Object Oriented Programming

By Julie VK3FOWL and Joe VK3YSP
www.sarcnet.org

What is OOP?

Object-Oriented Programming (OOP) is a methodology for analysing, designing and implementing computer software by first declaring a common class of functions, or "methods", and their associated data elements, or "attributes". One or more objects can then be created, or "instantiated", based on this class. Each object then inherits the behaviour of the class. For more on Object Oriented Programming using Arduino or Raspberry Pi see or OOP Workshop at: https://www.sarcnet.org/workshops.html.

Why use OOP?

1. **Architectural Reasons**
    a. **Abstraction** – Only public methods and attributes are exposed to the user: A user can create high-level objects simply by including a free class library in their programs. A developer can fully complete the top-level object-oriented design of a program while deferring the detailed workings of the objects for later development.
    b. **Encapsulation** - All private methods and attributes are hidden from the user: A user does not have to know complex inner workings of an object to use it. A developer can protect the object's internals from inadvertent access.
    c. **Inheritance** – Every object in a class has the same methods and attributes: A user can rely on every single object functioning the same way. A developer need only to declare and implement each of the class methods and attributes once.
    d. **Polymorphism** – The same method can perform an operation on different data types or in different situations. A user need not worry about using the same method for different applications. A developer can create multiple methods with the same name to handle different situations.
2. **Practical Reasons**
    a. **Modularity** – All the object's methods and attributes are defined by the one class.
    b. **Reusability** – The same class can be used in many different applications, even in other classes.
    c. **Maintainability** – The class can be easily checked for errors and fixes are localised to that class.
    d. **Flexibility** – The class often provides all the possible methods and attributes for using the object.
    e. **Extensibility** – The class provides a framework for all new object methods and attributes.
    f. **Portability** – Class libraries can be easily published, distributed and included.
    g. **Hierarchical** – Programs can be designed and built from the top down, by deferring complexity.

A Side-By-Side Comparison of Coding Methods

Example: A simple program to control 4 LEDs. First setup the I/O pins to control 4 LEDs then repeat the following forever: Turn all LEDs on. Wait one second. Turn all LEDs off. Wait one second. Flash all LEDs in a sequence.

C on Arduino

```
const int led1 = 2;
const int led2 = 3;
const int led3 = 4;
const int led4 = 5;
void setup() {
   pinMode(led1, OUTPUT);
   pinMode(led2, OUTPUT);
   pinMode(led3, OUTPUT);
   pinMode(led4, OUTPUT);
   digitalWrite(led1, LOW);
   digitalWrite(led2, LOW);
   digitalWrite(led3, LOW);
   digitalWrite(led4, LOW);
}
void loop() {
   digitalWrite(led1, HIGH);
   digitalWrite(led2, HIGH);
   digitalWrite(led3, HIGH);
   digitalWrite(led4, HIGH);
   delay(1000);
   digitalWrite(led1, LOW);
   digitalWrite(led2, LOW);
   digitalWrite(led3, LOW);
   digitalWrite(led4, LOW);
   delay(1000);
   digitalWrite(led1, HIGH);
   delay(100);
   digitalWrite(led1, LOW);
   delay(100);
   digitalWrite(led2, HIGH);
   delay(100);
   digitalWrite(led2, LOW);
   delay(100);
   digitalWrite(led3, HIGH);
   delay(100);
   digitalWrite(led3, LOW);
   delay(100);
   digitalWrite(led4, HIGH);
   delay(100);
   digitalWrite(led4, LOW);
   delay(100);
}
```

C++ on Arduino

```
class Led{
   public:
      Led(int pin);
      void on();
      void off();
      void flash(int onTime, int
      offTime);
   private:
      int _pin;
};
Led::Led(int pin) {
   _pin = pin
   pinMode(_pin, OUTPUT);
   off();
}
void Led::on() {
   digitalWrite(_pin, HIGH);
}
void Led::off() {
   digitalWrite(_pin, LOW);
}
void Led::flash(int onTime, int
offTime) {
   on();
   delay(onTime);
   off();
   delay(offTime);
}
Led led1(2),led2(3),led3(4),led4(5);
void setup() {
}
void loop() {
   led1.on();
   led2.on();
   led3.on();
   led4.on();
   delay(1000);
   led1.off();
   led2.off();
   led3.off();
   led4.off();
   delay(1000);
   led1.flash(100,100);
   led2.flash(100,100);
   led3.flash(100,100);
   led4.flash(100,100);
}
```

Python on Raspberry Pi

```
from RPi.GPIO import
setwarnings, setmode, setup,
output, BOARD, OUT, HIGH,
LOW
from time import sleep
class Led(object):
   def __init__(self,pin):
      self.__pin = pin #Note:
      __ means private
      setwarnings(False)
      setmode(BOARD)
      setup(self.__pin,OUT)
      self.off()
   def on(self):
      output(self.__pin,
      HIGH)
   def off(self):
      output(self.__pin, LOW)
   def
   flash(self,onTime,offTime
   ):
      self.on()
      sleep(onTime)
      self.off()
      sleep(offTime)
led1 = Led(12)
led2 = Led(16)
led3 = Led(18)
led4 = Led(22)
while True:
   led1.on()
   led2.on()
   led3.on()
   led4.on()
   sleep(1)
   led1.off()
   led2.off()
   led3.off()
   led4.off()
   sleep(1)
   led1.flash(0.1,0.1)
   led2.flash(0.1,0.1)
   led3.flash(0.1,0.1)
   led4.flash(0.1,0.1)
```

Object Oriented Programming in C++ using the Arduino IDE

Code	Steps for writing an object oriented C++ program

`class Led{`
`  public:`
`  Led(int pin);`
`  void on();`
`  void off();`
`  void flash(int onTime, int offTime);`
`  private:`
`  int _pin;`
`};`
	Step 3: Create your class, declaring its public and private methods and attributes.
	
	E.G. I'll create a Led class. I know the Led class must have a constructor method called "Led" and I want it to handle setting the led pin numbers.
	
	I already know what methods I want: on, off and flash. Each led object will need to remember its own pin number. So I'll create a private attribute called "pin" for that. Only the Led class methods will be able to use the pin number, because it is private.

`Led::Led(int pin) {`
`  _pin = pin;`
`  pinMode(_pin, OUTPUT);`
`  off()`
`}`
 	
`void Led::on() {`
`  digitalWrite(_pin, HIGH);`
`}`
 	
`void Led::off() {`
`  digitalWrite(_pin, LOW);`
`}`
 	
`void Led::flash(int onTime, int offTime) {`
`  on();`
`  delay(onTime);`
`  off();`
`  delay(offTime);`
`}`
	Step 4: Code the class methods. The good news is that you only have to do this once, no matter how many objects you create!
	
	All the other methods and attributes are immediately available to each method. So use them if you can. E.G. flash() uses both on() and off().
	
	Notes: Prefix the name of each method with the name of the class followed by two colons E.G. Led:: Note: This goes after the method's return type.
	
	The class constructor always has the same name as the class.
	
	The class constructor is special and does not need a type.
	
	That's it. Your Object Oriented C++ program is finished.

`Led led1(2),led2(3),led3(4),led4(5);`	Step 2: Create objects, based on your class. E.G. The led objects will be based on the Led class, of course (By convention, the class name always starts with a capital letter). When the objects are created I'll need to specify the pin number to use for each one. So the pin number will have to be a parameter.

`void setup() { }`
`void loop() {`
`  led1.on();`
`  led2.on();`
`  led3.on();`
`  led4.on();`
`  delay(1000);`
`  led1.off();`
`  led2.off();`
`  led3.off();`
`  led4.off();`
`  delay(1000);`
`  led1.flash(100,100);`
`  led2.flash(100,100);`
`  led3.flash(100,100);`
`  led4.flash(100,100);`
`}`
	Step 1: Create your main program
	
	Using your knowledge of how objects, methods and attributes work, simply assume they already exist and write down what you would like your main program to do. E.G. I want to have 4 LEDs all turn on, all turn off and then flash in sequence.
	
	I know: I'll use four led objects (led1 – led4) each with the following methods:
	on() – Makes LEDs go on
	off() – Makes LED go off
	flash() – This one will need on-time and off-time parameters

Creating and Installing Arduino C++ Libraries using the Arduino IDE

Code	Steps for creating and installing Arduino C++ libraries

//led.h - Arduino LED library
//Copyright (c) 2018 - Name
//Released under the GNU Public Licence
 	
#ifndef LED_H
#def LED_H
 	
class Led{
…
};
 	
#endif
	Step 1: Put your class in a separate header file.
	
	E.G.
	1. Create a new tab called "led.h" in the Arduino IDE
	2. Add your own copyright header
	3. Add the compiler directives to protect the file from being included multiple times.
	4. Cut and paste your class to that tab just above #endif

//led.cpp - Arduino LED library
//Copyright (c) 2018 - Name
//Released under the GNU Public Licence
 	
#include "led.h"
#include "arduino.h"
 	
Led::Led(){
…
}
 	
…
	Step 2: Put your class methods in a separate C++ file.
	
	E.G.
	1. Create a new tab called "led.cpp" in the Arduino IDE
	2. Add your own copyright header
	3. Add the compiler directive to include your header file
	4. Add the compiler directive to include arduino.h. Note this is automatically included in your Arduino sketch file, but not here.
	5. Cut and paste your class methods to that tab just under #include "arduino.h"

//led.ino – Test file for Arduino LED library
//Copyright (c) 2018 - Name
//Released under the GNU Public Licence
 	
#include "led.h"
 	
…
	Step 3: Update your sketch file.
	
	E.G.
	1. Add your own copyright header
	2. Add the compiler directive to include your header file
	3. Recompile to check everything still works ok.

#include <led.h>
 	
//led.ino – Test file for Arduino LED library
//Copyright (c) 2018 - Name
//Released under the GNU Public Licence
 	
Led led1(2),led2(3),led3(4),led4(5);
 	
void setup() { }
void loop() {
led1.on();
…
}
	Step 4: Create and install an Arduino library
	
	E.G.
	1. Delete #include "led.h" from your sketch
	2. Select Sketch | Show Sketch Folder
	3. Select and Copy the led.h and led.cpp files
	4. Open the parent folder and enter the "libraries" folder
	5. Create a new folder in libraries called "Led". Use a capital.
	6. Open the new Led folder
	7. Paste the copied led.h and led.cpp files into the Led folder
	8. Close and re-open the IDE to allow it to read the new library
	9. Select Sketch | Include Library | Led
	10. Check #include <led.h> is inserted at the top of your sketch
	11. Recompile to check everything still works ok.
	12. You have created a new Arduino library that others can use.
	13. Zip up the Led folder and send them a copy

```
//led.h - LED library.
//Copyright (c) 2018, Julie Gonzales VK3FOWL and Joe Gonzales VK3YSP.
//www.sarcnet.org
//Released under the GNU General Public License.
//Provides a flexible Led object with built-in Non-Blocking flash timer

#ifndef LED_H
#define LED_H
#include "timer.h"

class Led {
  public:
    Led(int pin);
    void on();
    void off();
    void state(bool state);
    void flash(long onTime, long offTime);
  private:
    int _pin;
    Timer _t1;
};
#endif

//led.cpp - LED library.
//Copyright (c) 2018, Julie Gonzales VK3FOWL and Joe Gonzales VK3YSP.
//Released under the GNU General Public License.
//Provides a flexible Led object with built-in Non-Blocking flash timer

#include "led.h"
#include "timer.h"
#include "arduino.h"

Led::Led(int pin){
  _pin = pin;
  pinMode(_pin, OUTPUT);
  digitalWrite(_pin, LOW);
}

void Led::on() {
  digitalWrite(_pin, HIGH);
}

void Led::off() {
  digitalWrite(_pin, LOW);
}

void Led::state(bool state) {
  if (state) {
    on();
  } else {
    off() ;
  }
}

void Led::flash(long onTime, long offTime) {
  long cycleTime = onTime + offTime;
  long cyclePoint = _t1.elapsed() % cycleTime;
  if (cyclePoint < onTime){
    on();
  } else {
    off();
  }
}
```

```
//switch.h - Switch library.
//Copyright (c) 2018, Julie Gonzales VK3FOWL and Joe Gonzales VK3YSP.
//www.sarcnet.org
//Released under the GNU General Public License.
//Provides a de-bounced, press-button or toggle switch object

#ifndef SWITCH_H
#define SWITCH_H
#define Press 0
#define Toggle 1
#define Pass 10

class Switch {
  public:
    Switch(int pin, int type);
    bool state();
  private:
    bool _switch();
    int _pin;
    int _type;
    bool _state;
    int _pass;
    int _count;
};
#endif

//switch.cpp - Switch library.
//Copyright (c) 2018, Julie Gonzales VK3FOWL and Joe Gonzales VK3YSP.
//www.sarcnet.org
//Released under the GNU General Public License.
//Provides a de-bounced, press-button or toggle switch object

#include "switch.h"
#include "arduino.h"

Switch::Switch(int pin, int type) {
  _pin = pin;
  _state = false;
  _type = type;
  _pass = 0;
  _count = 0;
  pinMode(_pin, INPUT_PULLUP);  //This provides an internal pullup for a switch to GND.
}

bool Switch::_switch() {
  return not digitalRead(_pin); //The switch state is the opposite of the pin state
}

bool Switch::state() {
  if (_type == Press) {
    _state = _switch();
    return _state;
  }
  if (_type == Toggle) {
    if (_switch() == true) {    //The switch is on
      if (_pass < Pass) {
        _pass++;
        if (_pass == Pass) {    //The threshold is reached for the first time
          _state = not _state;  //Toggle the state of the switch
          _count++;             //Increment the switch count
        }
      }
    } else {                    //The switch is off
      _pass = 0;                //Reset the pass counter
    }
  }
  return (_state);              //Return the current state of the switch
}
```

```
//filter.h - Filter library.
//Copyright (c) 2015-2018 Julie VK3FOWL and Joe VK3YSP
//www.sarcnet.org
//For more information please visit http://www.sarcnet.org
//Released under the GNU General Public License.
//Provides low pass filters for the sensors and motors

#ifndef FILTER_H
#define FILTER_H

class Filter {
  public:
    Filter(float Alpha);
    float lpf(float Value);
  private:
    float _last;
    float _alpha;
};

#endif

//filter.cpp - Filter library.
//Copyright (c) 2015-2018 Julie VK3FOWL and Joe VK3YSP
//www.sarcnet.org
//For more information please visit http://www.sarcnet.org
//Released under the GNU General Public License.
//Provides low pass filters for the sensors and motors

#include "filter.h"
#include "arduino.h"

//Public methods

Filter::Filter(float alpha) {
  //Constructor
  _alpha = alpha;
  _last = 0;
}

float Filter::lpf(float value) {
  //Low pass filter - Decrease alpha to increase damping factor
  float result = (_alpha * value) + _last * (1 - _alpha);
  _last = result;
  return result;
}
```

```
//timer.h - Timer library.
//Copyright (c) 2018, Julie Gonzales VK3FOWL and Joe Gonzales VK3YSP.
//www.sarcnet.org
//Released under the GNU General Public License.
//Provides a Non-Blocking timer object to replace the delay() function

#ifndef TIMER_H
#define TIMER_H
class Timer {
  public:
    Timer(void);
    Timer(long period);
    void reset();
    void reset(long period);
    long elapsed();
    long periods();
    bool timeout();
    bool toggle();
  private:
    long _period;
    long _start;
};
#endif

//timer.cpp - Timer library.
//Copyright (c) 2018, Julie Gonzales VK3FOWL and Joe Gonzales VK3YSP.
//www.sarcnet.org
//Released under the GNU General Public License.
//Provides a Non-Blocking timer object to replace the delay() function

#include "timer.h"
#include "arduino.h"

Timer::Timer(void) {
  reset();
}

Timer::Timer(long period) {
  _period = period;
  reset();
}

void Timer::reset() {
  _start = millis();
}

void Timer::reset(long period) {
  _period = period;
  _start = millis();
}

long Timer::elapsed() {
  return millis() - _start;
}

long Timer::periods() {
  return long(elapsed() / _period);
}

bool Timer::timeout() {
  return (elapsed() > _period);
}

bool Timer::toggle() {
  return (periods() % 2);
}
```

Simple Led, Switch and Timer Example

```
//oop.ino - Demonstration of Object oriented programming using Led, Switch and Timer library.
//Copyright (c) 2018, Julie Gonzales VK3FOWL and Joe Gonzales VK3YSP.
//www.sarcnet.org
//Released under the GNU General Public License.
//Provides a simple demonstration of the Led, Switch and Timer objects.

#include <led.h>
#include <switch.h>
#include <timer.h>

/*
This demonstration includes:
  Use of the Led, Switch and Timer library
  led1 responds to press button switch s1.
  led2 responds to toggle switch s2
  led3 toggles every 100ms with timer t1
  led4 toggles every 200ms with timer t2
  led5 flashes for 400ms on and 400ms off
  led6 flashes for 800ms on and 800ms off
  NO delay() functions allowed
*/

#include "led.h"
#include "switch.h"
#include "timer.h"

Led led1(2), led2(3), led3(4), led4(5), led5(6), led6(7);
Switch sw1(8, Press), sw2(9, Toggle);
Timer t1(100),t2(200);

void setup() {
}

void loop() {
  led1.state(sw1.state());
  led2.state(sw2.state());
  led3.state(t1.toggle());
  led4.state(t2.toggle());
  led5.flash(400,400);
  led6.flash(800,800);
}
```

Traffic Lights with a Pedestrian Crossing – Using a Finite State Machine Main Loop

```
//TrafficLightsDemo.ino - Demonstration of Led, Switch and Timer library.
//Copyright (c) 2018, Julie Gonzales VK3FOWL and Joe Gonzales VK3YSP.
//www.sarcnet.org
//Released under the GNU General Public License.
//Provides a more complex demo of the Led, Switch and Timer objects using a finite state machine.

#include <led.h>
#include <switch.h>
#include <timer.h>

/* Traffic Lights State Transition Table
  State Road1 Road2 Cross1  Cross2  Time  Input Next
  GRWD  G1    R2    W1      D2      10000 P2    GRFD
  GRFD  G1    R2    F1      D2      3000        GRDD
  GRDD  G1    R2    D1      D2      2000        YRDD
  YRDD  Y1    R2    D1      D2      2000        RRDDa
  RRDDa R1    R2    D1      D2      1000        RGDW
  RGDW  R1    G2    D1      W2      10000 P1    RGDF
  RGDF  R1    G2    D1      F2      3000        RGDD
  RGDD  R1    G2    D1      D2      2000        RYDD
  RYDD  R1    Y2    D1      D2      2000        RRDDb
  RRDDb R1    R2    D1      D2      1000        GRWD
*/

//Declare Traffic Light States
enum State {GRWD, GRFD, GRDD, YRDD, RRDDa, RGDW, RGDF, RGDD, RYDD, RRDDb};
State state = GRWD, lastState = RRDDb;

//Declare Red, Yellow, Green, Don't Walk and Walk LEDs for roads 1 and 2
Led R1(2), Y1(3), G1(4), R2(5), Y2(6), G2(7), D1(15), W1(14), D2(16), W2(10);

//Declare Pedestrian Crossing switches for roads 1 and 2
Switch P1(8, Press), P2(9, Press);

//Declare timer
Timer t1;

void off() {
  //Turn off all lights (just prior to turning four lights on)
  R1.off(); Y1.off(); G1.off();
  R2.off(); Y2.off(); G2.off();
  D1.off(); W1.off();
  D2.off(); W2.off();
}

void initState(long period) {
  //Initialise the state
  if (lastState != state) { //Only run this on entering the state for the first time
    lastState = state;      //Update the last state
    t1.reset(period);       //Reset the timer
    off();                  //Turn off all lights
  }
}

void setup() {
}

void loop() {
  //Traffic light finite state machine
  switch (state) {
    case GRWD:
      //GRWD  G1    R2    W1      D2      10000        GRFD
      initState(10000);                                //Entry
      G1.on(); R2.on(); W1.on(); D2.on();              //Outputs
      if (t1.timeout() || P2.state()) state = GRFD;    //Exit
      break;
```

```
    case GRFD:
      //GRFD  G1    R2    F1      D2      3000        GRDD
      initState(3000);                                //Entry
      G1.on(); R2.on(); D1.flash(200, 200); D2.on();  //Outputs
      if (t1.timeout()) state = GRDD;                 //Exit
      break;
    case GRDD:
      //GRDD  G1    R2    D1      D2      2000        YRDD
      initState(2000);                                //Entry
      G1.on(); R2.on(); D1.on(); D2.on();             //Outputs
      if (t1.timeout()) state = YRDD;                 //Exit
      break;
    case YRDD:
      //YRDD  Y1    R2    D1      D2      2000        RRDDa
      initState(2000);                                //Entry
      Y1.on(); R2.on(); D1.on(); D2.on();             //Outputs
      if (t1.timeout()) state = RRDDa;                //Exit
      break;
    case RRDDa:
      //RRDDa R1    R2    D1      D2      1000        RGDW
      initState(1000);                                //Entry
      R1.on(); R2.on(); D1.on(); D2.on();             //Outputs
      if (t1.timeout()) state = RGDW;                 //Exit
      break;
    case RGDW:
      //RGDW  R1    G2    D1      W2      10000        RGDF
      initState(10000);                                //Entry
      R1.on(); G2.on(); D1.on(); W2.on();              //Outputs
      if (t1.timeout() || P1.state()) state = RGDF;    //Exit
      break;
    case RGDF:
      //RGDF  R1    G2    D1      F2      3000        RGDD
      initState(3000);                                //Entry
      R1.on(); G2.on(); D1.on(); D2.flash(200, 200);  //Outputs
      if (t1.timeout()) state = RGDD;                 //Exit
      break;
    case RGDD:
      //RGDD  R1    G2    D1      D2      2000        RYDD
      initState(2000);                                //Entry
      R1.on(); G2.on(); D1.on(); D2.on();             //Outputs
      if (t1.timeout()) state = RYDD;                 //Exit
      break;
    case RYDD:
      //RYDD  R1    Y2    D1      D2      2000        RRDDb
      initState(2000);                                //Entry
      R1.on(); Y2.on(); D1.on(); D2.on();             //Outputs
      if (t1.timeout()) state = RRDDb;                //Exit
      break;
    case RRDDb:
      //RRDDb R1    R2    D1      D2      1000        GRWD
      initState(1000);                                //Entry
      R1.on(); R2.on(); D1.on(); D2.on();             //Outputs
      if (t1.timeout()) state = GRWD;                 //Exit
      break;
  }
}
```

```
//motor.h - Library for DC motor drive subsystem.
//Copyright (c) 2018, Julie Gonzales VK3FOWL and Joe Gonzales VK3YSP.
//www.sarcnet.org
//Released under the GNU General Public License.
//Provides forward and reverse PWM control of a DC motor via an L298N driver.

#include "filter.h"

#ifndef MOTOR_H
#define MOTOR_H

class Motor {
  public:
    Motor(int fwdPin, int revPin, float alpha);
    void spd(int spd);
  private:
    int _fwdPin; //Left forward drive pin
    int _revPin; //Left reverse drive pin
    Filter _f1;
};

#endif
```

```
//motor.cpp - Library for DC motor drive subsystem.
//Copyright (c) 2018, Julie Gonzales VK3FOWL and Joe Gonzales VK3YSP.
//www.sarcnet.org
//Released under the GNU General Public License.
//Provides forward and reverse PWM control of a DC motor via an L298N driver.

#include "motor.h"
#include "filter.h"
#include "arduino.h"

Motor::Motor(int fwdPin, int revPin, float alpha): _f1(alpha) {
  //Drive class constructor
  _fwdPin = fwdPin;
  _revPin = revPin;
  //Set the pin modes
  pinMode(_fwdPin, OUTPUT);
  pinMode(_revPin, OUTPUT);
  //Make it safe
  analogWrite(_fwdPin, 0);
  analogWrite(_revPin, 0);
}

void Motor::spd(int spd) {
  //Set the motor speed. Note: spd in the range -255..255
  //Where: -255 means full reverse, 255 means full forward and 0 means stopped
  spd = constrain(spd, -255, 255);
  spd = _f1.lpf(spd);
  if (spd > 0) {
    analogWrite(_fwdPin, spd);
  } else {
    analogWrite(_revPin, -spd);
  }
}
```

```
//motor.ino - Motor drive example of motor drive subsystem using C++ programming.
//Copyright (c) 2015, Julie Gonzales VK3FOWL and Joe Gonzales VK3YSP.
//www.sarcnet.org
//Released under the GNU General Public License.

#include "motor.h"
#include "timer.h"

//Declare Constants
const int TFWD = 1800;
const int TURN = 2800;

//Declare Motor states
enum State {FWD1, LFT1, FWD2, LFT2, FWD3, LFT3, FWD4, LFT4};
State state = FWD1, lastState = LFT4;

//Decalre Motor objects
Motor m1(3, 5, 0.001);
Motor m2(6, 9, 0.001);

//Declare timer object
Timer t1;

/* Motor Demo State Transition Table
  State m1  m2  Time  Next
  FWD1  255 255 TFWD  LFT1
  LFT1  255 0   TURN  FWD2
  FWD2  255 255 TFWD  LFT2
  LFT2  255 0   TURN  FWD3
  FWD3  255 255 TFWD  LFT3
  LFT3  255 0   TURN  FWD4
  FWD4  255 255 TFWD  LFT4
  LFT4  255 0   TURN  FWD2
*/

void initState(long period) {
  //Initialise the state
  if (lastState != state) { //Only run this on entering the state for the first time
    lastState = state;      //Update the last state
    t1.reset(period);       //Reset the timer
  }
}
void setup() {
}

void loop() {
  //Motor demo finite state machine
  switch (state) {
    case FWD1:
      //FWD1  255 255 FWD  LFT1
      initState(TFWD);                    //Entry
      m1.spd(255); m2.spd(255);           //Outputs
      if (t1.timeout()) state = LFT1;     //Exit
      break;
    case LFT1:
      //LFT1  255 0   TURN  FWD2
      initState(TURN);                    //Entry
      m1.spd(255); m2.spd(0);             //Outputs
      if (t1.timeout()) state = FWD2;     //Exit
      break;
    case FWD2:
      //FWD2  255 255 FWD  LFT2
      initState(TFWD);                    //Entry
      m1.spd(255); m2.spd(255);           //Outputs
      if (t1.timeout()) state = LFT2;     //Exit
      break;
    case LFT2:
      //LFT2  255 0   TURN  FWD3
```

```
      initState(TURN);                  //Entry
      m1.spd(255); m2.spd(0);           //Outputs
      if (t1.timeout()) state = FWD3;   //Exit
      break;
    case FWD3:
      //FWD3  255 255 FWD  LFT3
      initState(TFWD);                  //Entry
      m1.spd(255); m2.spd(255);         //Outputs
      if (t1.timeout()) state = LFT3;   //Exit
      break;
    case LFT3:
      //LFT3  255 0   TURN  FWD4
      initState(TURN);                  //Entry
      m1.spd(255); m2.spd(0);           //Outputs
      if (t1.timeout()) state = FWD4;   //Exit
      break;
    case FWD4:
      //FWD4  255 255 FWD  LFT4
      initState(TFWD);                  //Entry
      m1.spd(255); m2.spd(255);         //Outputs
      if (t1.timeout()) state = LFT4;   //Exit
      break;
    case LFT4:
      //LFT4  255 0   TURN  FWD2
      initState(TURN);                  //Entry
      m1.spd(255); m2.spd(0);           //Outputs
      if (t1.timeout()) state = FWD1;   //Exit
      break;
  }
}
```