the `heartbeat1` program for this purpose. Does the metronome time drift appreciably? Can you quantify the drift?

Show your running `metronome` to an instructor or TA.

Stoplight Controller

Next, build 6 LED digital outputs: 2 red, 2 yellow, and 2 green. Arrange them physically such that they resemble the indicators for a stoplight (i.e., 1 red 1 yellow and 1 green LED represent a street running north/south and the other 3 LEDs represent a street running east/west.

Design a program called `stoplight` which controls your stoplight. The timing should be specified using an appropriate set of parameters which you set via `const int` statements in C. What are the states of the intersection that it is appropriate for the stoplight controller to retain?

Show your running `stoplight` to an instructor or TA.

Submitting Your Work

When you have finished your studio exercises, return the Arduino hardware to the instructor or TA and make sure they record your attendance and participation.