Al-Mustaqbal University

College Of Engineering & Technology

Department of Computer Engineering Techniques (Stage: 3)

Digital Control

Lecture 2

Introduction to Arduino

Dr.: Fanar Ali Joda

Variables

A variable is a way of naming and storing a numerical value for later use by the program. As their namesake suggests, variables are numbers that can be continually changed as opposed to constants whose value never changes. A variable needs to be declared and optionally assigned to the value needing to be stored. The following code declares a variable called inputVariable and then assigns it the value obtained on analog input pin 2:

```
int inputVariable = 0;        // declares a variable and
                              // assigns value of 0
inputVariable = analogRead(2); // set variable to value of
                               // analog pin 2
```

‘inputVariable’ is the variable itself. The first line declares that it will contain an int, short for integer. The second line sets the variable to the value at analog pin 2. This makes the value of pin 2 accessible elsewhere in the code.

Once a variable has been assigned, or re-assigned, you can test its value to see if it meets certain conditions, or you can use its value directly. As an example to illustrate three useful operations with variables, the following code tests whether the inputVariable is less than 100, if true it assigns the value 100 to inputVariable, and then sets a delay based on inputVariable which is now a minimum of 100:

```
if (inputVariable < 100) // tests variable if less than 100

{

  inputVariable = 100;   // if true assigns value of 100

}

delay(inputVariable);    // uses variable as delay
```

Note: Variables should be given descriptive names, to make the code more readable. Variable names like tiltSensor or pushButton help the programmer and anyone else reading the code to understand what the variable represents. Variable names like var or value, on the other hand, do little to make the code readable and are only used here as examples. A variable can be named any word that is not already one of the keywords in the Arduino language.

Variable declaration

All variables have to be declared before they can be used. Declaring a variable means defining its value type, as in int, long, float, etc., setting a specified name, and optionally assigning an initial value.

```
int inputVariable = 0;
```

byte

Byte stores an 8-bit numerical value without decimal points. They have a range of 0-255.

```
byte someVariable = 180;   // declares 'someVariable'
                           // as a byte type
```

int

Integers are the primary datatype for storage of numbers without decimal points and store a 16-bit value with a range of 32,767 to -32,768.

```
int someVariable = 1500;   // declares 'someVariable'
                           // as an integer type
```

Note: Integer variables will roll over if forced past their maximum or minimum values by an assignment or comparison. For example, if x = 32767 and a subsequent statement adds 1 to x, x = x + 1 or x++, x will then rollover and equal -32,768.

long

Extended size datatype for long integers, without decimal points, stored in a 32-bit value with a range of 2,147,483,647 to -2,147,483,648.

```
long someVariable = 90000; // declares 'someVariable'
                           // as a long type
```

float

A datatype for floating-point numbers, or numbers that have a decimal point. Floating- point numbers have greater resolution than integers and are stored as a 32-bit value with a range of 3.4028235E+38 to -3.4028235E+38.

```
float someVariable = 3.14; // declares 'someVariable'
                           // as a floating-point type
```

arrays

An array is a collection of values that are accessed with an index number. Any value in the array may be called upon by calling the name of the array and the index number of the value. Arrays are zero indexed, with the first value in the array beginning at index number 0. An array needs to be declared and optionally assigned values before they can be used.

```
int myArray[] = {value0, value1, value2...}
```

Likewise it is possible to declare an array by declaring the array type and size and later assign values to an index position:

```
 int myArray[5];    // declares integer array w/ 6 positions
myArray[3] = 10;   // assigns the 4th index the value 10
```

To retrieve a value from an array, assign a variable to the array and index position:

```
x = myArray[3];    // x now equals 10
```

Arrays are often used in for loops, where the increment counter is also used as the index position for each array value. The following example uses an array to flicker an LED. Using a for loop, the counter begins at 0, writes the value contained at index position 0 in the array flicker[], in this case 180, to the PWM pin 10, pauses for 200ms, then moves to the next index position.

```
int ledPin = 10;                    // LED on pin 10
byte flicker[] = {180, 30, 255, 200, 10, 90, 150, 60};
                                    // above array of 8
void setup()                        // different values
{
  pinMode(ledPin, OUTPUT);          // sets OUTPUT pin
}
void loop()
{
  for(int i=0; i<7; i++)            // loop equals number
  {                                 // of values in array
    analogWrite(ledPin, flicker[i]); // write index value
    delay(200);                     // pause 200ms
  }
}
```

arithmetic

Arithmetic operators include addition, subtraction, multiplication, and division. They return the sum, difference, product, or quotient (respectively) of two operands.

y = y + 3;

x = x - 7;

i = j * 6;

r = r / 5;

compound assignments

Compound assignments combine an arithmetic operation with a variable assignment. These are commonly found in for loops as described later. The most common compound assignments include:

x ++   // same as x = x + 1, or increments x by +1

x -- // same as x = x - 1, or decrements x by -1

x += y   // same as x = x + y, or increments x by +y

x -= y   // same as x = x - y, or decrements x by -y

x *= y   // same as x = x * y, or multiplies x by y

x /= y   // same as x = x / y, or divides x by y

comparison operators

Comparisons of one variable or constant against another are often used in if statements to test if a specified condition is true. In the examples found on the following pages, ?? is used to indicate any of the following conditions:

```
x == y    // x is equal to y

x != y    // x is not equal to y

x <  y    // x is less than y

x >  y    // x is greater than y

x <= y    // x is less than or equal to y

x >= y    // x is greater than or equal to y
```

logical operators

Logical operators are usually a way to compare two expressions and return a TRUE or FALSE depending on the operator. There are three logical operators, AND, OR, and NOT, that are often used in if statements:

Logical AND:

```
if (x > 0 && x < 5)    // true only if both

                 // expressions are true
```

Logical OR:

```
if (x > 0 || y > 0)    // true if either

                 // expression is true
```

Logical NOT:

```
if (!x > 0)          // true only if

                 // expression is false
```

true/false

These are Boolean constants that define logic levels. FALSE is easily defined as 0 (zero) while TRUE is often defined as 1, but can also be anything else except zero. So in a Boolean sense, -1, 2, and -200 are all also defined as TRUE.

```
if (b == TRUE);
{
  doSomething;
}
```

high/low

These constants define pin levels as HIGH or LOW and are used when reading or writing to digital pins. HIGH is defined as logic level 1, ON, or 5 volts while LOW is logic level 0, OFF, or 0 volts.

```
digitalWrite(13, HIGH);
```

input/output

Constants used with the pinMode() function to define the mode of a digital pin as either INPUT or OUTPUT.

```
pinMode(13, OUTPUT);
```