

420-N23-LA Introduction to IOT

Arduino Programming – Part 3

Arduino Programming Part 3

Arrays

Arrays

■ Initialization of an Array

- Standard without element specification
 - `int myArray[]; //this array will hold integers`
 - `dogs myArray[]; // this array will hold dogs`
- Specify Data
 - `dogs myArray[] = {spot, pluto, clifford, ruff};`
 - `dogs myArray[4] = {spot, pluto, clifford, ruff};`
- A multi-dimensional array
 - `int b[2][2] = { { 1, 2 }, { 3, 4 } };`
 - `int b[2][2] = { { 1 }, { 3, 4 } };`

Arduino Arrays

- Use a CONSTANT when defining an array

- `Const int arraySize = 10;`

- Then define the array:

- `int temps[arraySize] = { 87, 68, 94, 100, 83, 78, 85, 91, 76, 87 };`

- Usage:

- Using the array is standard for C based languages:

- `int x = temps[5] // Gets the 6th element of the array.`

Passing an array to a function:

■ Passing an array to a function:

- A standard way to include an array in a function is to pass the array along with the SIZE of the array (as an integer).
- `void doSomething(int b[], int sizeOfArray) { ... }`
- For Example:

```
void modifyArray( int b[], int sizeOfArray )  
{ // multiply each array element by 2  
  for ( int k = 0 ; k < sizeOfArray ; ++k )  
    b[ k ] *= 2;  
}
```

Calculate the Array Size

- To calculate the size programmatically, you cannot just get number of elements in the array (or sizeof) the array.
- Each element takes up a predetermined size based on its type.
- Example, an “int” takes up 2 bytes per value. An array of 10 ints would be 20 bytes long. The same “long” array would take up 40 bytes.

Calculate the Array Size

You can find the length of an array (number of elements in the array) by dividing the size of the array by the size of each element .

$$\frac{\text{array_size}}{\text{element_size}}$$

```
int arr[] = {1,2,3};  
int arraysize = sizeof(arr) / sizeof(arr[0]);
```

Sizeof with Objects

- Nothing changes, same formula. An object is a reference value which is a huge 16 bytes on the Arduino.

```
Flasher f1(4, 10,200);  
Flasher f2(5, 100,200);  
Flasher f3(6, 1000,200);  
Flasher fa[3] = {f1, f2, f3};
```

```
Serial.println(sizeof(fa)); // 48  
Serial.println(sizeof(fa[0])); // 16  
Serial.println(sizeof(fa)/sizeof(fa[0])); // 3
```

Calculating Size for a 2D Array

- Make the distinction between rows and columns, see the example here;

```
int ary[][5]
    = { {1, 2, 3, 4, 5},
        {6, 7, 8, 9, 0}};

// 2 rows
int rows
    = sizeof ary / sizeof ary[0];

// 5 cols
int cols
    = sizeof ary[0] / sizeof(int);
```

Arduino Programming Part 3

Classes and Libraries

Classes Introduction

- Classes can be added to your Arduino project to:
 - Reduce code duplication.
 - Reduce the size of your setup and loop methods.
 - Reuse your class in other projects.
- The classes are written similar to classes in generic C++ code.
- Two examples are given here;
 - One: A class within the “.ino” file.
 - Two: A class within a separate CPP class library.

Empty Class Example

The class and the Setup/Loop are all in the same file.

Class LED_CONTROLS

```
{  
    public:  
    void begin (in pin){  
        pinMode(ledPin, OUTPUT);  
    }  
    void on(ledPin){  
        digitalWrite ( ledPin, HIGH);  
    }  
    void off(ledPin){  
        digitalWrite ( ledPin, LOW);}  
};
```

LED_CONTROLS LED;

```
void Setup() {  
    LED.begin (LED_BUILTIN);  
}  
  
Void Loop()  
{  
    LED.on (LED_BUILTIN);  
    Delay(1000);  
    LED.off ((LED_BUILTIN);  
    Delay(1000);  
}
```

Class Library

- You need to create two files:
 - **A header file (with extension .h)**
 - This has definitions of the class within it. It lists everything in the class without the code that implements it.
 - Many programs require this file when including the source code.
 - **A source-code file (with extension cpp)**

Review an Example: Morse Code

```
int pin = 13;

void setup()
{
  pinMode(pin, OUTPUT);
}

void loop()
{
  dot(); dot(); dot();
  dash(); dash(); dash();
  dot(); dot(); dot();
  delay(3000);
}
```

```
void dot()
{
  digitalWrite(pin, HIGH);
  delay(250);
  digitalWrite(pin, LOW);
  delay(250);
}

void dash()
{
  digitalWrite(pin, HIGH);
  delay(1000);
  digitalWrite(pin, LOW);
  delay(250);
}
```

Turning the Sketch Into a Library!



- The sketch has a few different parts that we'll need to bring into our library.
 - The dot() and dash() functions that do the actual blinking.
 - The ledPin variable which the functions use to determine which pin to use.
 - Finally, there's the call to pinMode() that initializes the pin as an output.

Turning the Sketch Into a Library!

- As mentioned earlier, we need at least two files for a library:
 - **a header file (w/ the extension .h) and the source file (w/ extension .cpp).**
 - The header file has definitions for the library: basically a listing of everything that's inside;
 - **while the source file has the actual code.**
 - We'll call our library "Morse", so our header file will be Morse.h. Let's take a look at what goes in it. It might seem a bit strange at first, but it will make more sense once you see the source file that goes with it.

Turning the Sketch Into a Library: Header File (1)

- The core of the header file consists of a line for each function in the library, wrapped up in a class along with any variables you need.

```
class Morse
{
    public:
        Morse(int pin);
        void dot();
        void dash();
    private:
        int _pin;
};
```

Turning the Sketch Into a Library: Header File (2)

- Add `#include` statement that gives you access to the standard types and constants of the Arduino language (this is automatically added to normal sketches, but not to libraries). It looks like this

```
#include "Arduino.h"
```

- It goes above the class definition.

Turning the Sketch Into a Library: Header File (3)

- Finally, to wrap the whole header file up in this construct:

```
#ifndef Morse_h
#define Morse_h

// the #include statment and code
go here...

#endif
```

Morese.h Header File (.h)

```
#ifndef Morse_h
#define Morse_h
#include "Arduino.h"
class Morse
{
  public:
    Morse(int pin);
    void dot();
    void dash();
  private:
    int _pin;
};
#endif
```

- The “ifndef” statement makes sure this header isn’t included twice – just in case.
- Including “Arduino.h” gives this class access to Arduino functions. This is required for most Arduino applications.

Turning the Sketch Into a Library: cpp file

- First comes a couple of #include statements.
 - These give the rest of the code access to the standard Arduino functions, and to the definitions in the header file:

```
#include "Arduino.h"  
#include "Morse.h"
```

- Then comes the constructor:

```
Morse::Morse(int pin)  
{  
    pinMode(pin, OUTPUT);  
    _pin = pin;  
}
```

Turning the Sketch Into a Library: cpp file

- Morse::
 - This says that the function is part of the Morse class.
- The underscore in the name private variable, `_pin`.
 - this variable can actually have any name you want, as long as it matches the definition in the header file.
 - Adding an underscore to the start of the name is a common convention to make it clear which variables are private, and also to distinguish the name from that of the argument to the function (pin in this case).

Turning the Sketch Into a Library: cpp file

- Next comes the actual code from the sketch that you're turning into a library. It looks pretty much the same, except with `Morse::` in front of the names of the functions and `_pin` instead of `pin`:

```
void Morse::dot()
{
    digitalWrite(_pin, HIGH);
    delay(250);
    digitalWrite(_pin, LOW);
    delay(250);
}

void Morse::dash()
{
    digitalWrite(_pin, HIGH);
    delay(1000);
    digitalWrite(_pin, LOW);
    delay(250);
}
```

The Source Code File (.cpp)

```
#include "Arduino.h"
#include "Morse.h"
```

References to
header files

```
Morse::Morse(int pin)
{
    pinMode(pin, OUTPUT);
    _pin = pin;
}
```

Constructor

```
void Morse::dot()
{
    digitalWrite(_pin, HIGH);
    delay(250);
    digitalWrite(_pin, LOW);
    delay(250);
}
```

Methods

```
void Morse::dash()
{
    digitalWrite(_pin, HIGH);
    delay(1000);
    digitalWrite(_pin, LOW);
    delay(250);
}
```

Add Library to Arduino (or Sketches) Folder

- You can add this as a global importable file for all Arduino project.
- Examples
 - D:\code\Arduino\libraries
 - C:\Program Files (x86)\Arduino\libraries
- Create a directory for your project (don't just put it in the libraries folder).

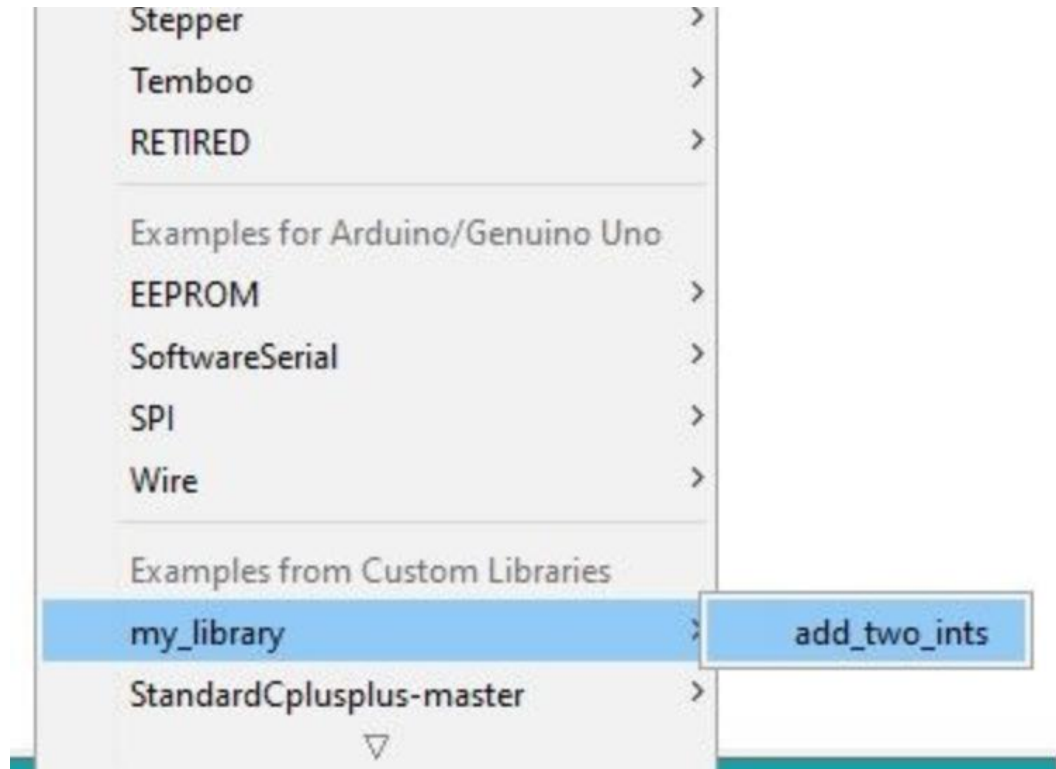
```
1. Documents/  
2.   └─ Arduino/  
3.     └─ libraries/  
4.         └─ my_library/  
5.             └─ my_library.h  
6.             └─ my_library.cpp
```

Adding Examples

- Adding an INO file showing BASIC usage of your class is very helpful.
- Just create an INO file here. It will be accessible in the examples folder.

```
1. Documents/  
2.   └─ Arduino/  
3.     └─ libraries/  
4.        └─ examples/  
5.           └─ blink_led/  
6.              └─ blink_led.ino  
7.     └─ Led/  
8.        └─ Led.h  
9.           └─ Led.cpp
```

- Library Access



Turning the Sketch Into a Library: cpp file

- Next comes the actual code from the sketch that you're turning into a library. It looks pretty much the same, except with Morse:: in front of the names of the functions and pin instead of pin:

```
#include <Morse.h>
Morse morse(13);
void setup() {
}
void loop() {
  morse.dot(); morse.dot(); morse.dot();
  morse.dash(); morse.dash(); morse.dash();
  morse.dot(); morse.dot(); morse.dot();
  delay(3000);
}
```

- **name** - the name of the library. Library names must contain only basic letters (A - Z or a - z) and numbers (0 - 9), spaces (), underscores (_), dots (.) and dashes (-). It cannot start or end with a space, and also it cannot start with a number.
- **version** - version of the library. Version should be [semver](#) compliant. 1.2.0 is correct; 1.2 is accepted; r5, 003, 1.1c are invalid
- **author** - name/nickname of the authors and their email addresses (not mandatory) separated by comma ","
- **maintainer** - name and email of the maintainer
- **sentence** - a sentence explaining the purpose of the library
- **paragraph** - a longer description of the library. The value of **sentence** always will be prepended, so you should start by writing the second sentence here
- **category** - (defaults to `uncategorized`) if present, one of these:
 - Display
 - Communication
 - Signal Input/Output
 - Sensors
 - Device Control
 - Timing
 - Data Storage
 - Data Processing
 - Other
- **url** - the URL of the library project, for a person to visit. For example, the library's GitHub page
- **architectures** - (defaults to `*`) a comma separated list of architectures supported by the library. If the library doesn't contain architecture specific code use `*` to match all architectures
- **dot_a_linkage** - (available from IDE 1.6.0 / `arduino-builder 1.0.0-beta13`) (optional) when set to `true` , the library will be compiled using a `.a` (archive) file. First, all source files are compiled into `.o` files as normal. Then instead of including all `.o` files in the linker command directly, all `.o` files are saved into a `.a` file, which is then included in the linker command. [1.5 format library folder structure](#) is required.
- **includes** - (available from IDE 1.6.10) (optional) a comma separated list of files to be added to the sketch as `#include <...>` lines. This property is used with the "Include library" command in the IDE. If the property is undefined all the headers files (`.h`) on the root source folder are included.
- **precompiled** - (feature not yet released, will be available in `arduino-builder >1.3.25`) (optional) set to `true` to allow the use of `.a` (archive) and `.so` (shared object) files. The `.a/.so` file must be located at `src/{build.mcu}` where `{build.mcu}` is the architecture name of the target the file was compiled for. Ex: `cortex-m3` for the Arduino DUE. The static library should be linked as an `ldflag`.
- **ldflags** - (feature not yet released, will be available in `arduino-builder >1.3.25`) (optional) the linker flags to be added. Ex: `ldflags=-lm`



<https://www.devdungeon.com/content/arduino-libraries-tutorial>

<https://www.teachmicro.com/create-arduino-library/>

<http://www.arduino.cc/en/Guide/Libraries>

More Information on Libraries