What is this lab about ?

1) Becoming familiar with the Arduino microcontroller integrated development environment (IDE), which is the tool that enables you to compose and compile code and then program the code to the chip.
2) Becoming familiar with the Arduino hardware's input and output capabilities: digitalWrite, serial.println, etc.
3) Reading a schematic to understand how the microcontroller chip interfaces to the external world.
4) Becoming familiar with the main sections of code used in any Arduino program. setup(), loop().
5) Using the serial monitor program to display what the microcontroller is doing
6) Using the serial monitor to control what the microcontroller is doing by processing characters typed in by a user (This is a simple human-machine interface).
7) Controlling how fast the microcontroller performs a task by using time delay functions.
8) Practicing writing source code, compiling code, uploading code to the Flash memory of the microcontroller and debugging the microcontroller's operation.
9) Practicing using function calls to make the code easier to read.

How to Succeed With This Lab:

Purchase your Arduino board and bring it to class every week.

Before executing the procedure, skim-read through the whole lab handout. (2 minutes)

Work together as a team with your lab partners to solve problems and overcome any snags (e.g. syntax issues with C, issues with Arduino programming interface -- called an IDE or Integrated Development Environment).

The number one problem students face doing this lab:

Not having the USB serial port in the IDE set to the COM port where the Arduino is connected (e.g. COM1 instead of COM3 or COM4). A second problem area is not having the serial monitor window configured to the same baud rate and format as the microcontroller.

Reading and References:

http://www.atmel.com/devices/atmega328p.aspx Rev I datasheet

http://Arduino.cc

http://Arduino.cc/en/Reference/HomePage

http://forum.Arduino.cc/

What should you be able to do at the end of the lab ? (Learning Outcomes)

Utilize Arduino IDE to edit, compile, load and run a program on the microcontroller.	
Find and fix syntax errors that prevent compilation.	
Demonstrate understanding of how to use Arduino serial terminal monitor window to debug a program using print statements placed at strategic locations in your code.	
Utilize delay statements to control the rate at which a program runs.	
Read the schematic of the Arduino board and be able to identify connection between pins on the headers on the printed circuit board and pins of the actual microcontroller chip (ATMEL 328)	
Create a simple menu system to control the operation of a microcontroller.	

Overview of Arduino Uno Microprocessor Specifications:

Microcontroller	ATmega328
Operating Voltage	5V
Input Voltage (recommended)	7-12V
Input Voltage (limits)	6-20V
Digital I/O Pins	14 (of which 6 provide PWM output)
Analog Input Pins	6
DC Current per I/O Pin	40 mA
DC Current for 3.3V Pin	50 mA
Flash Memory	32 KB (ATmega328) of which 0.5 KB used by bootloader
SRAM	2 KB (ATmega328)
EEPROM	1 KB (ATmega328)
Clock Speed	16 MHz

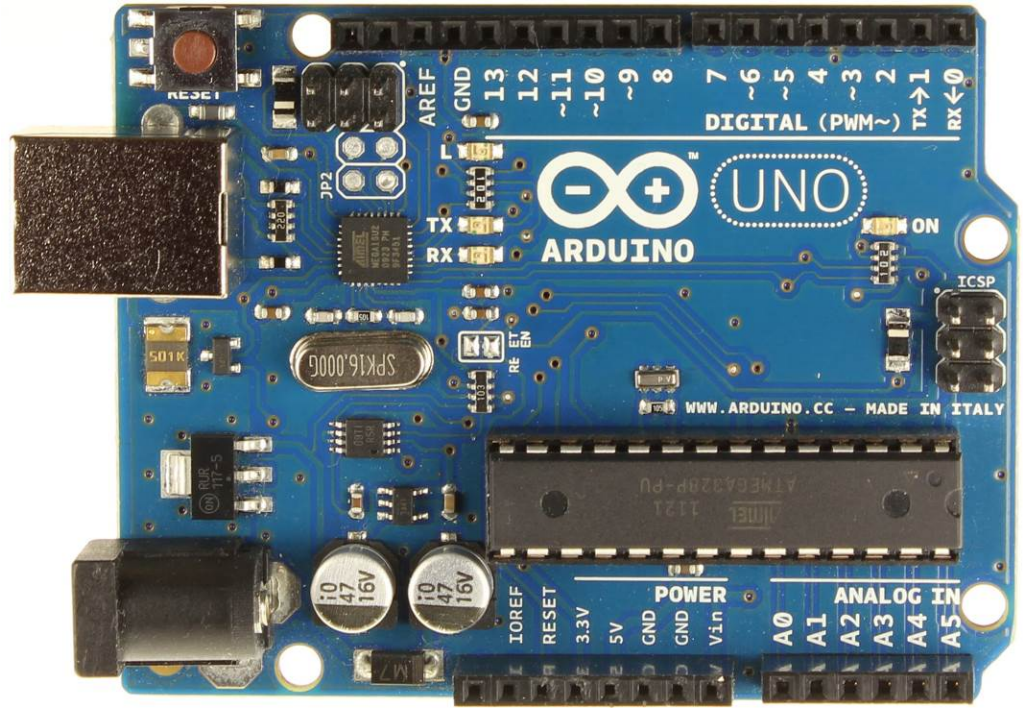


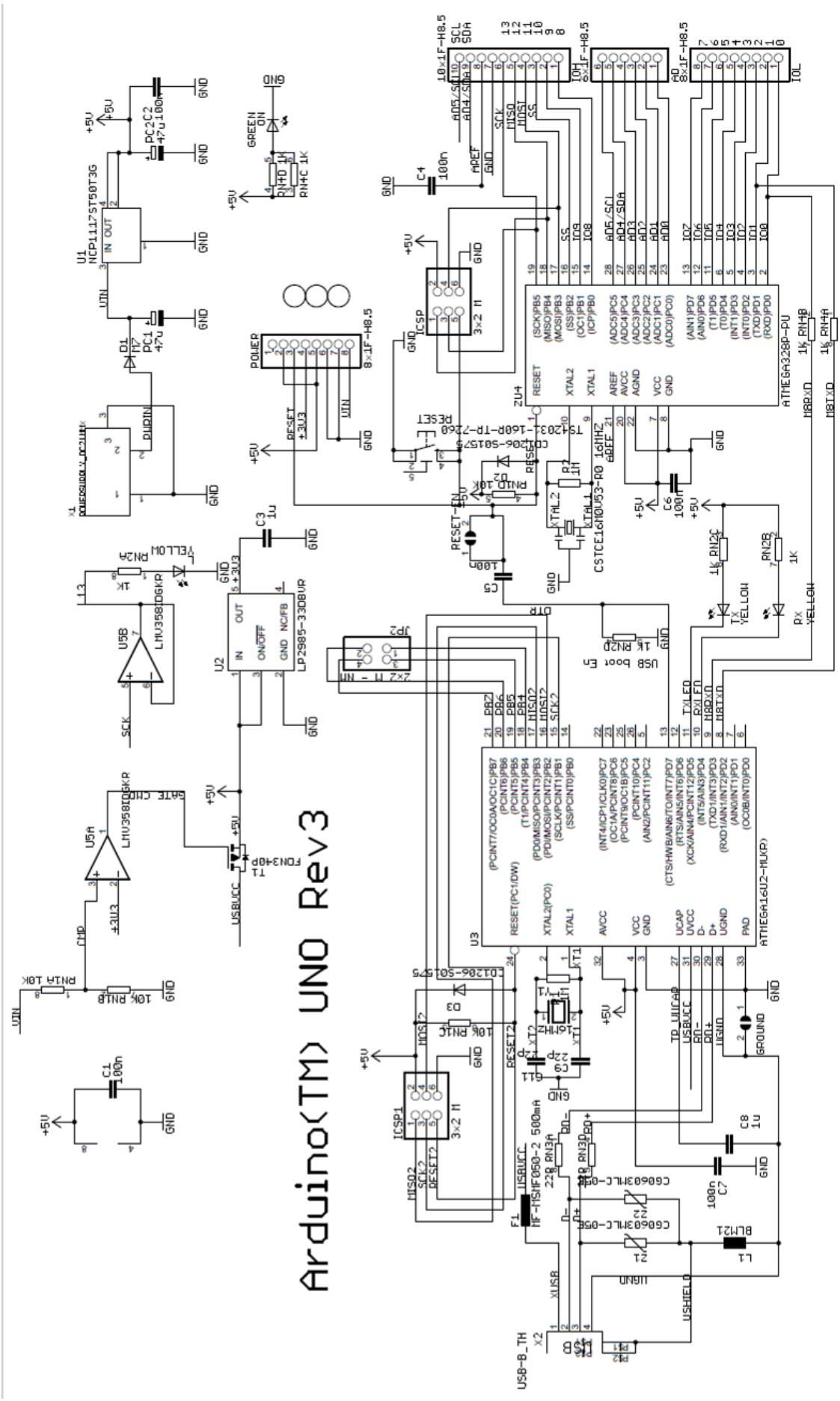
image ref: http://arduino.cc/en/Main/ArduinoBoardUno

Arduino Uno R3 Schematic:

Questions about the schematic:

1) How many LEDs are in this schematic?

2) To which microprocessor pins are the LED's attached to in this schematic? e.g. PB1?


Arduino(TM) UNO Rev3

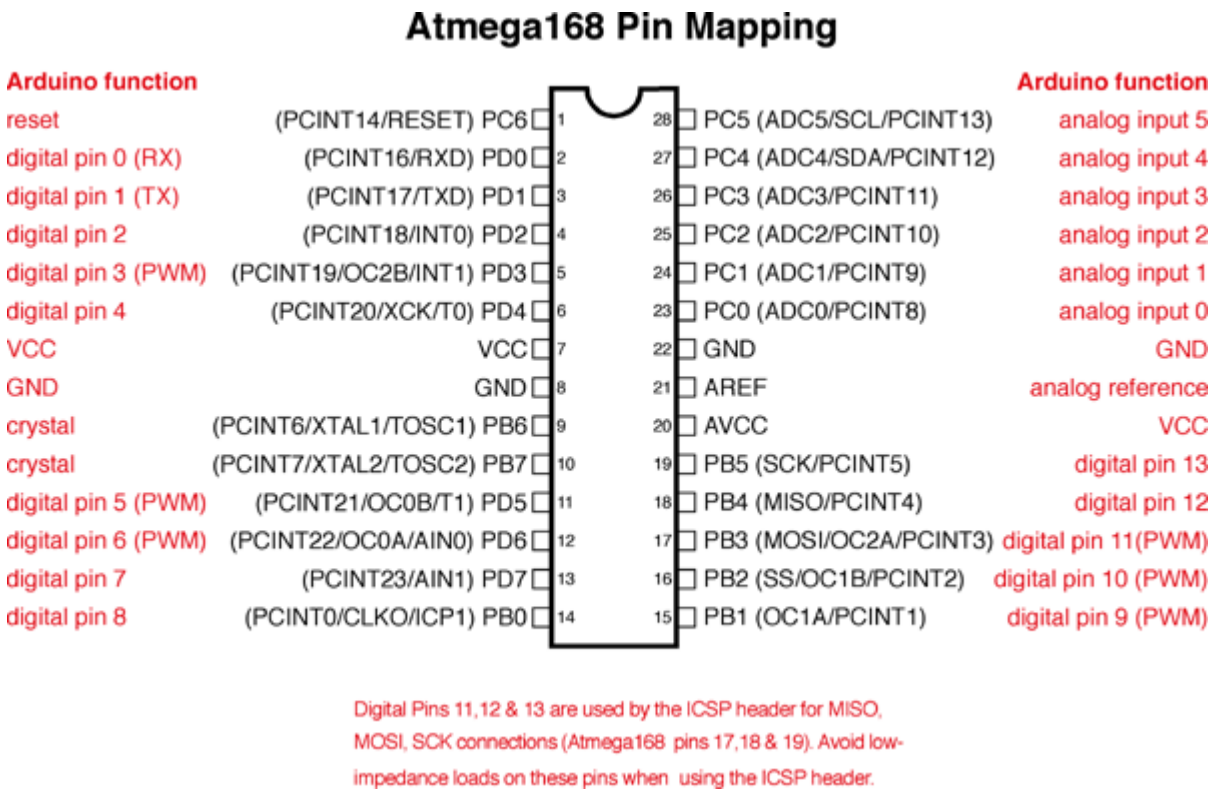


image from: http://playground.arduino.cc/Learning/PortManipulation

Atmega168 Register Descriptions

PORTD maps to Arduino digital pins 0 to 7
DDRD - The Port D Data Direction Register
PORTD - The Port D Data Register
PIND - The Port D Input Pins Address

PORTB maps to Arduino digital pins 8 to 13 The two high bits (6 & 7) map to the crystal pins and are not usable
DDRB - The Port B Data Direction Register
PORTB - The Port B Data Register
PINB - The Port B Input Pins Address

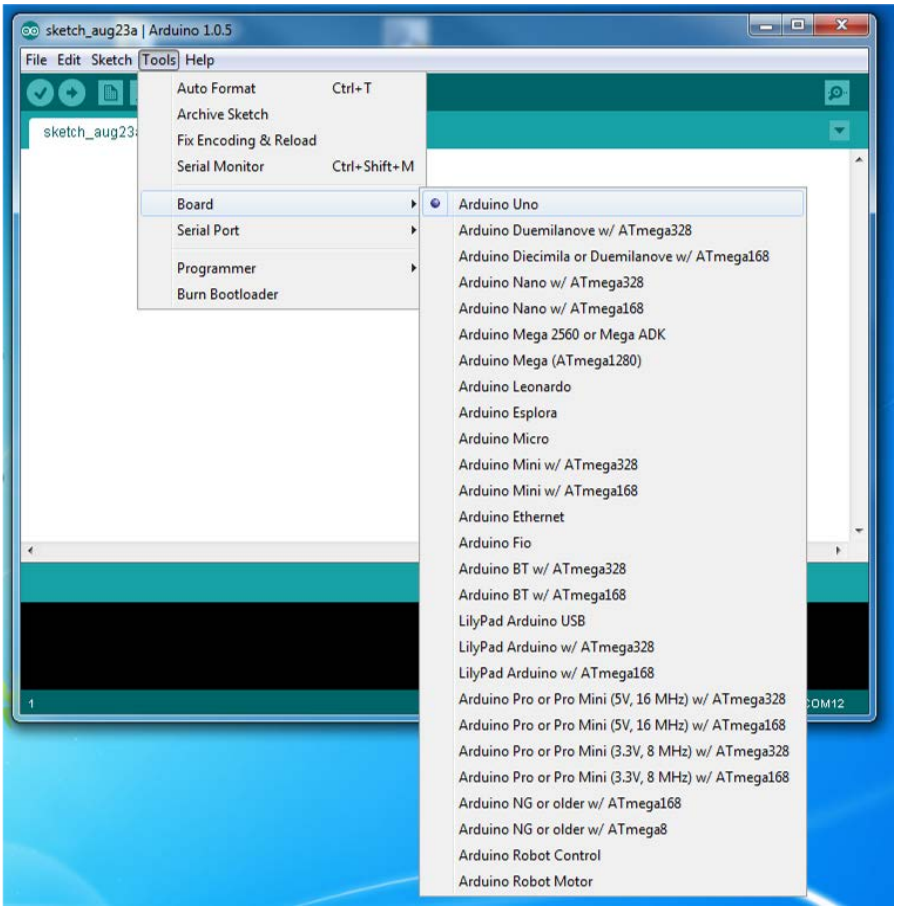
PORTC maps to Arduino analog pins 0 to 5
DDRC - The Port C Data Direction Register
PORTC - The Port C Data Register
PINC - The Port C Input Pins Address

Programming First Steps – Make the LED blink

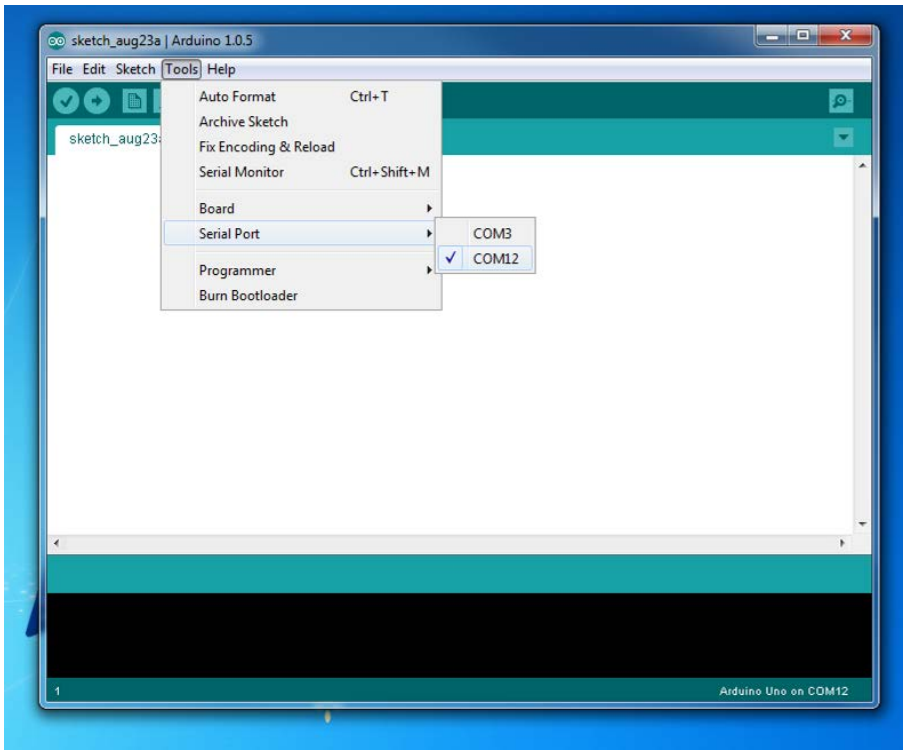
Procedure:

1) Setup the following items in the lab.
    a. Arduino Uno microprocessor board
    b. USB cable from Uno to host PC.

2) Launch the Arduino.exe program located in the C:software folder.
3) Configure the Arduino host program to talk to the correct board (Uno)

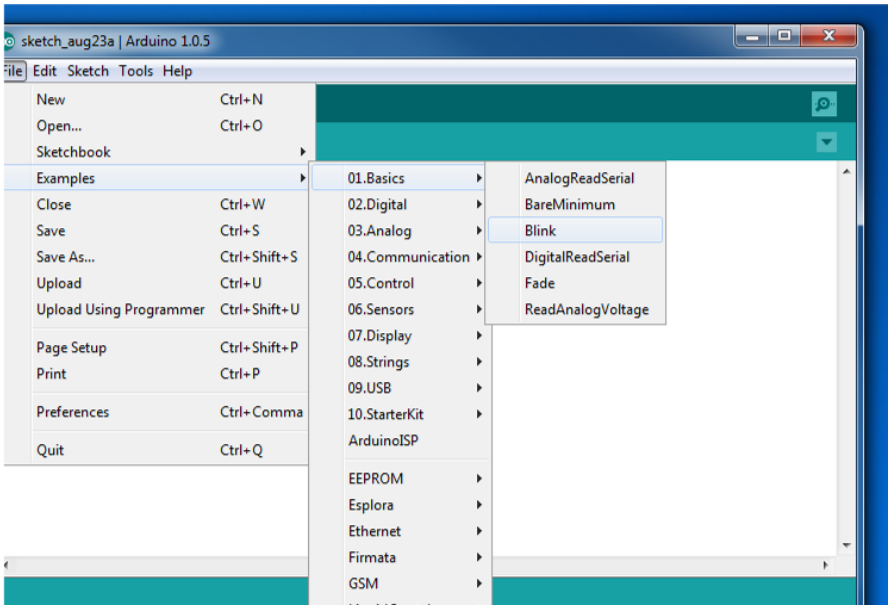


4) Configure the Arduino IDE to talk to the Uno board on the correct USB serial COM port



5) Load the Blink program.

a. File-> Examples->1.Basic->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
}
```

What does this code do?

This code blinks the LED attached to pin 13. The LED is turned on for 1 second (1000 msec) and off for 1 second (1000 msec). First the code defines the pin where the LED is attached "`int led = 13;`". Then that pin is set to be an output pin "`pinMode(led, OUTPUT);''` This is done in the setup portion of the code "`void setup() {`". Then the digitalWrite command is used to make the LED pin go HIGH and then LOW repeatedly in the main part of the code. " `digitalWrite(led, HIGH)`"

In Arduino, the main part of the code is in a section called loop "`void loop() {` ". The code cycles through the loop section over and over again, blinking the light on and off. The microcontroller is very fast (16 MHz) and would turn the light on and off so fast that you could not see it blink. Therefore, a time delay is added so the light stays on for a long enough period to see it. "`delay(1000);`" The delay command hold the program at this line for 1000 milliseconds.

b. Now, upload the program to the microprocessor File->Upload, or Ctrl+U. This will compile the code, check for errors, and upload the file to flash memory.

c. If you see an error message that says file did not upload, go to Tools-> Serial Port and change the serial port e.g. select COM6 instead of COM1. Try uploading again.

d. The code will automatically start running and you should see the LED blink on and off slowly.

6) Modify the code so that there is a shorter delay between blinks:

a. Change the delay from 1000 milliseconds to 200 milliseconds:

```
/*
  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(200);               // wait for a second
  digitalWrite(led, LOW);    // turn the LED off by making the voltage LOW
  delay(200);               // wait for a second
}
```

b. Upload the code to the microprocessor File->Upload, or Ctrl+U. Verify that the LED blinks steadily at a faster rate. Try 50 msec. Have your lab instructor sign off on the 50 msec blink rate.

7) Summary Comments:

The Arduino IDE has numerous built-in code examples. These are a great way to learn the capabilities of the micro and learn how to program it. The Arduino website and forums have a large collection of FAQs as well as responses to individual debugging questions. You will find these website very helpful if you do a project with the Arduino microcontroller and helpful even for other micros or other programming projects.

In the next section, we will start using custom code to explore specific ways of debugging your programs and using more features of the microcontroller.

SECTION 2 – Using the Serial Monitor to Observe Operation of the Microcontroller

Code:

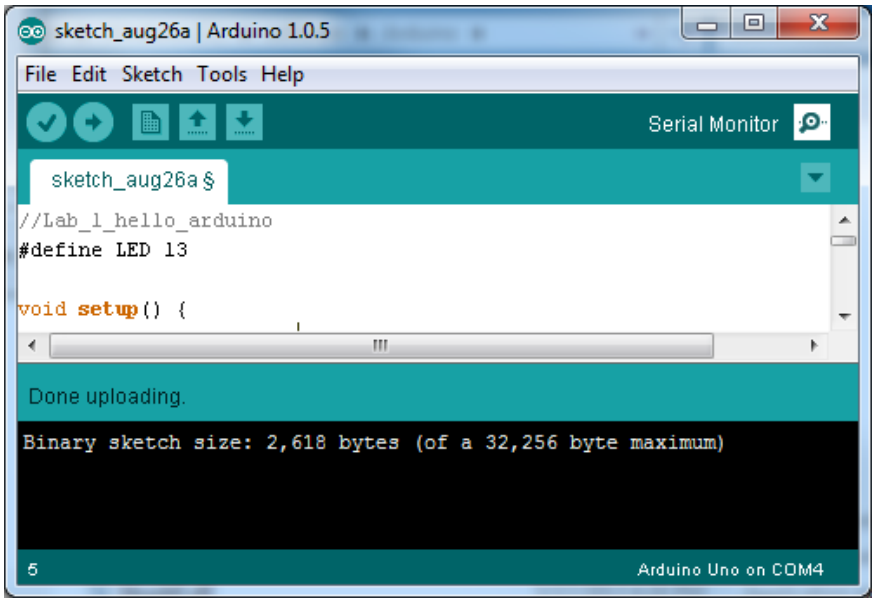
```
//Lab_1_hello_arduino
#define LED 13

void setup() {
  pinMode(LED, OUTPUT);
  digitalWrite(LED, LOW);

  // Set console termination for 'No line ending'
  Serial.begin(9600);
  Serial.println("Lab 1: hello arduino v0.0\n");
  delay(10000);
}

void loop() {
  digitalWrite(LED, HIGH);   // turn the LED on (HIGH is the voltage level)
  Serial.println("On");
  delay(500);               // wait for a second
  digitalWrite(LED, LOW);    // turn the LED off by making the voltage LOW
  Serial.println("Off");
  delay(500);               // wait for a second
} // loop()
```

What does this code do?

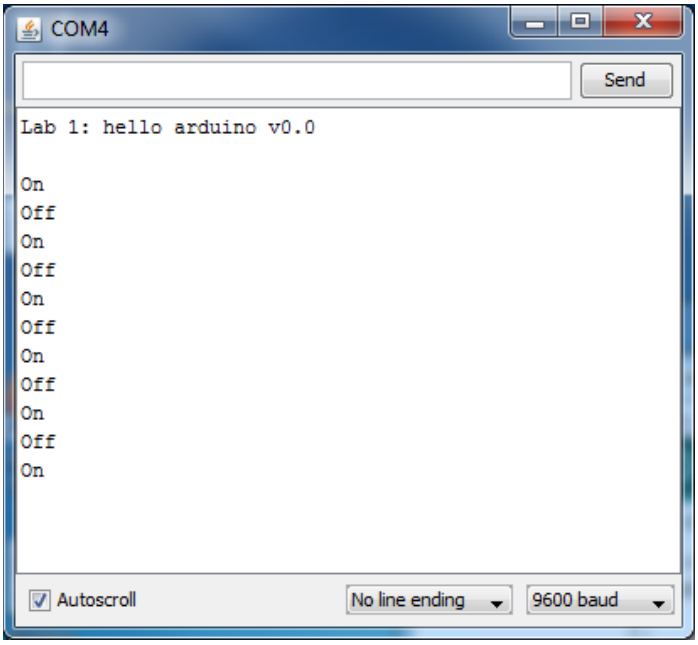
This code blinks the LED on and off for 500 milliseconds. It also prints text to the screen, telling you what the code is doing. First, a serial port is configured to talk at 9600 baud "`Serial.begin(9600);`" Then it prints a line of text out the serial port which is then displayed on the computer display "`Serial.println("Lab 1: hello arduino v0.0\n");`" Later in the main loop() the state of the LED is printed out to the serial port as "On" and "Off".

Procedure:

Copy the code from the above text box and paste it into the Arduino IDE, compile it and upload to the Arduino UNO. Click on the upload icon or use the keyboard shortcut CTRL+U.



When the code uploads, what do you see? You probably won't see anything except that the LED is blinking. To be able to see the serial data, you have to launch the serial monitor in the Arduino IDE. You can do this by clicking on the Serial Monitor icon in the upper right hand corner of the IDE or by the keyboard shortcut CTRL+SHIFT+M. Try this and observe the text that comes up in the serial monitor window.



SECTION 3 – Using the Serial Monitor to Control Operation of the Microcontroller

Code:

```
//Lab_1_hello_arduino
#define LED 13
char inChar;

void setup() {
  pinMode(LED, OUTPUT);
  digitalWrite(LED, LOW);

  // Set serial monitor console termination for 'No line ending'
  Serial.begin(9600);
  Serial.println("Lab 1: hello arduino v0.1\n");
  delay(5000);
}

void loop() {
  if (Serial.available()) {
    inChar = Serial.read();
    Serial.print("Serial input detected: ");
    Serial.println(inChar);
  }
  if (inChar == 'n') digitalWrite(led, HIGH); // oN
  if (inChar == 'f') digitalWrite(led, LOW);  // oFf
} // loop()
```

What does this code do?

This code allows the user to control whether the LED is on or off by typing a command from the LED on and off for 500 milliseconds. By typing the character ‘n’ and hitting return in the serial monitor, you can turn on the LED. Try it. You can turn off the LED by typing the character ‘f’. Try it. The character you type in is defined as a character datatype named inChar. “`char inChar;`”
In the main loop, an if-statement checks to see if serial data is available at the microcontroller, coming from the PC. “`Serial.available()`”. If there is data from the PC, then read it and printout what it was. Then another set of if-statements turns the LED on or off based on the input character.

Procedure:

Copy the code from the above text box and paste it into the Arduino IDE, compile it and upload to the Arduino UNO. Click on the upload icon or use the keyboard shortcut CTRL+U.

Note that when you try to compile the code, it will not compile. There is a syntax error. Examine the error messages by scrolling through the black status box section at the bottom of the IDE. Start by fixing the first error in the list and then recompile as often just the first error causes all the subsequent error messages. If there is still an error, do the same thing, fix the first error, recompile and keep going.

When the code uploads, what do you see? You probably won’t see anything except that the LED is blinking. To be able to see the serial data, you have to launch the serial monitor in the

Arduino IDE. You can do this by clicking on the Serial Monitor icon in the upper right hand corner of the IDE or by the keyboard shortcut CTRL+SHIFT+M. Try this and observe the text that comes up in the serial monitor window.

SECTION 4 – Adding more menu items – and introducing a bug

Code:

```
//Lab_1_hello_arduino
#define LED 13
char inChar;

void setup() {
  pinMode(LED, OUTPUT);
  digitalWrite(LED, LOW);

  // Set serial monitor console termination for 'No line ending'
  Serial.begin(9600);
  Serial.println("Lab 1: hello arduino v0.2\n");
  delay(5000);
}

void loop() {
  if (Serial.available()) {
    inChar = Serial.read();
    Serial.print("Serial input detected: ");
    Serial.println(inChar);
  }
  if (inChar == 'n') digitalWrite(LED, HIGH); // oN
  if (inChar == 'f') digitalWrite(LED, LOW);  // oFf

  if (inChar == 'b') { // blink with 25% duty cycle
    digitalWrite(LED, HIGH);
    delay(250);
    digitalWrite(LED, LOW);
    delay(750);
  }

  // Discover 't' persistence bug by observing high rate LED blink
  if (inChar == 't') { // toggle
    digitalWrite(LED, !digitalRead(LED));
  }

} // loop()
```

What does this code do?

This code adds a blink option. Type ‘b’ to blink LED. It also add the ability to toggle the state of the LED by typing ‘t’. The code “**`digitalWrite(LED, !digitalRead(LED));`**” first reads the current state of the LED pin “**`digitalRead(LED)`**” and the writes the LED pin to the opposite value “**`!digitalRead(LED)`**”. The exclamation point symbol is the same as saying a logical NOT statement.

Procedure:

Copy the code from the above text box and paste it into the Arduino IDE, compile it and upload to the Arduino UNO. Click on the upload icon or use the keyboard shortcut CTRL+U. Try the blink function first by typing ‘b’ and then hit return.

Next try turning the LED on with’n’. Then try toggle ‘t’. What does the LED do in toggle mode? Is this what you expected? Compare the LED brightness in ‘t’ mode to when the LED is turned on using ‘n’.

The LED is dim in ‘t’ mode because the inChar is ‘t’ and it remains ‘t’ everytime through the loop until you type something new. So if inChar is ‘t’ then everytime through the loop, the LED is switched from on to off, or off to on. So it is on about 50% of the time. That is why the LED appears dim compared to when it is simply turned on.

SECTION 5 – Fixing the bug

To fix the ‘t’ bug, the code has to be modified so that it only changes the LED state if a new character has been typed (a fresh character). The code in the next textbox implements this using ‘T’ as the command.

```
//Lab_1_hello_arduino
#define LED 13
char inChar;
boolean isFreshInChar;

void setup() {
  pinMode(LED, OUTPUT);
  digitalWrite(LED, LOW);

  // Set serial monitor console termination for 'No line ending'
  Serial.begin(9600);
  Serial.println("Lab 1: hello arduino v0.3\n");
  delay(5000);
}

void loop() {
  isFreshInChar = false;
  if (Serial.available()) {
    inChar = Serial.read();
    Serial.print("Serial input detected: ");
    Serial.println(inChar);
    isFreshInChar = true;
  }

  if (inChar == 'n') digitalWrite(LED, HIGH); // oN
  if (inChar == 'f') digitalWrite(LED, LOW);  // oFf

  if (inChar == 'b') { // blink with 25% duty cycle
    digitalWrite(LED, HIGH);
    delay(250);
    digitalWrite(LED, LOW);
    delay(750);
  }

  // Discover 't' persistence bug by observing high rate LED blink
  if (inChar == 't') { // toggle
    digitalWrite(LED, !digitalRead(LED));
  }

  // Add state change detection to get proper toggle action.
  if (inChar == 'T') { // toggle
    if (isFreshInChar) digitalWrite(LED, !digitalRead(LED));
  }

} // loop()
```

What does this code do?

This code defines a new Boolean variable that acts a flag to indicate if new character has just been typed. The Boolean flag is "**`isFreshInChar"`** and it is set to false at the start of the loop. If new serial data is available, then the flag is set to true. At the end of the code, if the "**`inChar == 'T'`**" then before toggling the LED, the code first checks to see if the 'T' is a fresh character, if so, then write the LED pin to the opposite of what it current is (toggle its state).

Procedure:

Copy the code from the above text box and paste it into the Arduino IDE, compile it and upload to the Arduino UNO. Click on the upload icon or use the keyboard shortcut CTRL+U. Try the blink function first by typing ‘b’ and then hit return.

Next try toggling the state of the LED by repeatedly typing ‘T’ and observing the LED turn on and off at your command. Compare this to typing ‘t’ and note how it is different.

SECTION 6 – Creating a function call to make the code more readable

It takes several lines of code to blink the LED off and on. As the code becomes more complex, it helps to use function calls, e.g. for the blink routine.

```
//Lab_1_hello_arduino
#define LED 13
char inChar;
boolean isFreshInChar;

void setup() {
  pinMode(LED, OUTPUT);
  digitalWrite(LED, LOW);

  // Set serial monitor console termination for 'No line ending'
  Serial.begin(9600);
  Serial.println("Lab 1: hello arduino v0.4\n");
  delay(5000);
}

void loop() {
  isFreshInChar = false;
  if (Serial.available()) {
    inChar = Serial.read();
    Serial.print("Serial input detected: ");
    Serial.println(inChar);
    isFreshInChar = true;
  }

  if (inChar == 'n') digitalWrite(LED, HIGH); // oN
  if (inChar == 'f') digitalWrite(LED, LOW);  // oFf

  if (inChar == 'b') { // blink with 25% duty cycle
    digitalWrite(LED, HIGH);
    delay(250);
    digitalWrite(LED, LOW);
    delay(750);
  }

  // function for readability: blink with 75% duty cycle
  if (inChar == 'B') blinkLED(1000,750);

  // Discover 't' persistence bug by observing high rate LED blink
  if (inChar == 't') { // toggle
    digitalWrite(LED, !digitalRead(LED));
  }

  // Add state change detection to get proper toggle action.
  if (inChar == 'T') { // toggle
    if (isFreshInChar) digitalWrite(LED, !digitalRead(LED));
  }

} // loop()

//********************************************************************************
void blinkLED(unsigned int msecT, unsigned int msecOn) {
  digitalWrite(LED, HIGH);
  delay(msecOn);
  digitalWrite(LED, LOW);
  delay(msecT - msecOn);
}
```

What does this code do?

At the bottom of the code, outside of the loop(), a function called "**`blinkLED()`**" is defined. This function blinks the LED on and off. The function receives two parameters. The first parameter is

the total period for the blink, say 1000 milliseconds. The second parameter is the number of milliseconds that the LED should be on, say 750 milliseconds. Both of these parameters are unsigned integer data types containing non-negative numbers.

The function "**`blinkLED()`**" is called in the main loop code when you type 'B'

" **`if (inChar == 'B') blinkLED(1000,750);`**"

Procedure:

Upload the code and try entering 'B'. Observe the light blinking. Enter 'b' and compare how the blink is different. Now the change the code so that the first parameter in the call to blinkLED is 500 and the second is 200.
Upload the code, run it, and type 'B'. Observe the change in the LED pattern.

Now change the code so that the first parameter is 750 and the second is 1000.

" **`if (inChar == 'B') blinkLED(750,1000);`**"

Upload the run the code.

First, type 'n' to turn the LED on.

Now type 'f' to turn it off.

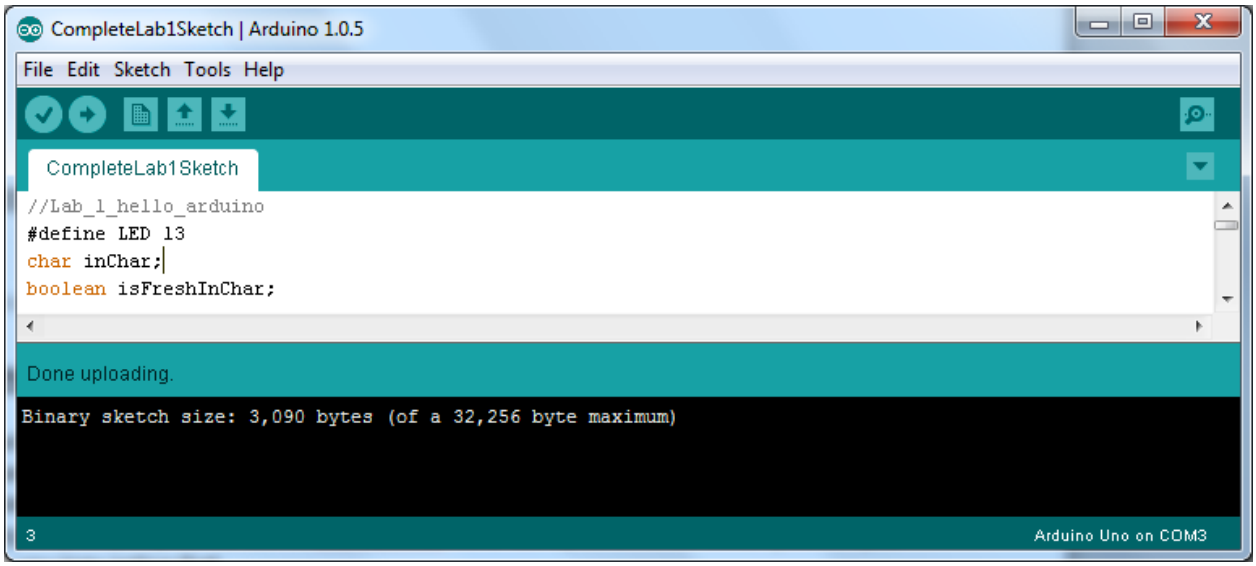
Now type 'B' and observe what happens. What frequency does the light blink at? What if you type 'b' what happens? Now type 'n' to turn the LED on. What happens? Why do get the results that you are getting? What is going on?

For the parameters you entered into the blinkLED function, what is the delay value that is calculated in the expression: "**`delay(msecT - msecOn);`**" ? You have found out what a runtime error can do to your program. How could this be avoided?

SECTION 6 – Microcontroller Memory Resources

The Atmel Mega328 chip that is used in the Arduino board has three types of memory, RAM, FLASH, and EEPROM. The RAM is used to execute programs and holds temporary variables for calculation and control. Only 2k bytes of RAM are available, which may be a lot or a little depending on your perspective. Some microcontrollers have 128 bytes of RAM. When you compile the code, you may notice that in the black status window, there is a message about the sketch size. E.g. 3,090 bytes in this case. This number represents the amount of FLASH memory used. There is no indication of RAM usage. Since RAM is in short supply, it is preferable to have information stored in FLASH where possible. The text that you print to the screen e.g.
"`Serial.println("Lab 1: hello arduino v0.4\n");`" is by default stored in RAM. This can quickly create a problem if you have a large menu system. Deciding how much memory you are using and where program and variables are stored in memory is a significant part of microcontroller system design.

To show how you can influence this, modify the print statement as follows so that the string that you print is stored in FLASH, not RAM. "`Serial.println(F("Lab 1: hello arduino v0.4\n"));`" Recompile the code and notice how the number of bytes has changed. What is the number of additional bytes of FLASH used? Note that less RAM will be needed after this change. In future labs, serial.println statement should store the string values in FLASH `Serial.println(F("…`



Back to the 'B' Problem

The function blinkLED could have made sure that any parameters given to it are valid. Enter the following code into your function, upload and retry the steps above that got you into trouble. Does the system behave differently when called with the parameters
" **`if (inChar == 'B') blinkLED(750,1000)`**" ?

As you just experienced, it is very important to check the bounds of any input parameters to your functions. This should be done in the function itself. Note that the compiler did not find this error, and the program ran normally until the function was called.

```
...

//*************************************************************************
void blinkLED(unsigned int msecT, unsigned int msecOn) {
  // Check for valid input parameters, if not valid, fix them so they are
  if (msecT < 2) msecT = 2;  // need at least 2 msec to blink with msec resolution
  if (msecOn >= msecT) msecOn = msecT/2; // 50% if ill posed

  digitalWrite(LED, HIGH);
  delay(msecOn);
  digitalWrite(LED, LOW);
  delay(msecT - msecOn);
}
...
```

Write Up:

There is no write up for this lab, just get the items on the cover sheet (next page) signed off by your lab instructor and submit just the cover sheet in class.

Cover Sheet: Lab #1 Intro to Arduino Microprocessor

Date: _____

Names: _____

Section: _____

LABORATORY CHECK OFFS	
Number of LED's determined	____
LED blinking with 50 mSec delay	____
Turn on and off LED with 'n' and 'f'	____
`if (inChar == 'B') blinkLED(750,1000);`	____
Number of additional bytes of Flash used with Serial.println(F())	____
Points	____