

Arduino Programming Part II

ME 120

Mechanical and Materials Engineering

Portland State University

<http://web.cecs.pdx.edu/~me120>

Overview

Review of Blink

Variable Declarations

Variable Assignments

Built-in I/O functions

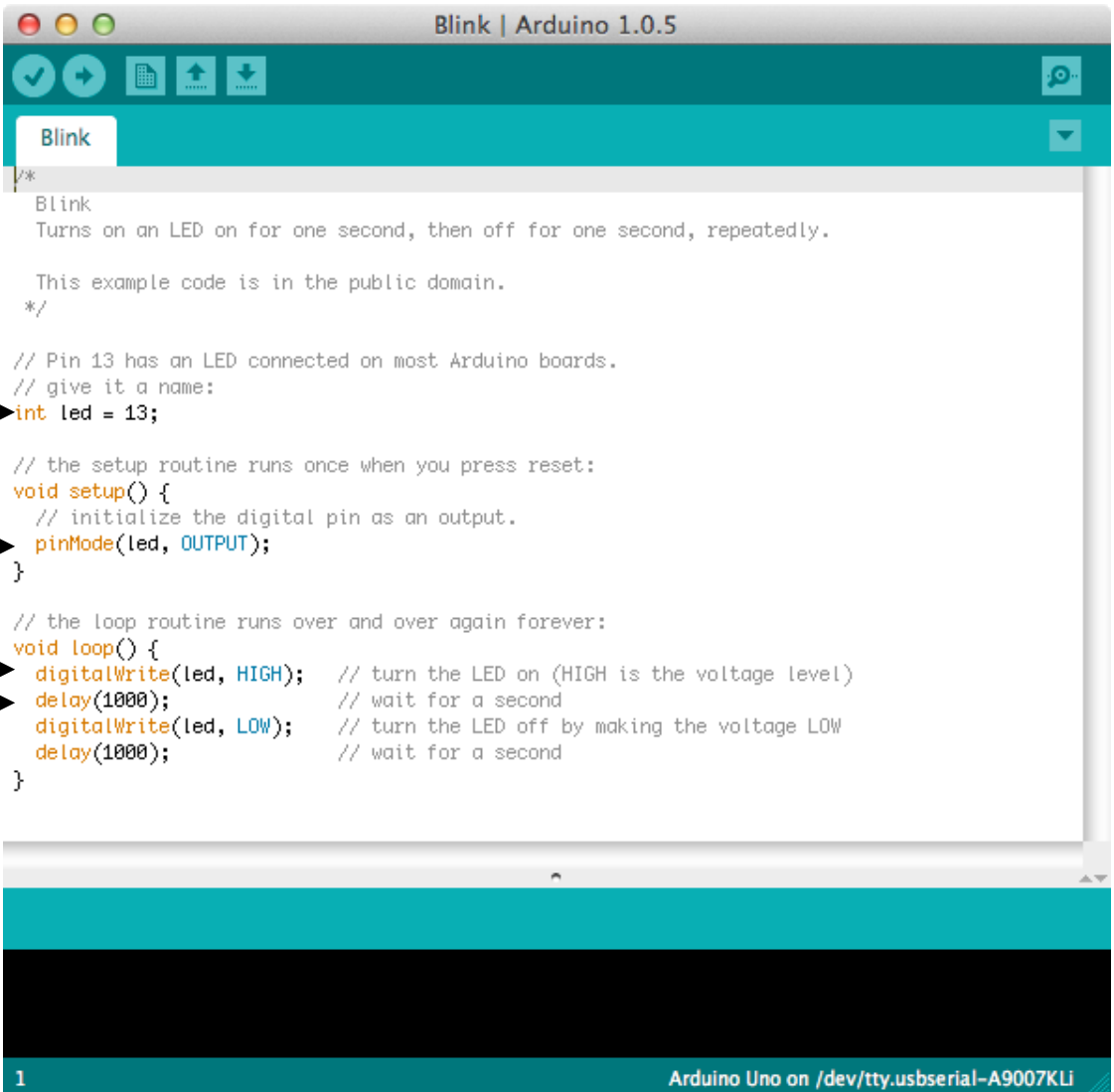
See on-line reference:

<http://arduino.cc/en/Reference/HomePage>

Blink code

Declare `led`
and assign a
value

Built-in functions:
`pinMode`
`digitalWrite`
`delay`



```
/*
 * 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
}
```

Variables in Arduino programs

Using Variables and Functions

Assigning values to a variable: “`int`” is a type of variable

```
int led = 12;
```

`pinMode` and `digitalWrite` expect “`int`” variables as inputs

```
pinMode (led, OUTPUT) ;
```

```
digitalWrite (led, HIGH) ;
```

`OUTPUT` and `HIGH` are pre-defined constants

See <http://arduino.cc/en/Reference/Constants>

Variable types

Three basic categories of variables

- ❖ integers
- ❖ floating point values
- ❖ character strings

Integers

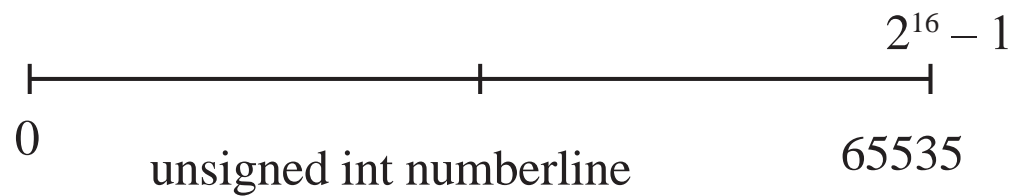
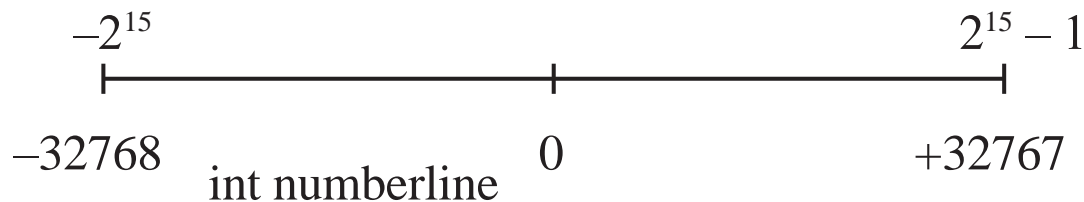
- ❖ No fractional part. Examples: 1, 2, 23, 0, -50213
- ❖ Used for counting and return values from some built-in functions
- ❖ Integer arithmetic results in truncation to integers

Floating point numbers

- ❖ Non-zero fractional parts. Examples 1.234, -2.728, 4.329×10^{-4}
- ❖ Large range of magnitudes
- ❖ Floating point arithmetic does not truncate, but has round-off

Integer types

<code>int</code>	integer in the range $-32,768$ to $32,767$
<code>long</code>	integer in the range $-2,147,483,648$ to $2,147,483,647$
<code>unsigned int</code>	positive integer in the range 0 to $65,535$
<code>unsigned long</code>	positive integer in the range 0 to $4,294,967,295$



See <http://arduino.cc/en/Reference/Int> and <http://arduino.cc/en/Reference/Long>

Floating point types

float values with approximately seven significant digits in the range $\pm(1.80 \times 10^{-38}$ to $3.40 \times 10^{38})$

double values with approximately thirteen significant digits in the range $\pm(2.2 \times 10^{-308}$ to $1.80 \times 10^{308})$

There is no double on an Arduino Uno. On an Uno, a double is the same as a float.

See <http://arduino.cc/en/Reference/Float>
and <http://arduino.cc/en/Reference/Double>

Declaring and assigning values

Declarations are necessary. Assignments are optional

```
int n;           // single declaration
int i,j,k,n;    // multiple declaration
int i=5;        // single declaration and assignment
int i=5, j=2;   // multiple declaration and assignment

float x;
float x,y,z;
float x=0.0, y=-1.23e5; // assignment with "e" notation
```

Notes

- ❖ Integer values do not use decimal points
- ❖ Floating point values can use “e” notation
 - 1.23e5 is equal to 1.23×10^5
 - DO NOT write $x = 1.23 * 10^5$ instead of $x = 1.23e5$

See <http://arduino.cc/en/Reference/Float>
and <http://arduino.cc/en/Reference/Double>

Assigning values

The equals sign is the *assignment operator*

- ❖ The statement `x = 3` assigns a value of 3 to x. The actual operation involves storing the value 3 in the memory location that is reserved for x.
- ❖ The equals sign does not mean that x and 3 are the same!
- ❖ Symbolically you can replace `x = 3` with `x ← 3`.

Consider the following sequence of statements

```
x = 3;  
y = x;  
x = 5;
```

The preceding statements are executed in sequence. The last assignment determines the value stored in x. There is no ambiguity in two “`x =`” statements. The `x = 5;` statement replaces the 3 stored in x with a new value, 5.

Test your understanding

What are the values of n and z at the end of the following sequences of statements?

```
int i,j,k,n;
```

```
i = 2;  
j = 3;  
k = i + 2*j;  
n = k - 5;
```

n = ?

```
int i,j,k,n;
```

```
i = 2;  
j = 3;  
n = j - i;  
n = n + 2;
```

n = ?

```
int n;  
float x,y,z;
```

```
x = 2.0;  
y = 3.0;  
z = y/x;  
n = z;
```

z = ?

n = ?

Test your understanding

What are the values of n and z at the end of the following sequences of statements?

```
int i,j,k,n;
```

```
i = 2;  
j = 3;  
k = i + 2*j;  
n = k - 5;
```

```
int i,j,k,n;
```

```
i = 2;  
j = 3;  
n = j - i;  
n = n + 2;
```

```
int n;  
float x,y,z;
```

```
x = 2.0;  
y = 3.0;  
z = y/x;  
n = z;
```

The `n = n + 2;` statement shows why it is helpful to think of the equal sign as a left facing arrow.

You can mentally replace `n = n + 2;` with `n ← n + 2;`

Integer arithmetic

We have to be aware of the rules of numerical computation used by Arduino hardware (and computers, in general).

Integer arithmetic always produces integers

```
int i, j;  
i = (2/3)*4;  
j = i + 2;
```

What values are stored in i and j?

Integer arithmetic

We have to be aware of the rules of numerical computation used by Arduino hardware (and computers, in general).

Integer arithmetic always produces integers

```
int i, j;  
i = (2/3)*4;  
j = i + 2;
```

What values are stored in i and j?

Answer: $i \leftarrow 0$, $j \leftarrow 2$

Integer arithmetic

Integer arithmetic always produces integers

```
int i,j;  
i = (2.0/3.0)*4.0;  
j = i + 2;
```

What values are stored in i and j?

Answer: $i \leftarrow 2$, $j \leftarrow 4$

Floating point arithmetic

Floating point arithmetic preserves the fractional part of numbers, but it does so approximately

```
float w,x,y,z;  
w = 3.0;  
x = 2.0;  
y = w/x;  
z = y - 1.5;
```

What values are stored in y and z?

Floating point arithmetic

Floating point arithmetic preserves the fractional part of numbers, but it does so approximately

```
float w,x,y,z;  
w = 3.0;  
x = 2.0;  
y = w/x;  
z = y - 1.5;
```

What values are stored in y and z?

Answer: $y \leftarrow 1.5$, $z \leftarrow 0$

Floating point arithmetic

Consider this alternate test*

```
float w,x,y,z;  
w = 4.0/3.0;  
x = w - 1;  
y = 3*x;  
z = 1 - y;
```

*See, e.g. C. Moler, *Numerical Computing in MATLAB*, 2004, SIAM, p. 38

Floating point arithmetic

Consider this alternate test*

```
float w,x,y,z;  
w = 4.0/3.0;  
x = w - 1;  
y = 3*x;  
z = 1 - y;
```

which produces $x = 0.333$ and $y = 1.000$ and $z = -1.19e-7$

*See, e.g. C. Moler, *Numerical Computing in MATLAB*, 2004, SIAM, p. 38

Global and local variables

In this sketch, `LED_pin` is a global variable, accessible to other functions in the file

```
int LED_pin = 13;
```

```
void setup() {  
  pinMode( LED_pin, OUTPUT );  
}  
  
void loop() {  
  digitalWrite( LED_pin, HIGH );  
  delay(1000);  
  digitalWrite( LED_pin, LOW );  
  delay(1000);  
}
```

In this sketch, `LED_pin` is a local variable in the `setup` function, and is not accessible to the code in the `loop` function. *This sketch will not compile. It cannot be run.*

```
void setup() {  
  int LED_pin = 13;  
  pinMode( LED_pin, OUTPUT );  
}  
  
void loop() {  
  digitalWrite( LED_pin, HIGH );  
  delay(1000);  
  digitalWrite( LED_pin, LOW );  
  delay(1000);  
}
```

In general, it is wise to avoid global variables unless you must. Since `LED_pin` must be accessible to `setup` and `loop`, it has to be a global variable.

Codes to demonstrate integer and floating point arithmetic

Serial communication with host computer (1)

Serial.begin(speed)

- ❖ Initializes the Serial port at speed. Typical speed is 9600

Serial.print(value)

- ❖ Sends **value** to the serial port
- ❖ **value** can be a single number or a character string
- ❖ *No newline* after **value** is sent

Serial.println(value)

- ❖ Sends **value** to the serial port
- ❖ **value** can be a single number or a character string
- ❖ *Add a newline* after **value** is sent

Integer arithmetic

```
// File: int_test.pde
//
// Demonstrate truncation with integer arithmetic
// ME 120, Lecture 5, Fall 2013

void setup() {
  int i,j;

  Serial.begin(9600);
  delay(3500);          // wait for user to open the serial monitor

  // -- First example: slide #13
  i = (2/3)*4;
  j = i + 2;
  Serial.println("First test");
  Serial.print(i);   Serial.print(" ");   Serial.println(j);

  // -- Second example: slide #15
  i = (2.0/3.0)*4.0;
  j = i + 2;
  Serial.println("Second test");
  Serial.print(i);   Serial.print(" ");   Serial.println(j);
}

void loop() {}        // Loop does nothing. Code in setup() is executed only once
```

Floating point arithmetic: test 1

```
// File: float_test.pde
//
// Demonstrate well-known round-off error problem with floating point arithmetic
// See, e.g., Cleve Moler, Numerical Computing in MATLAB, p. 38

void setup() {
  float w,x,y,z;

  Serial.begin(9600);
  delay(3500);          // wait for user to open the serial monitor

  // -- Computations that return results that you would expect; No rounding
  w = 3.0;
  x = 2.0;
  y = w/x;
  z = y - 1.5;
  Serial.println("First test");
  Serial.print(w,8); Serial.print(" "); // Spec # of digits w/ 2nd argument of print
  Serial.print(x,8); Serial.print(" ");
  Serial.print(y,8); Serial.print(" ");
  Serial.print(z,8); Serial.print(" ");
  Serial.println(z*1.0e7,8);
}

void loop() {}        // Loop does nothing. Code in setup() is executed only once
```


Floating point arithmetic: test 2

```
// File: float_test.pde
//
// Demonstrate well-known round-off error problem with floating point arithmetic
// See, e.g., Cleve Moler, Numerical Computing in MATLAB, p. 38

void setup() {
  float w,x,y,z;

  Serial.begin(9600);
  delay(3500);          // wait for user to open the serial monitor

  // -- Computations that show rounding
  w = 4.0/3.0;
  x = w - 1;
  y = 3*x;
  z = 1 - y;
  Serial.println("\nSecond test");
  Serial.print(w,8);  Serial.print(" ");
  Serial.print(x,8);  Serial.print(" ");
  Serial.print(y,8);  Serial.print(" ");
  Serial.print(z,8);  Serial.print(" ");
  Serial.println(z*1.0e7,8);
}

void loop() {}        // Loop does nothing. Code in setup() is executed only once
```

Built-in Arduino functions

All sketches have `setup()` and `loop()`

`void setup()`

- ❖ Executed only once
- ❖ No input arguments: parentheses are empty
- ❖ No return values: function type is `void`

`void loop()`

- ❖ Executed repeatedly
- ❖ No input arguments: parenthesis are empty
- ❖ No return values: function type is `void`

Digital input and output (1)

Digital I/O pins 0 through 13 can respond to input or be sources of output

pinMode(pin, mode)

- ❖ Configures a digital I/O pin for input or output
- ❖ pin – specifies the digital I/O channel: 0 to 13
- ❖ mode – one of: INPUT, OUTPUT or INPUT_PULLUP
 - we use OUTPUT to set the pin as a power source for an LED
 - we use INPUT when we read a digital input, such as a button
- ❖ No return value: function type is **void**

Digital input and output (2)

digitalWrite (pin, value)

- ❖ Sets the state of a digital I/O pin
- ❖ pin – specifies the digital I/O channel: 0 to 13
- ❖ value – one of: HIGH or LOW
- ❖ No return value: function type is **void**

digitalRead (pin)

- ❖ Reads the state of a digital I/O pin
- ❖ pin – specifies the digital I/O channel: 0 to 13
- ❖ Returns an int that is equivalent to either LOW or HIGH

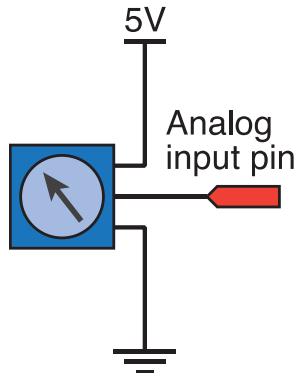
See <http://arduino.cc/en/Reference/DigitalWrite>
and <http://arduino.cc/en/Reference/DigitalRead>
and <http://arduino.cc/en/Tutorial/DigitalPins>

Analog input

`analogRead(pin)`

- ❖ Reads the voltage on an analog input pin
- ❖ `pin` – an integer that specifies the analog input channel: 0 to 5.
`pin` can also be referred to by name as A0, A1, A2, A3, A4 or A5
- ❖ Returns an `int` in the range 0 to 1023 (for an Arduino Uno)

Example: Read a potentiometer



```
void setup() {  
    Serial.begin(9600);  
}  
  
void loop() {  
    int reading;  
    reading = analogRead(A0);  
    Serial.println(reading);  
}
```