Second Lecture

Arduino Programming

By:

Ayub Othman Abdulrahman

University Of Halabja

Introduction

- The Arduino language is a modified of the C++ programming language, but it uses built-in libraries to simplify complex coding jobs to make it easier for users to pick up

Declaring Variables

- **#define** is a useful C component that allows the programmer to give a name to a constant value before the program is compiled. Defined constants in Arduino don't take up any program memory space on the chip. The compiler will replace references to these constants with the defined value at compile time.

How to use #define

- #define ledPin 3 // The compiler will replace any mention of ledPin with the value 3 at compile time.
- **Notes and Warnings**
- There is no semicolon after the #define statement. If you include one, the compiler will throw cryptic errors further down the page.
- #define ledPin 3; // this is an error
- Similarly, including an equal sign after the #define statement will also generate a cryptic compiler error further down the page.
- #define ledPin = 3 // this is also an error

Integers

- Integers are your primary data-type for number storage.
- Syntax
- int var = val;
- var - your int variable name
- val - the value you assign to that variable

const

- The const keyword stands for constant. It is a variable *qualifier* that modifies the behavior of the variable, making a variable "*read-only*". This means that the variable can be used just as any other variable of its type, but its value cannot be changed.

const

- const float pi = 3.14;
- float x;
- x = pi * 2; // it's fine to use consts in math
- pi = 7; // illegal - you can't write to (modify)
- **#define or const**
- You can use either const or #define for creating numeric or string constants. In general const is preferred over #define for defining constants.

Int VS Const int

- Basically, an int is read/write and a const int is read only. You use a const int for something that never changes value, such as a pin number. The advantage is less memory is used.
- RAM is very limited on the AVRs so it is best to const when using variables to hold read-only values to save ram space.

Unsigned int

- On the Uno and other ATMEGA based boards, unsigned ints (unsigned integers) are the same as ints in that they store a 2 byte value. Instead of storing negative numbers however they only store positive values, yielding a useful range of 0 to 65,535 ((2^16) - 1).
- **Example Code**
- unsigned int var= 6;

Unsigned int

- When unsigned variables are made to exceed their maximum capacity they "roll over" back to 0, and also the other way around:
- unsigned int x;
- x = 0;
- x = x - 1; // x now contains 65535 - rolls over in neg direction
- x = x + 1; // x now contains 0 - rolls over

void

- The void keyword is used only in function declarations. It indicates that the function is expected to return no information to the function from which it was called.

```
void setup()
{ // ...
}
void loop()
{ // ...
}
```

Constants

- Constants are predefined expressions in the Arduino language. They are used to make the programs easier to read. We classify constants in groups:
- Defining Logical Levels: true and false (Boolean Constants)
- Defining Pin Levels: HIGH and LOW

Boolean Constants

- There are two constants used to represent truth and falsity in the Arduino language: true, and false.
- false
- false is the easier of the two to define. false is defined as 0 (zero).
- true
- true is often said to be defined as 1, which is correct, but true has a wider definition. Any integer which is non-zero is true, in a Boolean sense. So -1, 2 and -200 are all defined as true, too, in a Boolean sense.

HIGH and LOW

- When reading or writing to a digital pin there are only two possible values a pin can take/be-set-to: HIGH and LOW.
- HIGH
- The meaning of HIGH (in reference to a pin) is somewhat different depending on whether a pin is set to an INPUT or OUTPUT. When a pin is configured as an INPUT, a voltage greater than 3.0V is present at the pin
- When a pin is configured to OUTPUT, the pin is at: 5 volts
- In this state it can source current, e.g. light an LED that is connected through a series resistor to ground.

HIGH and LOW

- LOW
- The meaning of LOW also has a different meaning depending on whether a pin is set to INPUT or OUTPUT. When a pin is configured as an INPUT, a voltage less than 1.5V is present at the pin (5V boards)
- When a pin is configured to OUTPUT, the pin is at 0 volts (both 5V and 3.3V boards).
- In this state it can sink current, e.g. light an LED that is connected through a series resistor to +5 volts (or +3.3 volts).

#include

- #include is used to include outside libraries in your sketch. This gives the programmer access to a large group of standard C libraries (groups of pre-made functions), and also libraries written especially for Arduino.
- #include <Library_Name.h>
- #include “Library_Name.h”

Structure {setup() }

- The setup() function is called when a sketch starts. Use it to initialize variables, pin modes, start using libraries, etc. The setup() function will only run once, after each powered up or reset of the Arduino board.
- **Example Code**
- int buttonPin = 3;
- void setup(){
- Serial.begin(9600);
- pinMode(buttonPin, INPUT);}

Structure {loop()}

- The loop() function does precisely what its name suggests, and loops consecutively, allowing your program to change and respond. Use it to actively control the Arduino board.
- void loop()
- {
- //
- }

Control Structure

- To control statements
- Break
- Continue
- Goto
- Switch Case
- And all types of loop
  - For loop
  - While loop
  - Do … while

Break

- break is used to exit from a for, while or do… while loop, bypassing the normal loop condition. It is also used to exit from a switch case statement.
- break;

Continue

- The continue statement skips the rest of the current iteration of a loop (for, while, do…while). It continues by checking the conditional expression of the loop, and proceeding with any subsequent iterations.

```
1.  for (int x = 0; x <= 255; x ++)  {
2.      if (x > 40 && x < 120){     // create jump in values
3.          continue;  }
4.      Serial.println(x);
5.      delay(100); }
```

Goto

- Transfers program flow to a labeled point in the program
- Syntax
- Label:
- goto label; // sends program flow to the label

```
1.  for(int r = 0; r < 255; r++){
2.  for(int b = 0; b < 255; b++){
3.   if (analogRead(0) > 250){ goto bailout;}
4.   // more statements ...
5.  } } }
6.   bailout:
```

Switch Case

- Like if statements, switch case controls the flow of programs by allowing programmers to specify different code that should be executed in various conditions. In particular, a switch statement compares the value of a variable to the values specified in case statements. When a case statement is found whose value matches that of the variable, the code in that case statement is run.

Switch Case

- The break keyword exits the switch statement, and is typically used at the end of each case. Without a break statement, the switch statement will continue executing the following expressions.

- Parameters
- var: a variable whose value to compare with various cases. **Allowed data types:** int, char
- label1, label2: constants.

Switch Case (Example Code)

```
switch (var) {
    case 1:   // label1
      //do something when var equals 1
      break;
    case 2:
      //do something when var equals 2
      break;
    default:
      // if nothing else matches, do the default
      // default is optional
      break;
  }
```

FUNCTIONS

- For controlling the Arduino board and performing computations.
- Digital I/O {pinMode(), digitalRead(), digitalWrite()}
- Analog I/O {analogRead(), analogWrite()}
- Time {delay() }
- Math {abs(), pow(), sq(), sqrt() }
- Communication {Serial, Serial.begin(), Serial.end(), Serial.available(), Serial.read(), Serial.write(), Serial.print()

pinMode()

- Configures the specified pin to behave either as an input or an output.

- Syntax
  - pinMode(pin, mode)
- Parameters
  - pin: the number of the pin
  - mode: INPUT, OUTPUT

digitalRead()

- Reads the value from a specified digital pin, either HIGH or LOW.
- The analog input pins can be used as digital pins, referred to as A0, A1, etc.

- Syntax
  - digitalRead(pin)
- Parameters
  - pin: the number of the digital pin you want to read
- Returns
  - HIGH or LOW

digitalWrite()

- Write a HIGH or a LOW value to a digital pin.
- If the pin has been configured as an OUTPUT with pinMode(), its voltage will be set to the corresponding value: 5V for HIGH, 0V (ground) for LOW.

- Syntax
  - digitalWrite(pin, value)
- Parameters
  - pin: the pin number
  - value: HIGH or LOW

digitalRead(), digitalWrite() {Example Code}

```
1.  int ledPin = 13;   // LED connected to digital pin 13
2.  int inPin = 7;     // pushbutton connected to digital pin 7
3.  int val ;       // variable to store the read value

4.  void setup()
5.  {
6.    pinMode(ledPin, OUTPUT);      // sets the digital pin 13 as output
7.    pinMode(inPin, INPUT);        // sets the digital pin 7 as input
8.  }
9.  void loop()
10. {
11.   val = digitalRead(inPin);     // read the input pin
12.   digitalWrite(ledPin, val);    // sets the LED to the button's value
13. }
```

analogRead()

- Reads the value from the specified analog pin. The Arduino board contains a 6 channel, 10-bit analog to digital converter. This means that it will map input voltages between 0 and 5 volts into integer values between 0 and 1023. This yields a resolution between readings of: 5 volts / 1024 units or, .0049 volts (4.9 mV) per unit.

analogRead()

- Syntax
  - analogRead(pin)
- Parameters
  - pin: the number of the analog input pin to read from (0 to 5 on most boards, 0 to 6 on MKR boards, 0 to 7 on the Mini and Nano, 0 to 15 on the Mega)
- Returns
  - int(0 to 1023)

analogRead() {Example Code}

```
1.  int analogPin = 3;  // potentiometer wiper connected to analog pin 3
2.  int val ;           // variable to store the value read

3.  void setup()
4.  {
5.    Serial.begin(9600);           //  setup serial
6.  }

7.  void loop()
8.  {
9.    val = analogRead(analogPin);    // read the input pin
10.   Serial.println(val);            // debug value
11. }
```

analogWrite()

- Writes an analog value (PWM wave) to a pin. Can be used to light a LED at varying brightnesses or drive a motor at various speeds. After a call to analogWrite(), the pin will generate a steady square wave of the specified duty cycle until the next call to analogWrite() (or a call to digitalRead() or digitalWrite()) on the same pin.

analogWrite()

- On most Arduino boards (those with the ATmega168 or ATmega328P), this function works on pins 3, 5, 6, 9, 10, and 11.
- You do not need to call pinMode() to set the pin as an output before calling analogWrite().
- The analogWrite function has nothing to do with the analog pins or the analogRead function.

analogWrite()

- Syntax
- analogWrite(pin, value)
- Parameters
- pin: the pin to write to. Allowed data types: int.
- value: the duty cycle: between 0 (always off) and 255 (always on).

- Notes and Warnings
- The PWM outputs generated on pins 5 and 6 will have higher-than-expected duty cycles. This is because of interactions with the millis() and delay() functions, which share the same internal timer used to generate those PWM outputs. This will be noticed mostly on low duty-cycle settings (e.g. 0 - 10) and may result in a value of 0 not fully turning off the output on pins 5 and 6.

analogWrite() {Example Code}

```
1.  int ledPin = 9;      // LED connected to digital pin 9
2.  int analogPin = 3;   // potentiometer connected to analog pin 3
3.  int val = 0;         // variable to store the read value

4.  void setup()
5.  {
6.    pinMode(ledPin, OUTPUT);   // sets the pin as output
7.  }

8.  void loop()
9.  {
10.   val = analogRead(analogPin);   // read the input pin
11.   analogWrite(ledPin, val / 4);  // analogRead values go from 0 to 1023,
      analogWrite values from 0 to 255
12. }
```

delay()

- Pauses the program for the amount of time (in milliseconds) specified as parameter. (There are 1000 milliseconds in a second.)
- Syntax
  - delay(ms)
- Parameters
  - ms: the number of milliseconds to pause (unsigned long)

abs()

- Calculates the absolute value of a number.

- Syntax
  - abs(x)
- Parameters
  - x: the number

- Notes and Warnings
- Because of the way the abs() function is implemented, avoid using other functions inside the brackets, it may lead to incorrect results.
- abs(a++); // avoid this - yields incorrect results

pow()

- Calculates the value of a number raised to a power. Pow() can be used to raise a number to a fractional power. This is useful for generating exponential mapping of values or curves.
- Syntax
- pow(base, exponent)
- Parameters
- base: the base number
- exponent: the power to which the base is raised

sq()

- Calculates the square of a number: the number multiplied by itself.

- Syntax
  - sq(x)
- Parameters
  - x: the number, any data type
- Returns
  - The square of the number.

sqrt()

- Calculates the square root of a number.
  - Syntax
    - sqrt(x)
  - Parameters
    - x: the number, any data type
  - Returns
    - The number's square root. (double)

Serial

- Used for communication between the Arduino board and a computer or other devices. All Arduino boards have at least one serial port (also known as a UART or USART): It communicates on digital pins 0 (RX) and 1 (TX) as well as with the computer via USB. Thus, if you use these functions, you cannot also use pins 0 and 1 for digital input or output.

Serial

- You can use the Arduino environment's built-in serial monitor to communicate with an Arduino board. Click the serial monitor button in the toolbar and select the same baud rate used in the call to begin().

Serial.begin()

- Sets the data rate in bits per second (baud) for serial data transmission. For communicating with the computer, use one of these rates: 300, 600, 1200, 2400, 4800, 9600, etc. You can, however, specify other rates.
- Serial.begin(9600); //opens serial port, sets data rate to 9600 bps

Serial.end()

- Disables serial communication, allowing the RX and TX pins to be used for general input and output. To re-enable serial communication, call Serial.begin()
- Syntax
- Serial.end()

Serial.available()

- Get the number of bytes (characters) available for reading from the serial port. This is data that's already arrived and stored in the serial receive buffer (which holds 64 bytes). available() inherits from the Stream utility class.
- Syntax
- Serial.available()

Serial.read()

- Reads incoming serial data.
  - Syntax
    - Serial.read()

- There are a couple of ways to go about it.
- You could read a byte (that's what Serial.read returns; not an int)
- Or, you could read all the available data, and store it in an array of characters.

Serial.read(),Serial.available() {Example Code}

```
1. int incomingByte = 0;          // for incoming serial data
2. void setup() {
3. Serial.begin(9600); }// opens serial port, sets data rate to 9600 bps

4. void loop() {

5. if (Serial.available() > 0) {      // reply only when you receive data:
6. incomingByte = Serial.read();    // read the incoming byte:
7. Serial.print("I received: "); // say what you got:
8. Serial.println(incomingByte, DEC);
9. }}
```

Serial.write()

- Writes binary data to the serial port. This data is sent as a byte or series of bytes; to send the characters representing the digits of a number use the print() function instead.
- Syntax
- Serial.write(val)
- Serial.write(str)
- Serial.write(buf, len)
- Parameters
- val: a value to send as a single byte
- str: a string to send as a series of bytes
- buf: an array to send as a series of bytes

Serial.print()

- Prints data to the serial port as human-readable ASCII text. This command can take many forms. Numbers are printed using an ASCII character for each digit. Floats are similarly printed as ASCII digits, defaulting to two decimal places. Bytes are sent as a single character. Characters and strings are sent as is. For example-
- Serial.print(78) gives "78"
- Serial.print(1.23456) gives "1.23"
- Serial.print('N') gives "N"
- Serial.print("Hello world.") gives "Hello world."

Serial.print()

- An optional second parameter specifies the base (format) to use; permitted values are BIN(binary, or base 2), OCT(octal, or base 8), DEC(decimal, or base 10), HEX(hexadecimal, or base 16). For floating point numbers, this parameter specifies the number of decimal places to use. For example-
- Serial.print(78, BIN) gives "1001110”
- Serial.print(78, OCT) gives "116”
- Serial.print(78, DEC) gives "78”
- Serial.print(78, HEX) gives "4E”
- Serial.print(1.23456, 0) gives "1”
- Serial.print(1.23456, 2) gives "1.23”
- Serial.print(1.23456, 4) gives "1.2346"

Thank You

End