Al-Mustaqbal University

College Of Engineering & Technology

Department of Computer Engineering Techniques (Stage: 3)

Digital Control

Lecture 3

Introduction to Arduino

Dr.: Fanar Ali Joda

if

if statements test whether a certain condition has been reached, such as an analog value being above a certain number, and executes any statements inside the brackets if the statement is true. If false the program skips over the statement. The format for an if test is:

```
if (someVariable ?? value)
{
  doSomething;
}
```

The above example compares some Variable to another value, which can be either a variable or constant. If the comparison, or condition in parentheses is true, the statements inside the brackets

are run. If not, the program skips over them and continues on after the brackets.

if… else

if… else allows for 'either-or' decisions to be made. For example, if you wanted to test a digital input, and do one thing if the input went HIGH or instead do another thing if the input was LOW, you would write that this way:

```
if (inputPin == HIGH)
{
  doThingA;
}
else
{
  doThingB;
}
```

else can also precede another if test, so that multiple, mutually exclusive tests can be run at the same time. It is even possible to have an unlimited number of these else branches. Remember though, only one set of statements will be run depending on the condition tests:

```
 if (inputPin < 500)
{
  doThingA;
```

```
}
else if (inputPin >= 1000)
{
  doThingB;
}
else
{
  doThingC;
}
```

for

The for statement is used to repeat a block of statements enclosed in curly braces a specified number of times. An increment counter is often used to increment and terminate the loop. There are three parts, separated by semicolons (;), to the for loop header:

```
for (initialization; condition; expression)
{
  doSomething;
}
```

The initialization of a local variable, or increment counter, happens first and only once. Each time through the loop, the following condition is tested. If the condition remains true, the following statements and expression are executed and the condition is tested again. When the condition becomes false, the loop ends.

The following example starts the integer i at 0, tests to see if i is still less than 20 and if true, increments i by 1 and executes the enclosed statements:

```
for (int i=0; i<20; i++)  // declares i, tests if less
{                         // than 20, increments i by 1
  digitalWrite(13, HIGH); // turns pin 13 on
  delay(250);             // pauses for 1/4 second
  digitalWrite(13, LOW);  // turns pin 13 off
  delay(250);             // pauses for 1/4 second
}
```

while

while loops will loop continuously, and infinitely, until the expression inside the parenthesis becomes false. Something must change the tested variable, or the while loop will never exit. This could be in your code, such as an incremented variable, or an external condition, such as testing a sensor.

```
while (someVariable ?? value)
{
  doSomething;
}
```

The following example tests whether 'someVariable' is less than 200 and if true executes the statements inside the brackets and will continue looping until 'someVariable' is no longer less than 200.

```
while (someVariable < 200) // tests if less than 200
{
  doSomething;          // executes enclosed statements
  someVariable++;       // increments variable by 1
}
```

do… while

The do loop is a bottom driven loop that works in the same manner as the while loop, with the exception that the condition is tested at the end of the loop, so the do loop will always run at least once.

```
do
{
  doSomething;
} while (someVariable ?? value);
```

The following example assigns readSensors() to the variable ‘x’, pauses for 50 milliseconds, then loops indefinitely until ‘x’ is no longer less than 100:

```
do
{
  x = readSensors();    // assigns the value of
                        // readSensors() to x
  delay (50);           // pauses 50 milliseconds
} while (x < 100);      // loops if x is less than 100
```

pinMode(pin, mode)

Used in void setup() to configure a specified pin to behave either as an INPUT or an OUTPUT.

pinMode(pin, OUTPUT); // sets 'pin' to output

Arduino digital pins default to inputs, so they don't need to be explicitly declared as inputs with pinMode(). Pins configured as INPUT are said to be in a high-impedance state.

There are also convenient 20KΩ pullup resistors built into the Atmega chip that can be accessed from software. These built-in pullup resistors are accessed in the following manner:

pinMode(pin, INPUT); // set 'pin' to input

digitalWrite(pin, HIGH); // turn on pullup resistors

digitalRead(pin)

Reads the value from a specified digital pin with the result either HIGH or LOW. The pin can be specified as either a variable or constant (0-13).

value = digitalRead(Pin); // sets 'value' equal to

// the input pin

digitalWrite(pin, value)

Outputs either logic level HIGH or LOW at (turns on or off) a specified digital pin. The pin can be specified as either a variable or constant (0-13).

```
digitalWrite(pin, HIGH);   // sets 'pin' to high
```

The following example reads a pushbutton connected to a digital input and turns on an LED connected to a digital output when the button has been pressed:

```
int led   = 13;   // connect LED to pin 13
int pin   =  7;   // connect pushbutton to pin 7
int value =  0;   // variable to store the read value
 void setup()
{
  pinMode(led, OUTPUT);    // sets pin 13 as output
  pinMode(pin, INPUT);     // sets pin 7 as input
}
void loop()
{
  value = digitalRead(pin);  // sets 'value' equal to
                             // the input pin
  digitalWrite(led, value);  // sets 'led' to the
}                            // button's value
```

analogRead(pin)

Reads the value from a specified analog pin with a 10-bit resolution. This function only works on the analog in pins (0-5). The resulting integer values range from 0 to 1023.

value = analogRead(pin); // sets 'value' equal to 'pin'

analogWrite(pin, value)

Writes a pseudo-analog value using hardware enabled pulse width modulation (PWM) to an output pin marked PWM. On newer Arduinos with the ATmega168 chip, this function works on pins 3, 5, 6, 9, 10, and 11. Older Arduinos with an ATmega8 only support pins 9, 10, and 11. The value can be specified as a variable or constant with a value from 0-255.

analogWrite(pin, value); // writes 'value' to analog 'pin'

A value of 0 generates a steady 0 volts output at the specified pin; a value of 255 generates a steady 5 volts output at the specified pin. For values in between 0 and 255, the pin rapidly alternates between 0 and 5 volts - the higher the value, the more often the pin is HIGH (5 volts). For example, a value of 64 will be 0 volts three-quarters of the time, and 5 volts one quarter of the time; a value of 128 will be at 0 half the time and 255 half the time; and a value of 192 will be 0 volts one quarter of the time and 5 volts three-quarters of the time.

Because this is a hardware function, the pin will generate a steady wave after a call to analogWrite in the background until the next call to analogWrite (or a call to digitalRead or digitalWrite on the same pin).

Note: Analog pins unlike digital ones, do not need to be first declared as INPUT nor OUTPUT.

The following example reads an analog value from an analog input pin, converts the value by dividing by 4, and outputs a PWM signal on a PWM pin:

```
int led = 10;    // LED with 220 resistor on pin 10
int pin =  0;    // potentiometer on analog pin 0
int value;       // value for reading
void setup(){}   // no setup needed
void loop()
{
  value =  analogRead(pin);  // sets 'value' equal to 'pin'
  value /= 4;                // converts 0-1023 to 0-255
  analogWrite(led, value);   // outputs PWM signal to led
}
```

delay(ms)

Pauses a program for the amount of time as specified in milliseconds, where 1000 equals 1 second.

```
delay(1000);   // waits for one second
```

Serial.begin(rate)

Opens serial port and sets the baud rate for serial data transmission. The typical baud rate for communicating with the computer is 9600 although other speeds are supported.

```
void setup()
{
  Serial.begin(9600);   // opens serial port
```

} // sets data rate to 9600 bps

Note: When using serial communication, digital pins 0 (RX) and 1 (TX) cannot be used at the same time.

Serial.println(data)

Prints data to the serial port, followed by an automatic carriage return and line feed.

This command takes the same form as Serial.print(), but is easier for reading data on the Serial Monitor.

Serial.println(analogValue);// sends the value of 'analogValue'

Note: For more information on the various permutations of the Serial.println() and Serial.print() functions please refer to the Arduino website. The following simple example takes a reading from analog pin0 and sends this data to the computer every 1 second.

```
void setup()
{
  Serial.begin(9600);          // sets serial to 9600bps
}
void loop()
{
  Serial.println(analogRead(0)); // sends analog value
  delay(1000);                 // pauses for 1 second
}
```