# mBot with Arduino

## Contents

## Get Started with the Arduino IDE

Time required: 30 minutes

The mBot is based on an open source microcontroller board called an Arduino. We are going to learn Arduino C to make our mBot move about. Arduino C is based on the C++ programming language.

### Install the Arduino IDE

An IDE is an Integrated Development Environment. mBlock is an IDE. An IDE contains everything necessary to create fully functional programs in whatever language you are writing in. The Arduino IDE is specifically designed for microcontroller boards like the mBot.

1. Go to https://www.arduino.cc/en/Main/Software
2. Download and install the **Arduino IDE**: **Windows Win 7 and newer**
3. Double Click the file downloaded file to start the installation.
4. Accept everything. Click Close when the installation is done.

### Install the Makeblock Libraries

The Arduino IDE doesn't know anything about the mBot. We will download the code needed to communicate with the mBot. This code is in a library.

1. Go to https://github.com/Makeblock-official/Makeblock-Libraries/archive/master.zip
2. This link will start a download of a file named **Makeblock-Libraries-master.zip**
3. Right Click on the file you just downloaded → Choose **Extract All**.
4. You will end up with a **Makeblock-Libraries-master** folder with some files and folders inside.
5. Double click the folder to find another **Makeblock-Libraries-master** folder.
6. Copy this folder → **Makeblock-Libraries-master** → into your **Documents\Libraries** folder.
7. Start the Arduino IDE.
8. Click **File** → **Examples** → **mBlockDrive**. This is one of the sample programs for the mBot.
9. You are ready to program your mBot in Arduino C.

## First Program: Blink LED's

Please read the directions carefully before beginning the assignment.

- Comment your code as shown in the tutorials and other code examples.
- Follow all directions carefully and accurately.
- Think of the directions as minimum requirements.

### Requirements

1. Blink LED's on the robot in a continuous loop.

### Tutorial Assignment

1. Start the Arduino IDE. Save the sketch as **Blinky**.
2. Complete and test the program as shown.

### Challenge

1. Add more LED blinks
2. Add different delay times.

### Upload a Program to the mBot

This is how to compile and upload your program to the robot. An Arduino device can only run one program at a time.

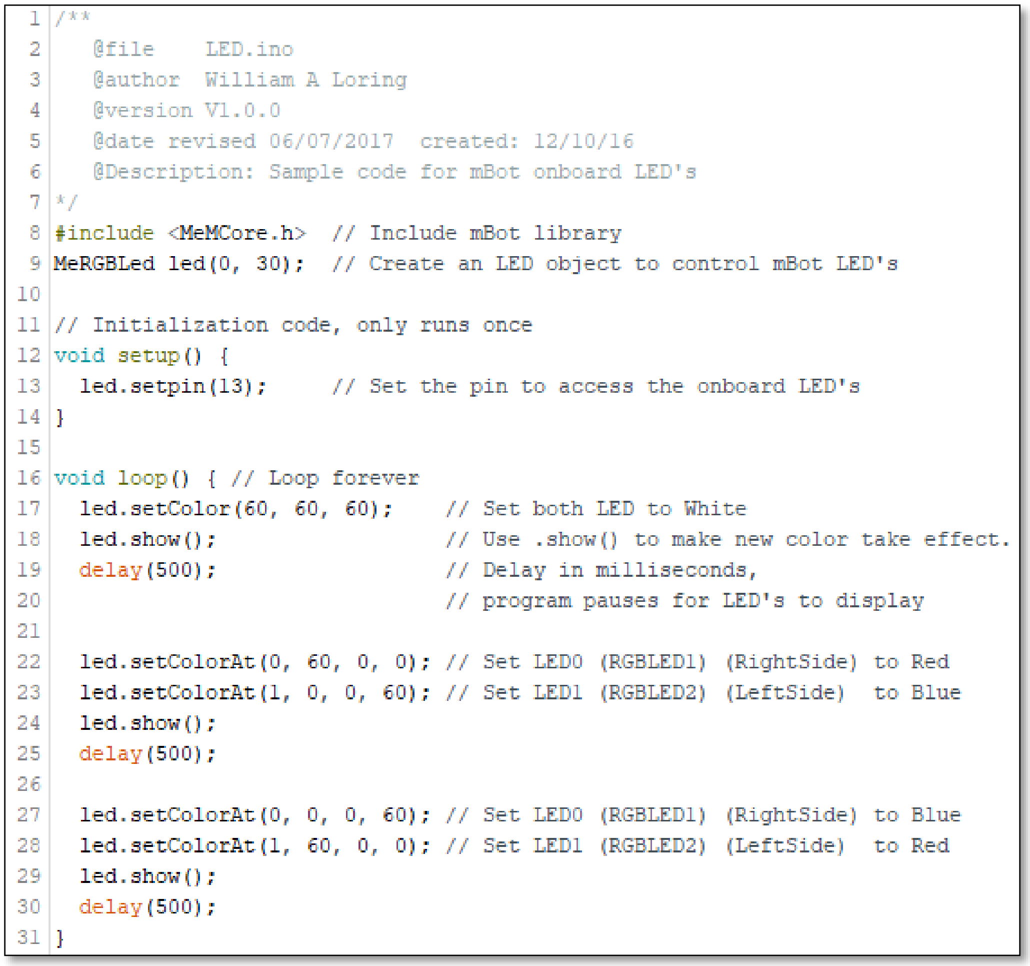
1. Power on the robot.
2. Connect the robot through the USB cable.
3. Run the **Arduino IDE** software. You may have to navigate to the C:\Arduino folder, where C is whatever drive you copied your Arduino folder to.)
4. Under **Tools → Board,** it should say **Arduino Uno**.
5. Select **Tools → Port → COMx**, where x is the highest number shown.
6. Select **Sketch → Upload**. This compiles and uploads the program to your robot.

## Assignment Submission

1. All students: Zip up the sketch folder. Attach the zip file to the assignment in Blackboard.

2. The assignment is demonstrated in class.

3. Online students: A link to a YouTube video recording showing your robot going through its motions is placed in the submission area in BlackBoard.



```
/**
   @file    LED.ino
   @author  William A Loring
   @version V1.0.0
   @date revised 06/07/2017  created: 12/10/16
   @Description: Sample code for mBot onboard LED's
*/
#include <MeMCore.h>  // Include mBot library
MeRGBLed led(0, 30);  // Create an LED object to control mBot LED's

// Initialization code, only runs once
void setup() {
  led.setpin(13);     // Set the pin to access the onboard LED's
}

void loop() { // Loop forever
  led.setColor(60, 60, 60);    // Set both LED to White
  led.show();                  // Use .show() to make new color take effect.
  delay(500);                  // Delay in milliseconds,
                               // program pauses for LED's to display

  led.setColorAt(0, 60, 0, 0); // Set LED0 (RGBLED1) (RightSide) to Red
  led.setColorAt(1, 0, 0, 60); // Set LED1 (RGBLED2) (LeftSide)  to Blue
  led.show();
  delay(500);

  led.setColorAt(0, 0, 0, 60); // Set LED0 (RGBLED1) (RightSide) to Blue
  led.setColorAt(1, 60, 0, 0); // Set LED1 (RGBLED2) (LeftSide)  to Red
  led.show();
  delay(500);
}
```

## Simple Buzzer

Time required: 20 minutes

Please read all the directions carefully before beginning the assignment.

- Comment your code as shown in the tutorials and other code examples.
- Follow all directions carefully and accurately.
- Think of the directions as minimum requirements.

### Understanding

Demonstrate understanding of:

**delay, buzzer, setup, loop**

### Knowledge Points

Every Arduino sketch has functions. A function is a block of code. Functions help to keep our code organized and tidy.

The **setup()** and **loop()** functions are required for all Arduino sketches.

**setup()** This function initializes anything needed for the program. It only runs once. Control is turned over to the loop() function when it is finished.

**loop()** This function does exactly what it says it does. It loops repeatedly as fast as it can. This is the main part of the program.

### Requirements

1. Create a sketch that uses the onboard buzzer to make some music.
2. The notes will keep playing until the mBot is turned off.

### Tutorial Assignment

1. Start the Arduino IDE. Save the sketch as **SimpleBuzzer**.
2. Complete and test the program as shown.

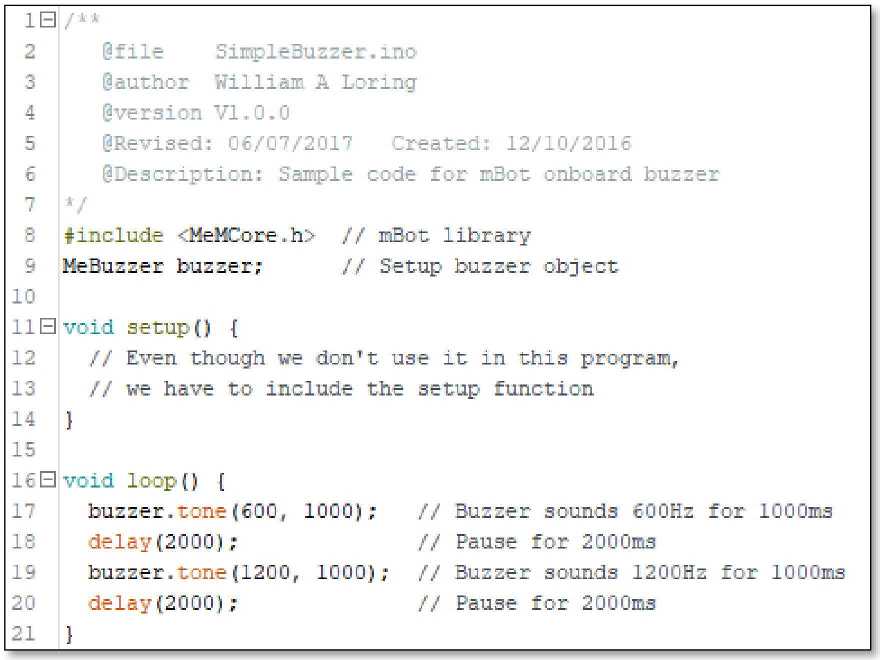
### Challenge

1. Add more buzzer notes

2. Add different delay times.

## Assignment Submission

1. All students: Zip up the sketch folder. Attach the zip file to the assignment in Blackboard.
2. The assignment is demonstrated in class.
3. Online students: A link to a YouTube video recording showing your robot going through its motions is placed in the submission area in BlackBoard.



```
/**
   @file    SimpleBuzzer.ino
   @author  William A Loring
   @version V1.0.0
   @Revised: 06/07/2017   Created: 12/10/2016
   @Description: Sample code for mBot onboard buzzer
*/
#include <MeMCore.h>  // mBot library
MeBuzzer buzzer;      // Setup buzzer object

void setup() {
  // Even though we don't use it in this program,
  // we have to include the setup function
}

void loop() {
  buzzer.tone(600, 1000);   // Buzzer sounds 600Hz for 1000ms
  delay(2000);              // Pause for 2000ms
  buzzer.tone(1200, 1000);  // Buzzer sounds 1200Hz for 1000ms
  delay(2000);              // Pause for 2000ms
}
```

## Simple Movement

Time required: 30 minutes

Please read all the directions carefully before beginning the assignment.

- Comment your code as shown in the tutorials and other code examples.
- Follow all directions carefully and accurately.
- Think of the directions as minimum requirements.

### Understanding

Demonstrate understanding of:

**delay, setup, loop, if statement**

### Knowledge Points

This program shows how to control the motors. The program doesn’t execute until the remote button up is pressed.

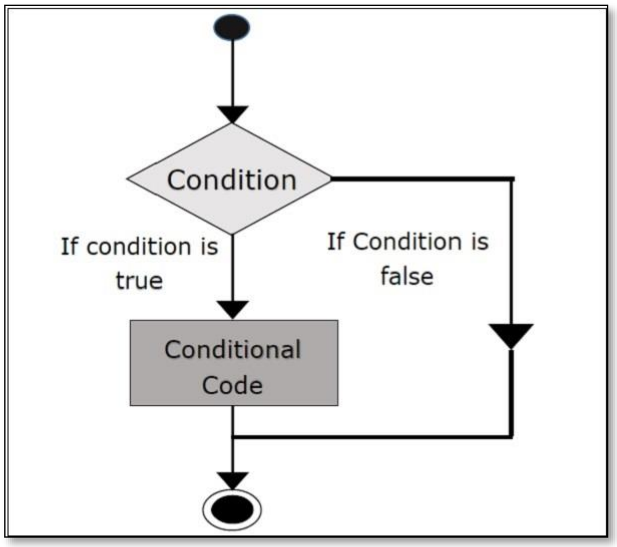
An `if` statement is a decision or control structure. The `if` statement checks for a condition and executes the following statement or set of statements if the condition is 'true'.

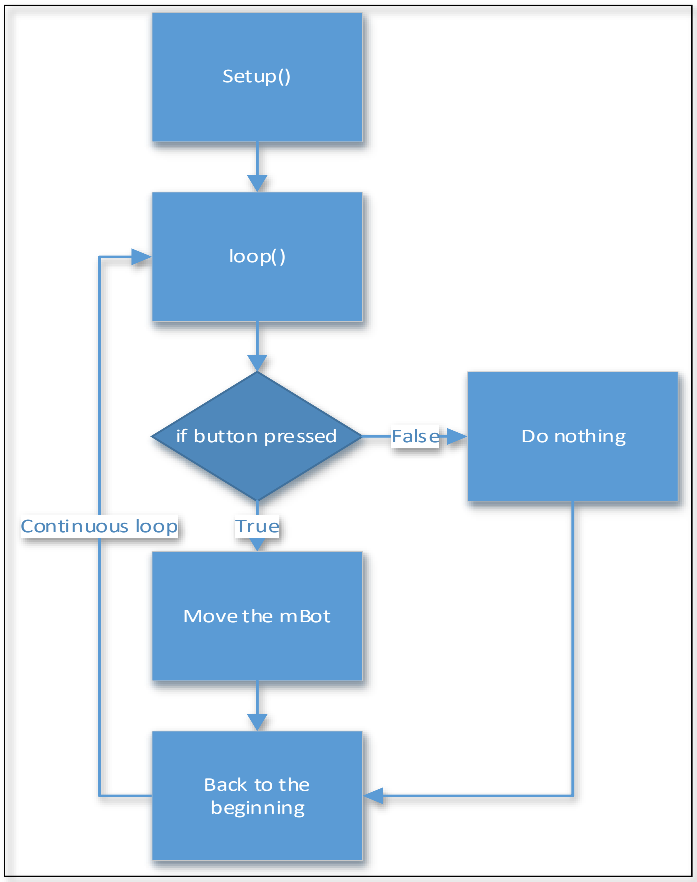
The == is the equal to comparison operator. Equal to ( == )returns true if the value on the left is equal to the value on the right, otherwise it returns false. This is also called a Boolean condition.

#### Syntax

```
if (condition == true) {
  // do stuff if the condition is true
}
```

This program uses an `if` control structure as shown in the diagrams.





## Requirements

Complete and successfully run the program as shown.

## Tutorial Assignment

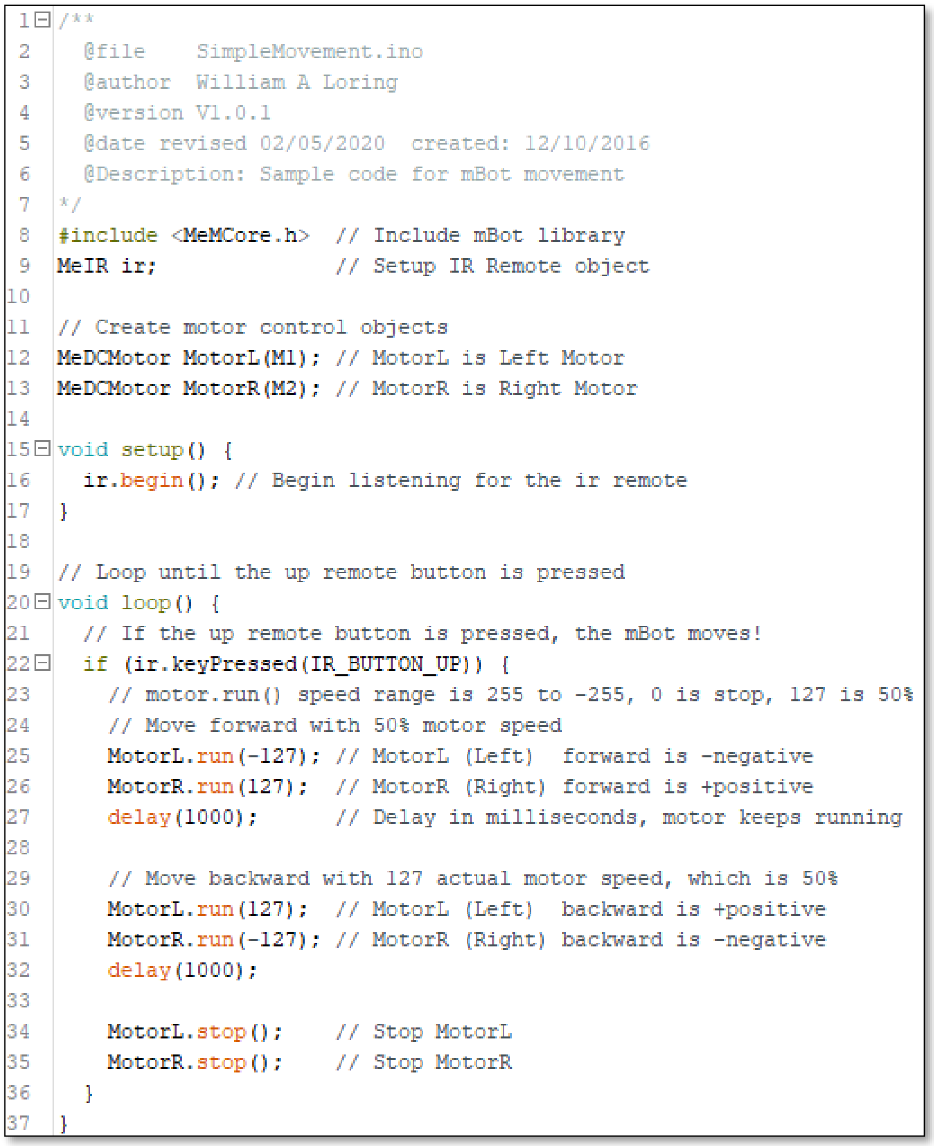
1. Start the Arduino IDE. Create a new sketch called **SimpleMovement**.
2. Complete and test the program as shown.

## Challenge

1. Add different movements to the program.

## Assignment Submission

1. All students: Zip up the sketch folder. Attach the zip file to the assignment in Blackboard.
2. The assignment is demonstrated in class.
3. Online students: A link to a YouTube video recording showing your robot going through its motions is placed in the submission area in BlackBoard.



```cpp
/**
  @file    SimpleMovement.ino
  @author  William A Loring
  @version V1.0.1
  @date revised 02/05/2020  created: 12/10/2016
  @Description: Sample code for mBot movement
*/
#include <MeMCore.h>  // Include mBot library
MeIR ir;              // Setup IR Remote object

// Create motor control objects
MeDCMotor MotorL(M1); // MotorL is Left Motor
MeDCMotor MotorR(M2); // MotorR is Right Motor

void setup() {
  ir.begin(); // Begin listening for the ir remote
}

// Loop until the up remote button is pressed
void loop() {
  // If the up remote button is pressed, the mBot moves!
  if (ir.keyPressed(IR_BUTTON_UP)) {
    // motor.run() speed range is 255 to -255, 0 is stop, 127 is 50%
    // Move forward with 50% motor speed
    MotorL.run(-127); // MotorL (Left)  forward is -negative
    MotorR.run(127);  // MotorR (Right) forward is +positive
    delay(1000);      // Delay in milliseconds, motor keeps running

    // Move backward with 127 actual motor speed, which is 50%
    MotorL.run(127);  // MotorL (Left)  backward is +positive
    MotorR.run(-127); // MotorR (Right) backward is -negative
    delay(1000);

    MotorL.stop();    // Stop MotorL
    MotorR.stop();    // Stop MotorR
  }
}
```

## Random Numbers

Time required: 15 minutes

Please read all the directions carefully before beginning the assignment.

1. Comment your code as shown in the tutorials and other code examples.
2. Follow all directions carefully and accurately.
3. Think of the directions as minimum requirements.

### Understanding

Demonstrate understanding of:

**random numbers, serial monitor**

### Requirements

Complete and successfully run the program as displayed.

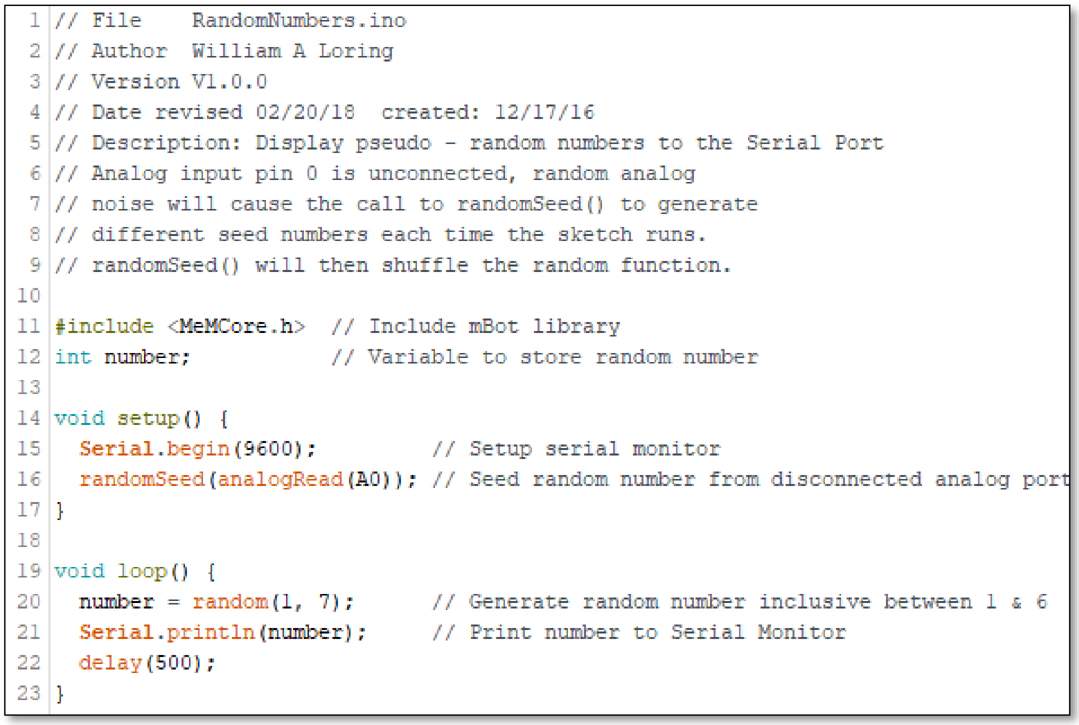
### Tutorial Assignment

We are going to generate some random numbers. This can be used to send the robot in random directions for random times or random LED light values. This sketch also shows how to use the Serial Monitor for debugging purposes.

1. Start the Arduino IDE. Save the sketch as **RandomNumbers**.
2. Complete and test the program as pictured with the requirements listed.
3. While running the sketch, go to **Tools → Serial Monitor** to display the random numbers. Please include this in your video.
4. Comment your code.

### Assignment Submission

1. All students: Zip up the sketch folder. Attach the zip file to the assignment in Blackboard.
2. The assignment is demonstrated in class.
3. Online students: A link to a YouTube video recording showing your robot going through its motions is placed in the submission area in BlackBoard.



```
// File    RandomNumbers.ino
// Author  William A Loring
// Version V1.0.0
// Date revised 02/20/18  created: 12/17/16
// Description: Display pseudo - random numbers to the Serial Port
// Analog input pin 0 is unconnected, random analog
// noise will cause the call to randomSeed() to generate
// different seed numbers each time the sketch runs.
// randomSeed() will then shuffle the random function.

#include <MeMCore.h>  // Include mBot library
int number;           // Variable to store random number

void setup() {
  Serial.begin(9600);         // Setup serial monitor
  randomSeed(analogRead(A0)); // Seed random number from disconnected analog port
}

void loop() {
  number = random(1, 7);      // Generate random number inclusive between 1 & 6
  Serial.println(number);     // Print number to Serial Monitor
  delay(500);
}
```

## Random LED’s

Time required: 30 minutes

Please read all the directions carefully before beginning the assignment.

1. Comment your code as shown in the tutorials and other code examples.
2. Follow all directions carefully and accurately.
3. Think of the directions as minimum requirements.

### Understanding

Demonstrate understanding of:

**random numbers, LED’s, constants, variables**

### Knowledge Points

In mBlock we had variables. Variables in Arduino C are the same idea. A variable stores a value in a memory location, and can be changed.

An int variable type is a whole number.

```
int a = 9;
```

A constant is declared once and never changes.

```
// Constant to store upper range of random LED colors
const int UPPER_RANDOM = 21;
```

### Tutorial Assignment

1. Start the Arduino IDE. Save the sketch as **RandomLED**.
2. Create and test the program as shown.

### Requirements

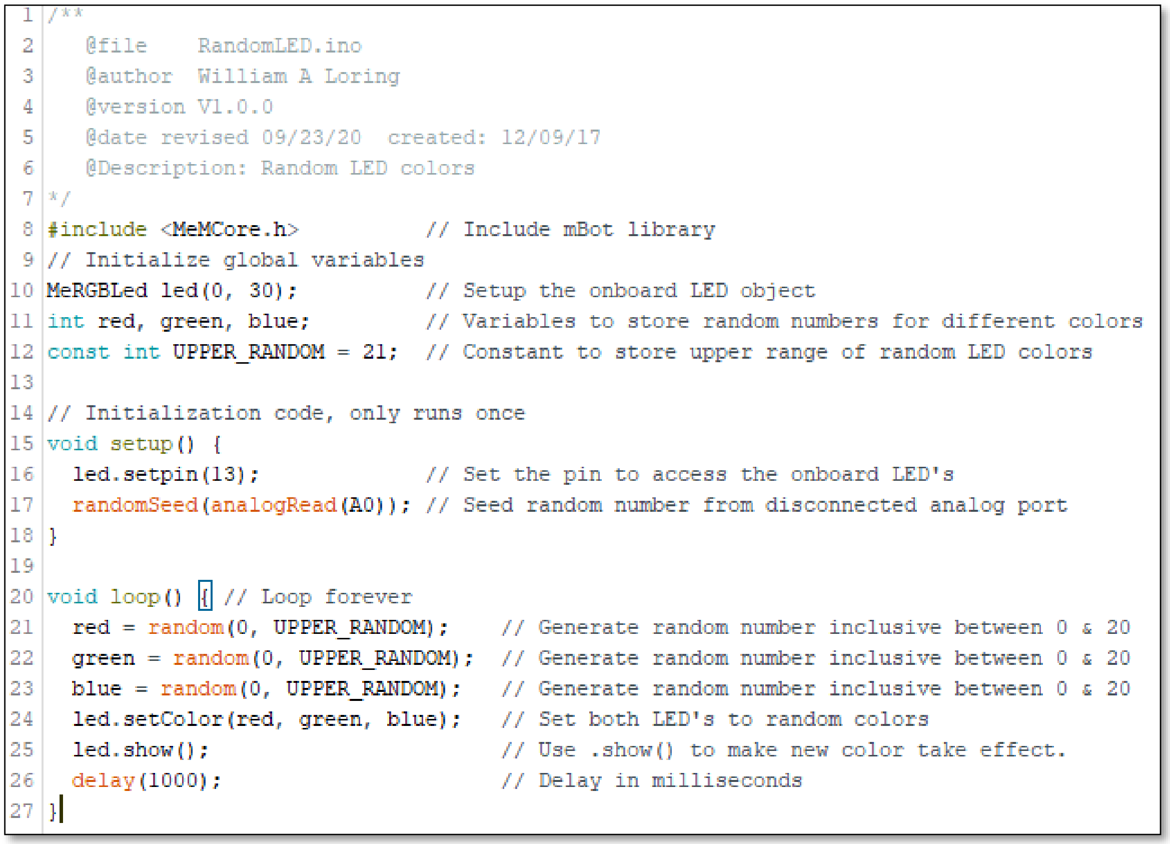
The onboard LED lights change randomly every second.

### Challenge

1. Choose a random value for one or two colors, set the others to a static value.
2. Change the range of random numbers.

## Assignment Submission

- Attach the programs, and submit the assignment in Blackboard.
- The challenge assignment is demonstrated in class or a link to a YouTube video recording showing your robot going through its motions is placed in the submission area in BlackBoard.



```
/**
   @file    RandomLED.ino
   @author  William A Loring
   @version V1.0.0
   @date revised 09/23/20  created: 12/09/17
   @Description: Random LED colors
*/
#include <MeMCore.h>          // Include mBot library
// Initialize global variables
MeRGBLed led(0, 30);          // Setup the onboard LED object
int red, green, blue;         // Variables to store random numbers for different colors
const int UPPER_RANDOM = 21;  // Constant to store upper range of random LED colors

// Initialization code, only runs once
void setup() {
  led.setpin(13);             // Set the pin to access the onboard LED's
  randomSeed(analogRead(A0)); // Seed random number from disconnected analog port
}

void loop() { // Loop forever
  red = random(0, UPPER_RANDOM);    // Generate random number inclusive between 0 & 20
  green = random(0, UPPER_RANDOM);  // Generate random number inclusive between 0 & 20
  blue = random(0, UPPER_RANDOM);   // Generate random number inclusive between 0 & 20
  led.setColor(red, green, blue);   // Set both LED's to random colors
  led.show();                       // Use .show() to make new color take effect.
  delay(1000);                      // Delay in milliseconds
}
```

## Twinkle Twinkle

Time required: 45 minutes

Please read all the directions carefully before beginning the assignment.

- Comment your code as shown in the tutorials and other code examples.
- Follow all directions carefully and accurately.
- Think of the directions as minimum requirements.

### Understanding

Demonstrate understanding of:

**libraries, functions**

### Libraries

A library file is a collection of code that you can use from an Arduino sketch. It allows you to easily reuse code. The library file notes.h is placed in the sketch folder.

### Requirements

Play Twinkle Twinkle, Little Star from a library file.

### Tutorial Assignment

1. Start the Arduino IDE. Save the sketch as **TwinkleTwinkle**.
2. Copy the **notes.h** file attached to the assignment into the **TwinkleTwinkle** sketch folder.
3. Complete and test the program as shown.

### Challenge

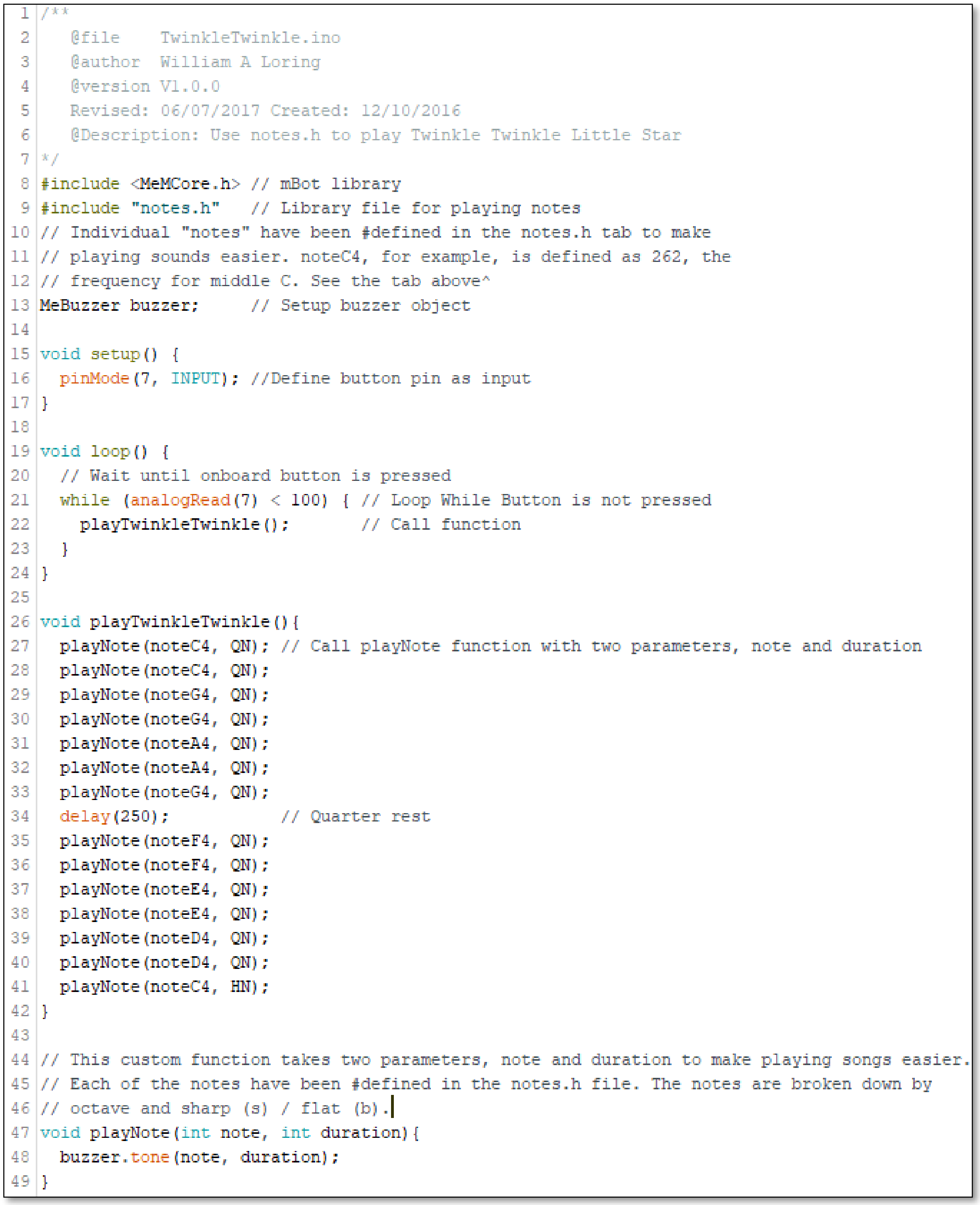
1. Add a new function call and function name to match the song you chose.
2. Play a small part of a song of your choosing or make up your own song. There is plenty of sheet music available on the web with notes and names of notes to help you figure out a different song.
   Here is a web site to get you started with a known song. https://noobnotes.net

## Assignment Submission

1. All students: Zip up the sketch folder. Attach the zip file to the assignment in Blackboard.

2. The assignment is demonstrated in class.

3. Online students: A link to a YouTube video recording showing your robot going through its motions is placed in the submission area in BlackBoard.



```
/**
   @file    TwinkleTwinkle.ino
   @author  William A Loring
   @version V1.0.0
   Revised: 06/07/2017 Created: 12/10/2016
   @Description: Use notes.h to play Twinkle Twinkle Little Star
*/
#include <MeMCore.h> // mBot library
#include "notes.h"   // Library file for playing notes
// Individual "notes" have been #defined in the notes.h tab to make
// playing sounds easier. noteC4, for example, is defined as 262, the
// frequency for middle C. See the tab above^
MeBuzzer buzzer;     // Setup buzzer object

void setup() {
  pinMode(7, INPUT); //Define button pin as input
}

void loop() {
  // Wait until onboard button is pressed
  while (analogRead(7) < 100) { // Loop While Button is not pressed
    playTwinkleTwinkle();       // Call function
  }
}

void playTwinkleTwinkle(){
  playNote(noteC4, QN); // Call playNote function with two parameters, note and duration
  playNote(noteC4, QN);
  playNote(noteG4, QN);
  playNote(noteG4, QN);
  playNote(noteA4, QN);
  playNote(noteA4, QN);
  playNote(noteG4, QN);
  delay(250);           // Quarter rest
  playNote(noteF4, QN);
  playNote(noteF4, QN);
  playNote(noteE4, QN);
  playNote(noteE4, QN);
  playNote(noteD4, QN);
  playNote(noteD4, QN);
  playNote(noteC4, HN);
}

// This custom function takes two parameters, note and duration to make playing songs easier.
// Each of the notes have been #defined in the notes.h file. The notes are broken down by
// octave and sharp (s) / flat (b).
void playNote(int note, int duration){
  buzzer.tone(note, duration);
}
```

## Simple Remote Control

Time required: 60 minutes

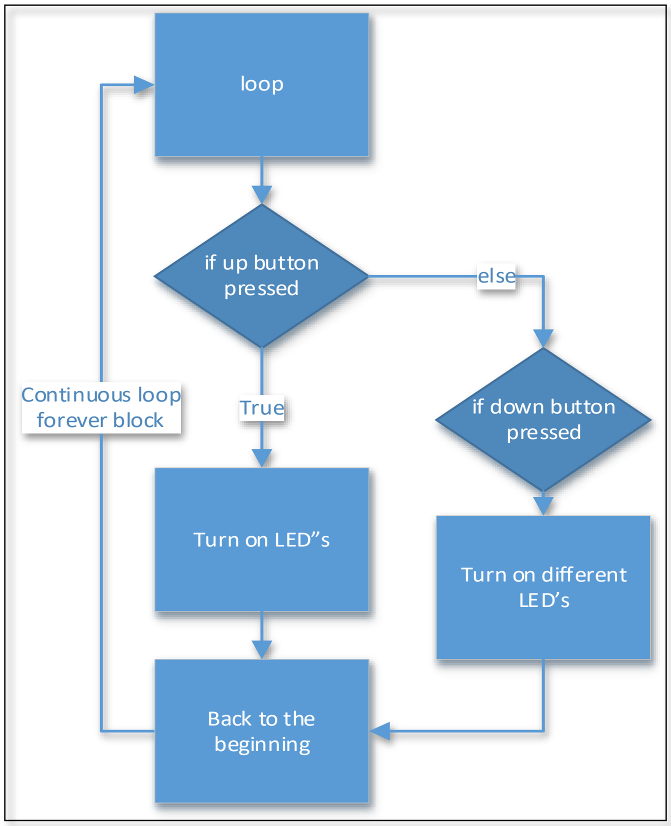
Please read all the directions carefully before beginning the assignment.

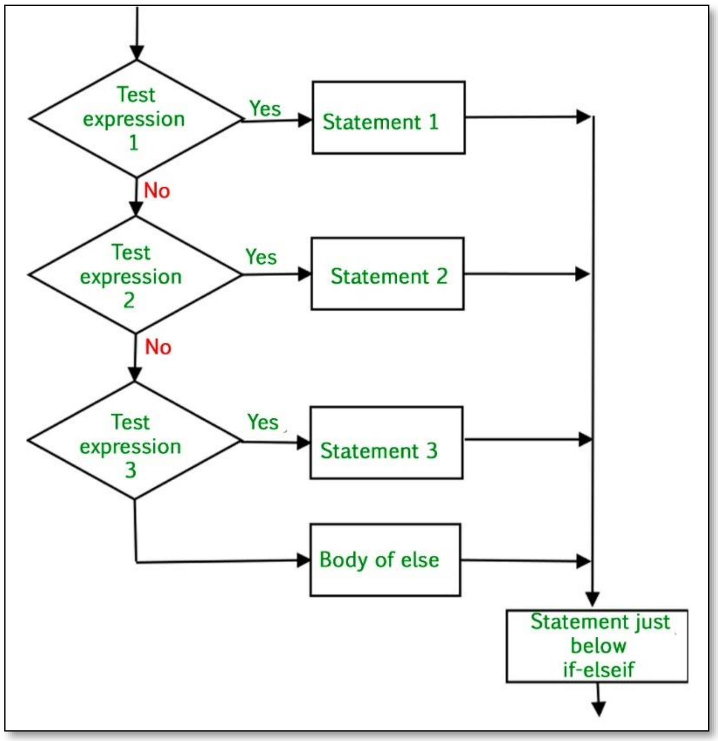
- Comment your code as shown in the tutorials and other code examples.
- Follow all directions carefully and accurately.
- Think of the directions as minimum requirements.

## Knowledge Points

### If Then Else

If then else extends our decision making. If then else is mutually exclusive. This means that only one of the choices or conditions can be true. You can stack multiple if else statements together.





## Debouncing

When you press a button on the remote, it is impossible to press it once. It makes contact several times, bouncing off the contact. It may make contact on one side, then the other, then it settles down. The Arduino scans for each ir code so fast that it can mistake one code for another. Debouncing is putting a tiny delay in between scanning for the ir codes to ensure smooth operation.

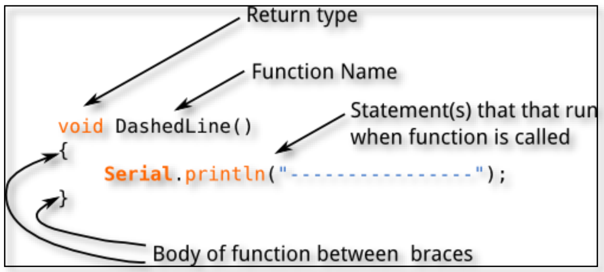
## Functions

As our code gets longer and more complex, it can get difficult to follow. Functions allow for reusable and modular code. A function is a code block. It wraps up everything needed to provide a service to the program. You can easily reuse the code in another sketch or the same sketch.

We have used pre written functions, such as **led.setColorAt();** and **delay(500);**. We will start writing our own.

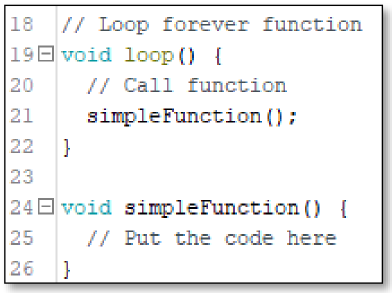
Please go to the following web site to learn more about functions.

https://startingelectronics.org/software/arduino/learn-to-program-course/15-functions/


Return type
Function Name
Statement(s) that that run when function is called
Body of function between braces

```
void DashedLine()
{
    Serial.println("----------------");
}
```

### Sample function



```
// Loop forever function
void loop() {
  // Call function
  simpleFunction();
}

void simpleFunction() {
  // Put the code here
}
```

## Tutorial Assignment

1. Start the Arduino IDE. Save the sketch as **SimpleRemoteControl**.
2. Include **notes.h** in the **SimpleRemoteControl** folder.
3. Complete and test the program as pictured with the requirements listed.

### Requirements

The robot will move in the direction of the arrow keys on the remote, then stop when the keys are released.

## Challenge

- Use LED's to indicate direction and movement.

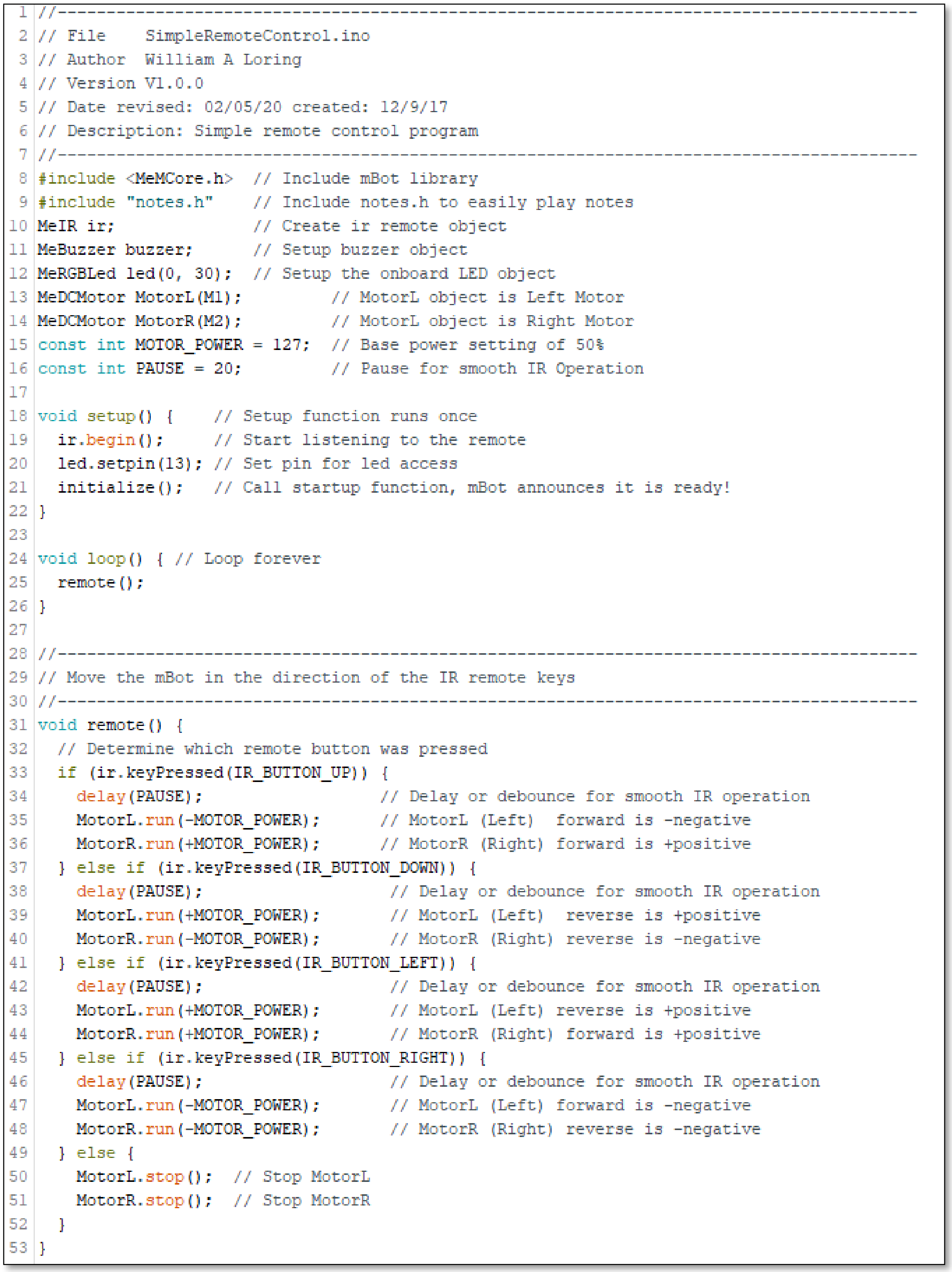
## Assignment Submission

1. All students: Zip up the sketch folder. Attach the zip file to the assignment in Blackboard.
2. The assignment is demonstrated in class.
3. Online students: A link to a YouTube video recording showing your robot going through its motions is placed in the submission area in BlackBoard.

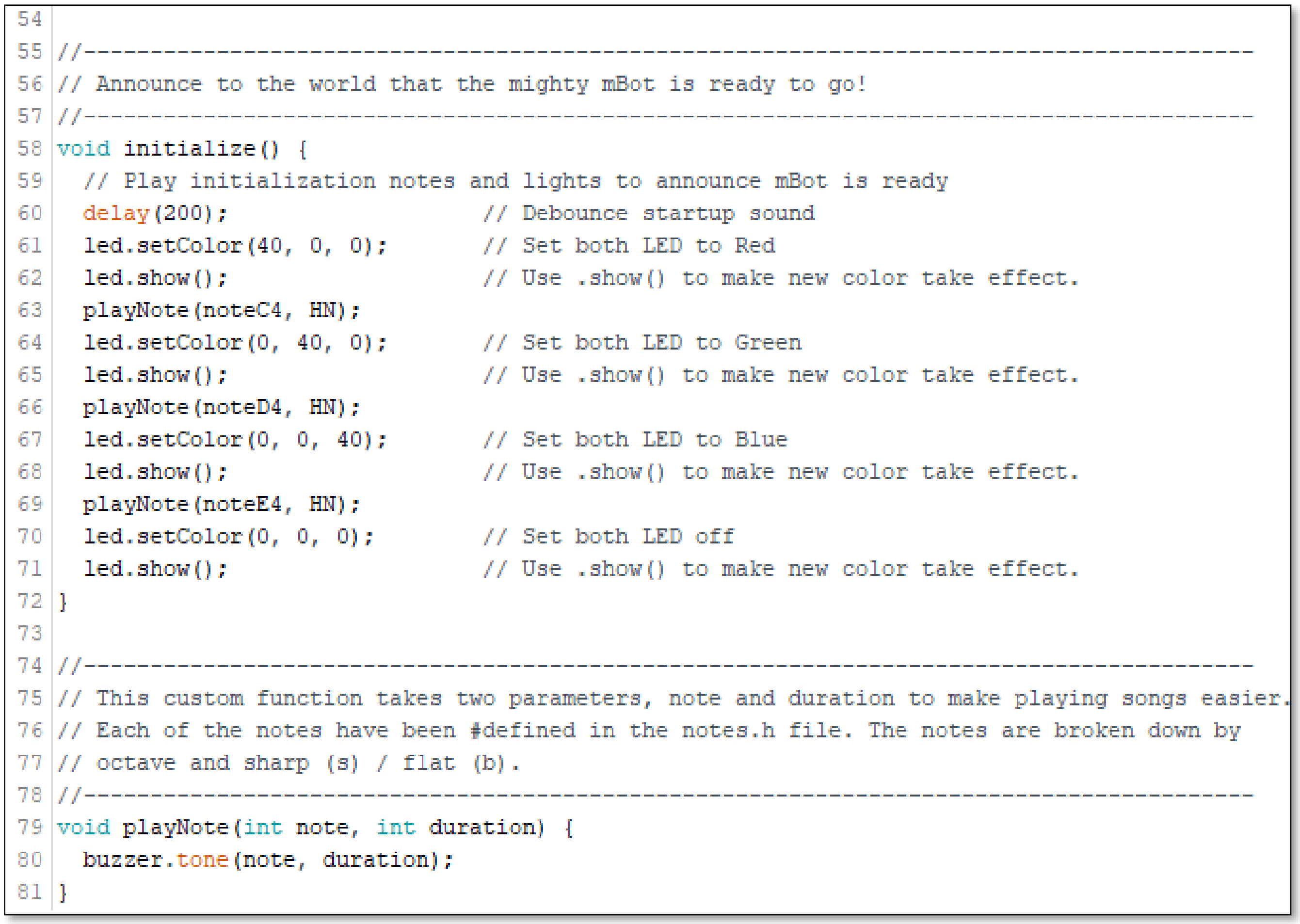
## IR Remote Constants for mBot

The following table is a reference for the constants for reading the IR remote.

| | | |
|---|---|---|
| IR_BUTTON_A | IR_BUTTON_SETTING | IR_BUTTON_0 |
| IR_BUTTON_B | IR_BUTTON_LEFT | IR_BUTTON_1 |
| IR_BUTTON_C | IR_BUTTON_RIGHT | IR_BUTTON_2 |
| IR_BUTTON_D | IR_BUTTON_UP | IR_BUTTON_3 |
| IR_BUTTON_E | IR_BUTTON_DOWN | IR_BUTTON_4 |
| IR_BUTTON_F | | IR_BUTTON_5 |
| | | IR_BUTTON_6 |
| | | IR_BUTTON_7 |
| | | IR_BUTTON_8 |
| | | IR_BUTTON_9 |



```
//-----------------------------------------------------------------------------------------
// File    SimpleRemoteControl.ino
// Author  William A Loring
// Version V1.0.0
// Date revised: 02/05/20 created: 12/9/17
// Description: Simple remote control program
//-----------------------------------------------------------------------------------------
#include <MeMCore.h>  // Include mBot library
#include "notes.h"    // Include notes.h to easily play notes
MeIR ir;              // Create ir remote object
MeBuzzer buzzer;      // Setup buzzer object
MeRGBLed led(0, 30);  // Setup the onboard LED object
MeDCMotor MotorL(M1);         // MotorL object is Left Motor
MeDCMotor MotorR(M2);         // MotorL object is Right Motor
const int MOTOR_POWER = 127;  // Base power setting of 50%
const int PAUSE = 20;         // Pause for smooth IR Operation

void setup() {    // Setup function runs once
  ir.begin();     // Start listening to the remote
  led.setpin(13); // Set pin for led access
  initialize();   // Call startup function, mBot announces it is ready!
}

void loop() { // Loop forever
  remote();
}

//-----------------------------------------------------------------------------------------
// Move the mBot in the direction of the IR remote keys
//-----------------------------------------------------------------------------------------
void remote() {
  // Determine which remote button was pressed
  if (ir.keyPressed(IR_BUTTON_UP)) {
    delay(PAUSE);                // Delay or debounce for smooth IR operation
    MotorL.run(-MOTOR_POWER);    // MotorL (Left)  forward is -negative
    MotorR.run(+MOTOR_POWER);    // MotorR (Right) forward is +positive
  } else if (ir.keyPressed(IR_BUTTON_DOWN)) {
    delay(PAUSE);                 // Delay or debounce for smooth IR operation
    MotorL.run(+MOTOR_POWER);     // MotorL (Left)  reverse is +positive
    MotorR.run(-MOTOR_POWER);     // MotorR (Right) reverse is -negative
  } else if (ir.keyPressed(IR_BUTTON_LEFT)) {
    delay(PAUSE);                 // Delay or debounce for smooth IR operation
    MotorL.run(+MOTOR_POWER);     // MotorL (Left) reverse is +positive
    MotorR.run(+MOTOR_POWER);     // MotorR (Right) forward is +positive
  } else if (ir.keyPressed(IR_BUTTON_RIGHT)) {
    delay(PAUSE);                 // Delay or debounce for smooth IR operation
    MotorL.run(-MOTOR_POWER);     // MotorL (Left) forward is -negative
    MotorR.run(-MOTOR_POWER);     // MotorR (Right) reverse is -negative
  } else {
    MotorL.stop();  // Stop MotorL
    MotorR.stop();  // Stop MotorR
  }
}
```



```

//----------------------------------------------------------------------------------------
// Announce to the world that the mighty mBot is ready to go!
//----------------------------------------------------------------------------------------
void initialize() {
  // Play initialization notes and lights to announce mBot is ready
  delay(200);                   // Debounce startup sound
  led.setColor(40, 0, 0);       // Set both LED to Red
  led.show();                   // Use .show() to make new color take effect.
  playNote(noteC4, HN);
  led.setColor(0, 40, 0);       // Set both LED to Green
  led.show();                   // Use .show() to make new color take effect.
  playNote(noteD4, HN);
  led.setColor(0, 0, 40);       // Set both LED to Blue
  led.show();                   // Use .show() to make new color take effect.
  playNote(noteE4, HN);
  led.setColor(0, 0, 0);        // Set both LED off
  led.show();                   // Use .show() to make new color take effect.
}

//----------------------------------------------------------------------------------------
// This custom function takes two parameters, note and duration to make playing songs easier.
// Each of the notes have been #defined in the notes.h file. The notes are broken down by
// octave and sharp (s) / flat (b).
//----------------------------------------------------------------------------------------
void playNote(int note, int duration) {
  buzzer.tone(note, duration);
}
```

## ~~Function Junction~~

Time required: 30 minutes

Please read all the directions carefully before beginning the assignment.

- Comment your code as shown in the tutorials and other code examples.
- Follow all directions carefully and accurately.
- Think of the directions as minimum requirements.

### Understanding

Demonstrate understanding of:

**light sensor, serial monitor**

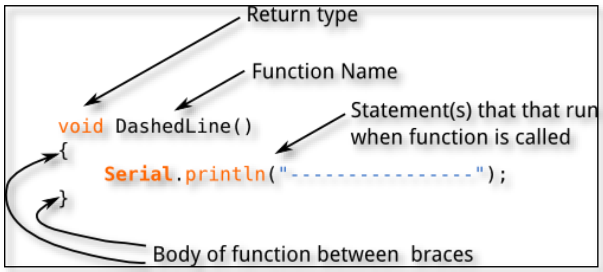
### Knowledge Points

As our code gets longer and more complex, it can get difficult to follow. Functions allow for reusable and modular code. A function is a code block. It wraps up everything needed to provide a service to the program. You can easily reuse the code in another sketch or the same sketch.

We have used pre written functions, such as **led.setColorAt();** and **delay(500);**. We will start writing our own.

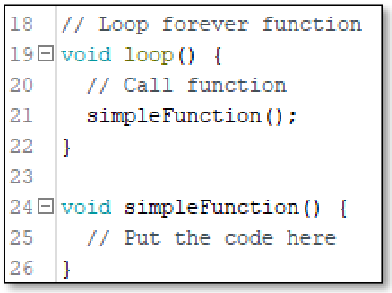
Please go to the following web site to learn more about functions.

[https://startingelectronics.org/software/arduino/learn-to-program-course/15-functions/](https://startingelectronics.org/software/arduino/learn-to-program-course/15-functions/)



```
void DashedLine()
{
    Serial.println("----------------");
}
```

### Sample function



```
// Loop forever function
void loop() {
  // Call function
  simpleFunction();
}

void simpleFunction() {
  // Put the code here
}
```

## Requirements

Break our code into smaller chunks by dividing our code into functions.

## Tutorial Assignment

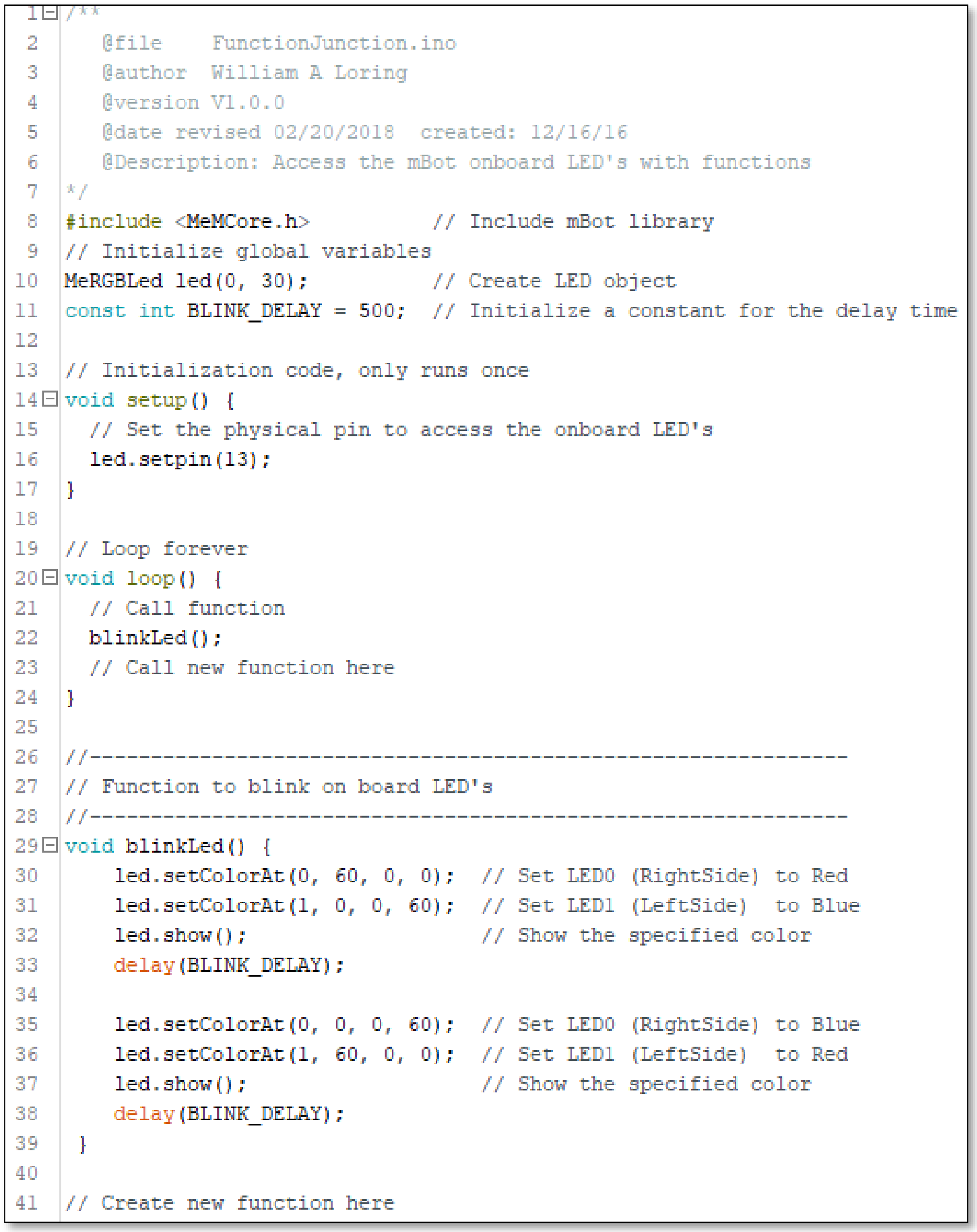
1. Start the Arduino IDE. Save the sketch as **FunctionJunction**.
2. Complete and test the program as shown.

## Challenge

- Create a new function that does something with LED's and/or sound.
- Call the new function.

## Assignment Submission

1. All students: Zip up the sketch folder. Attach the zip file to the assignment in Blackboard.
2. The assignment is demonstrated in class.
3. Online students: A link to a YouTube video recording showing your robot going through its motions is placed in the submission area in BlackBoard.



```
/**
   @file    FunctionJunction.ino
   @author  William A Loring
   @version V1.0.0
   @date revised 02/20/2018  created: 12/16/16
   @Description: Access the mBot onboard LED's with functions
*/
#include <MeMCore.h>          // Include mBot library
// Initialize global variables
MeRGBLed led(0, 30);          // Create LED object
const int BLINK_DELAY = 500;  // Initialize a constant for the delay time

// Initialization code, only runs once
void setup() {
  // Set the physical pin to access the onboard LED's
  led.setpin(13);
}

// Loop forever
void loop() {
  // Call function
  blinkLed();
  // Call new function here
}

//--------------------------------------------------------------
// Function to blink on board LED's
//--------------------------------------------------------------
void blinkLed() {
    led.setColorAt(0, 60, 0, 0);  // Set LED0 (RightSide) to Red
    led.setColorAt(1, 0, 0, 60);  // Set LED1 (LeftSide)  to Blue
    led.show();                   // Show the specified color
    delay(BLINK_DELAY);

    led.setColorAt(0, 0, 0, 60);  // Set LED0 (RightSide) to Blue
    led.setColorAt(1, 60, 0, 0);  // Set LED1 (LeftSide)  to Red
    led.show();                   // Show the specified color
    delay(BLINK_DELAY);
 }

// Create new function here
```

## Ambulance

Time required: 45 minutes

IDE: Arduino

Please read all the directions carefully before beginning the assignment.

- Comment your code as shown in the tutorials and other code examples.
- Follow all directions carefully and accurately.
- Think of the directions as minimum requirements.

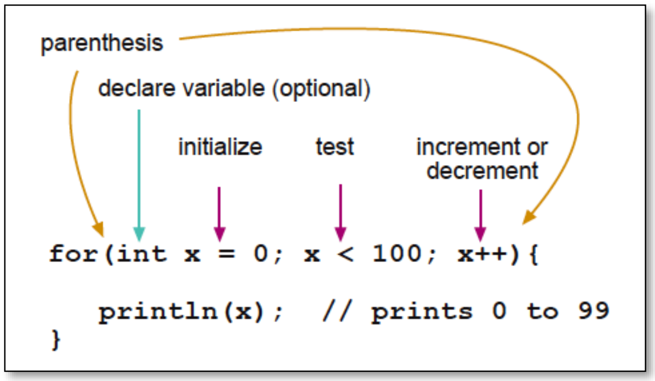
### Understanding

Demonstrate understanding of:

**random numbers, LED's, for loops**

### Knowledge Points

The for statement is used to repeat a block of statements enclosed in curly braces. An increment counter is usually used to increment and terminate the loop. The for statement is useful for any repetitive operation.



```
for(int x = 0; x < 100; x++){
    println(x);  // prints 0 to 99
}
```

This is an example of a for loop which iterates 5 times.

```
for (int i = 0; i < 5; i++ ){
// Do something
}
```

## Tutorial Assignment

Use the pictured mBlock program as a model for this program. Convert the code concepts into the corresponding Arduino code. Notice how the blocks similar to the Arduino C code.

1. Start the Arduino IDE. Save the sketch as **Ambulance**.
2. Use a for loop to repeat the siren 5 times.

### How to Use The Buzzer:

```
buzzer.tone(600, 1000); //Buzzer sounds 600Hz for 1000ms
```

### How to Use Left and Right LED's

```
led.setColorAt(0, 40, 0, 0); // Set LED0 (RGBLED1) (RightSide) to Red
led.setColorAt(1, 0, 0, 0);  // Set LED1 (RGBLED2) (LeftSide)  to Blue
led.show();
```

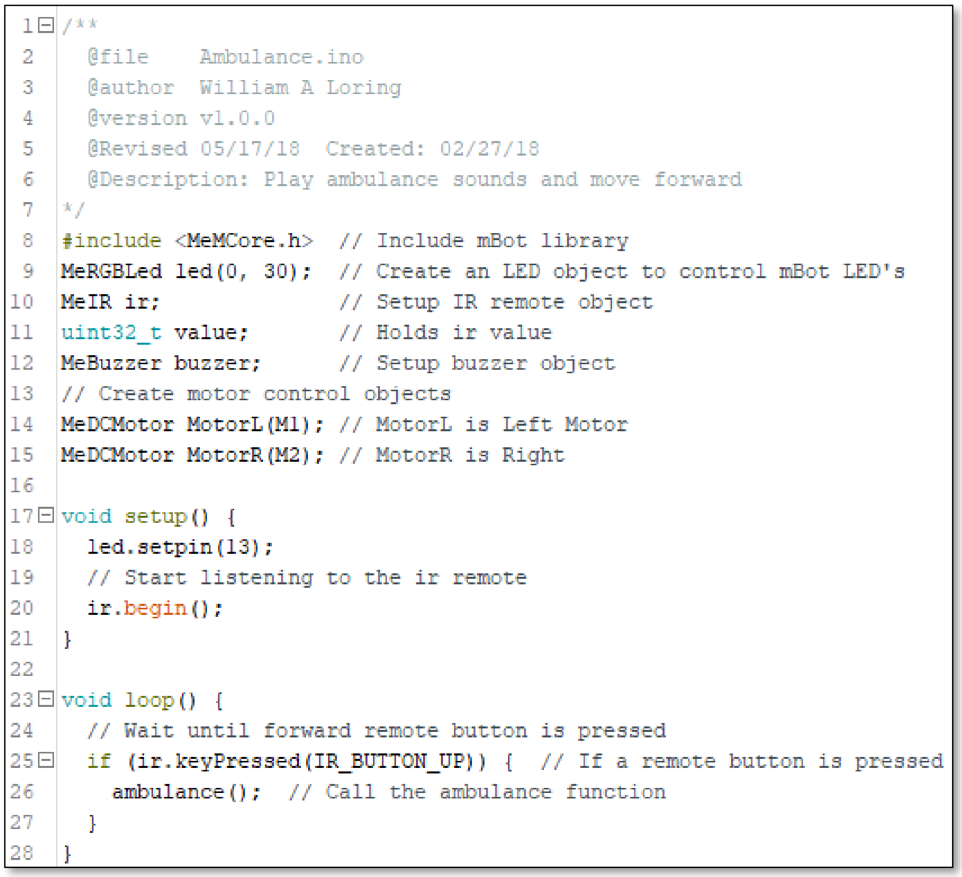
### Requirements

- The program will run when you press the remote button on the mBot.
- The program will play an ambulance siren and move forward.
- Comment your code.

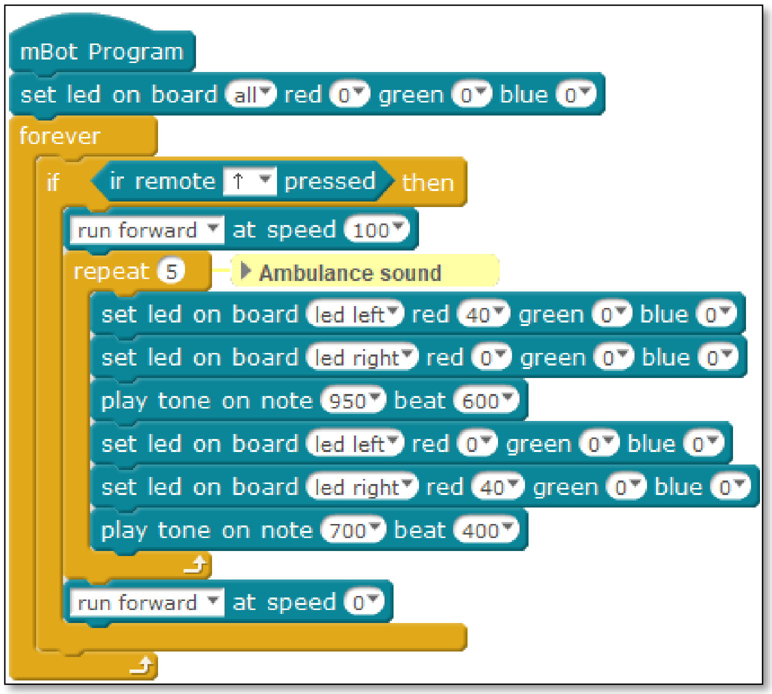
## Assignment Submission

1. All students: Zip up the sketch folder. Attach the zip file to the assignment in Blackboard.
2. The assignment is demonstrated in class.
3. Online students: A link to a YouTube video recording showing your robot going through its motions is placed in the submission area in BlackBoard.

## Program Starter Code



```
/**
  @file    Ambulance.ino
  @author  William A Loring
  @version v1.0.0
  @Revised 05/17/18  Created: 02/27/18
  @Description: Play ambulance sounds and move forward
*/
#include <MeMCore.h>  // Include mBot library
MeRGBLed led(0, 30);  // Create an LED object to control mBot LED's
MeIR ir;              // Setup IR remote object
uint32_t value;       // Holds ir value
MeBuzzer buzzer;      // Setup buzzer object
// Create motor control objects
MeDCMotor MotorL(M1); // MotorL is Left Motor
MeDCMotor MotorR(M2); // MotorR is Right

void setup() {
  led.setpin(13);
  // Start listening to the ir remote
  ir.begin();
}

void loop() {
  // Wait until forward remote button is pressed
  if (ir.keyPressed(IR_BUTTON_UP)) {  // If a remote button is pressed
    ambulance();  // Call the ambulance function
  }
}
```



```
mBot Program
set led on board all red 0 green 0 blue 0
forever
  if ir remote ↑ pressed then
    run forward at speed 100
    repeat 5    Ambulance sound
      set led on board led left red 40 green 0 blue 0
      set led on board led right red 0 green 0 blue 0
      play tone on note 950 beat 600
      set led on board led left red 0 green 0 blue 0
      set led on board led right red 40 green 0 blue 0
      play tone on note 700 beat 400
    run forward at speed 0
```

## Remote Control with Backup Sounds

Time required: 30 minutes

Please read all the directions carefully before beginning the assignment.

- Comment your code as shown in the tutorials and other code examples.
- Follow all directions carefully and accurately.
- Think of the directions as minimum requirements.

### Understanding

Demonstrate understanding of:

**variables, for loops**

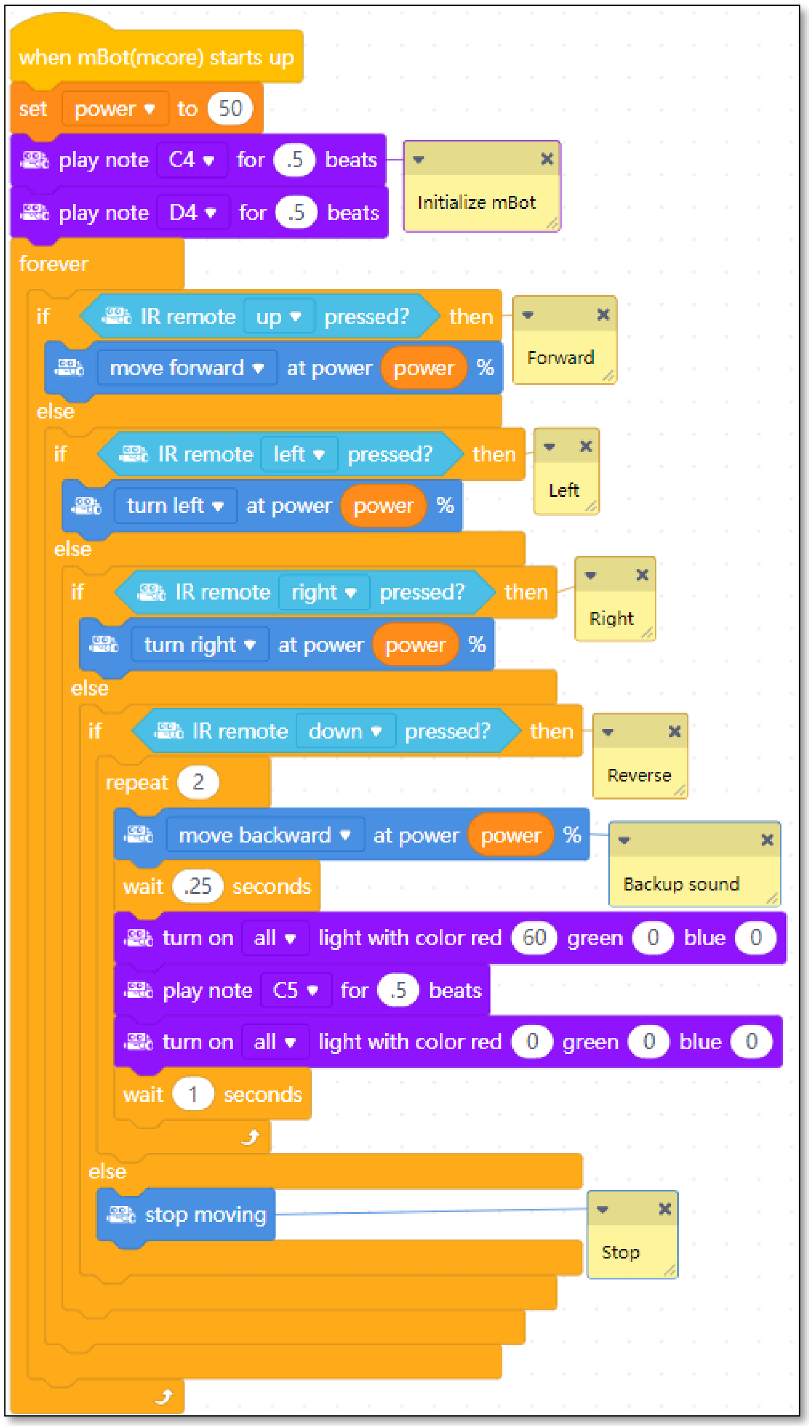
### Knowledge Points

### Requirements

This program simulates the backup sound of a dump truck or payloader.

### Tutorial Assignment

1. Save your **SimpleRemoteControl** sketch as **RemoteControlBackup**
2. Add some backup sounds to your simple remote sketch using a for loop. Use the mBlock program shown as a guide to build your backup sounds.



```
when mBot(mcore) starts up
set power ▾ to 50
play note C4 ▾ for .5 beats                                  [Initialize mBot]
play note D4 ▾ for .5 beats
forever
  if IR remote up ▾ pressed? then                            [Forward]
    move forward ▾ at power (power) %
  else
    if IR remote left ▾ pressed? then                        [Left]
      turn left ▾ at power (power) %
    else
      if IR remote right ▾ pressed? then                     [Right]
        turn right ▾ at power (power) %
      else
        if IR remote down ▾ pressed? then                    [Reverse]
          repeat 2
            move backward ▾ at power (power) %               [Backup sound]
            wait .25 seconds
            turn on all ▾ light with color red 60 green 0 blue 0
            play note C5 ▾ for .5 beats
            turn on all ▾ light with color red 0 green 0 blue 0
            wait 1 seconds
        else
          stop moving                                        [Stop]
```

## Challenge

1. Add a variation to the Backup Sound to your Remote Control with Backup Sounds program.

## Assignment Submission

- All students attach the finished program to the assignment in Blackboard.
- Each assignment can be demonstrated in class.
- For online students, a link to a YouTube video recording showing the assignment can be placed in the submission area in BlackBoard.

## Fire Engine

Time required: 30 minutes

Please read all the directions carefully before beginning the assignment.

- Comment your code as shown in the tutorials and other code examples.
- Follow all directions carefully and accurately.
- Think of the directions as minimum requirements.

### Understanding

Demonstrate understanding of:

**variables, constants, for and while loops**

### Knowledge Points

This program simulates the sound of a fire engine siren.

### Fire Engine Sire Simulation

The fire engine siren's low frequency sound is 500Hz, its high frequency sound is between 1500Hz. The siren sound is generated by repeating the following pattern: the low frequency sound amplifies to a high frequency sound in 1.5 seconds, and then drops back to the lower frequency in 3.5 seconds. Therefore, the fire engine siren sounds can be programmatically simulated as follows:

Set the low frequency to be 500Hz, then set the high frequency sound to be 1500HZ, repeatedly playing the buzzer in a range from 500Hz to 1500Hz and then back to 700Hz. The ratio of amplification time to the drop time is 1.5:3.5, which is 3:7, so the ratio of frequency amplification to the drop needs to be 7:3. By tuning the sound time and amplification vs. drop's amplitude, the fire engine siren is simulated.

#### Why "repeat until frequency > 1500", not "frequency = 1500"?

That is because in the example of simulating the fire engine sound effects, it is hard to define the sound frequency each time when it is increased from 500Hz to 1500Hz and what increment it should be each time the frequency is increased. If we set frequency=1500, the final frequency should reach 1500 so that it can break the loop, or the frequency will be increased again and again, making it hard to debug. So we use frequency > 1500, and when the frequency is above 1500, the loop will be broken and the following program decreasing the frequency will be executed.

## Tutorial Assignment

1. Start the Arduino IDE. Save the program as **FireEngine**
2. Complete and test the program as pictured with the requirements listed.

## Requirements

The program runs as shown.

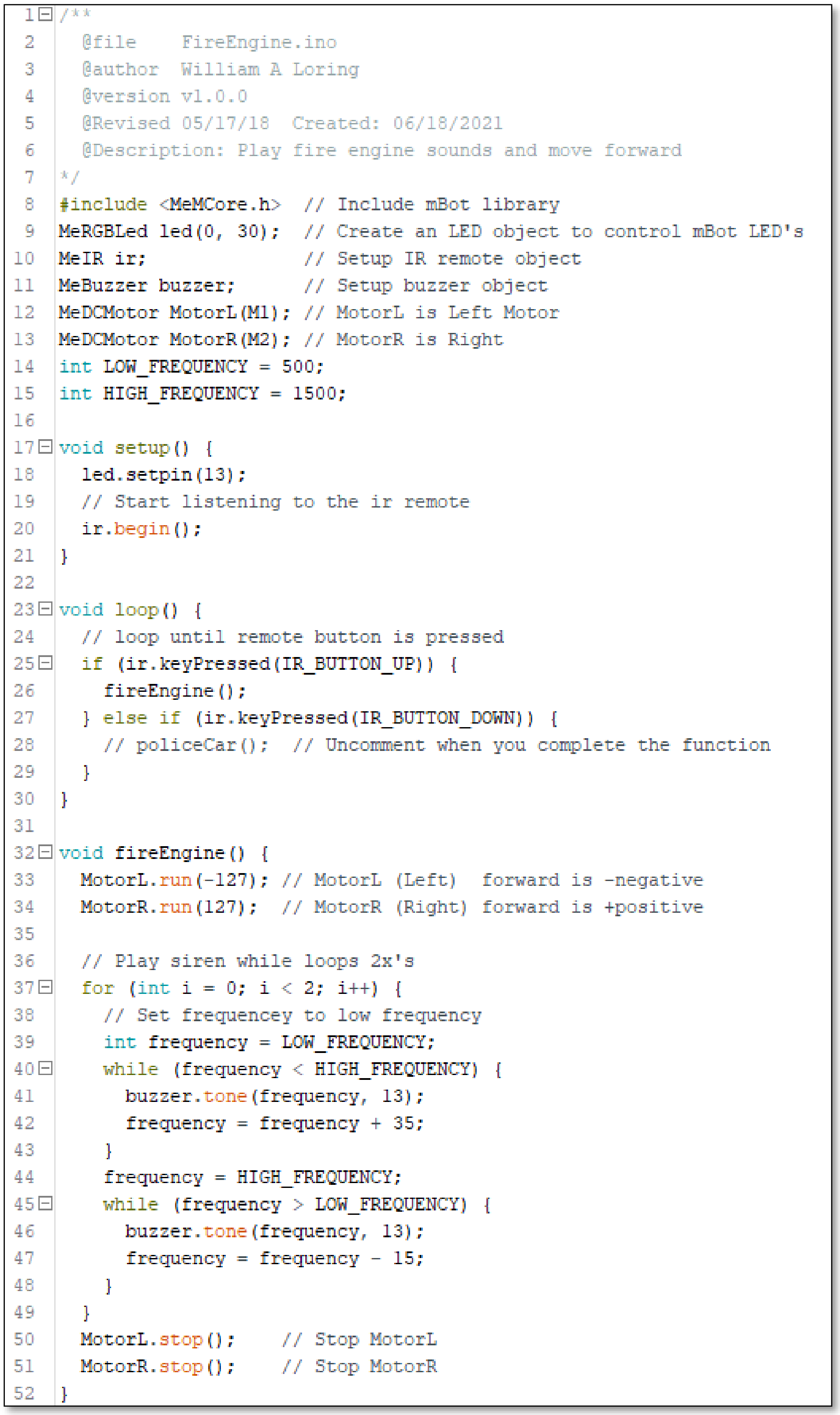
## Challenge

Keep the fire siren. Add a police car sound. The tutorial shows a policeCar function which you can complete

How to simulate police car sound effects: Low-frequency sound is set at 500Hz and high-frequency sound at 1500Hz. It takes 23 ms to raise a low-frequency sound to a high-frequency one and 23 ms to lower a high-frequency sound to a low-frequency one.

## Assignment Submission

- All students attach the finished program to the assignment in Blackboard.
- Each assignment can be demonstrated in class.
- For online students, a link to a YouTube video recording showing the assignment can be placed in the submission area in BlackBoard.



```
/**
  @file    FireEngine.ino
  @author  William A Loring
  @version v1.0.0
  @Revised 05/17/18  Created: 06/18/2021
  @Description: Play fire engine sounds and move forward
*/
#include <MeMCore.h>  // Include mBot library
MeRGBLed led(0, 30);  // Create an LED object to control mBot LED's
MeIR ir;              // Setup IR remote object
MeBuzzer buzzer;      // Setup buzzer object
MeDCMotor MotorL(M1); // MotorL is Left Motor
MeDCMotor MotorR(M2); // MotorR is Right
int LOW_FREQUENCY = 500;
int HIGH_FREQUENCY = 1500;

void setup() {
  led.setpin(13);
  // Start listening to the ir remote
  ir.begin();
}

void loop() {
  // loop until remote button is pressed
  if (ir.keyPressed(IR_BUTTON_UP)) {
    fireEngine();
  } else if (ir.keyPressed(IR_BUTTON_DOWN)) {
    // policeCar();  // Uncomment when you complete the function
  }
}

void fireEngine() {
  MotorL.run(-127); // MotorL (Left)  forward is -negative
  MotorR.run(127);  // MotorR (Right) forward is +positive

  // Play siren while loops 2x's
  for (int i = 0; i < 2; i++) {
    // Set frequencey to low frequency
    int frequency = LOW_FREQUENCY;
    while (frequency < HIGH_FREQUENCY) {
      buzzer.tone(frequency, 13);
      frequency = frequency + 35;
    }
    frequency = HIGH_FREQUENCY;
    while (frequency > LOW_FREQUENCY) {
      buzzer.tone(frequency, 13);
      frequency = frequency - 15;
    }
  }
  MotorL.stop();    // Stop MotorL
  MotorR.stop();    // Stop MotorR
}
```

## Functions with Parameters (Sounds and Lights)

Time required: 45 minutes

Please read all the directions carefully before beginning the assignment.

- Comment your code as shown in the tutorials and other code examples.
- Follow all directions carefully and accurately.
- Think of the directions as minimum requirements.

### Requirements

This program has two functions that behave differently based on the data that is passed in.

### Understanding

Demonstrate understanding of:

**LED's, buzzer, functions, parameters**

### Knowledge Points

Functions allow for reusable and modular code. The information or data passed into the function each time it is called can be different.

Please go to the following web site to learn more about functions.

https://startingelectronics.org/software/arduino/learn-to-program-course/15-functions/

```
// Call a function with 2 arguments
my_function(1, 2)
my_function(4, 60)

// A function with two parameters
Void my_function(int num1, int num2){
    delay(num1);
    buzzer(num1, num2);
}
```

### Tutorial Assignment

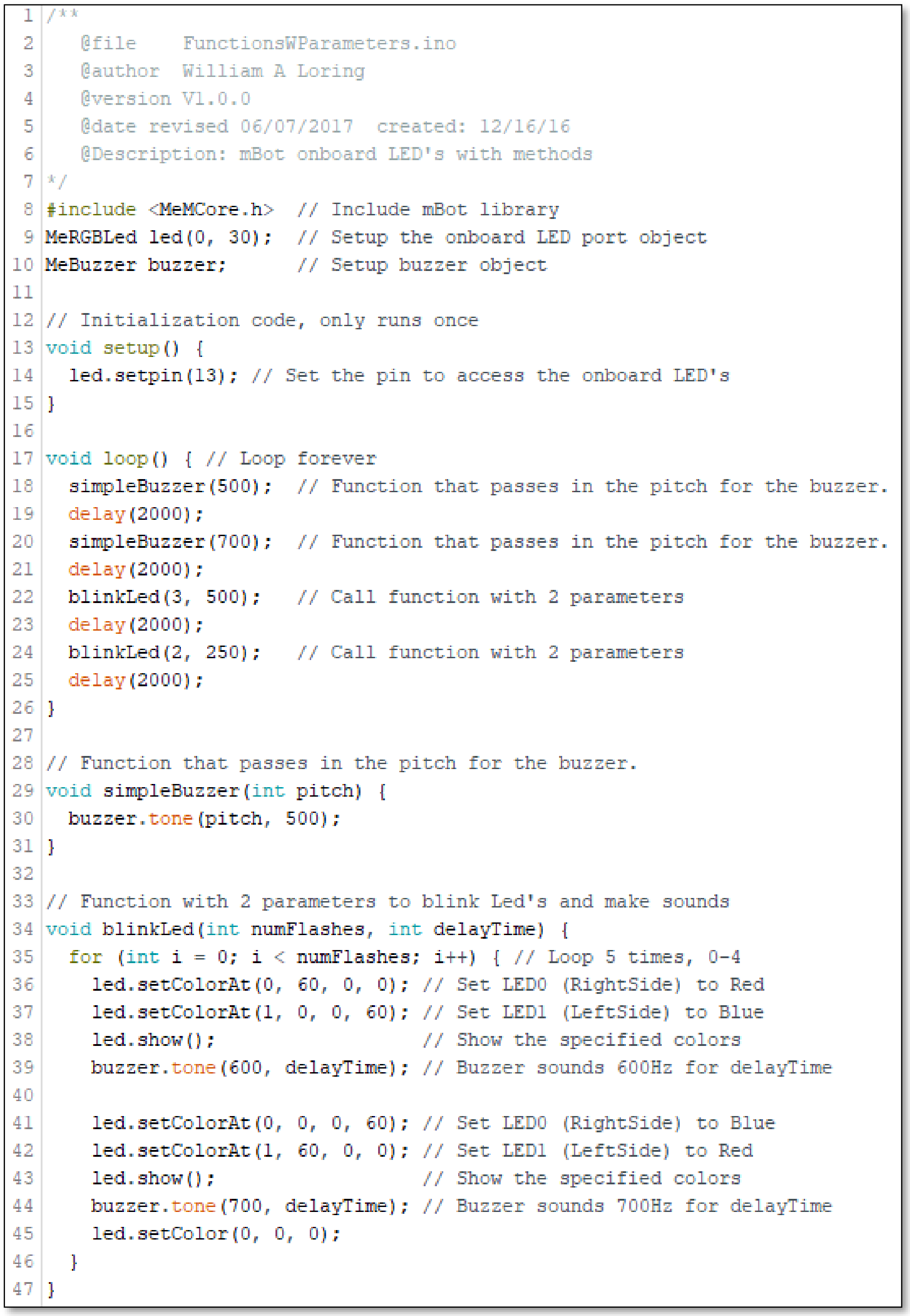
1. Start the Arduino IDE. Save the sketch as **FunctionsWParameters**.
2. Complete and test the tutorial program as shown.

## Challenge

- Add another simple function with a parameter or parameters that does something with LED’s or sound.
- Call the new function.

## Assignment Submission

1. All students: Zip up the sketch folder. Attach the zip file to the assignment in Blackboard.
2. The assignment is demonstrated in class.
3. Online students: A link to a YouTube video recording showing your robot going through its motions is placed in the submission area in BlackBoard.



```cpp
/**
   @file    FunctionsWParameters.ino
   @author  William A Loring
   @version V1.0.0
   @date revised 06/07/2017  created: 12/16/16
   @Description: mBot onboard LED's with methods
*/
#include <MeMCore.h>  // Include mBot library
MeRGBLed led(0, 30);  // Setup the onboard LED port object
MeBuzzer buzzer;      // Setup buzzer object

// Initialization code, only runs once
void setup() {
  led.setpin(13); // Set the pin to access the onboard LED's
}

void loop() { // Loop forever
  simpleBuzzer(500);  // Function that passes in the pitch for the buzzer.
  delay(2000);
  simpleBuzzer(700);  // Function that passes in the pitch for the buzzer.
  delay(2000);
  blinkLed(3, 500);   // Call function with 2 parameters
  delay(2000);
  blinkLed(2, 250);   // Call function with 2 parameters
  delay(2000);
}

// Function that passes in the pitch for the buzzer.
void simpleBuzzer(int pitch) {
  buzzer.tone(pitch, 500);
}

// Function with 2 parameters to blink Led's and make sounds
void blinkLed(int numFlashes, int delayTime) {
  for (int i = 0; i < numFlashes; i++) { // Loop 5 times, 0-4
    led.setColorAt(0, 60, 0, 0); // Set LED0 (RightSide) to Red
    led.setColorAt(1, 0, 0, 60); // Set LED1 (LeftSide) to Blue
    led.show();                  // Show the specified colors
    buzzer.tone(600, delayTime); // Buzzer sounds 600Hz for delayTime

    led.setColorAt(0, 0, 0, 60); // Set LED0 (RightSide) to Blue
    led.setColorAt(1, 60, 0, 0); // Set LED1 (LeftSide) to Red
    led.show();                  // Show the specified colors
    buzzer.tone(700, delayTime); // Buzzer sounds 700Hz for delayTime
    led.setColor(0, 0, 0);
  }
}
```

## Sound and Light Gradient

Time required: 45 minutes

Please read all the directions carefully before beginning the assignment.

- Comment your code as shown in the tutorials and other code examples.
- Follow all directions carefully and accurately.
- Think of the directions as minimum requirements.

### Understanding

Demonstrate understanding of:

**libraries, functions, loops, variables**

### Requirements

1. Use variables

### Tutorial Assignment

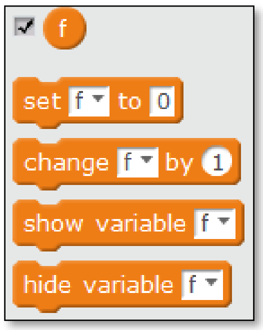
This assignment experiments with loops, variables, and changing values in the variables.

This assignment introduces variables and Blocks. Modular code is easier to maintain. Blocks are under My Blocks.

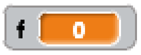
### Define a Variable

Go to **Variables**. Click

to create a variable. Enter the name of the variable and create it. 4 blocks relating to it will appear automatically, as shown in the following picture.



The first two blocks are used to define the variable value and the varied value of the variable. The other two blocks are rarely used.

On the left upper corner of the mBlock stage, there will be 

, where the number is the variable value. This is to facilitate users' observation of the variable value changing.

### Use of Variables

Variables are values that change when a program runs. A variable has a name and a value.

In mBlock, we can imagine that a variable is a box containing data. A program can store and extract the data within the box. The name of the box is "variable name" and the data in the box is "variable value". When you create a variable, the box is empty. When you set a variable value, you fill the box with data. When you change the value you take the data out and put into new data.

The logic of this example is as follows:

1. Define and create a variable `f`.
2. Use the block `set f to 500` to set the initial value of `f` to 500.
3. Use the block `play tone on note f beat 50` to define Variable `f` to the frequency value of the buzzer.
4. Use Block `change f by 20` to increase the value of Variable `f` by 20 each time, i.e. the frequency of the buzzer increases by 20Hz each time.
5. Use `repeat 50` to limit the number of changes of `f` to 50, and the final value of `f` is 500+20*50=1500Hz.

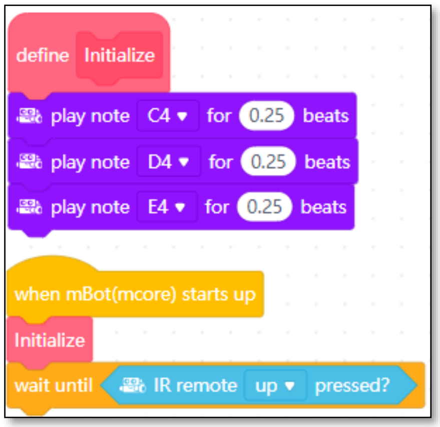
The buzzer sound effects of this example is: It first sounds 500Hz and then the frequency increases by 20Hz each time, i.e. the buzzer sounds respectively 500Hz, 520Hz, and 540Hz … (you can observe the change of the value in `f 0` under online debugging status), and each sound lasts for 50ms. It will stop sounding upon the 50th frequency increase.

**Note:** If the change in the value is set to `change f by -20` -20, the value of `f` will be decreased by 20 each time.

## Tutorial Assignment

### Create and Use a Block

A Block is a chunk of modular code that can be reused in the program without having to write the code again and again.



```
define Initialize
play note C4 for 0.25 beats
play note D4 for 0.25 beats
play note E4 for 0.25 beats

when mBot(mcore) starts up
Initialize
wait until IR remote up pressed?
```

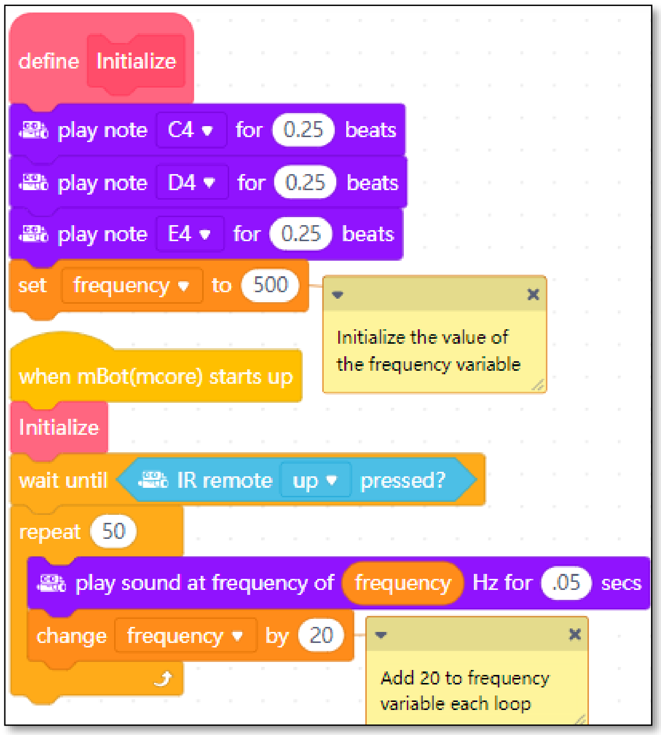
1. Start mBlock and save the program as **Sound Gradient**.
2. Go to **My Blocks**. Click **Make a Block**.
3. Name the block **Initialize**. Click OK.
4. Use the **define** block to create the code for the block. In this example the block is called **Initialize**.
5. Drag the other part of the block as shown in the example.
6. Upload and test the program. You should hear the three notes.
7. Turn the mBot off and then on. You should hear the three notes indicating the mBot is ready to go.

### Sound Gradient

The sound starts at a frequency of 500 hertz. Each time through the repeat loop the frequency increases by 20 hertz.

**NOTE:** Only use letters to start a variable or block name. Don't use spaces in a variable or block name.

1. Create and test the program as pictured.



```
define Initialize
play note C4 for 0.25 beats
play note D4 for 0.25 beats
play note E4 for 0.25 beats
set frequency to 500

when mBot(mcore) starts up
Initialize
wait until IR remote up pressed?
repeat 50
    play sound at frequency of frequency Hz for .05 secs
    change frequency by 20
```

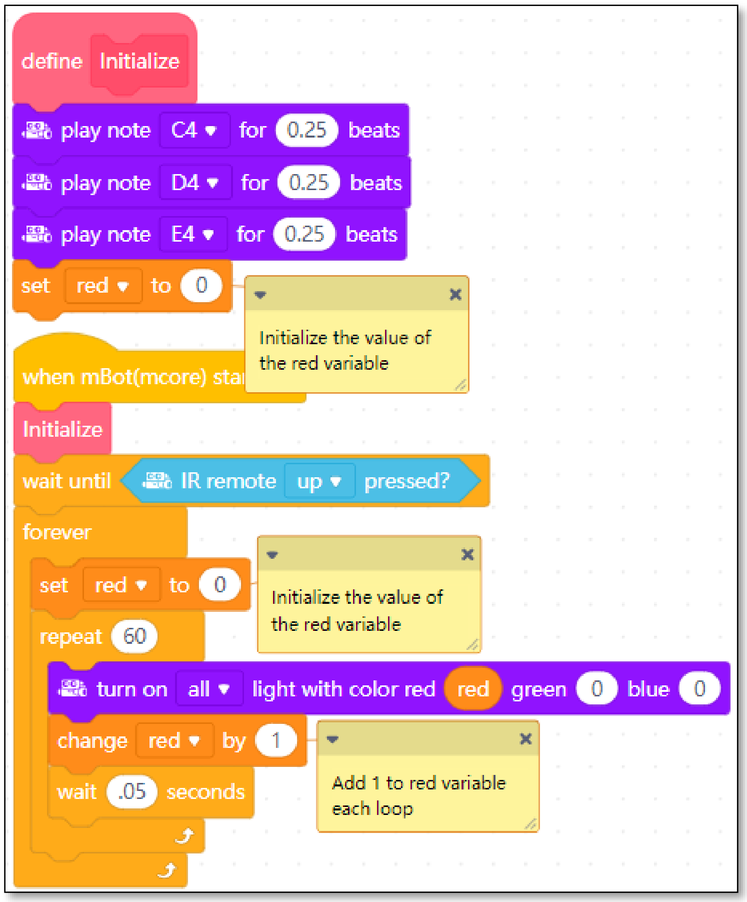
Initialize the value of the frequency variable

Add 20 to frequency variable each loop

## Brightness Gradient

The onboard LED lights start dark and the brightness increases gradually. When the brightness reaches a certain value, the onboard LED turns off. Repeat this process.

1. Save this program as **Brightness Gradient**.
2. Complete and test the program as pictured with the requirements listed.



```
define Initialize
play note C4 for 0.25 beats
play note D4 for 0.25 beats
play note E4 for 0.25 beats
set red to 0                    Initialize the value of
                                the red variable

when mBot(mcore) sta
Initialize
wait until IR remote up pressed?
forever
    set red to 0                Initialize the value of
                                the red variable
    repeat 60
        turn on all light with color red red green 0 blue 0
        change red by 1         Add 1 to red variable
                                each loop
        wait .05 seconds
```

## Requirements

- Complete and successfully run the programs as displayed.

## Challenge

- Change either program to loop until a different condition is met.
- Change something else based on a variable.

## Assignment Submission

- Attach the program and submit the assignment in Blackboard.
- Each assignment can be demonstrated in class. For online students, a link to a YouTube video recording showing the challenge part of the assignment can be placed in the submission area in BlackBoard.

## Movement with Functions

Time required: 60 minutes

Please read all the directions carefully before beginning the assignment.

1. Comment your code as shown in the tutorials and other code examples.
2. Follow all directions carefully and accurately.
3. Think of the directions as minimum requirements.

### Understanding

Demonstrate understanding of:

**functions**

We will use functions to modularize our code.

### Tutorial Assignment

1. Start the Arduino IDE. Create a new sketch called **MovementWFunctions**.
2. Complete and test the program as pictured with the requirements listed.

### Requirements

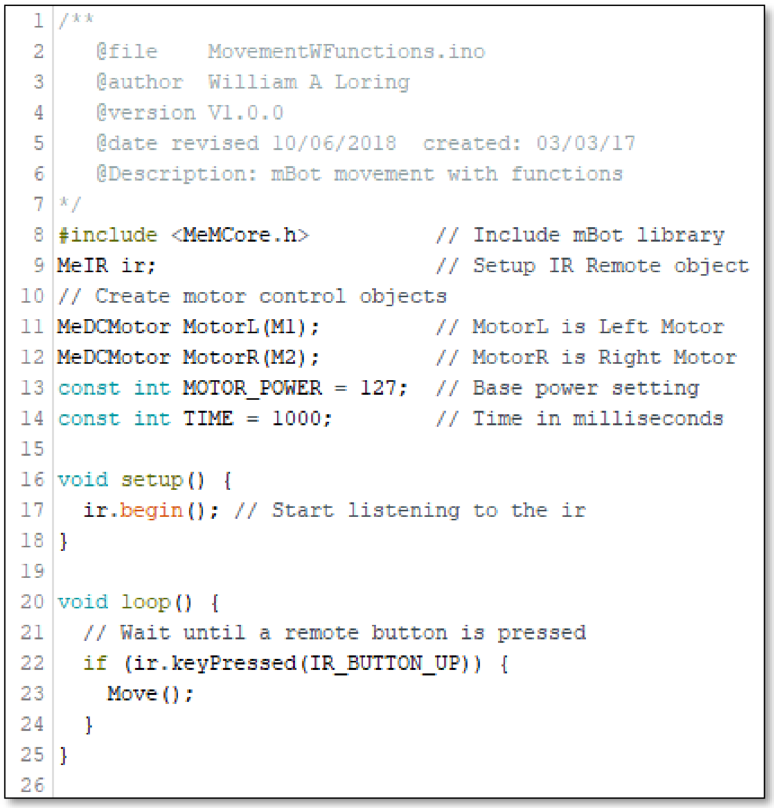
Complete and successfully run the program as shown.

### Challenge

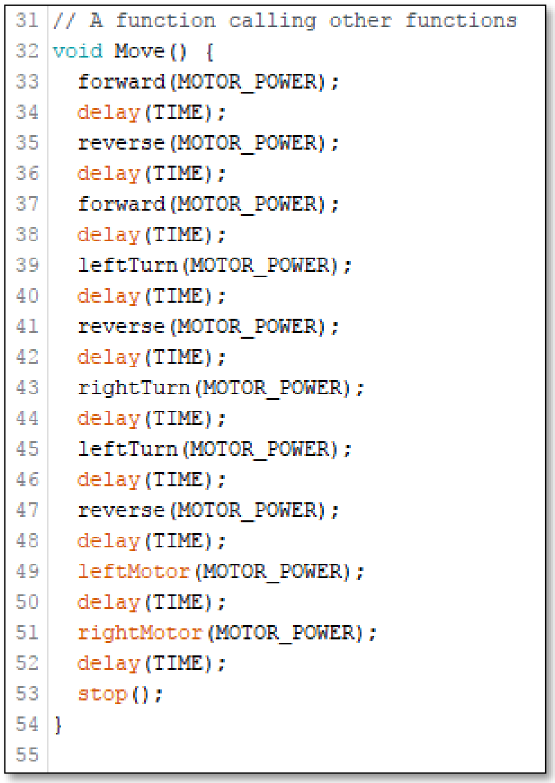
1. Add another function with a different remote button that calls a different combination of movements.

### Assignment Submission

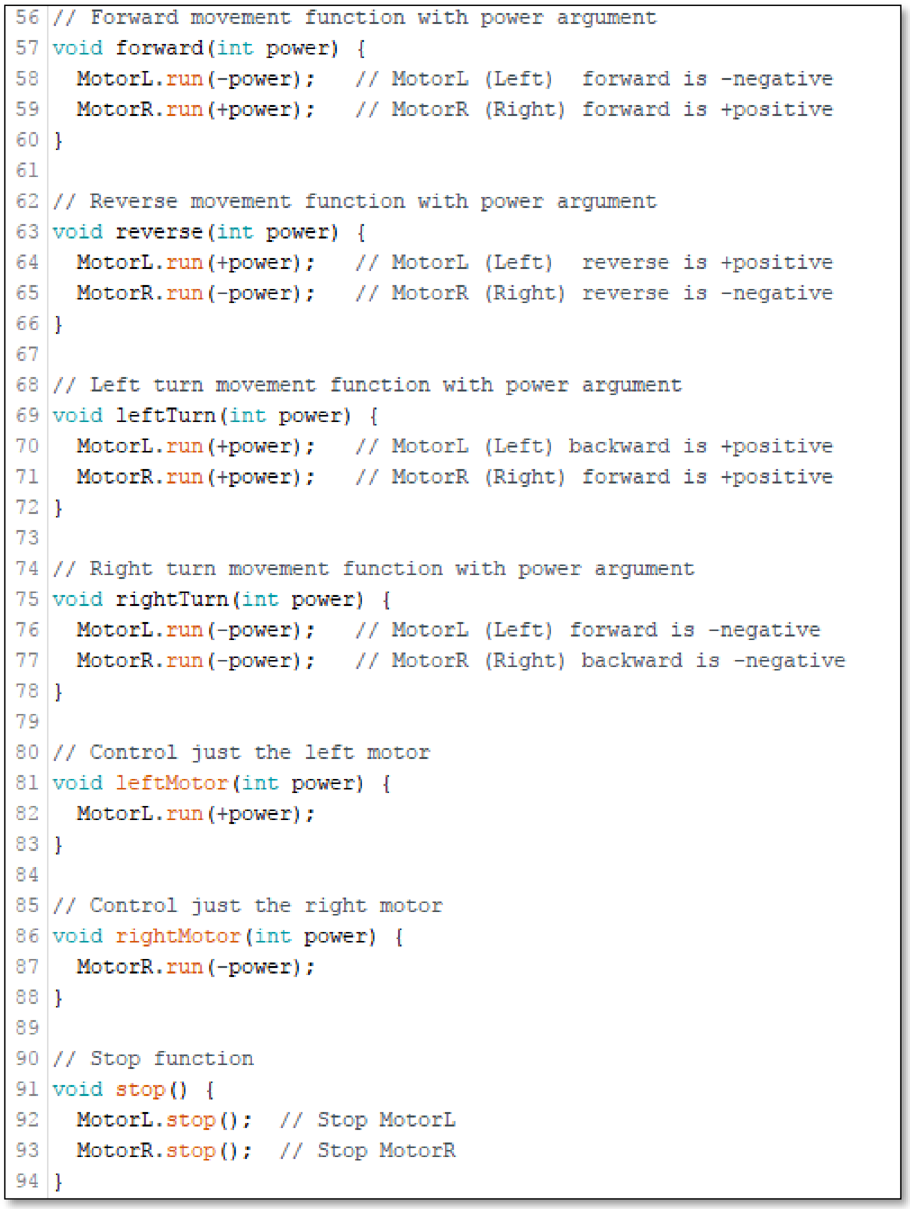
1. All students: Zip up the sketch folder. Attach the zip file to the assignment in Blackboard.
2. The assignment is demonstrated in class.
3. Online students: A link to a YouTube video recording showing your robot going through its motions is placed in the submission area in BlackBoard.



```
/**
   @file    MovementWFunctions.ino
   @author  William A Loring
   @version V1.0.0
   @date revised 10/06/2018  created: 03/03/17
   @Description: mBot movement with functions
*/
#include <MeMCore.h>          // Include mBot library
MeIR ir;                      // Setup IR Remote object
// Create motor control objects
MeDCMotor MotorL(M1);         // MotorL is Left Motor
MeDCMotor MotorR(M2);         // MotorR is Right Motor
const int MOTOR_POWER = 127;  // Base power setting
const int TIME = 1000;        // Time in milliseconds

void setup() {
  ir.begin(); // Start listening to the ir
}

void loop() {
  // Wait until a remote button is pressed
  if (ir.keyPressed(IR_BUTTON_UP)) {
    Move();
  }
}

```



```
// A function calling other functions
void Move() {
  forward(MOTOR_POWER);
  delay(TIME);
  reverse(MOTOR_POWER);
  delay(TIME);
  forward(MOTOR_POWER);
  delay(TIME);
  leftTurn(MOTOR_POWER);
  delay(TIME);
  reverse(MOTOR_POWER);
  delay(TIME);
  rightTurn(MOTOR_POWER);
  delay(TIME);
  leftTurn(MOTOR_POWER);
  delay(TIME);
  reverse(MOTOR_POWER);
  delay(TIME);
  leftMotor(MOTOR_POWER);
  delay(TIME);
  rightMotor(MOTOR_POWER);
  delay(TIME);
  stop();
}

```



```
// Forward movement function with power argument
void forward(int power) {
  MotorL.run(-power);   // MotorL (Left)  forward is -negative
  MotorR.run(+power);   // MotorR (Right) forward is +positive
}

// Reverse movement function with power argument
void reverse(int power) {
  MotorL.run(+power);   // MotorL (Left)  reverse is +positive
  MotorR.run(-power);   // MotorR (Right) reverse is -negative
}

// Left turn movement function with power argument
void leftTurn(int power) {
  MotorL.run(+power);   // MotorL (Left) backward is +positive
  MotorR.run(+power);   // MotorR (Right) forward is +positive
}

// Right turn movement function with power argument
void rightTurn(int power) {
  MotorL.run(-power);   // MotorL (Left) forward is -negative
  MotorR.run(-power);   // MotorR (Right) backward is -negative
}

// Control just the left motor
void leftMotor(int power) {
  MotorL.run(+power);
}

// Control just the right motor
void rightMotor(int power) {
  MotorR.run(-power);
}

// Stop function
void stop() {
  MotorL.stop();  // Stop MotorL
  MotorR.stop();  // Stop MotorR
}
```

## Calibrate Movement

Time required: 60 minutes

Please read all the directions carefully before beginning the assignment.

1. Comment your code as shown in the tutorials and other code examples.
2. Follow all directions carefully and accurately.
3. Think of the directions as minimum requirements.

### Understanding

Demonstrate understanding of:

**variables, constants, if else if**

### Knowledge Points

We are going to calibrate the robot to go a certain distance and turn accurately. The battery charge can make a difference, make sure the batteries are charged. The surface the robot is traveling on will make a difference. You may want to keep track of the settings for each surface. The sample program settings are for a smooth surface.

**NOTE:** The mBot is not an accurate robot. As the batteries discharge and the conditions change, it will behave differently. The only things we can change is power and time. Just try to get close.

This program will allow us to calibrate our mBot for the following:

1. Driving straight by adjust the comp variable.
2. Distance by adjusting the time variable to drive 48”.
3. Turns by adjusting turnTime360 to have the mBot start and start in approximately the same orientation while making a square.

### Tutorial Assignment

1. Save the sketch as **CalibrateMovement**
2. Complete and test the program as shown.

### Requirements

1. Use your mBlock program settings as a starting point.

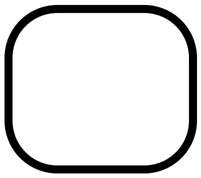
2. **Create** a program that has a block that move forward 48”, does a square turning to the right, and another square turning to the left.
3. Vary the compensation and time in milliseconds to make an exact square.
4. When doing the square, the robot should end where it started.

| | **Power** | **90$^o$** |
|---|---|---|
| Left | 127 * 1.1 | 520 |
| Right | 127 | |

- Use the arrow keys on the remote to drive forward and start the squares.

5. Turning while moving changes the turning rate, you may have to recalibrate your 360TurnTime.

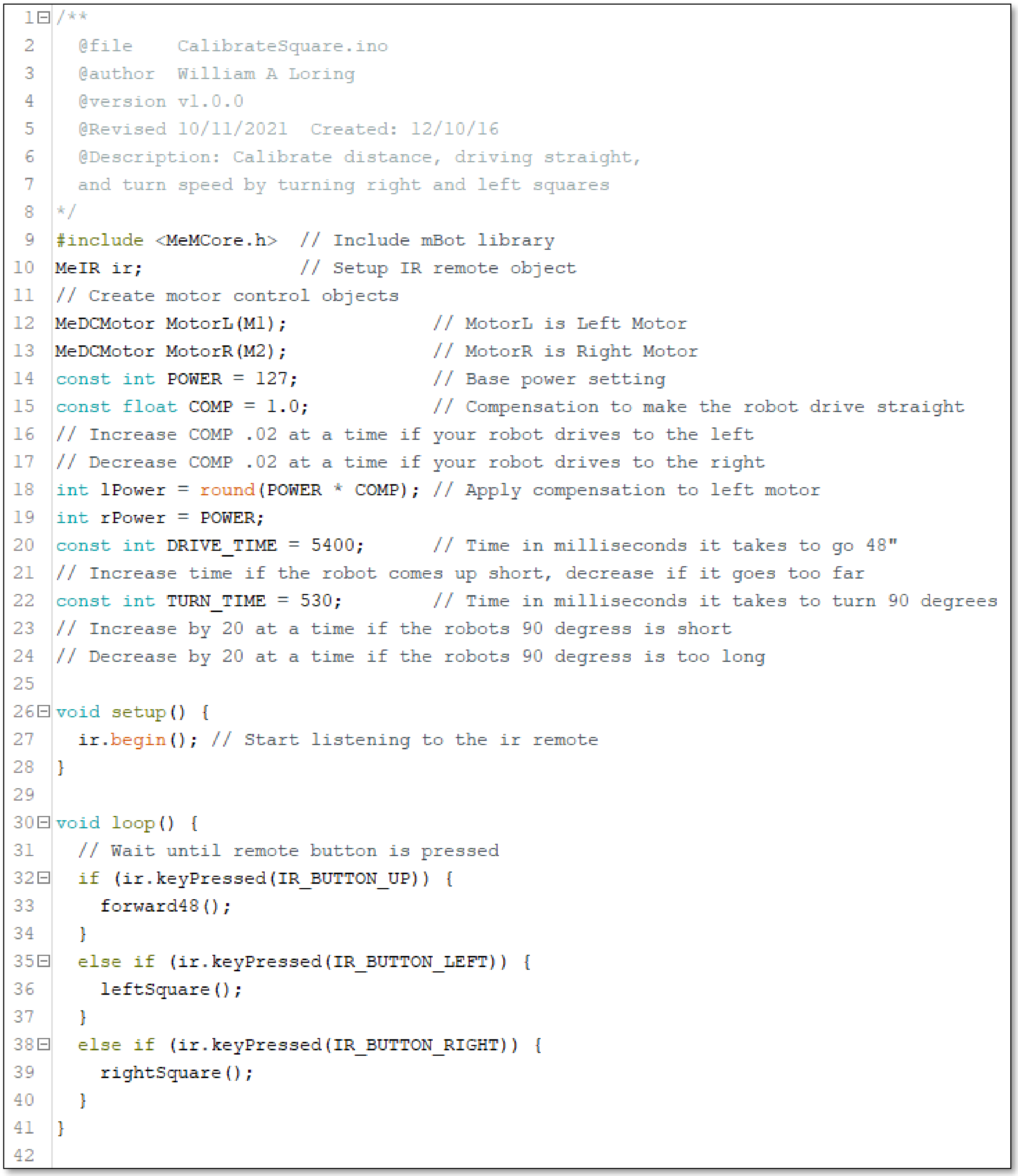
## Extra Credit Challenge Assignment



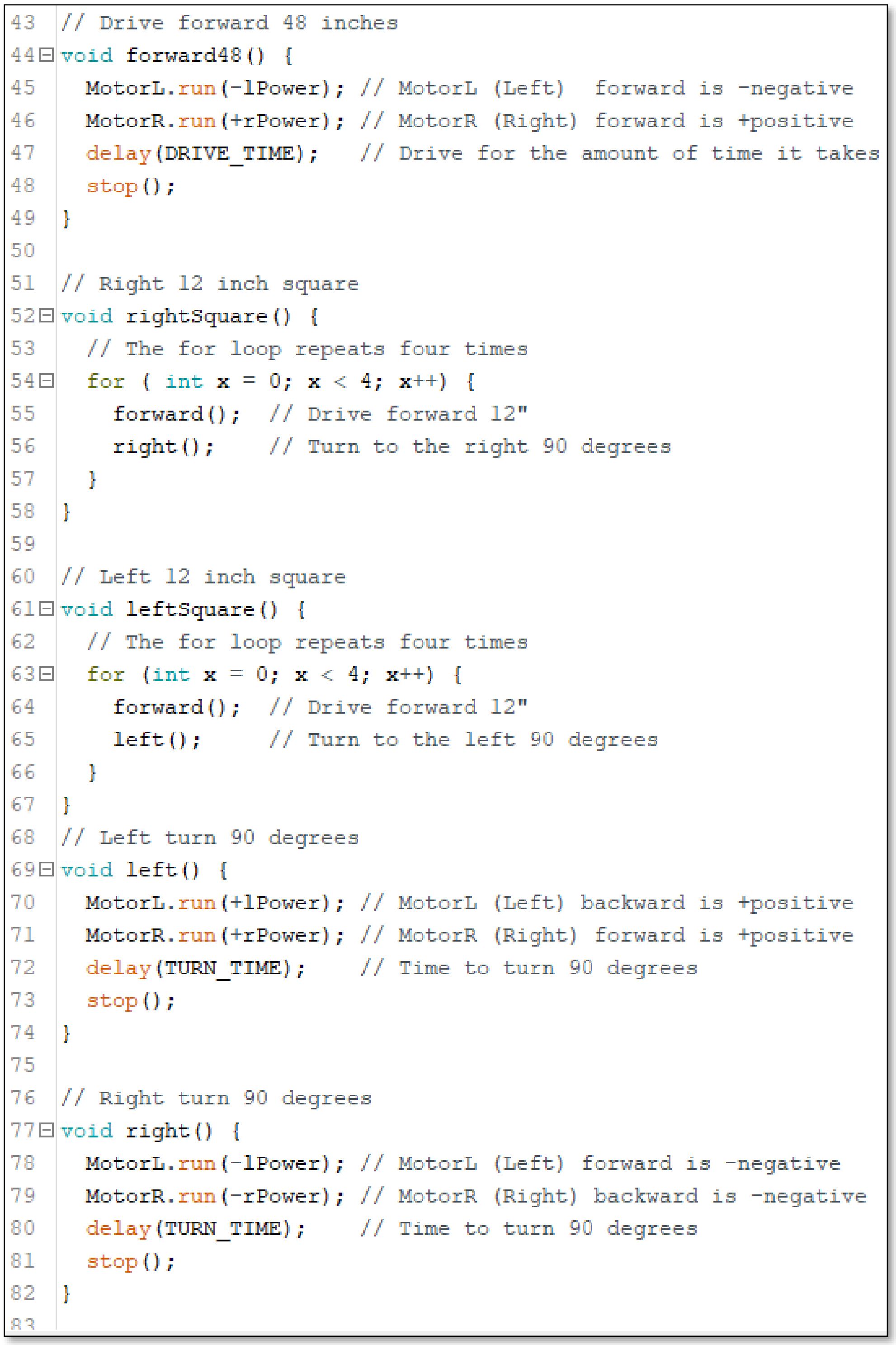
1. Add a **new** function to calibrate a right car turn.
2. Create a new turning function to trace a right handed square with rounded corners. Combine a differential turn with a 12” straight line and loop it four times.
3. **Comment** your challenge code.

## Assignment Submission

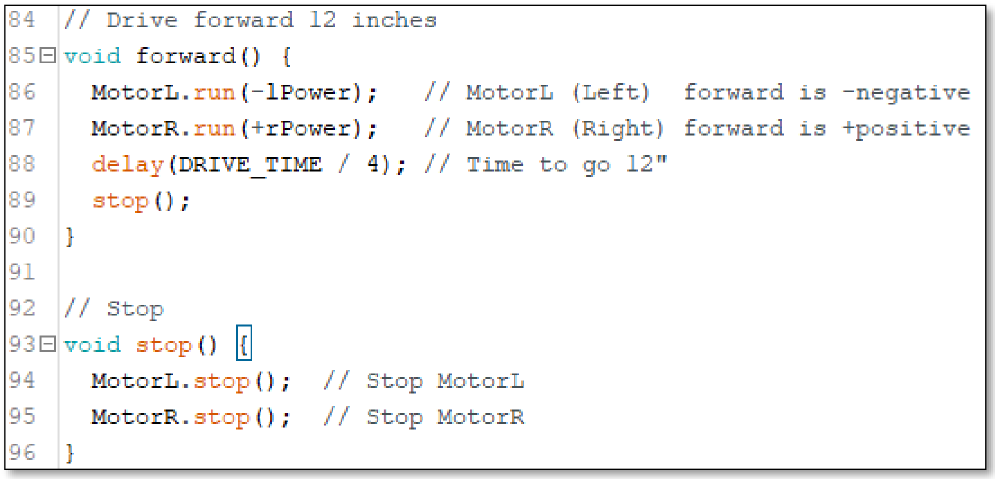
1. All students: Zip up the sketch folder. Attach the zip file to the assignment in Blackboard.
2. The assignment is demonstrated in class.
3. Online students: A link to a YouTube video recording showing your robot going through its motions is placed in the submission area in BlackBoard.



```
/**
  @file    CalibrateSquare.ino
  @author  William A Loring
  @version v1.0.0
  @Revised 10/11/2021  Created: 12/10/16
  @Description: Calibrate distance, driving straight,
  and turn speed by turning right and left squares
*/
#include <MeMCore.h>  // Include mBot library
MeIR ir;              // Setup IR remote object
// Create motor control objects
MeDCMotor MotorL(M1);             // MotorL is Left Motor
MeDCMotor MotorR(M2);             // MotorR is Right Motor
const int POWER = 127;            // Base power setting
const float COMP = 1.0;           // Compensation to make the robot drive straight
// Increase COMP .02 at a time if your robot drives to the left
// Decrease COMP .02 at a time if your robot drives to the right
int lPower = round(POWER * COMP); // Apply compensation to left motor
int rPower = POWER;
const int DRIVE_TIME = 5400;      // Time in milliseconds it takes to go 48"
// Increase time if the robot comes up short, decrease if it goes too far
const int TURN_TIME = 530;        // Time in milliseconds it takes to turn 90 degrees
// Increase by 20 at a time if the robots 90 degress is short
// Decrease by 20 at a time if the robots 90 degress is too long

void setup() {
  ir.begin(); // Start listening to the ir remote
}

void loop() {
  // Wait until remote button is pressed
  if (ir.keyPressed(IR_BUTTON_UP)) {
    forward48();
  }
  else if (ir.keyPressed(IR_BUTTON_LEFT)) {
    leftSquare();
  }
  else if (ir.keyPressed(IR_BUTTON_RIGHT)) {
    rightSquare();
  }
}

```



```
// Drive forward 48 inches
void forward48() {
  MotorL.run(-lPower); // MotorL (Left)  forward is -negative
  MotorR.run(+rPower); // MotorR (Right) forward is +positive
  delay(DRIVE_TIME);   // Drive for the amount of time it takes
  stop();
}

// Right 12 inch square
void rightSquare() {
  // The for loop repeats four times
  for ( int x = 0; x < 4; x++) {
    forward();  // Drive forward 12"
    right();    // Turn to the right 90 degrees
  }
}

// Left 12 inch square
void leftSquare() {
  // The for loop repeats four times
  for (int x = 0; x < 4; x++) {
    forward();  // Drive forward 12"
    left();     // Turn to the left 90 degrees
  }
}
// Left turn 90 degrees
void left() {
  MotorL.run(+lPower); // MotorL (Left) backward is +positive
  MotorR.run(+rPower); // MotorR (Right) forward is +positive
  delay(TURN_TIME);    // Time to turn 90 degrees
  stop();
}

// Right turn 90 degrees
void right() {
  MotorL.run(-lPower); // MotorL (Left) forward is -negative
  MotorR.run(-rPower); // MotorR (Right) backward is -negative
  delay(TURN_TIME);    // Time to turn 90 degrees
  stop();
}
```



```
// Drive forward 12 inches
void forward() {
  MotorL.run(-lPower);   // MotorL (Left)  forward is -negative
  MotorR.run(+rPower);   // MotorR (Right) forward is +positive
  delay(DRIVE_TIME / 4); // Time to go 12"
  stop();
}

// Stop
void stop() {
  MotorL.stop();  // Stop MotorL
  MotorR.stop();  // Stop MotorR
}
```

## Accurate Movement (We Like to Move It!)

Time required: 75 minutes

Please read all the directions carefully before beginning the assignment.

1. Comment your code as shown in the tutorials and other code examples.
2. Follow all directions carefully and accurately.
3. Think of the directions as minimum requirements.

### Understanding

Demonstrate understanding of:

**libraries, functions**

We know how much time it takes to move a certain distance at a certain power. We can input the distance for accurate movement. We can also calculate the amount of time it take to turn a specific angle.

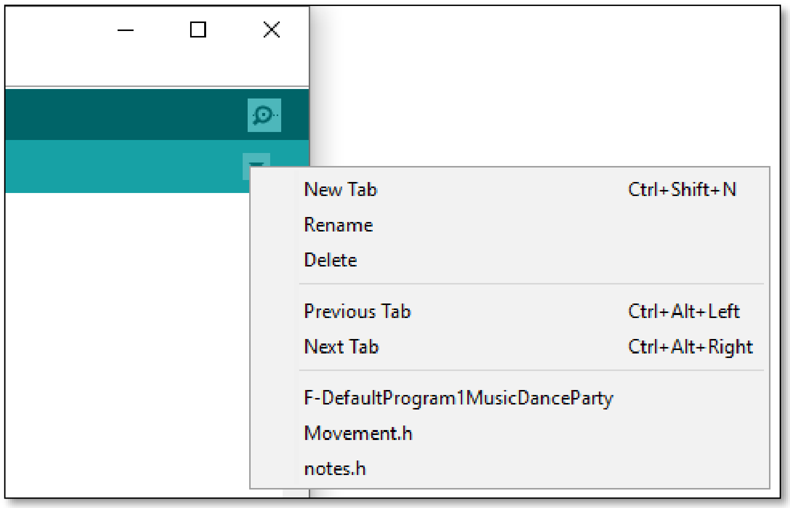
We will create a resuable header file called Movement.h. We will use this file to store our movement code and copy it from sketch to sketch. This allows for easily resuable code.

avgSpeed (inches per second) = (Distance(inches) / Time)

Example: (4'/7400) = 6.5 inches per second

### Tutorial Assignment

1. Open **CalibrateMovement**. Save the sketch as **AccurateMovement**.
2. On the right side of the Arduino IDE, click the down triangle → Click **New Tab** → **Filename** → **Movement.h** Click OK.



3. Cut and paste the code from the top of the main ino file to **Movement.h**. Look at the code at the end of this document to tell which code to copy and paste.
4. You can delete the code at the bottom of the ino file.
5. Complete and test the program as pictured with the requirements listed.

## Requirements

1. Create a function that tests each of the movements.

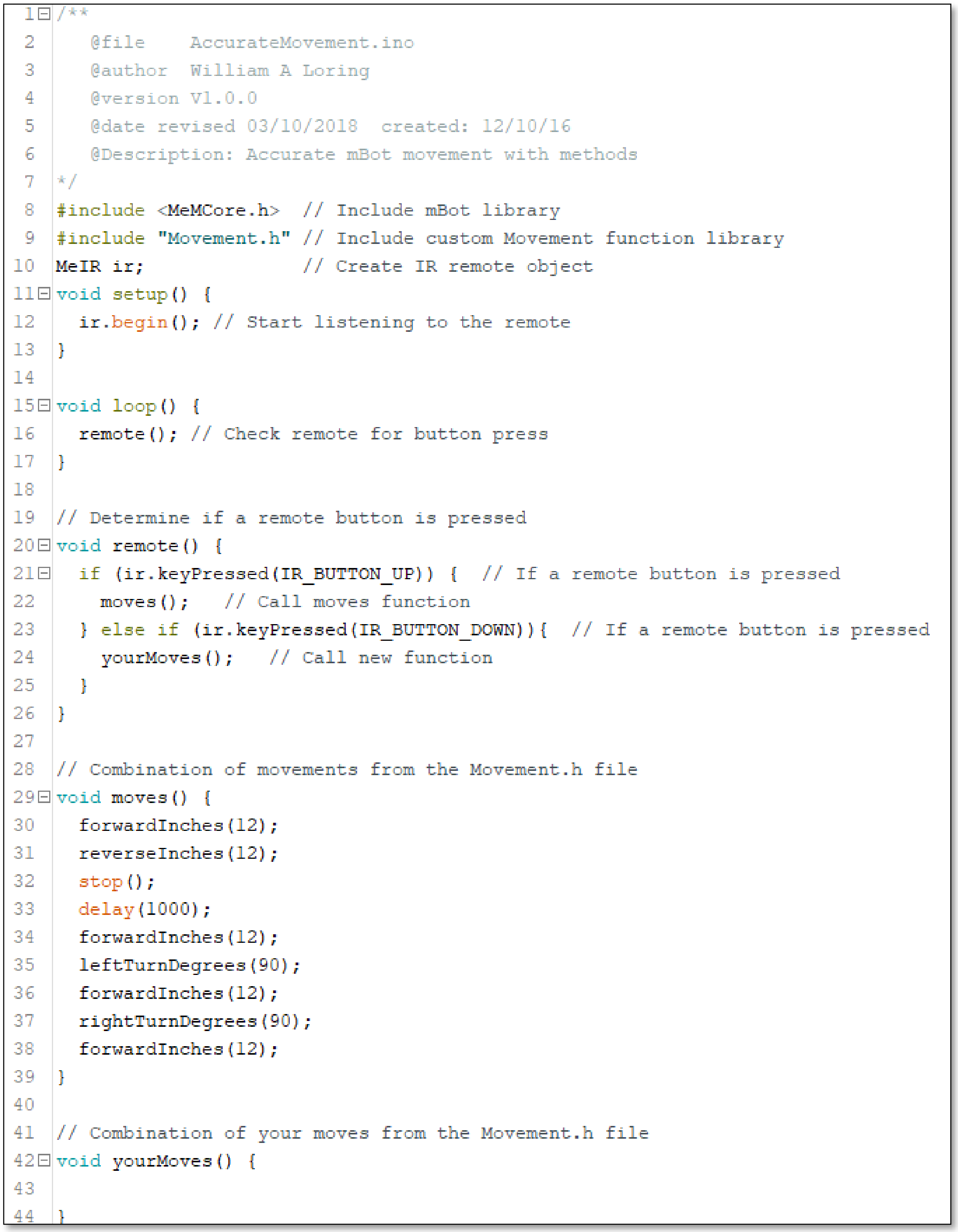
## Challenge

1. Create another function in the main sketch similar to the move function with different moves from the **Movement.h** file.
2. Use a different remote key to trigger the new function.
3. The following code is how to call another function with another key. You can name your second function as you wish. An example is in the code at the bottom of this assignment.

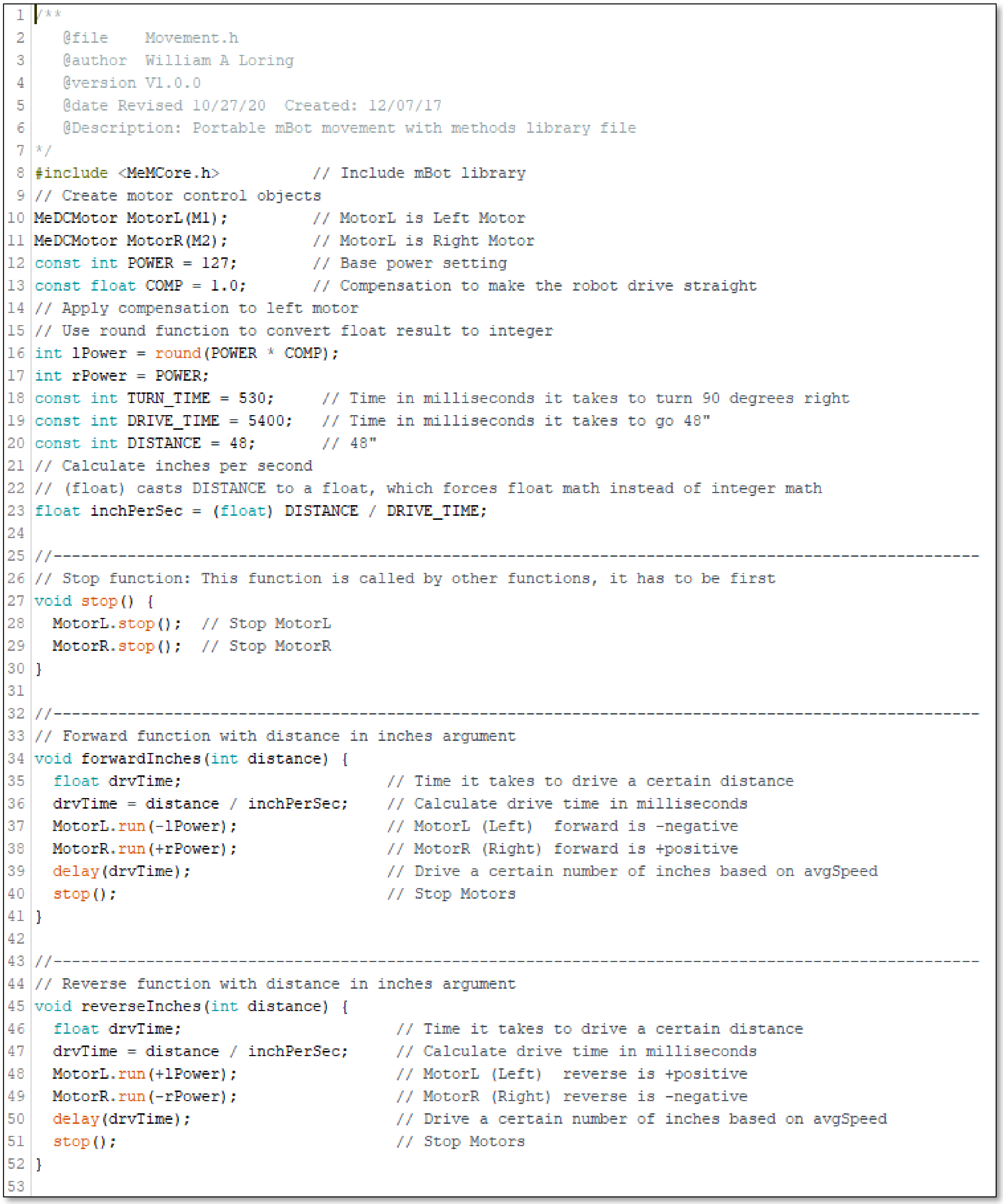
```
// Determine if a remote button is pressed
void remote() {
  if (ir.keyPressed(IR_BUTTON_UP)) {  // If a remote button is pressed
    move();   // Call move function
    else if (ir.keyPressed(IR_BUTTON_DOWN))  // If a remote button is pressed
    moove();   // Call move function
  }
}
```

## Assignment Submission

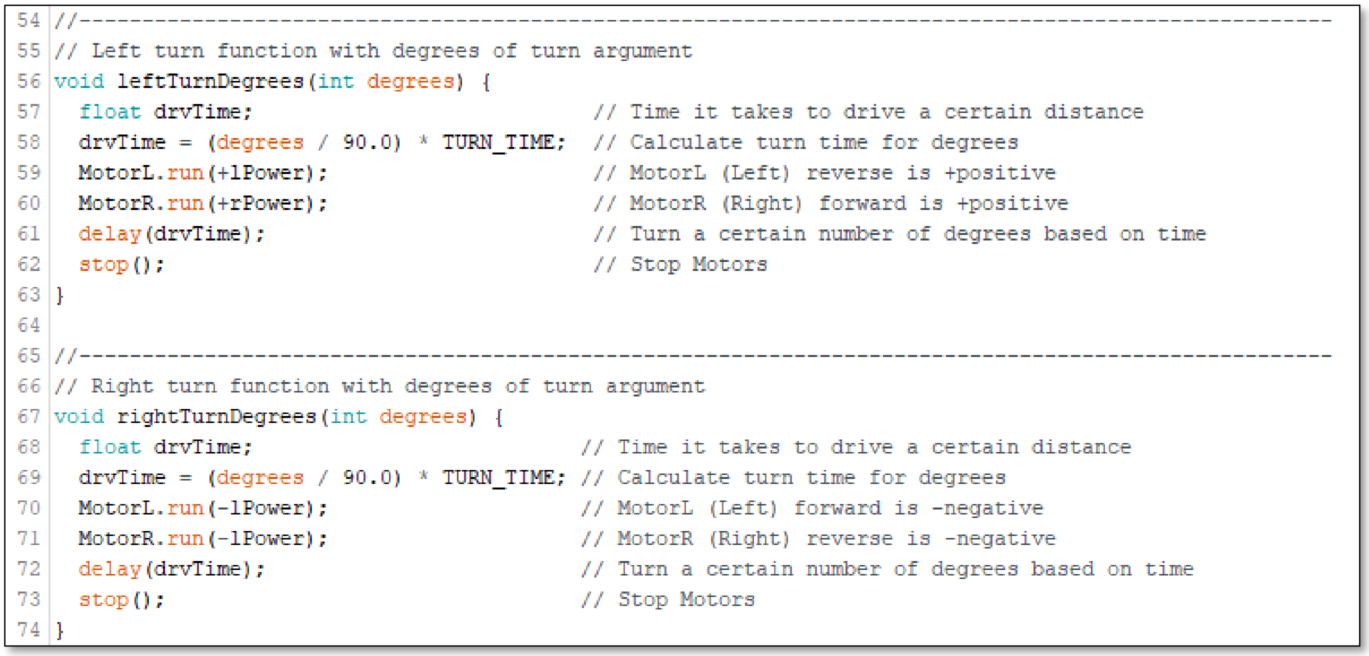
1. All students: Zip up the sketch folder. Attach the zip file to the assignment in Blackboard.
2. The assignment is demonstrated in class.
3. Online students: A link to a YouTube video recording showing your robot going through its motions is placed in the submission area in BlackBoard.



```
/**
   @file    AccurateMovement.ino
   @author  William A Loring
   @version V1.0.0
   @date revised 03/10/2018  created: 12/10/16
   @Description: Accurate mBot movement with methods
*/
#include <MeMCore.h>  // Include mBot library
#include "Movement.h" // Include custom Movement function library
MeIR ir;              // Create IR remote object
void setup() {
  ir.begin(); // Start listening to the remote
}

void loop() {
  remote(); // Check remote for button press
}

// Determine if a remote button is pressed
void remote() {
  if (ir.keyPressed(IR_BUTTON_UP)) {  // If a remote button is pressed
    moves();   // Call moves function
  } else if (ir.keyPressed(IR_BUTTON_DOWN)){  // If a remote button is pressed
    yourMoves();   // Call new function
  }
}

// Combination of movements from the Movement.h file
void moves() {
  forwardInches(12);
  reverseInches(12);
  stop();
  delay(1000);
  forwardInches(12);
  leftTurnDegrees(90);
  forwardInches(12);
  rightTurnDegrees(90);
  forwardInches(12);
}

// Combination of your moves from the Movement.h file
void yourMoves() {

}
```



```
/**
   @file    Movement.h
   @author  William A Loring
   @version V1.0.0
   @date Revised 10/27/20  Created: 12/07/17
   @Description: Portable mBot movement with methods library file
*/
#include <MeMCore.h>          // Include mBot library
// Create motor control objects
MeDCMotor MotorL(M1);         // MotorL is Left Motor
MeDCMotor MotorR(M2);         // MotorL is Right Motor
const int POWER = 127;        // Base power setting
const float COMP = 1.0;       // Compensation to make the robot drive straight
// Apply compensation to left motor
// Use round function to convert float result to integer
int lPower = round(POWER * COMP);
int rPower = POWER;
const int TURN_TIME = 530;     // Time in milliseconds it takes to turn 90 degrees right
const int DRIVE_TIME = 5400;   // Time in milliseconds it takes to go 48"
const int DISTANCE = 48;       // 48"
// Calculate inches per second
// (float) casts DISTANCE to a float, which forces float math instead of integer math
float inchPerSec = (float) DISTANCE / DRIVE_TIME;

//----------------------------------------------------------------------------------------------------
// Stop function: This function is called by other functions, it has to be first
void stop() {
  MotorL.stop();  // Stop MotorL
  MotorR.stop();  // Stop MotorR
}

//----------------------------------------------------------------------------------------------------
// Forward function with distance in inches argument
void forwardInches(int distance) {
  float drvTime;                     // Time it takes to drive a certain distance
  drvTime = distance / inchPerSec;   // Calculate drive time in milliseconds
  MotorL.run(-lPower);               // MotorL (Left)  forward is -negative
  MotorR.run(+rPower);               // MotorR (Right) forward is +positive
  delay(drvTime);                    // Drive a certain number of inches based on avgSpeed
  stop();                            // Stop Motors
}

//----------------------------------------------------------------------------------------------------
// Reverse function with distance in inches argument
void reverseInches(int distance) {
  float drvTime;                      // Time it takes to drive a certain distance
  drvTime = distance / inchPerSec;    // Calculate drive time in milliseconds
  MotorL.run(+lPower);                // MotorL (Left)  reverse is +positive
  MotorR.run(-rPower);                // MotorR (Right) reverse is -negative
  delay(drvTime);                     // Drive a certain number of inches based on avgSpeed
  stop();                             // Stop Motors
}

```



```
//-----------------------------------------------------------------------------------------------
// Left turn function with degrees of turn argument
void leftTurnDegrees(int degrees) {
  float drvTime;                           // Time it takes to drive a certain distance
  drvTime = (degrees / 90.0) * TURN_TIME;  // Calculate turn time for degrees
  MotorL.run(+lPower);                     // MotorL (Left) reverse is +positive
  MotorR.run(+rPower);                     // MotorR (Right) forward is +positive
  delay(drvTime);                          // Turn a certain number of degrees based on time
  stop();                                  // Stop Motors
}

//-----------------------------------------------------------------------------------------------
// Right turn function with degrees of turn argument
void rightTurnDegrees(int degrees) {
  float drvTime;                          // Time it takes to drive a certain distance
  drvTime = (degrees / 90.0) * TURN_TIME; // Calculate turn time for degrees
  MotorL.run(-lPower);                    // MotorL (Left) forward is -negative
  MotorR.run(-lPower);                    // MotorR (Right) reverse is -negative
  delay(drvTime);                         // Turn a certain number of degrees based on time
  stop();                                 // Stop Motors
}
```

## Ultrasonic Sensor Test

Time required: 10 minutes

Please read all the directions carefully before beginning the assignment.

1. Comment your code as shown in the tutorials and other code examples.
2. Follow all directions carefully and accurately.
3. Think of the directions as minimum requirements.

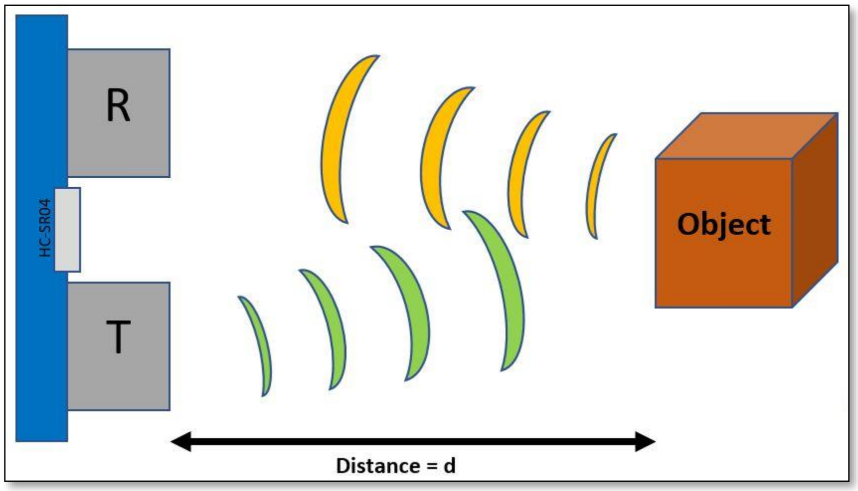
### Understanding

Demonstrate understanding of:

**ultrasonic sensor, serial monitor**

### Knowledge Points

The robot has an ultrasonic sensor on the front that detects objects and how far away they are. One "eye" emits ultrasonic sound waves while the other receives the signals bounced back. The distance is calculated based on how long it takes for the sound to return, much like a sonar



### Tutorial Assignment

Test the ultrasonic sensor in inches or cm. Go to **Tools → Serial Monitor** to view the feedback.

1. Start the Arduino IDE. Save the sketch as **UltrasonicSensorTest**.

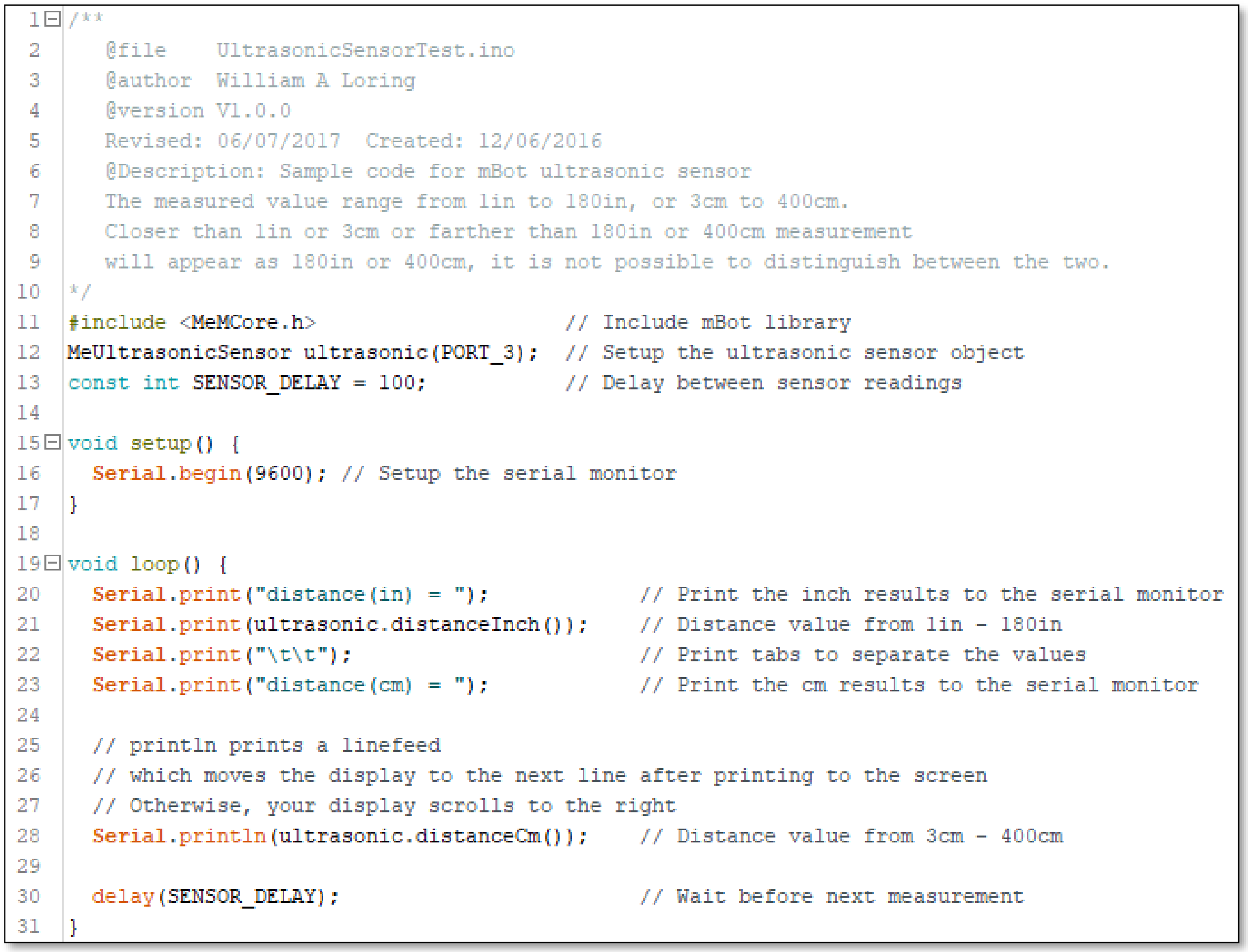
2. Complete and test the program as shown.

## Assignment Submission

1. All students: Zip up the sketch folder. Attach the zip file to the assignment in Blackboard.

2. The assignment is demonstrated in class.

3. Online students: A link to a YouTube video recording showing your robot going through its motions is placed in the submission area in BlackBoard.



```
/**
   @file    UltrasonicSensorTest.ino
   @author  William A Loring
   @version V1.0.0
   Revised: 06/07/2017  Created: 12/06/2016
   @Description: Sample code for mBot ultrasonic sensor
   The measured value range from 1in to 180in, or 3cm to 400cm.
   Closer than 1in or 3cm or farther than 180in or 400cm measurement
   will appear as 180in or 400cm, it is not possible to distinguish between the two.
*/
#include <MeMCore.h>                    // Include mBot library
MeUltrasonicSensor ultrasonic(PORT_3);  // Setup the ultrasonic sensor object
const int SENSOR_DELAY = 100;           // Delay between sensor readings

void setup() {
  Serial.begin(9600); // Setup the serial monitor
}

void loop() {
  Serial.print("distance(in) = ");            // Print the inch results to the serial monitor
  Serial.print(ultrasonic.distanceInch());    // Distance value from 1in - 180in
  Serial.print("\t\t");                       // Print tabs to separate the values
  Serial.print("distance(cm) = ");            // Print the cm results to the serial monitor

  // println prints a linefeed
  // which moves the display to the next line after printing to the screen
  // Otherwise, your display scrolls to the right
  Serial.println(ultrasonic.distanceCm());    // Distance value from 3cm - 400cm

  delay(SENSOR_DELAY);                        // Wait before next measurement
}
```

## Simple Obstacle Avoidance

Time required: 30 minutes

Please read all the directions carefully before beginning the assignment.

1. Comment your code as shown in the tutorials and other code examples.
2. Follow all directions carefully and accurately.
3. Think of the directions as minimum requirements.

### Understanding

Demonstrate understanding of:

**ultrasonic sensor, if statements**

### Tutorial Assignment

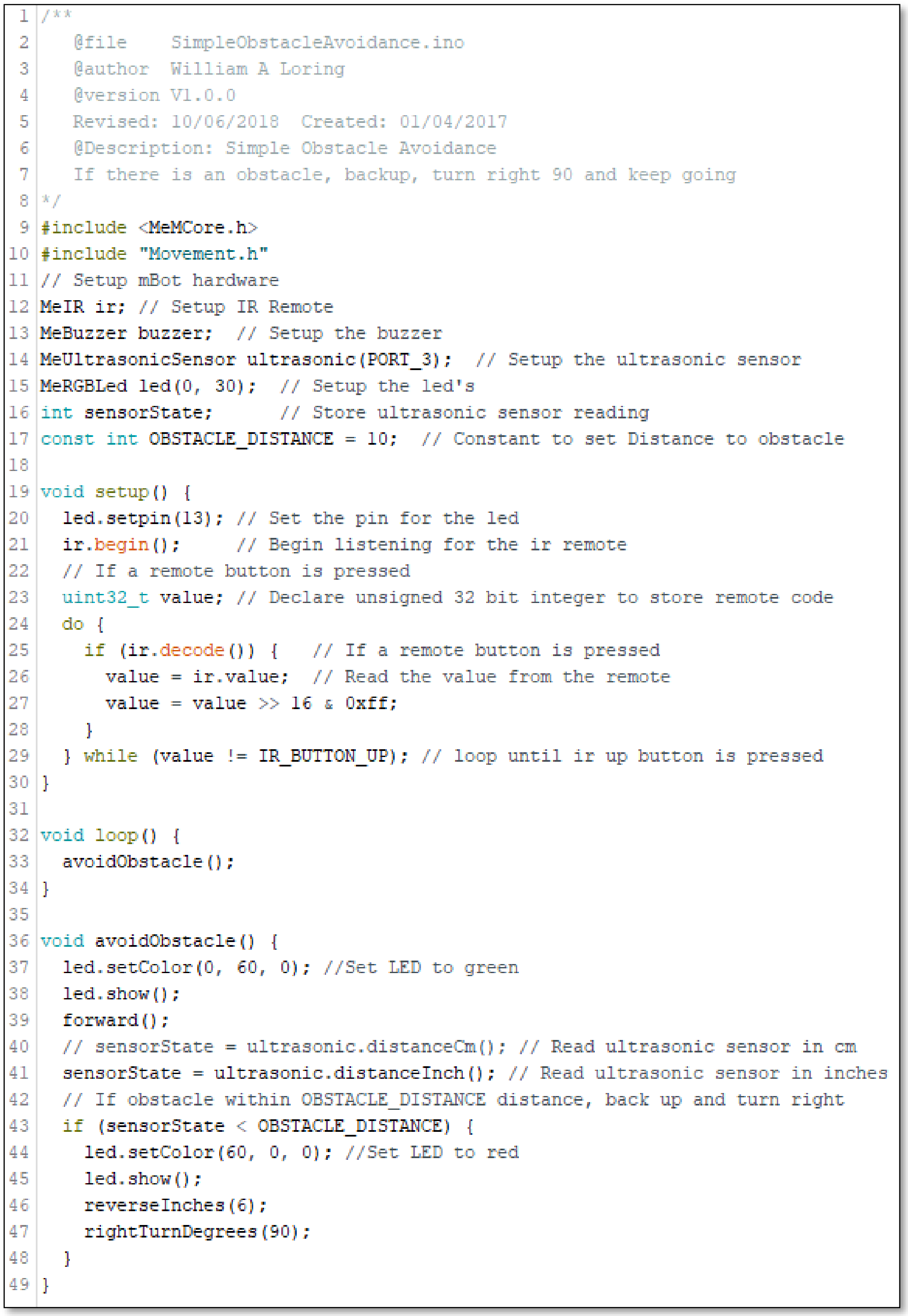
1. Start the Arduino IDE. Save the sketch as **SimpleObstacleAvoidance**.
2. Complete and test the program as pictured with the requirements listed.
3. Comment your code.

### Requirements

- Avoid obstacles by backing up, turning right, then continue moving.
- Include **Movement.h**.

### Assignment Submission

1. All students: Zip up the sketch folder. Attach the zip file to the assignment in Blackboard.
2. The assignment is demonstrated in class.
3. Online students: A link to a YouTube video recording showing your robot going through its motions is placed in the submission area in BlackBoard.



```
/**
   @file    SimpleObstacleAvoidance.ino
   @author  William A Loring
   @version V1.0.0
   Revised: 10/06/2018  Created: 01/04/2017
   @Description: Simple Obstacle Avoidance
   If there is an obstacle, backup, turn right 90 and keep going
*/
#include <MeMCore.h>
#include "Movement.h"
// Setup mBot hardware
MeIR ir; // Setup IR Remote
MeBuzzer buzzer;  // Setup the buzzer
MeUltrasonicSensor ultrasonic(PORT_3);  // Setup the ultrasonic sensor
MeRGBLed led(0, 30);  // Setup the led's
int sensorState;      // Store ultrasonic sensor reading
const int OBSTACLE_DISTANCE = 10;  // Constant to set Distance to obstacle

void setup() {
  led.setpin(13); // Set the pin for the led
  ir.begin();     // Begin listening for the ir remote
  // If a remote button is pressed
  uint32_t value; // Declare unsigned 32 bit integer to store remote code
  do {
    if (ir.decode()) {   // If a remote button is pressed
      value = ir.value;  // Read the value from the remote
      value = value >> 16 & 0xff;
    }
  } while (value != IR_BUTTON_UP); // loop until ir up button is pressed
}

void loop() {
  avoidObstacle();
}

void avoidObstacle() {
  led.setColor(0, 60, 0); //Set LED to green
  led.show();
  forward();
  // sensorState = ultrasonic.distanceCm(); // Read ultrasonic sensor in cm
  sensorState = ultrasonic.distanceInch(); // Read ultrasonic sensor in inches
  // If obstacle within OBSTACLE_DISTANCE distance, back up and turn right
  if (sensorState < OBSTACLE_DISTANCE) {
    led.setColor(60, 0, 0); //Set LED to red
    led.show();
    reverseInches(6);
    rightTurnDegrees(90);
  }
}
```

## Light Sensor Test

Time required: 15 minutes

Please read all the directions carefully before beginning the assignment.

- Comment your code as shown in the tutorials and other code examples.
- Follow all directions carefully and accurately.
- Think of the directions as minimum requirements.

### Understanding

Demonstrate understanding of:

**light sensor, serial monitor**

### Requirements

The mBot has a light sensor which can be used to create interactivity with the robot. The onboard light sensor has a sensitivity of 0 (dark) - 1024 (light).

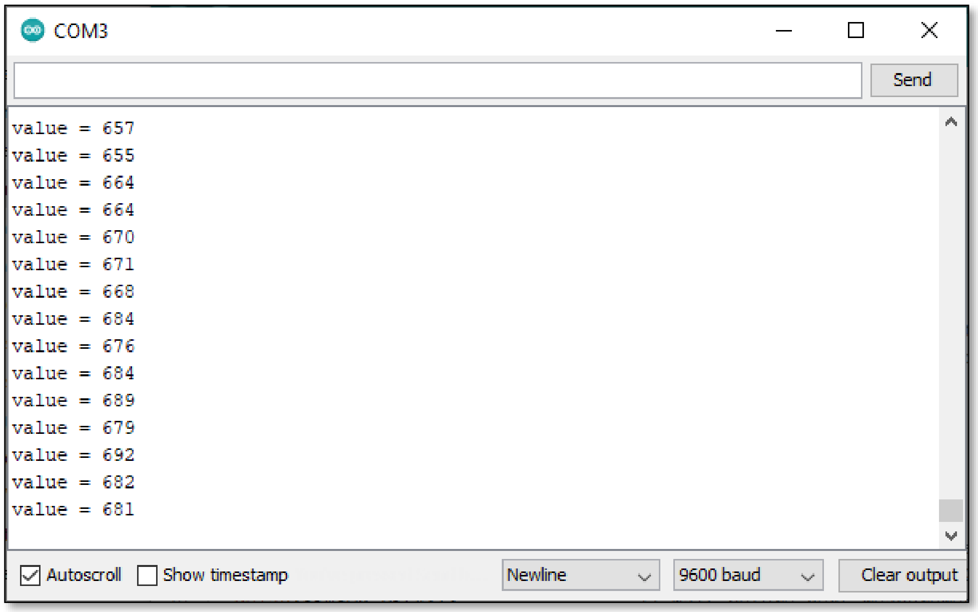
This program uses the serial monitor to show the readings coming from the light sensor.

### Tutorial Assignment

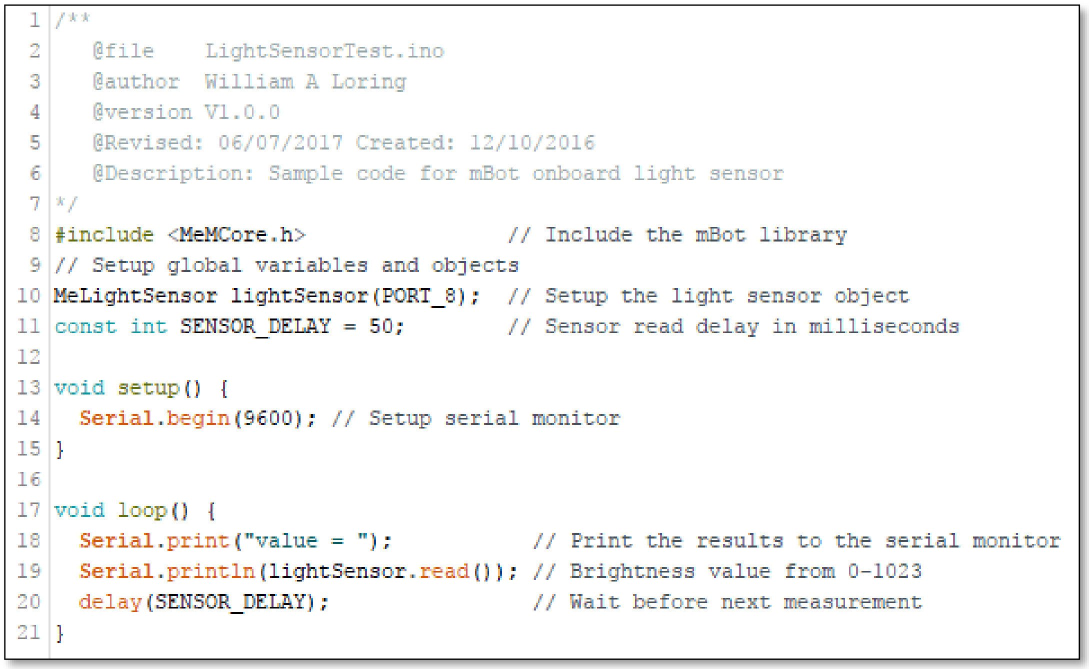
1. Start the Arduino IDE. Save the sketch as **LightSensorTest**.
2. Complete and test the program as shown.
3. While the sketch is running: In the Arduino IDE, go to **Tools → Serial Monitor** to display the real time reading from the light sensor. Move your hand back and forth on top of the mBot. Notice the number changes.
4. Please include the serial monitor in your screencast.

### Assignment Submission

1. All students: Zip up the sketch folder. Attach the zip file to the assignment in Blackboard.
2. The assignment is demonstrated in class.
3. Online students: A link to a YouTube video recording showing your robot going through its motions is placed in the submission area in BlackBoard.



```
value = 657
value = 655
value = 664
value = 664
value = 670
value = 671
value = 668
value = 684
value = 676
value = 684
value = 689
value = 679
value = 692
value = 682
value = 681
```



```
/**
   @file    LightSensorTest.ino
   @author  William A Loring
   @version V1.0.0
   @Revised: 06/07/2017 Created: 12/10/2016
   @Description: Sample code for mBot onboard light sensor
*/
#include <MeMCore.h>                    // Include the mBot library
// Setup global variables and objects
MeLightSensor lightSensor(PORT_8);  // Setup the light sensor object
const int SENSOR_DELAY = 50;        // Sensor read delay in milliseconds

void setup() {
  Serial.begin(9600); // Setup serial monitor
}

void loop() {
  Serial.print("value = ");          // Print the results to the serial monitor
  Serial.println(lightSensor.read()); // Brightness value from 0-1023
  delay(SENSOR_DELAY);               // Wait before next measurement
}
```

## Light Sensor (Nightime Dance Party!)

Time required: 45 minutes

Please read the directions carefully before beginning the assignment.

- Comment your code as shown in the tutorials and other code examples.
- Follow all directions carefully and accurately.
- Think of the directions as minimum requirements.

### Introduction of the Light Sensor

Sensors are used to detect events or changes in the environment and send information to the electronic components of other electronic devices. While the program is running and debugging, it is often required to collect real-time sensor values to help us understand the environment light, sound, distance and other information.

Light sensor value range: 0~1000, exposed under sunshine (> 500), evening (0 ~ 100), lighting (100 to 500).

The following program will give you a robot bedtime dance party!

### Understanding

Demonstrate understanding of:

**light sensor**

### Knowledge Points

We will do some math to determine the normal light in a room when the program first runs. If the light level falls below 80% of the room illumination, the robot sounds an alarm!

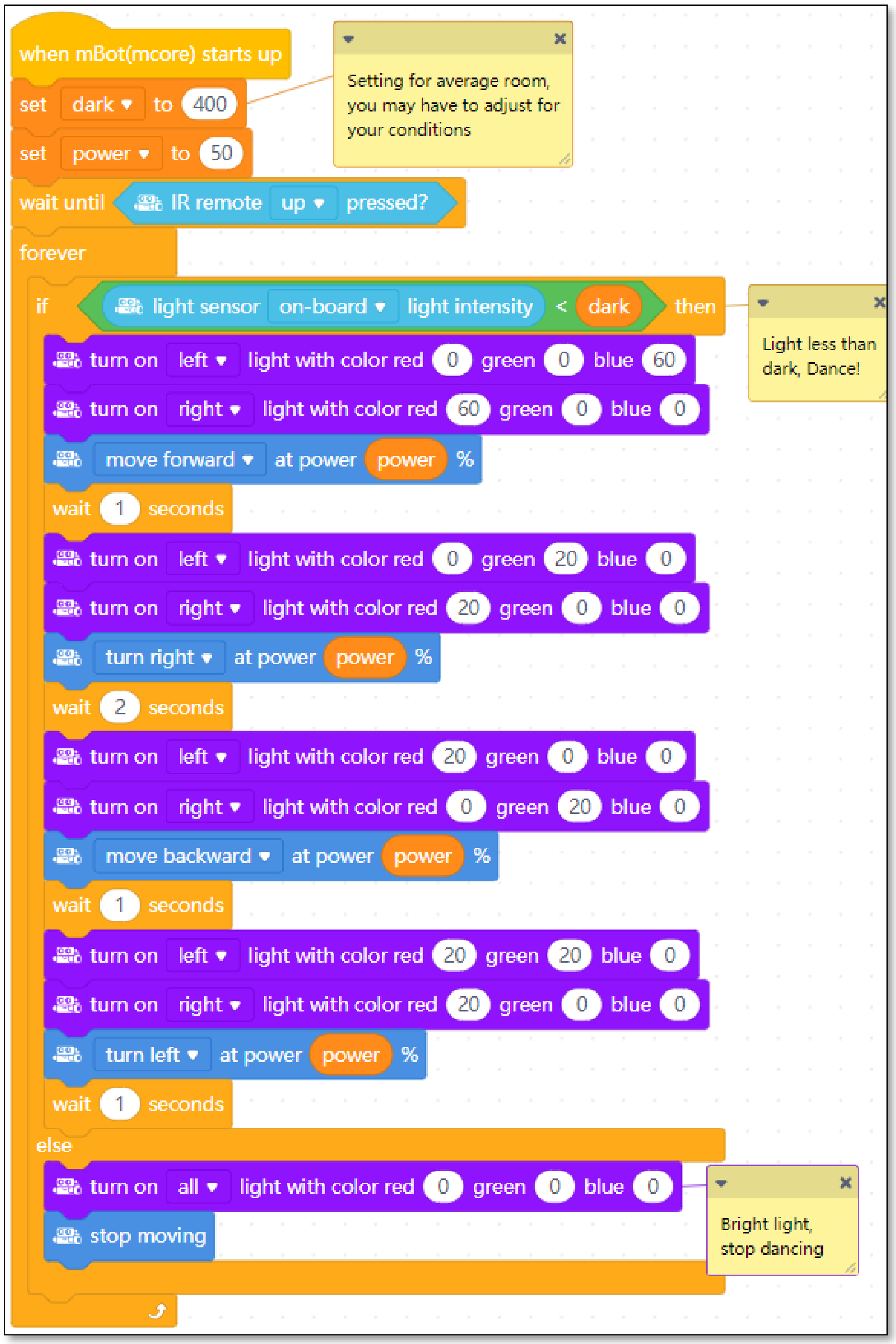
### Requirements

- The program will run when you power on the robot.
- When the lights are off, the robot does a dance.
- When the lights are on, the robot is quiet.

### Tutorial Assignment

1. Save the sketch as **DanceParty**

2. Complete and test the program as pictured with the requirements listed. The mBlock program shown is to get you started.



```
when mBot(mcore) starts up
set dark ▾ to 400
set power ▾ to 50
wait until IR remote up ▾ pressed?
forever
  if light sensor on-board ▾ light intensity < dark then
    turn on left ▾ light with color red 0 green 0 blue 60
    turn on right ▾ light with color red 60 green 0 blue 0
    move forward ▾ at power power %
    wait 1 seconds
    turn on left ▾ light with color red 0 green 20 blue 0
    turn on right ▾ light with color red 20 green 0 blue 0
    turn right ▾ at power power %
    wait 2 seconds
    turn on left ▾ light with color red 20 green 0 blue 0
    turn on right ▾ light with color red 0 green 20 blue 0
    move backward ▾ at power power %
    wait 1 seconds
    turn on left ▾ light with color red 20 green 20 blue 0
    turn on right ▾ light with color red 20 green 0 blue 0
    turn left ▾ at power power %
    wait 1 seconds
  else
    turn on all ▾ light with color red 0 green 0 blue 0
    stop moving
```

Setting for average room, you may have to adjust for your conditions

Light less than dark, Dance!

Bright light, stop dancing

## Requirements

- The program will run when you press the robot's remote control.

- There will be variety in movement, sights and sounds.
- The dance will start when the lights go out, and stop when the lights turn on.

## Challenge

Get creative and create your own version of Bedtime Dance Party!

## Assignment Submission

- All students attach the finished program to the assignment in Blackboard.
- Each assignment can be demonstrated in class.
- For online students, a link to a YouTube video recording showing the assignment can be placed in the submission area in BlackBoard.

## Car and Tank Turns

Time required: 20 minutes

Please read all the directions carefully before beginning the assignment.

- Comment your code as shown in the tutorials and other code examples.
- Follow all directions carefully and accurately.
- Think of the directions as minimum requirements.

### Understanding

Demonstrate understanding of:

**separate motor control, variables**

By varying the power to each motor, the mBot can make different types of turns.

### Requirements

- The program will run when you press the robot’s remote control button.
- Add your new movements into **Movement.h**
- Turn like a CAR with differential turns.

### Tutorial Assignment

1. Save the sketch as **CarAndTankTurns**.
2. Complete and test the program with the requirements listed.

## Assignment Submission

- All students attach the finished program to the assignment in Blackboard.
- Each assignment can be demonstrated in class.
- For online students, a link to a YouTube video recording showing the assignment can be placed in the submission area in BlackBoard.

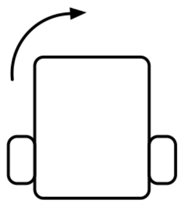


*Figure 2: Car*

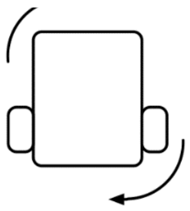


*Figure 2: Tank*

## Keep Away

Time required: 60 minutes

Please read all the directions carefully before beginning the assignment.

- Comment your code as shown in the tutorials and other code examples.
- Follow all directions carefully and accurately.
- Think of the directions as minimum requirements.

### Understanding

Demonstrate understanding of:

**Ultrasonic sensor, relational operators**

### Knowledge Points

An ultrasonic sensor can detect the distance from the object in front of it. A critical value is the distance between the object in front and mBot's ultrasonic sensor can be defined as the threshold to determine whether mBot should move forward (a threshold is a value of the condition under which an object is changed, which is also called critical value).

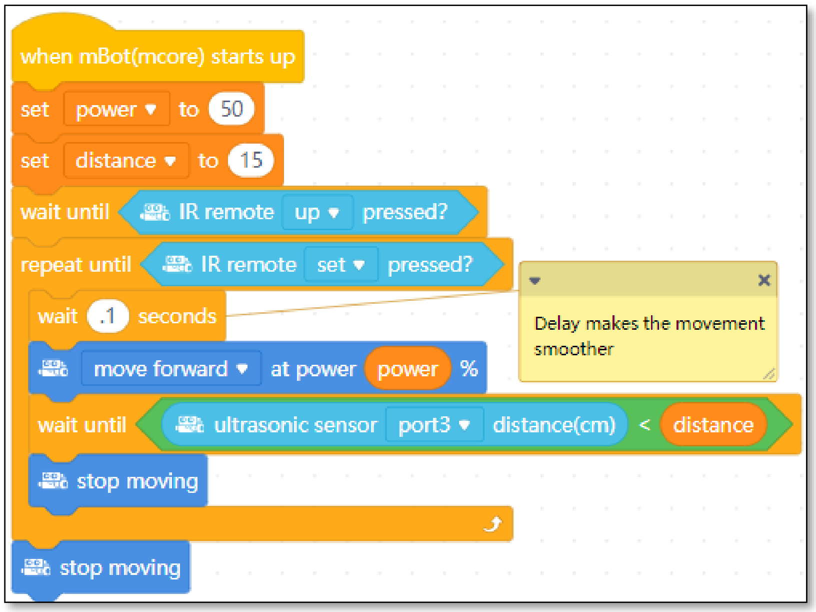
### While Loop

A while loop is similar to the repeat until block. This loop keeps going until a condition is met. In the example program, the keep away part of the program will continue to repeat until the set button is pressed. The program exits the loop. The mBot stops moving.

In this example the value of the ultrasonic sensor indicates the distance between mBot and the object in front of it. Given the threshold of 15cm, mBot will keep moving forward until its distance from the object is less than 15cm; the mBot will stop immediately when its distance from the object is less than 15cm.

### Tutorial Assignment

1. Open the Arduino IDE. Save the sketch as **KeepAway**
2. Complete and test the program. Use the mBlock as an example to get started.



```
when mBot(mcore) starts up
set power ▾ to 50
set distance ▾ to 15
wait until < IR remote up ▾ pressed? >
repeat until < IR remote set ▾ pressed? >
    wait .1 seconds
    move forward ▾ at power (power) %
    wait until < (ultrasonic sensor port3 ▾ distance(cm)) < (distance) >
    stop moving
stop moving
```

Delay makes the movement smoother

## Requirements

- The robot detects an object within 15 cm and stops.
- When the object is moved, the mBot starts moving forward.
- Test the keep away with your foot.

## Challenge

- Add lights when an object is detected.
- Add a sound when an object is detected. Make it very short. You will want to replace the wait with the sound.

## Extra Credit Challenge

Can you make the robot also move backwards if the barrier is moved closer to the mBot?

## Assignment Submission

- All students attach the finished program to the assignment in Blackboard.
- Each assignment can be demonstrated in class.
- For online students, a link to a YouTube video recording showing the assignment can be placed in the submission area in BlackBoard.

## Simple Line Following (What's My Line?)

Time required: 45 minutes

Please read all the directions carefully before beginning the assignment.

1. Comment your code as shown in the tutorials and other code examples.
2. Follow all directions carefully and accurately.
3. Think of the directions as minimum requirements.

---

### Understanding

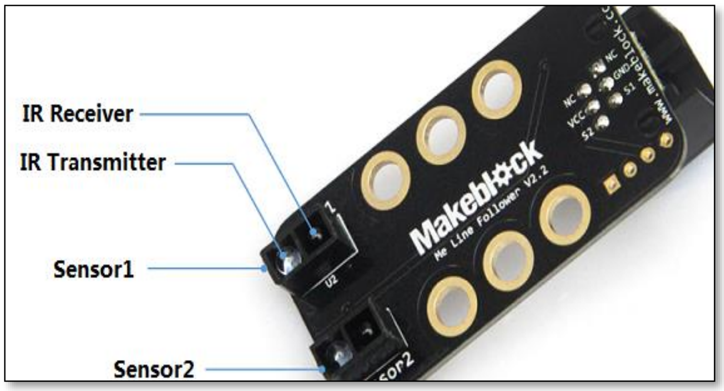
Demonstrate understanding of:

**line-follower sensor, if then else**

---

### Principles of the Line-follower Sensor

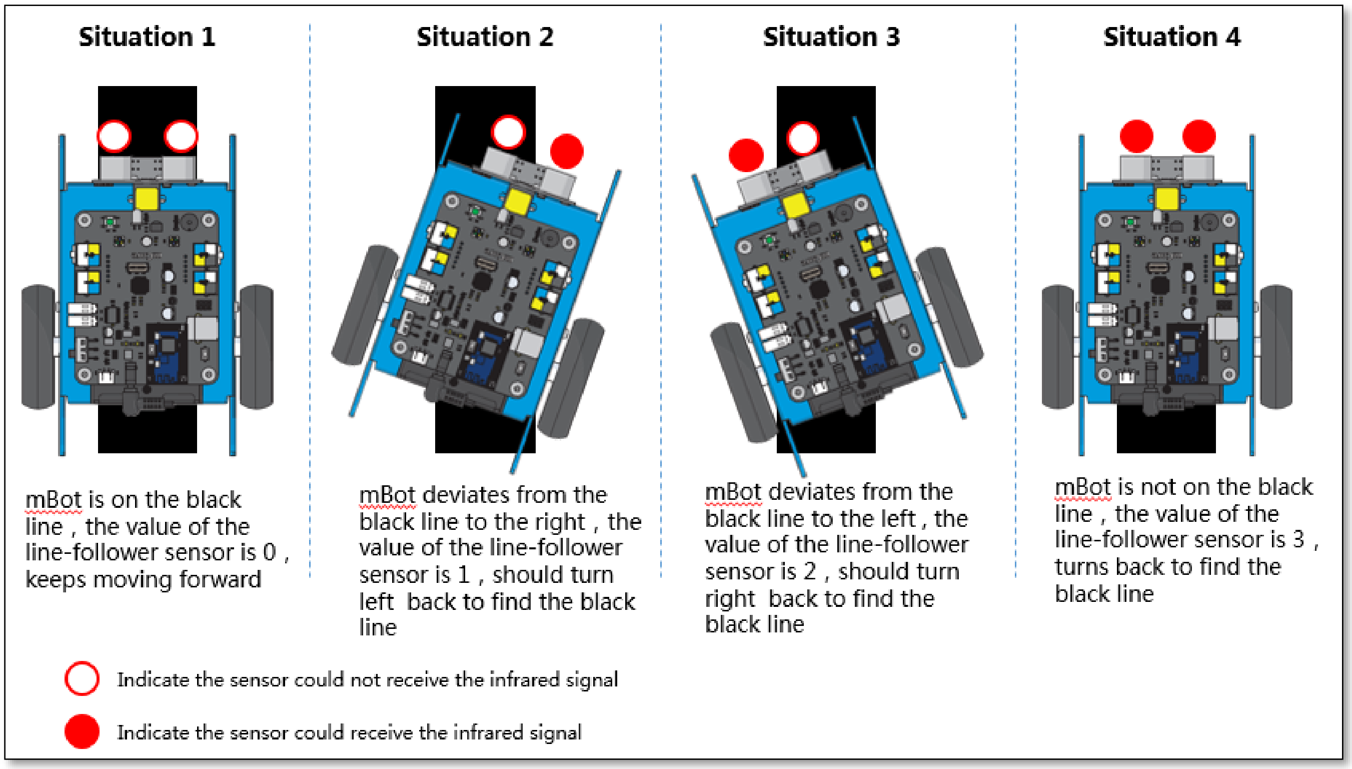
The line-follower sensor is below the robot (see the attached diagram), which consists of two sensors, Sensor 1 and 2, each consisting of an infrared emitter and an infrared receiver (see the attached diagram). As it is often used to keep the robot moving straight, it is called a line-follower sensor. Its detection range is 1 to 2 cm.



The infrared emitter continually emits infrared light during the mBot moving:

- If the infrared light is reflected (encountering white or other light color surfaces), the receiver receives the infrared signal and output the value 1 (now you can see the blue LED on the back of the line-follower sensor is lighted);
- If the infrared light is absorbed or cannot be reflected, the receiver will not receive the infrared signal but output the value 0.

The mBot line-follower sensors can detect a white line on a black surface, or a black line on a white surface.



**Situation 1**

mBot is on the black line , the value of the line-follower sensor is 0 , keeps moving forward

**Situation 2**

mBot deviates from the black line to the right , the value of the line-follower sensor is 1 , should turn left back to find the black line

**Situation 3**

mBot deviates from the black line to the left , the value of the line-follower sensor is 2 , should turn right back to find the black line

**Situation 4**

mBot is not on the black line , the value of the line-follower sensor is 3 , turns back to find the black line

○ Indicate the sensor could not receive the infrared signal

● Indicate the sensor could receive the infrared signal

- **Situation 1:** Line follower = 0. Both sensors detect a line indicated by both blue lights shutting off.
- **Situation 2:** Line follower = 1. The right sensor no longer detects a line indicated by the right blue light turning on. In order to get the mBot back on the line, therefore, we turn the mBot left until both sensors are activated and the mBot continues moving forward.
- **Situation 3:** Line follower = 2. The left sensor no longer detects a line indicated by the left blue light turning on. So we turn the mBot right until both sensors are activated and the mBot continues moving forward again.
- **Situation 4:** Line follower = 3. Both sensors no longer detect a line. Run backward until the robot detects a line.

| **Sensor Position** | **Value** |
|---|---|
| Both sensors over the line | 0 |
| Right sensor off line | 1 |
| Left sensor off line | 2 |

| | |
|---|---|
| Both sensors off line | 3 |

**Line Follower Track:** A line follower track can be made with foam board and black tape. Automotive cloth wiring harness, electrical tape duct tape works well.

## Relational Operators

Relational operators test for true or false by comparing one value to another. In this program we will compare the distance the sensor detects to the distance that we have set.

| Operator | Interpretation | Examples | Result |
|---|---|---|---|
| > | Greater than | 9 < 10<br>10 < 10 | true<br>false |
| >= | Greater than or equal to | 10 >= 10<br>10 >= 11 | true<br>false |
| < | Less than | 9 < 10<br>10 < 10 | true<br>false |
| <= | Less than or equal to | 10 <= 10<br>10 <=-9 | true<br>true |
| == | Equal to | 9 == 9 | true |
| != | Not equal to | 9 != 9 | false |

## Simple Line Following

This is a sketch shows how the line following sensor works with the mBot in Arduino C. Four possible states of the line sensor provides five different motor responses. You will modify and use **Movement.h** to control the robot.

| **Left Sensor** | **Right Sensor** | **Sensor Reading** | **Motor Response** |
|---|---|---|---|
| In | In | S1_IN_S2_IN<br>Both on line | Go Straight |
| In | Out | S1_IN_S2_OUT<br>Right off line | Left turn |
| Out | In | S1_OUT_S2_IN<br>Left off line | Right turn |

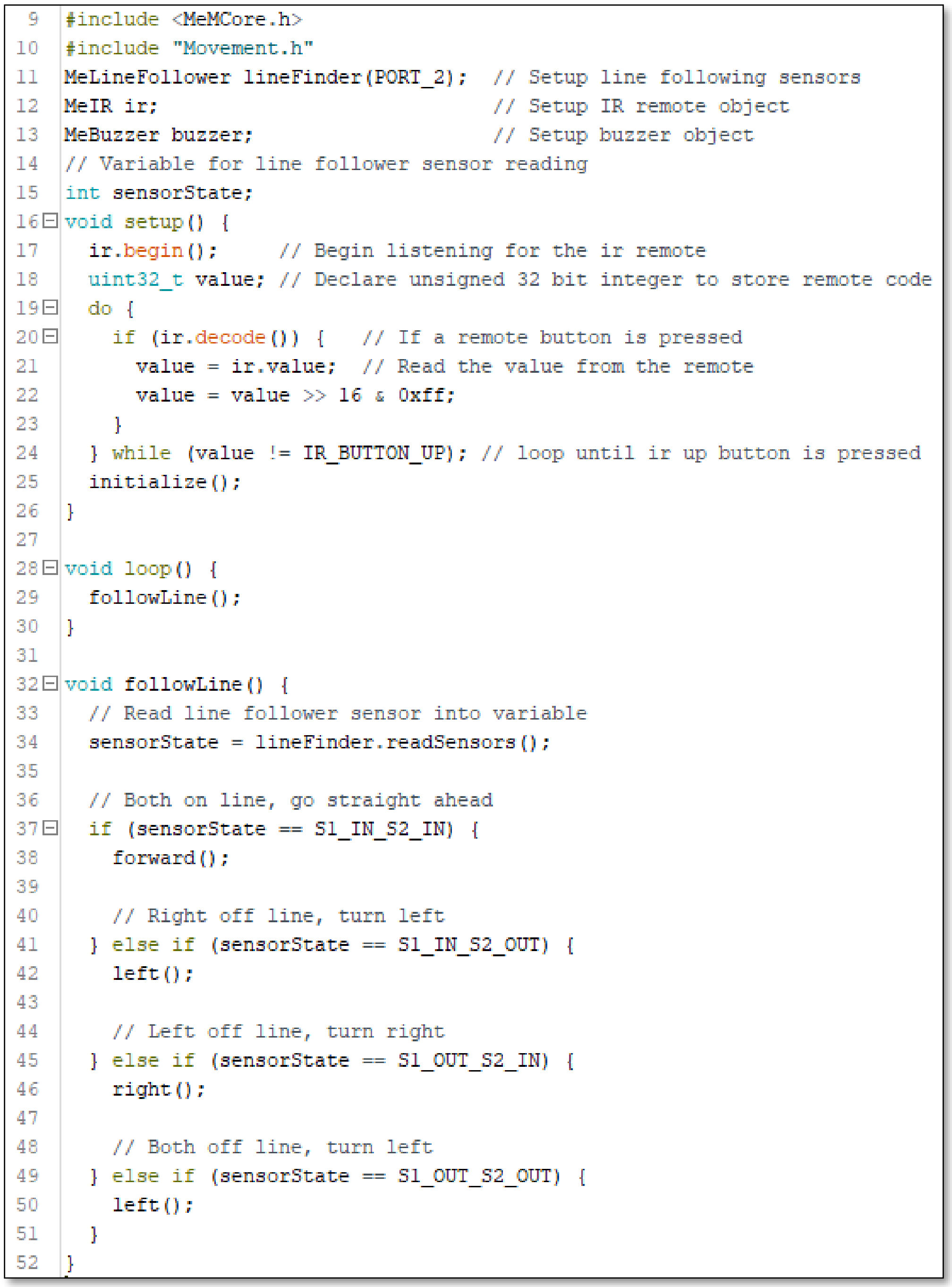
|  |  |  |  |
|---|---|---|---|
| Out | Out | S1_OUT_S2_OUT Both off line | (If previously left turn) Left Turn |
|  |  |  | (If previously right turn) Right Turn |

## Tutorial Assignment

1. Start the Arduino IDE. Save the sketch as **SimpleLineFollowing**.
2. Copy the file **Movement.h** into the sketch folder. **Movement.h** will need to be modified as shown below.
3. Complete and test the program as pictured.

## Assignment Submission

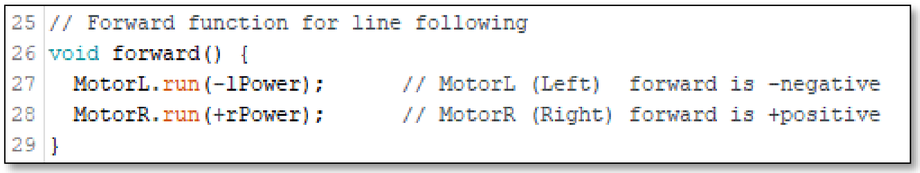
1. All students: Zip up the sketch folder. Attach the zip file to the assignment in Blackboard.
2. The assignment is demonstrated in class.
3. Online students: A link to a YouTube video recording showing your robot going through its motions is placed in the submission area in BlackBoard.



```
#include <MeMCore.h>
#include "Movement.h"
MeLineFollower lineFinder(PORT_2);  // Setup line following sensors
MeIR ir;                            // Setup IR remote object
MeBuzzer buzzer;                    // Setup buzzer object
// Variable for line follower sensor reading
int sensorState;
void setup() {
  ir.begin();     // Begin listening for the ir remote
  uint32_t value; // Declare unsigned 32 bit integer to store remote code
  do {
    if (ir.decode()) {   // If a remote button is pressed
      value = ir.value;   // Read the value from the remote
      value = value >> 16 & 0xff;
    }
  } while (value != IR_BUTTON_UP); // loop until ir up button is pressed
  initialize();
}

void loop() {
  followLine();
}

void followLine() {
  // Read line follower sensor into variable
  sensorState = lineFinder.readSensors();

  // Both on line, go straight ahead
  if (sensorState == S1_IN_S2_IN) {
    forward();

    // Right off line, turn left
  } else if (sensorState == S1_IN_S2_OUT) {
    left();

    // Left off line, turn right
  } else if (sensorState == S1_OUT_S2_IN) {
    right();

    // Both off line, turn left
  } else if (sensorState == S1_OUT_S2_OUT) {
    left();
  }
}
```

## Movement.h Modifications

Add forward, reverse, left and right functions to your Movement.h file. A forward function is given to start with. Add reverse, left and right functions.



```
// Forward function for line following
void forward() {
  MotorL.run(-lPower);      // MotorL (Left)  forward is -negative
  MotorR.run(+rPower);      // MotorR (Right) forward is +positive
}
```

## Driving School

Time required: 90 minutes

Please read all the directions carefully before beginning the assignment.

1. Comment your code as shown in the tutorials and other code examples.
2. Follow all directions carefully and accurately.
3. Think of the directions as minimum requirements.

### Understanding

Demonstrate understanding of:

**libraries, loops, variables**

**NOTE:** The mBot is not an accurate robot. As the batteries discharge and the conditions change, it will behave differently. The only things we can change is power and time. Just try to get close.

We can accurately move and turn. You will combine the movement and turning programs into one. There isn't an example program, it is up to you to figure it out.

Charge your batteries. Calibrate your robot with the **CalibrateMovement** program.

The sample program will get you started. It is time to put your mBot through its paces. Can you pass the driving tests?

### Assignment

- Save **AccurateMovement** as **DrivingSchool**
- Use **Movement.h** for your movements.
- Assign each shape to a different remote control button as shown.
- Use a for loop for repeated code.
- Add sounds and lights to make the program more interesting.
- Use the example program to get started.

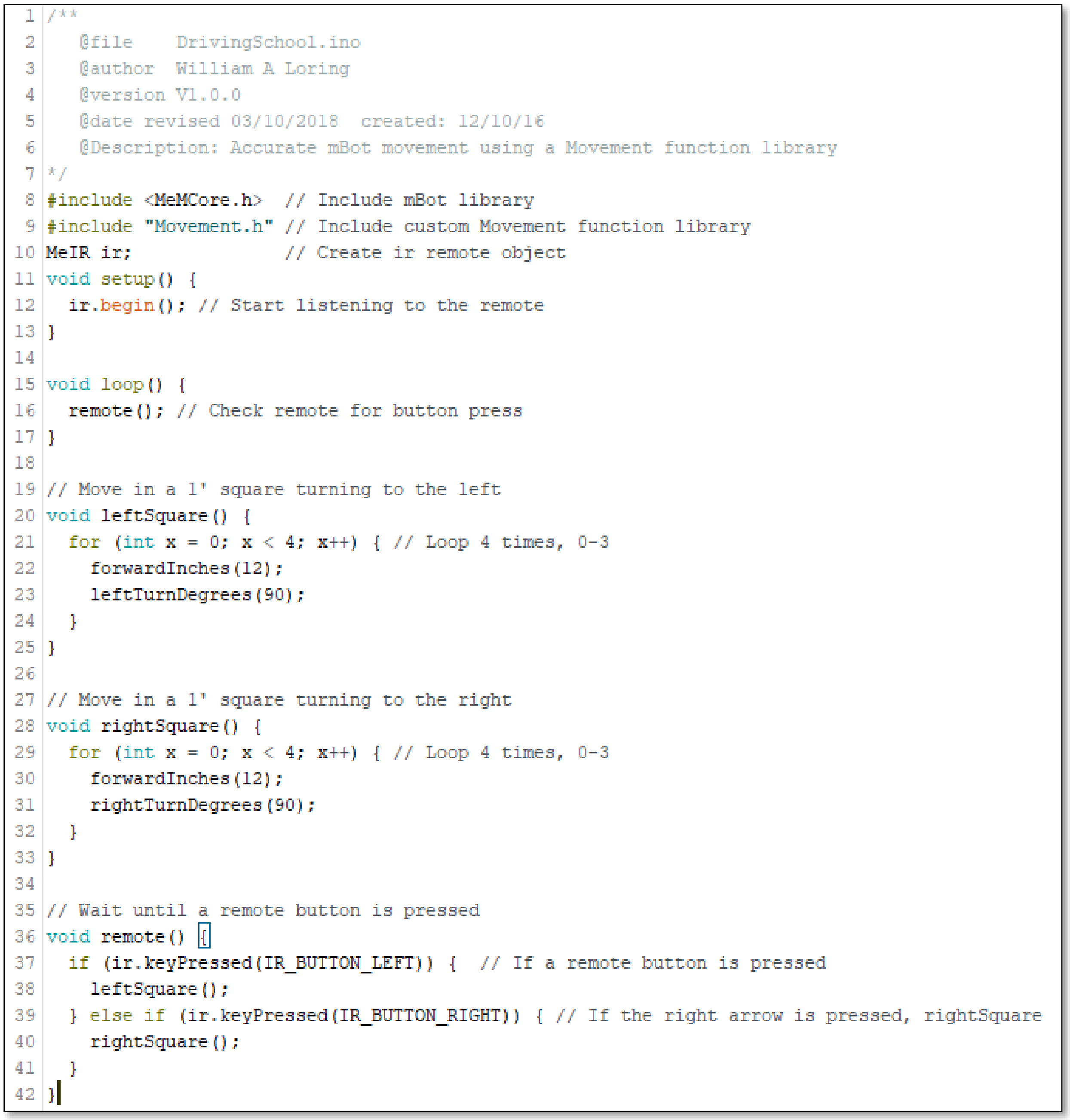
### Requirements

1. **Square** - Trace the path of a square that is 1-foot square. Start and end in the same place and the same orientation.

2. **Rectangle** - Trace the path of a rectangle that is 1-foot x 2-foot. Start and end in the same place and the same orientation.
3. **Sentry** - Trace a 1-foot square around an object. Start the square one way, then turn around and go the other way.
4. **Retrace** - Move in a 1-foot square forward, and then move in reverse to retrace that same square backwards to the beginning point and orientation.

## Assignment Submission

1. All students: Zip up the sketch folder. Attach the zip file to the assignment in Blackboard.
2. The assignment is demonstrated in class.
3. Online students: A link to a YouTube video recording showing your robot going through its motions is placed in the submission area in BlackBoard.



```
/**
   @file    DrivingSchool.ino
   @author  William A Loring
   @version V1.0.0
   @date revised 03/10/2018  created: 12/10/16
   @Description: Accurate mBot movement using a Movement function library
*/
#include <MeMCore.h>  // Include mBot library
#include "Movement.h" // Include custom Movement function library
MeIR ir;              // Create ir remote object
void setup() {
  ir.begin(); // Start listening to the remote
}

void loop() {
  remote(); // Check remote for button press
}

// Move in a 1' square turning to the left
void leftSquare() {
  for (int x = 0; x < 4; x++) { // Loop 4 times, 0-3
    forwardInches(12);
    leftTurnDegrees(90);
  }
}

// Move in a 1' square turning to the right
void rightSquare() {
  for (int x = 0; x < 4; x++) { // Loop 4 times, 0-3
    forwardInches(12);
    rightTurnDegrees(90);
  }
}

// Wait until a remote button is pressed
void remote() {
  if (ir.keyPressed(IR_BUTTON_LEFT)) {  // If a remote button is pressed
    leftSquare();
  } else if (ir.keyPressed(IR_BUTTON_RIGHT)) { // If the right arrow is pressed, rightSquare
    rightSquare();
  }
}
```

## Driving School Part 2

Time required: 90 minutes

Please read all the directions carefully before beginning the assignment.

1. Comment your code as shown in the tutorials and other code examples.
2. Follow all directions carefully and accurately.
3. Think of the directions as minimum requirements.

### Understanding

Demonstrate understanding of:

**libraries, functions**

Charge your batteries. Calibrate your robot with the **CalibrateMovement** program.

### Assignment

- Use **Movement.h** for your movements.
- Assign each shape to a different remote control button.
- Add to the **DrivingSchool** program you created in the earlier exercise. Save the sketch as **DrivingSchool2**.

### Requirements

1. **ForwardReverse** - Move forward 12”, turn 180°, move backwards 12” (which will be the same direction), turn 180° again, and then continue to move forward 12”. The robot should move in one direction, but do part of the trip moving backwards.
2. **Octagon** - Move a 12” octagon. Each turn is a 45° angle. Start and end in the same place and the same orientation.
3. **Equilateral Triangle** – Move in a 12” equilateral triangle. Start and end in the same place and the same orientation. An equilateral triangle has an inside angle of 60 degrees. Subtract that from 180 degrees to find out how far the robot should turn for each side.
4. **5-Point Star** – Rrace a 5-point 12” star. Start and end at the same location and orientation. Look up the inside angle and subtract from 180 degrees.

## Extra Credit Challenges

1. Modify the program to trace the outline of a pentagon.
2. Modify the program to trace the outline of a hexagon.

## Assignment Submission

1. All students: Zip up the sketch folder. Attach the zip file to the assignment in Blackboard.
2. The assignment is demonstrated in class.
3. Online students: A link to a YouTube video recording showing your robot going through its motions is placed in the submission area in BlackBoard.

## Obstacle Avoidance with Warning and Random Turns (Look Out!)

Time required: 30 minutes

Please read all the directions carefully before beginning the assignment.

1. Comment your code as shown in the tutorials and other code examples.
2. Follow all directions carefully and accurately.
3. Think of the directions as minimum requirements.

### Understanding

Demonstrate understanding of:

**ultrasonic sensor**

### Tutorial Assignment

1. Save the sketch as **ObstacleAvoidanceWithWarning**
2. Complete and test the program with the requirements listed.
3. Use the mBlock program as a starting point.

### Requirements

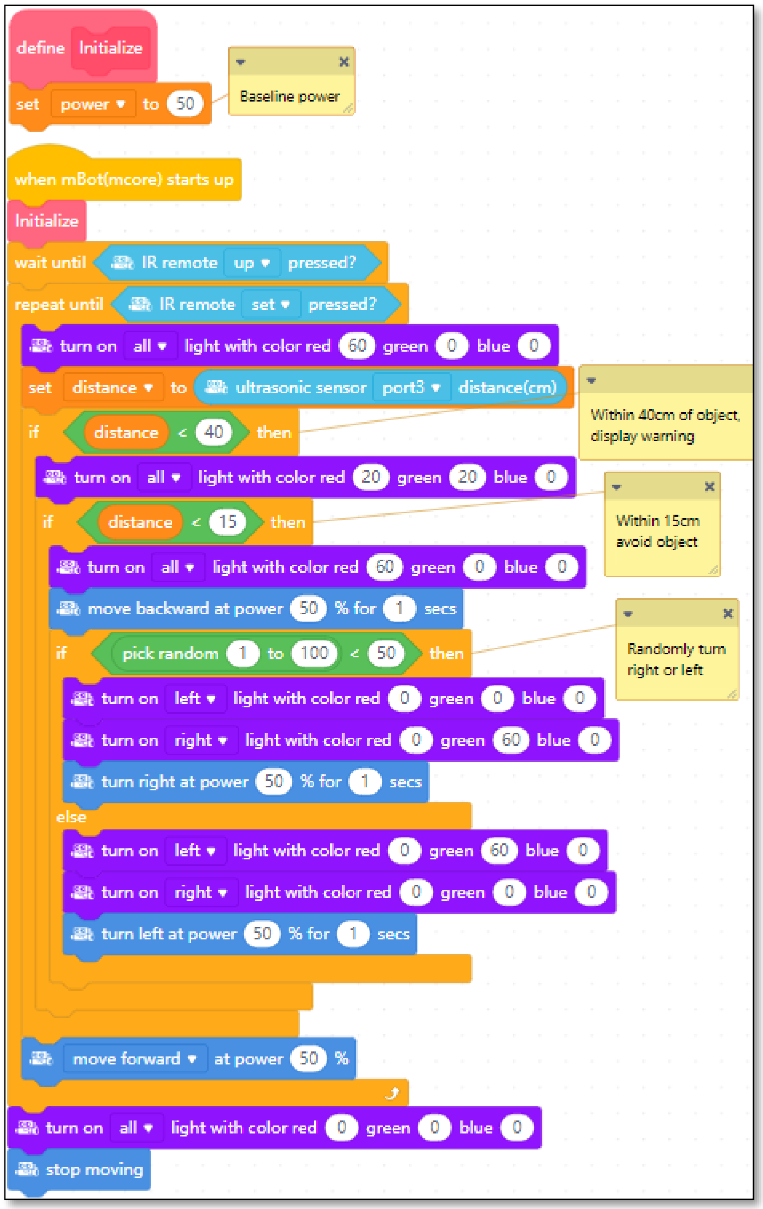
- The robot randomly turns left or right to avoid an obstacle and gives a visual warning.
- Test the obstacle avoidance with your foot.

### Challenges

- Play a sound when an object is detected.
- Experiment with the detection distance.

### Assignment Submission

- All students attach the finished program to the assignment in Blackboard.
- Each assignment can be demonstrated in class.
- For online students, a link to a YouTube video recording showing the assignment can be placed in the submission area in BlackBoard.



```
define Initialize
set power to 50                                   [Baseline power]

when mBot(mcore) starts up
Initialize
wait until IR remote up pressed?
repeat until IR remote set pressed?
    turn on all light with color red 60 green 0 blue 0
    set distance to ultrasonic sensor port3 distance(cm)
    if distance < 40 then                         [Within 40cm of object, display warning]
        turn on all light with color red 20 green 20 blue 0
        if distance < 15 then                     [Within 15cm avoid object]
            turn on all light with color red 60 green 0 blue 0
            move backward at power 50 % for 1 secs
            if pick random 1 to 100 < 50 then     [Randomly turn right or left]
                turn on left light with color red 0 green 0 blue 0
                turn on right light with color red 0 green 60 blue 0
                turn right at power 50 % for 1 secs
            else
                turn on left light with color red 0 green 60 blue 0
                turn on right light with color red 0 green 0 blue 0
                turn left at power 50 % for 1 secs
    move forward at power 50 %
turn on all light with color red 0 green 0 blue 0
stop moving
```

## Driving School Curves

Time required: 90 minutes

Please read all the directions carefully before beginning the assignment.

1. Comment your code as shown in the tutorials and other code examples.
2. Follow all directions carefully and accurately.
3. Think of the directions as minimum requirements.

### Understanding

Demonstrate understanding of:

**libraries, functions**

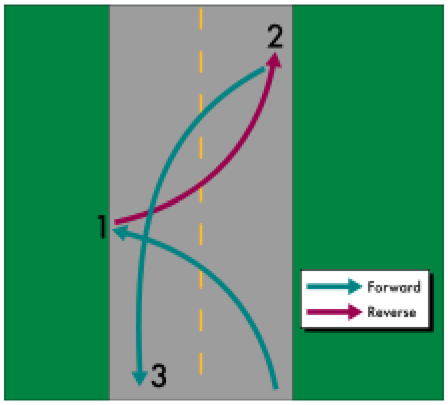
Charge your batteries. Calibrate your robot with the **CalibrateMovement** program.

### Assignment

- Complete each program clockwise and counterclockwise.
- Use **Movement.h** for your movements.
- Add a **rightCircle** and **leftCircle** function to your Movement.h library.
  **HINT**: Adjust the power of your left and right motors to create a left half circle function and right half circle function. Use these new functions to make your curved shapes.
- Assign each shape to a different remote control button.
- Add to the **DrivingSchool2** sketch. Save the sketch as **DrivingSchoolCurves**.

### Requirements

1. **3-PointTurn** - Using 3 or more turns, make a 3-point turn, like a regular car. You don’t have to do curves, you can use straight angles if you wish.



2. **Circle** - Trace the path of a circle that is 1 foot in diameter. It will start and end in the same location, and in the same orientation.
   **HINT**: Adjust the power of your left and right motors to create a left half circle block and right half circle block. Put those together to make your curved shapes.
3. **S-Shape** – Trace two half-circles to create an S-shaped curve. Your robot will start and end in the same orientation, and the two half-circles will be the same size.
4. **Figure-8** - Move in a figure-8 shape. You did this in an earlier assignment. ∞

## Assignment Submission

1. All students: Zip up the sketch folder. Attach the zip file to the assignment in Blackboard.
2. The assignment is demonstrated in class.
3. Online students: A link to a YouTube video recording showing your robot going through its motions is placed in the submission area in BlackBoard.

## Smart Line Following

Time required: 45 minutes

Please read all the directions carefully before beginning the assignment.

1. Comment your code as shown in the tutorials and other code examples.
2. Follow all directions carefully and accurately.
3. Think of the directions as minimum requirements.

### Understanding

Demonstrate understanding of:

**line follower,**

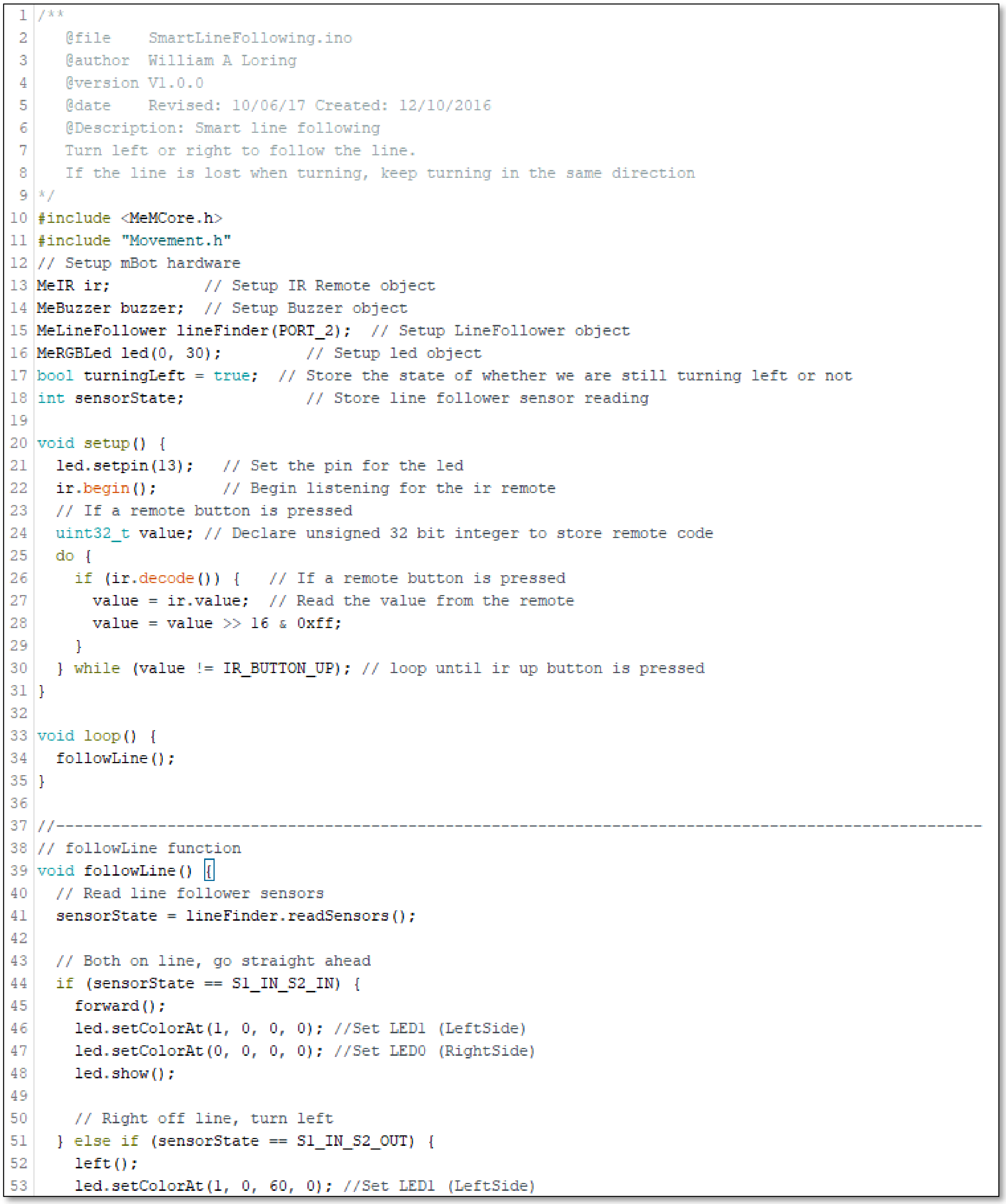
This sketch is based on the mBlock version. This version uses if, else if and nested if statements.

### Tutorial Assignment

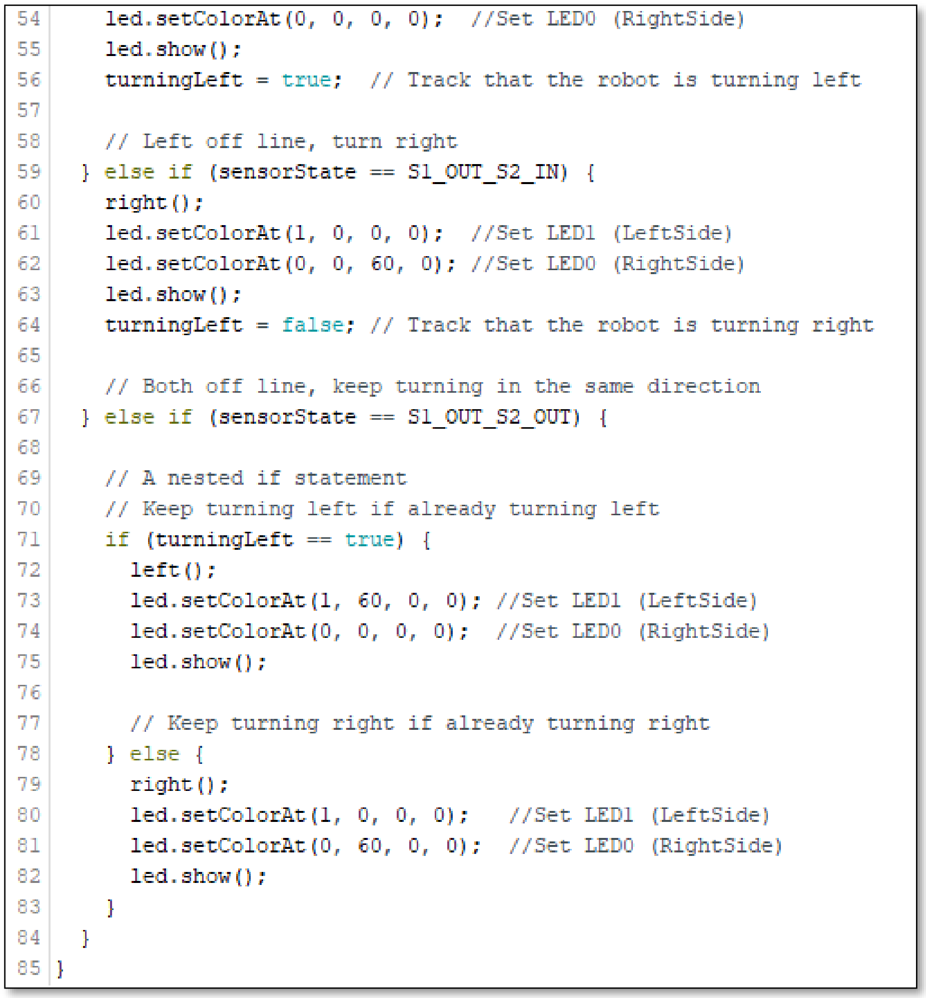
1. Start the Arduino IDE. Save the sketch as **SmartLineFollowing**.
2. Copy the file **Movement.h** into the sketch folder.
3. Complete and test the program as shown with the requirements listed.

### Assignment Submission

1. All students: Zip up the sketch folder. Attach the zip file to the assignment in Blackboard.
2. The assignment is demonstrated in class.
3. Online students: A link to a YouTube video recording showing your robot going through its motions is placed in the submission area in BlackBoard.



```
/**
   @file    SmartLineFollowing.ino
   @author  William A Loring
   @version V1.0.0
   @date    Revised: 10/06/17 Created: 12/10/2016
   @Description: Smart line following
   Turn left or right to follow the line.
   If the line is lost when turning, keep turning in the same direction
*/
#include <MeMCore.h>
#include "Movement.h"
// Setup mBot hardware
MeIR ir;          // Setup IR Remote object
MeBuzzer buzzer;  // Setup Buzzer object
MeLineFollower lineFinder(PORT_2);  // Setup LineFollower object
MeRGBLed led(0, 30);         // Setup led object
bool turningLeft = true;  // Store the state of whether we are still turning left or not
int sensorState;             // Store line follower sensor reading

void setup() {
  led.setpin(13);   // Set the pin for the led
  ir.begin();       // Begin listening for the ir remote
  // If a remote button is pressed
  uint32_t value; // Declare unsigned 32 bit integer to store remote code
  do {
    if (ir.decode()) {   // If a remote button is pressed
      value = ir.value;  // Read the value from the remote
      value = value >> 16 & 0xff;
    }
  } while (value != IR_BUTTON_UP); // loop until ir up button is pressed
}

void loop() {
  followLine();
}

//----------------------------------------------------------------------------------------------------
// followLine function
void followLine() {
  // Read line follower sensors
  sensorState = lineFinder.readSensors();

  // Both on line, go straight ahead
  if (sensorState == S1_IN_S2_IN) {
    forward();
    led.setColorAt(1, 0, 0, 0); //Set LED1 (LeftSide)
    led.setColorAt(0, 0, 0, 0); //Set LED0 (RightSide)
    led.show();

    // Right off line, turn left
  } else if (sensorState == S1_IN_S2_OUT) {
    left();
    led.setColorAt(1, 0, 60, 0); //Set LED1 (LeftSide)
```



```
    led.setColorAt(0, 0, 0, 0);  //Set LED0 (RightSide)
    led.show();
    turningLeft = true;  // Track that the robot is turning left

    // Left off line, turn right
  } else if (sensorState == S1_OUT_S2_IN) {
    right();
    led.setColorAt(1, 0, 0, 0);  //Set LED1 (LeftSide)
    led.setColorAt(0, 0, 60, 0); //Set LED0 (RightSide)
    led.show();
    turningLeft = false; // Track that the robot is turning right

    // Both off line, keep turning in the same direction
  } else if (sensorState == S1_OUT_S2_OUT) {

    // A nested if statement
    // Keep turning left if already turning left
    if (turningLeft == true) {
      left();
      led.setColorAt(1, 60, 0, 0); //Set LED1 (LeftSide)
      led.setColorAt(0, 0, 0, 0);  //Set LED0 (RightSide)
      led.show();

      // Keep turning right if already turning right
    } else {
      right();
      led.setColorAt(1, 0, 0, 0);   //Set LED1 (LeftSide)
      led.setColorAt(0, 60, 0, 0);  //Set LED0 (RightSide)
      led.show();
    }
  }
}
```

## Smooth Line Following

Time required: 45 minutes

Please read all the directions carefully before beginning the assignment.

1. Comment your code as shown in the tutorials and other code examples.
2. Follow all directions carefully and accurately.
3. Think of the directions as minimum requirements.

### Understanding

Demonstrate understanding of:

**line follower**

Use **SmartLineFollowing** to start with.

### Tutorial Assignment

1. Start the Arduino IDE. Save the sketch as **SmoothLineFollowing**.
2. Copy the file **Movement.h** into the sketch folder.
3. Take a look at your mBlock Smart Line Following program. Try to make the line following smoother and able to follow sharper turns. Try a car turn. Don't go below 70 for power. Try reducing one side, and increasing the other. Try speeding up when you are going straight.
4. Add different left turn, right turn and forward functions with different names. Start with the code from left, right, and forward. Change those functions to optimize the line following for speed and accuracy.

### Assignment Submission

1. All students: Zip up the sketch folder. Attach the zip file to the assignment in Blackboard.
2. The assignment is demonstrated in class.
3. Online students: A link to a YouTube video recording showing your robot going through its motions is placed in the submission area in BlackBoard.

## Smart Obstacle Avoidance

Time required: 60 minutes

Please read all the directions carefully before beginning the assignment.

1. Comment your code as shown in the tutorials and other code examples.
2. Follow all directions carefully and accurately.
3. Think of the directions as minimum requirements.

### Understanding

Demonstrate understanding of:

**ultrasonic sensor**

### Requirements

- Open the **SimpleObstacleAvoidance** sketch and save it as **SmartObstacleAvoidance**.
- Use the shape of the mBlock version of this program to guide your coding.
- Avoid obstacles by looking left, then right, then turning in the direction with the longest distance.
- Use the following obstacle detection functions from the mBlock Obstacle Avoidance with Smart Turns as examples.
    - obstacleAvoidance
    - detectObstacle
    - avoidObstacle
- Create a **boolean** variable **isObstacleDetected** to track whether there is an obstacle or not. Use **isObstacleDetected** = true or **isObstacleDetected** = false
- Include Movement.h

### Assignment Submission

1. All students: Zip up the sketch folder. Attach the zip file to the assignment in Blackboard.
2. The assignment is demonstrated in class.

3. Online students: A link to a YouTube video recording showing your robot going through its motions is placed in the submission area in BlackBoard.

## mBot Music Dance Party!

Time required: 60 minutes

The mBot can appear to be playing a song and moving at the same time. It looks like the mBot is dancing to the music. You will want a catchy song.

Here is a web site to get you started with a known song. https://noobnotes.net

### Requirements

- Add this to your Arduino Default Program. Save it as **Default_Program_Dance_Party**.
- You will want to use **notes.h** and your **movement.h** file.
- Spin in circles, wiggle back and forth, make turns, move forward and backward, etc.
- **Movement function without a parameter:** The music will keep playing while the mBot moves. The mBot will keep moving until you change to another movement or stop.
- **Movement function with a parameter:** The music will stop and wait until the movement is complete.
- **Slow song:** If you use a slow song, it can have more movements changes per number of notes played.
- **Fast song:** If you use a fast song, it would have less movement changes per the number of notes played.
- You do not have to do the whole song, just a part of it.
- The music dance party should last a minimum of 15 seconds.
- Comment your code. Please put the name of the song in the comments.

### Assignment Submission

- Attach the program, and submit the assignment in Blackboard.
- Each assignment is demonstrated in class or a link to a YouTube video recording showing your robot going through its motions is placed in the submission area in BlackBoard.

### Examples

Be creative: Find your own song and your own path!

- mBot Imperial March
- 11/19/18 I wish you a merry mBot Christmas from Andrew

- [11/19/18 4 mBots moving and playing Mario!]()

```
//----------------------------------------------------------------------------------------
// Music Dance Party function Imperial March
// This is a slow song, it can have more movements changes per number of notes played
// A fast song would need less movement changes per the number of notes played
//----------------------------------------------------------------------------------------
void musicDanceParty() {
  if (modeFlag == 4) {
    delay(1000);
    forward();
    playNote(noteA4, HN);
    playNote(noteA4, HN);
    left();
    playNote(noteA4, HN);
    playNote(noteF4, EN3);
    playNote(noteC5, EN);
    right();
    playNote(noteA4, HN);
    playNote(noteF4, EN3);
    playNote(noteC5, EN);
    right();
    playNote(noteA4, HN);
    stop();
    modeFlag = 0;         // Stop the dance party, return to remote control
  }
}
```

## State Machine (Flags)

Time required: 30 minutes

Please read all the directions carefully before beginning the assignment.

1. Comment your code as shown in the tutorials and other code examples.
2. Follow all directions carefully and accurately.
3. Think of the directions as minimum requirements.

This program demonstrates how to store the state of the machine (mBot) in a flag variable. Flags are a way of keeping track of the state or history of a robot. The robot can then access that history and make a decision based on that history. Flags allow for fast switching and checking of task states. Checking a flag is a common Arduino practice for modular programming.

In this program, we are changing modes, which allows the remote buttons to be reused for other code blocks. This is how the default program that came with the mBot works. When the mBot is in **ModeA**, you can set remote button actions in that code block. Switch to **ModeB**, the buttons can have other actions in that code block.

### How it Works

1. The **ModeFlag** is set to 0 in the Initialize block, the **ModeA** code block is active.
2. The forever loop checks for a remote key press in the **SetMode** block.
3. The **SetMode** block changes the **ModeFlag** to 1.
4. The **ModeA and ModeB** code blocks keep testing for a ModeFlag change. When **ModeB** sees the **ModeFlag** 1, it executes and **ModeA** stops.
5. When **SetMode** changes the **Modeflag** to 0, we go back to **ModeA**.

### Tutorial Assignment

1. Save the sketch as **StateMachine**.
2. Use the mBlock program shown as a starting place.
3. Complete and test the program with the requirements listed.

#### Requirements

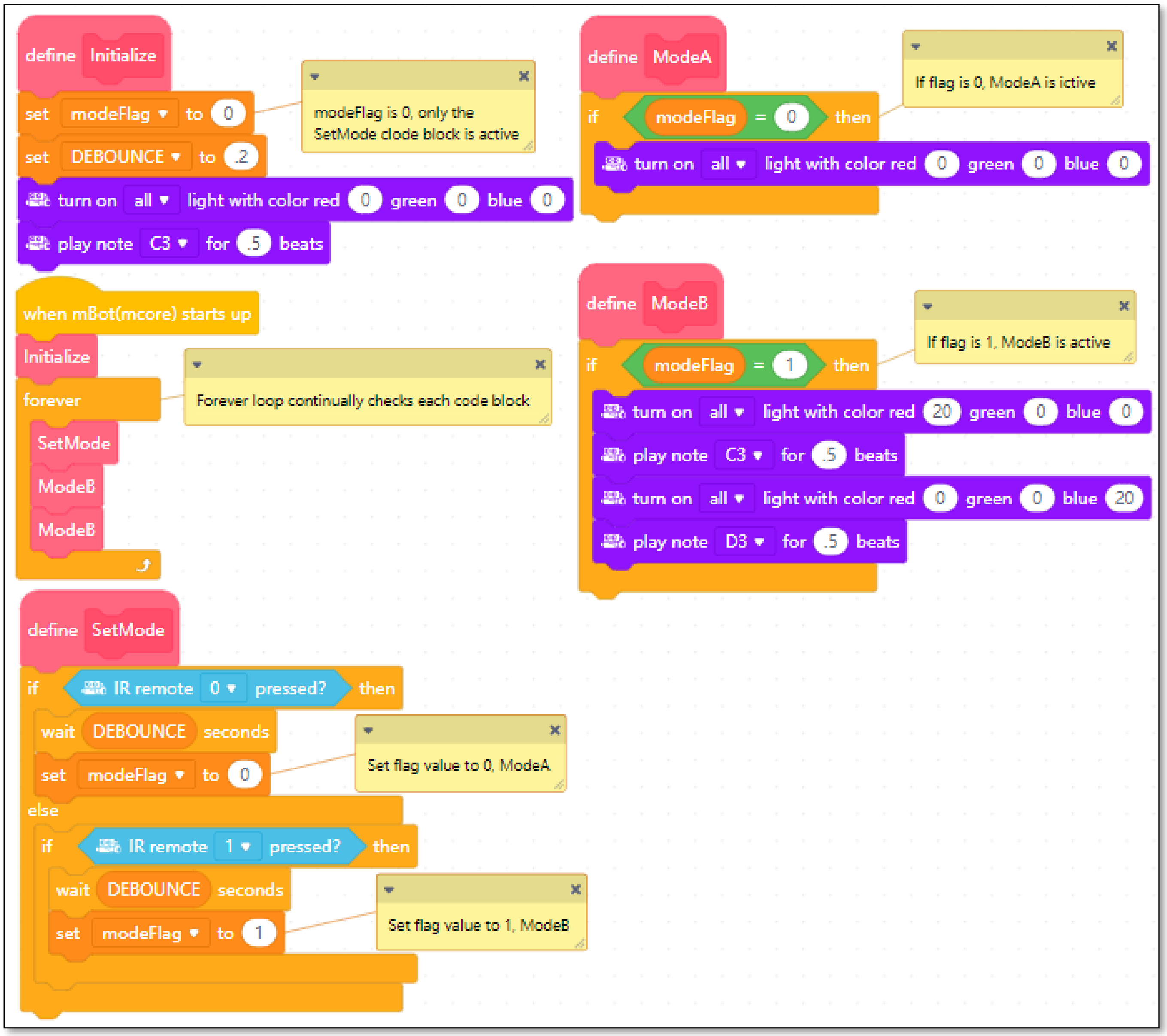
Create and test the program.

## Challenge

- Add **ModeC** to the program. The Flag value would be 2.
- Have **ModeC** do something else.

## Assignment Submission

- All students attach the finished program to the assignment in Blackboard.
- Each assignment can be demonstrated in class.
- For online students, a link to a YouTube video recording showing the assignment can be placed in the submission area in BlackBoard.



```
define Initialize
set modeFlag to 0
set DEBOUNCE to .2
turn on all light with color red 0 green 0 blue 0
play note C3 for .5 beats
```

modeFlag is 0, only the SetMode clode block is active

```
when mBot(mcore) starts up
Initialize
forever
    SetMode
    ModeB
    ModeB
```

Forever loop continually checks each code block

```
define SetMode
if IR remote 0 pressed? then
    wait DEBOUNCE seconds
    set modeFlag to 0
else
    if IR remote 1 pressed? then
        wait DEBOUNCE seconds
        set modeFlag to 1
```

Set flag value to 0, ModeA

Set flag value to 1, ModeB

```
define ModeA
if modeFlag = 0 then
    turn on all light with color red 0 green 0 blue 0
```

If flag is 0, ModeA is ictive

```
define ModeB
if modeFlag = 1 then
    turn on all light with color red 20 green 0 blue 0
    play note C3 for .5 beats
    turn on all light with color red 0 green 0 blue 20
    play note D3 for .5 beats
```

If flag is 1, ModeB is active

## Default Program 1: Remote Control

Time required: 120 minutes

Please read all the directions carefully before beginning the assignment.

1. Comment your code as shown in the tutorials and other code examples.
2. Follow all directions carefully and accurately.
3. Think of the directions as minimum requirements.

### Understanding

Demonstrate understanding of:

**functions**

### Tutorial Assignment

Let's begin building the mBot default program in Arduino starting with remote control.

1. Open the **DrivingSchool** program and save it as **DefaultProgram1**.
2. The **Movement.h** file should still be in the program folder..
3. Include the **notes.h** file to the sketch folder. Use this for audio feedback for the sketch.
4. Complete and test the program with the requirements listed.

### Requirements

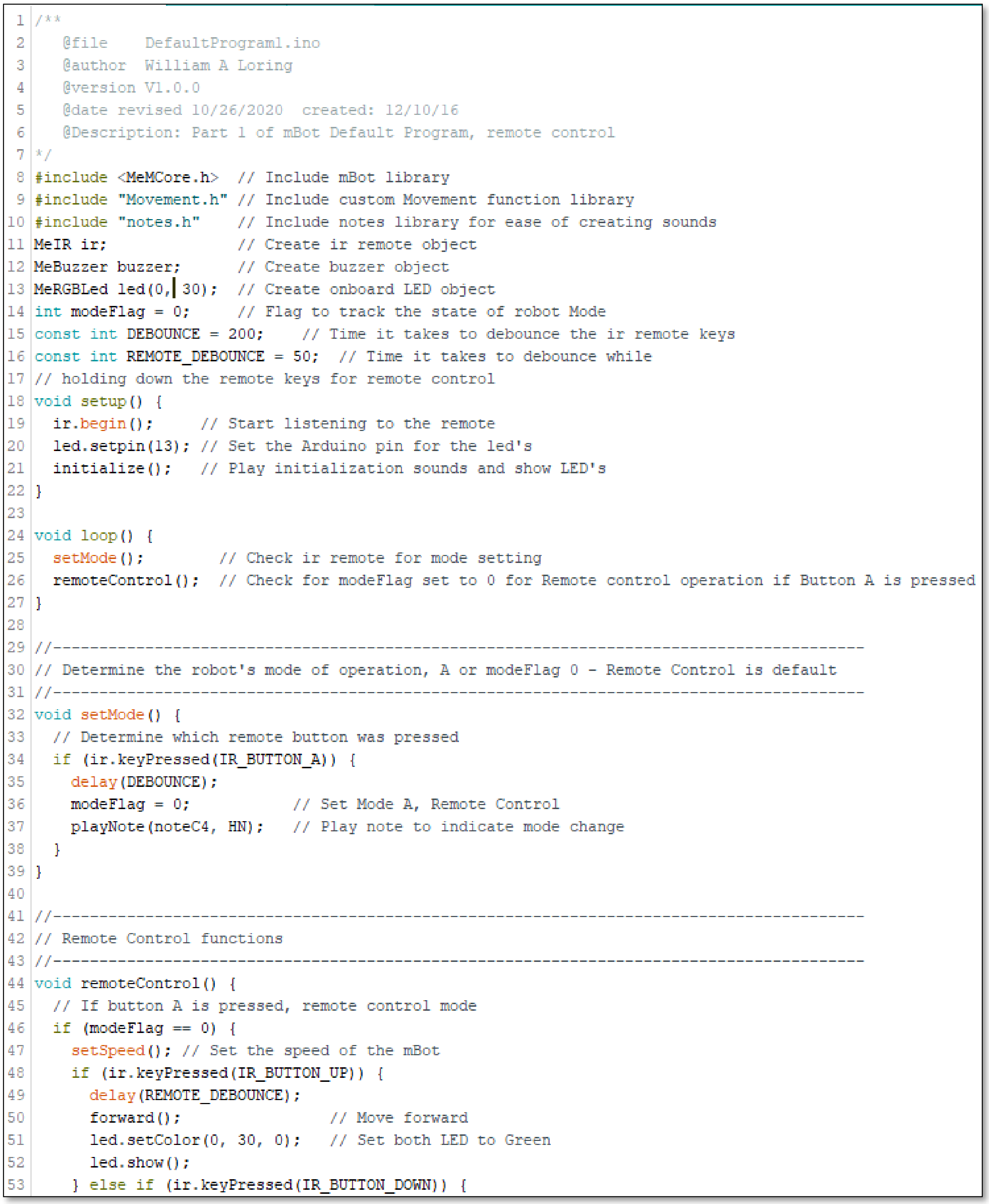
- Base your mode switching on your mBlock Default program.
- Add the necessary variables. For example, **int modeFlag = 0;** to track the mode in the main sketch.
- Create the **setMode**, **remoteControl**, **setSpeed, and speedSet** functions in the main sketch.
- Add the changes shown in the **Movement.h** file.
- Add the **playNote** function from previous Arduino programs to the main sketch. This allows you to easily add audio feedback.

## Assignment Submission

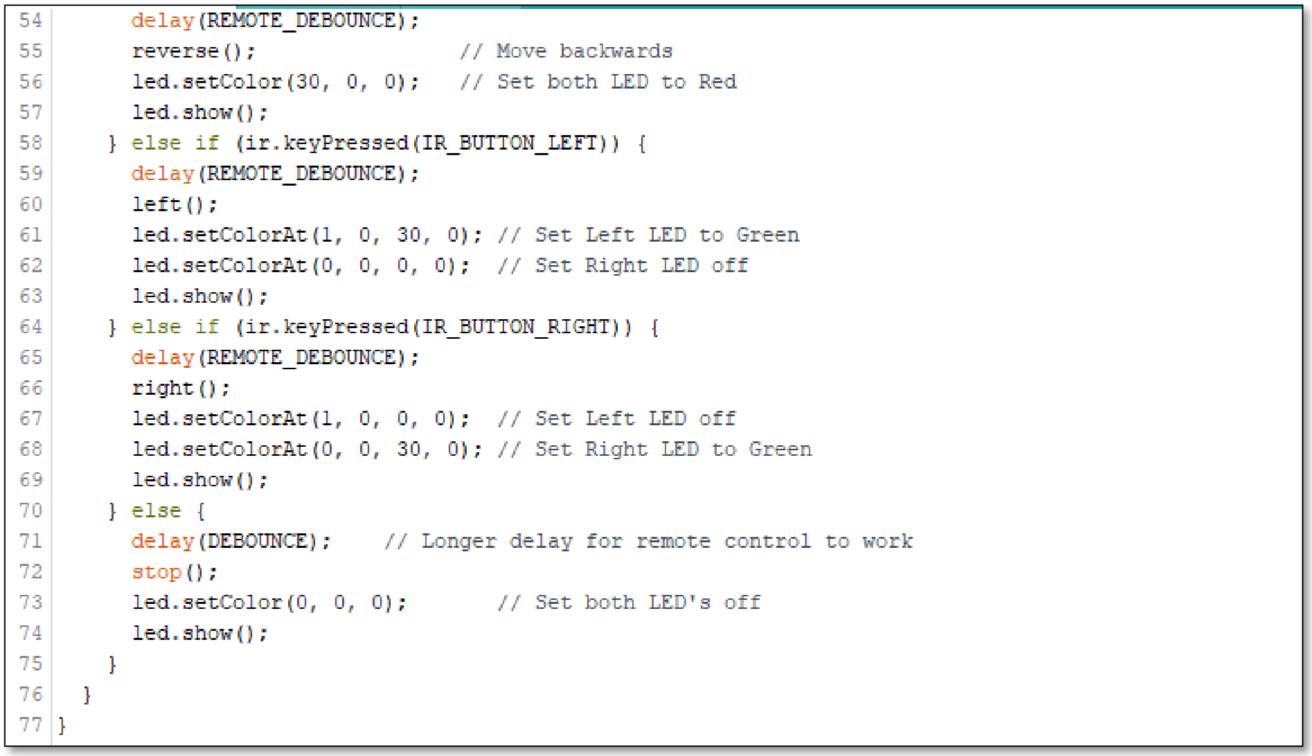
1. All students: Zip up the sketch folder. Attach the zip file to the assignment in Blackboard.

2. The assignment is demonstrated in class.

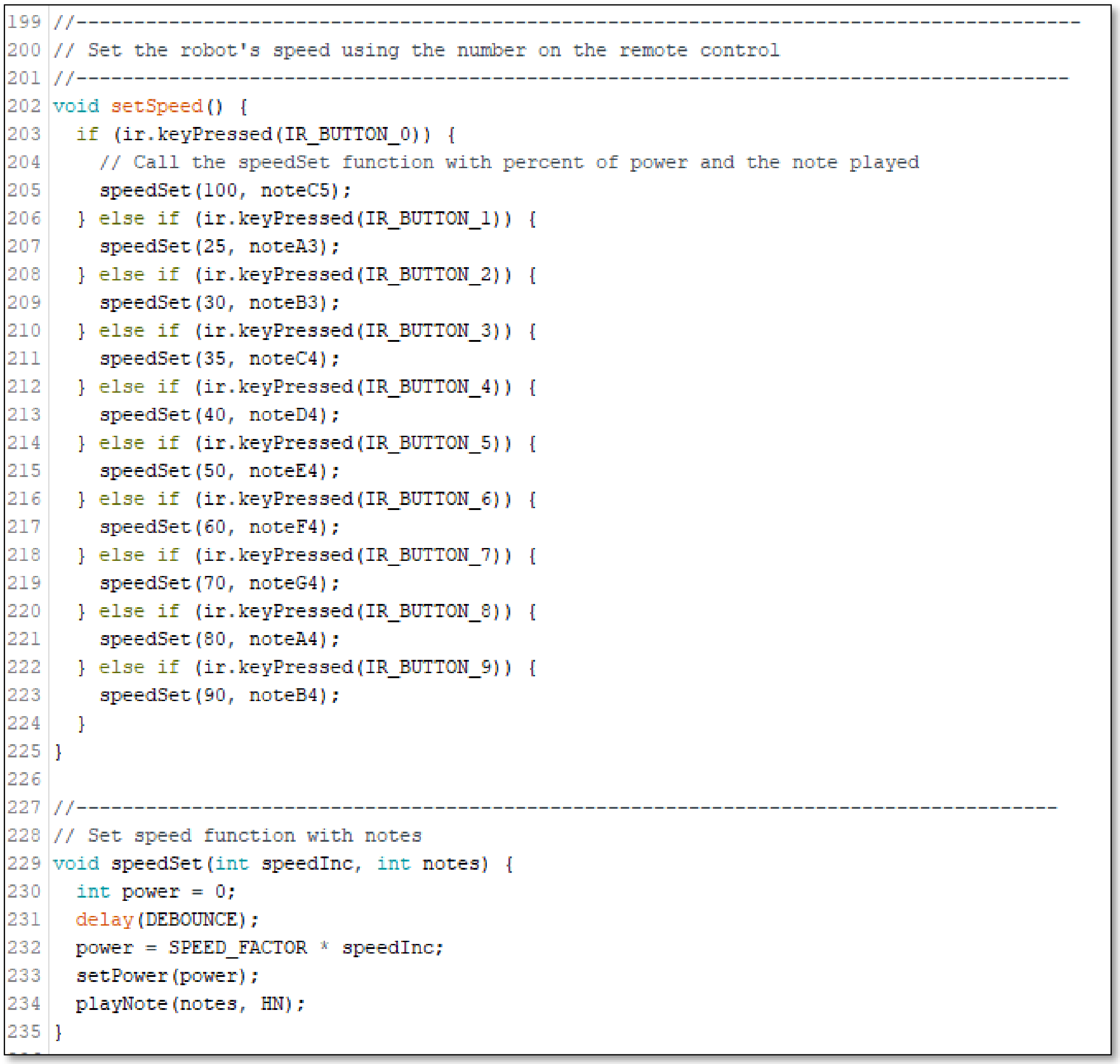
3. Online students: A link to a YouTube video recording showing your robot going through its motions is placed in the submission area in BlackBoard.



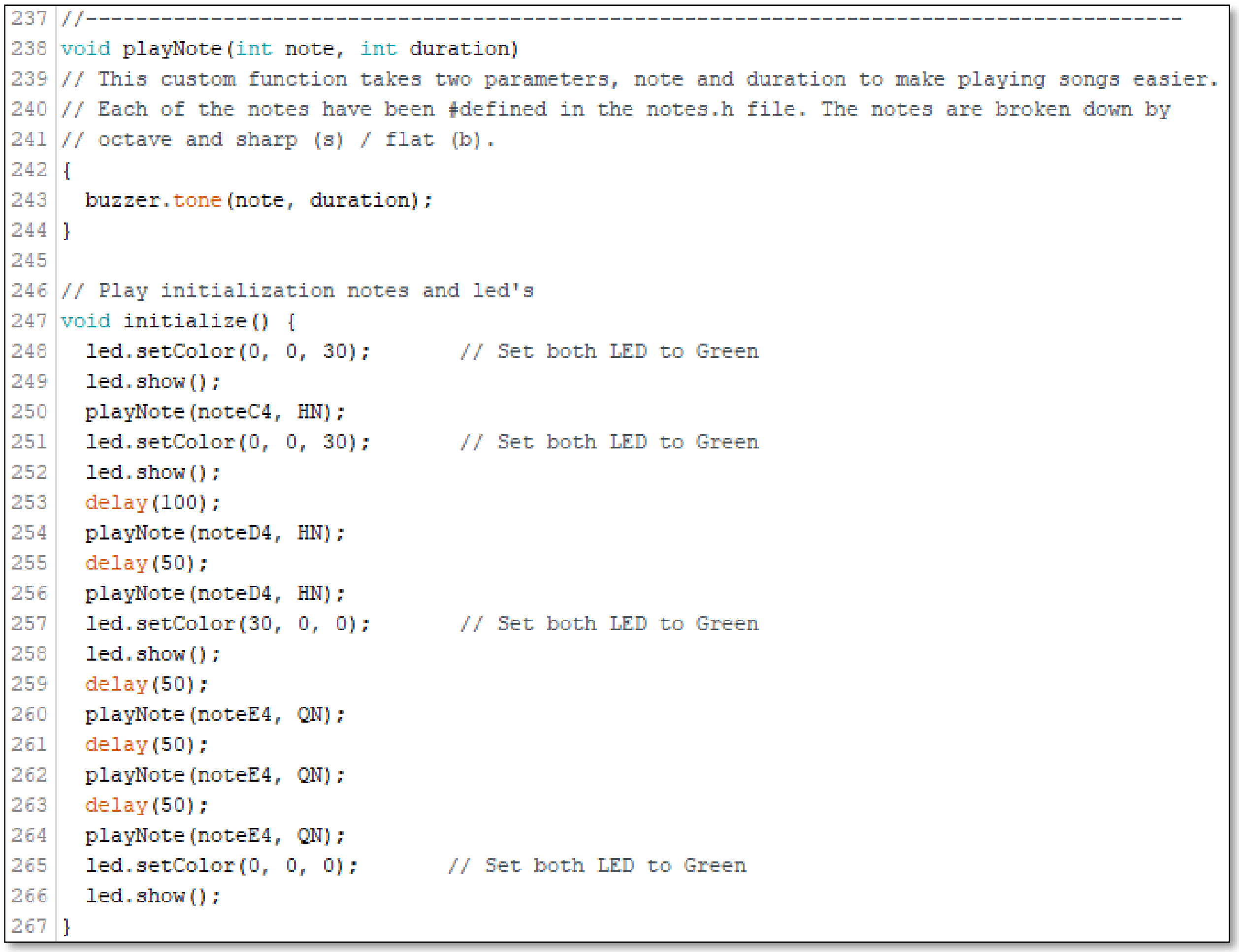
```
/**
   @file    DefaultProgram1.ino
   @author  William A Loring
   @version V1.0.0
   @date revised 10/26/2020  created: 12/10/16
   @Description: Part 1 of mBot Default Program, remote control
*/
#include <MeMCore.h>  // Include mBot library
#include "Movement.h" // Include custom Movement function library
#include "notes.h"    // Include notes library for ease of creating sounds
MeIR ir;              // Create ir remote object
MeBuzzer buzzer;      // Create buzzer object
MeRGBLed led(0, 30);  // Create onboard LED object
int modeFlag = 0;     // Flag to track the state of robot Mode
const int DEBOUNCE = 200;    // Time it takes to debounce the ir remote keys
const int REMOTE_DEBOUNCE = 50;  // Time it takes to debounce while
// holding down the remote keys for remote control
void setup() {
  ir.begin();     // Start listening to the remote
  led.setpin(13); // Set the Arduino pin for the led's
  initialize();   // Play initialization sounds and show LED's
}

void loop() {
  setMode();         // Check ir remote for mode setting
  remoteControl();  // Check for modeFlag set to 0 for Remote control operation if Button A is pressed
}

//----------------------------------------------------------------------------------------
// Determine the robot's mode of operation, A or modeFlag 0 - Remote Control is default
//----------------------------------------------------------------------------------------
void setMode() {
  // Determine which remote button was pressed
  if (ir.keyPressed(IR_BUTTON_A)) {
    delay(DEBOUNCE);
    modeFlag = 0;           // Set Mode A, Remote Control
    playNote(noteC4, HN);   // Play note to indicate mode change
  }
}

//----------------------------------------------------------------------------------------
// Remote Control functions
//----------------------------------------------------------------------------------------
void remoteControl() {
  // If button A is pressed, remote control mode
  if (modeFlag == 0) {
    setSpeed(); // Set the speed of the mBot
    if (ir.keyPressed(IR_BUTTON_UP)) {
      delay(REMOTE_DEBOUNCE);
      forward();               // Move forward
      led.setColor(0, 30, 0);  // Set both LED to Green
      led.show();
    } else if (ir.keyPressed(IR_BUTTON_DOWN)) {
```



```
      delay(REMOTE_DEBOUNCE);
      reverse();                // Move backwards
      led.setColor(30, 0, 0);   // Set both LED to Red
      led.show();
    } else if (ir.keyPressed(IR_BUTTON_LEFT)) {
      delay(REMOTE_DEBOUNCE);
      left();
      led.setColorAt(1, 0, 30, 0); // Set Left LED to Green
      led.setColorAt(0, 0, 0, 0);  // Set Right LED off
      led.show();
    } else if (ir.keyPressed(IR_BUTTON_RIGHT)) {
      delay(REMOTE_DEBOUNCE);
      right();
      led.setColorAt(1, 0, 0, 0);  // Set Left LED off
      led.setColorAt(0, 0, 30, 0); // Set Right LED to Green
      led.show();
    } else {
      delay(DEBOUNCE);    // Longer delay for remote control to work
      stop();
      led.setColor(0, 0, 0);       // Set both LED's off
      led.show();
    }
  }
}
```



```
//---------------------------------------------------------------------------------------
// Set the robot's speed using the number on the remote control
//--------------------------------------------------------------------------------------
void setSpeed() {
  if (ir.keyPressed(IR_BUTTON_0)) {
    // Call the speedSet function with percent of power and the note played
    speedSet(100, noteC5);
  } else if (ir.keyPressed(IR_BUTTON_1)) {
    speedSet(25, noteA3);
  } else if (ir.keyPressed(IR_BUTTON_2)) {
    speedSet(30, noteB3);
  } else if (ir.keyPressed(IR_BUTTON_3)) {
    speedSet(35, noteC4);
  } else if (ir.keyPressed(IR_BUTTON_4)) {
    speedSet(40, noteD4);
  } else if (ir.keyPressed(IR_BUTTON_5)) {
    speedSet(50, noteE4);
  } else if (ir.keyPressed(IR_BUTTON_6)) {
    speedSet(60, noteF4);
  } else if (ir.keyPressed(IR_BUTTON_7)) {
    speedSet(70, noteG4);
  } else if (ir.keyPressed(IR_BUTTON_8)) {
    speedSet(80, noteA4);
  } else if (ir.keyPressed(IR_BUTTON_9)) {
    speedSet(90, noteB4);
  }
}

//-------------------------------------------------------------------------------------
// Set speed function with notes
void speedSet(int speedInc, int notes) {
  int power = 0;
  delay(DEBOUNCE);
  power = SPEED_FACTOR * speedInc;
  setPower(power);
  playNote(notes, HN);
}
```



```
//---------------------------------------------------------------------------------------
void playNote(int note, int duration)
// This custom function takes two parameters, note and duration to make playing songs easier.
// Each of the notes have been #defined in the notes.h file. The notes are broken down by
// octave and sharp (s) / flat (b).
{
  buzzer.tone(note, duration);
}

// Play initialization notes and led's
void initialize() {
  led.setColor(0, 0, 30);       // Set both LED to Green
  led.show();
  playNote(noteC4, HN);
  led.setColor(0, 0, 30);       // Set both LED to Green
  led.show();
  delay(100);
  playNote(noteD4, HN);
  delay(50);
  playNote(noteD4, HN);
  led.setColor(30, 0, 0);       // Set both LED to Green
  led.show();
  delay(50);
  playNote(noteE4, QN);
  delay(50);
  playNote(noteE4, QN);
  delay(50);
  playNote(noteE4, QN);
  led.setColor(0, 0, 0);       // Set both LED to Green
  led.show();
}
```

Add to **Movement.h**

```
/**
   @file    Movement.h
   @author  William A Loring
   @version V1.0.0
   @date Revised 10/30/20  Created: 12/07/17
   @Description: Portable mBot movement with methods library file
*/
#include <MeMCore.h>          // Include mBot library
// Create motor control objects
MeDCMotor MotorL(M1);         // MotorL is Left Motor
MeDCMotor MotorR(M2);         // MotorL is Right Motor
const int POWER = 127;              // Base power setting
const float COMP = 1.0;       // Compensation to make the robot drive straight
// Apply compensation to left motor
// Use round function to convert float result to integer
int lPower = round(POWER * COMP);
int rPower = POWER;
const int TURN_TIME = 530;     // Time in milliseconds it takes to turn 90 degrees right
const int DRIVE_TIME = 5400;   // Time in milliseconds it takes to go 48"
const int DISTANCE = 48;       // 48"
// Calculate inches per second
// (float) converts the integer DISTANCE to a float, otherwise there would be integer math
float inchPerSec = (float) DISTANCE / DRIVE_TIME;
// Set to this number for maximum speed to go straight with COMP
const float SPEED_FACTOR = 2.42; // Constant to change speed with remote

//----------------------------------------------------------------------------------------------------
// Reset power variables for remote speed control
void setPower(int pwr) {
  // Use round function from math.h to convert float result to integer
  lPower = round(pwr * COMP);  // Apply compensation to left motor
  rPower = pwr;                // Set right motor power
}
```

## Default Program 2: Smart Obstacle Avoidance

Time required: 120 minutes

Please read all the directions carefully before beginning the assignment.

1. Comment your code as shown in the tutorials and other code examples.
2. Follow all directions carefully and accurately.
3. Think of the directions as minimum requirements.

### Understanding

Demonstrate understanding of:

**ultrasonic sensor**

Add Smart Obstacle Avoidance to the mBot default program in Arduino.

### Assignment

1. Start the Arduino IDE. Open **DefaultProgram1 a**nd save it as **DefaultProgram2**.
2. Add the appropriate code from the **SmartObstacleAvoidance** program to the default program. You should be able to copy and paste some of the code.
3. Add an if else statement to access the B button as shown.
4. Use the **ObstacleAvoidance** function to catch the mode change.
5. **NOTE:** Remove the led code from the **ObstacleAvoidance** function. For some reason you can’t change to another mode with led code in the mode change function.
6. Modify the **Movement.h** file if necessary.
7. Complete and test the program with the requirements listed.

### Requirements

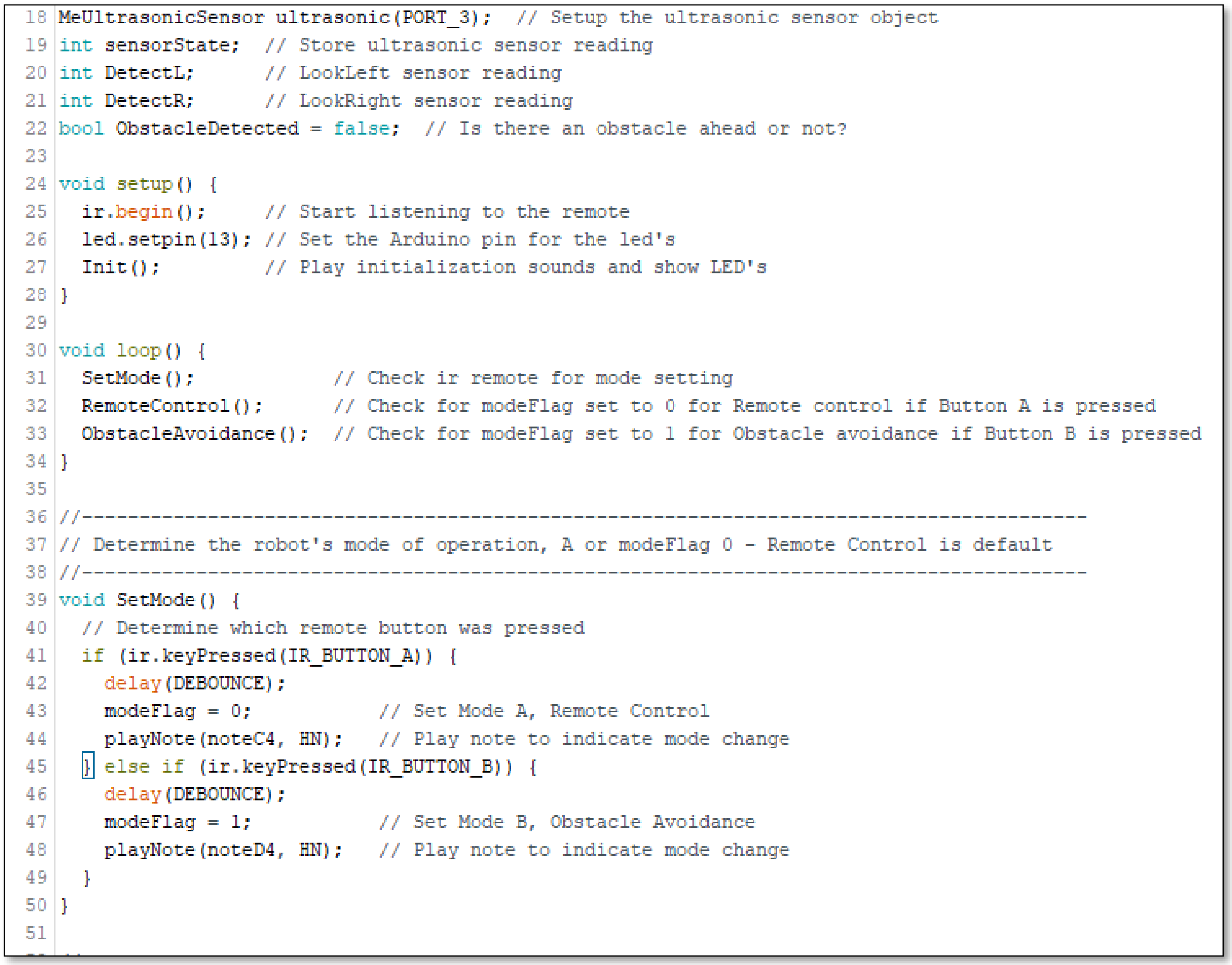
- Test the obstacle avoidance.
- Button A: Remote Control
- Button B: Smart Obstacle Avoidance.

## Assignment Submission

1. All students: Zip up the sketch folder. Attach the zip file to the assignment in Blackboard.

2. The assignment is demonstrated in class.

3. Online students: A link to a YouTube video recording showing your robot going through its motions is placed in the submission area in BlackBoard.



```
MeUltrasonicSensor ultrasonic(PORT_3);  // Setup the ultrasonic sensor object
int sensorState;  // Store ultrasonic sensor reading
int DetectL;      // LookLeft sensor reading
int DetectR;      // LookRight sensor reading
bool ObstacleDetected = false;  // Is there an obstacle ahead or not?

void setup() {
  ir.begin();     // Start listening to the remote
  led.setpin(13); // Set the Arduino pin for the led's
  Init();         // Play initialization sounds and show LED's
}

void loop() {
  SetMode();             // Check ir remote for mode setting
  RemoteControl();       // Check for modeFlag set to 0 for Remote control if Button A is pressed
  ObstacleAvoidance();   // Check for modeFlag set to 1 for Obstacle avoidance if Button B is pressed
}

//------------------------------------------------------------------------------------------
// Determine the robot's mode of operation, A or modeFlag 0 - Remote Control is default
//------------------------------------------------------------------------------------------
void SetMode() {
  // Determine which remote button was pressed
  if (ir.keyPressed(IR_BUTTON_A)) {
    delay(DEBOUNCE);
    modeFlag = 0;           // Set Mode A, Remote Control
    playNote(noteC4, HN);   // Play note to indicate mode change
  } else if (ir.keyPressed(IR_BUTTON_B)) {
    delay(DEBOUNCE);
    modeFlag = 1;           // Set Mode B, Obstacle Avoidance
    playNote(noteD4, HN);   // Play note to indicate mode change
  }
}

```

## Default Program 3: Smooth Line Following

Time required: 120 minutes

Please read all the directions carefully before beginning the assignment.

1. Comment your code as shown in the tutorials and other code examples.
2. Follow all directions carefully and accurately.
3. Think of the directions as minimum requirements.

### Understanding

Demonstrate understanding of:

**line follower**

### Assignment

1. Start the Arduino IDE. Open **DefaultProgram2 a**nd save it as **DefaultProgram3**.
2. Add the **SmoothLineFollowing** program to the mBot default program in Arduino. You should be able to copy and paste some of the code.

**NOTE:** Remove the led calls from the **SmoothLineFollowing** function, these interfere with the ability to change modes.

3. Modify the **Movement.h** file if necessary.
4. Complete and test the program with the requirements listed.

### Requirements

- Button A: Remote Control
- Button B: Smart Obstacle Avoidance
- Button C: Smooth Line Following

### Assignment Submission

1. All students: Zip up the sketch folder. Attach the zip file to the assignment in Blackboard.
2. The assignment is demonstrated in class.

3. Online students: A link to a YouTube video recording showing your robot going through its motions is placed in the submission area in BlackBoard.

## Default Program 4: Stay Inside the Line

Time required: 120 minutes

Please read all the directions carefully before beginning the assignment.

1. Comment your code as shown in the tutorials and other code examples.
2. Follow all directions carefully and accurately.
3. Think of the directions as minimum requirements.

### Understanding

Demonstrate understanding of:

**line follower**

The linefollowing sensor can be used for other tasks besides following a line. The linefollowing sensor detects dark and light. This can be used to keep the mBot inside a circle or other shape that is outlined in black. You can use your line following track or some sort of black tape to outline a shape. Automotive cloth wiring harness tape works really well. You could also use the paper line following track that came with the mBot by turning it upside down, and placing it on a dark surface. The mBot will stay on the white surface.

**NOTE:** Start the mBot inside the line.

### Tutorial Assignment

1. Start the Arduino IDE. Open **DefaultProgram3 a**nd save it as **DefaultProgram4**.
2. Complete and test the program with the requirements listed.

### Requirements

Example: https://youtu.be/ZhRuerC0Xgw

- Button A: Remote Control
- Button B: Smart Obstacle Avoidance
- Button C: Smooth Line Following
- Button D: Stay Inside the Line
- Your mBot should stay inside any black line.
- You can conceptually  base your sketch on the mBlock version picture.

- Use the Remote Control setSpeed function to slow down the mBot, otherwise it will overrun the line.
- You can compare the linefollowing sensor reading to < 3 in Arduino just like the mBlock program.

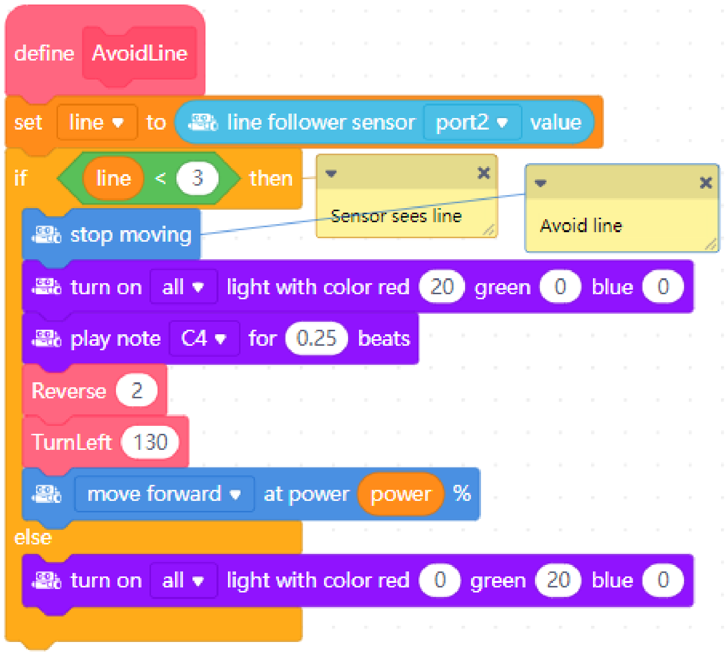
```
if (sensorState < 3) {  // Detect black and turn
```

## Challenge

- Randomly turn left or right to avoid the line. You might also backup first.
- For the avoidance, add a short sound and/or lights to indicate hitting the line.

## Assignment Submission

1. All students: Zip up the sketch folder. Attach the zip file to the assignment in Blackboard.
2. The assignment is demonstrated in class.
3. Online students: A link to a YouTube video recording showing your robot going through its motions is placed in the submission area in BlackBoard.



```
define AvoidLine
set line to line follower sensor port2 value
if line < 3 then
    stop moving
    turn on all light with color red 20 green 0 blue 0
    play note C4 for 0.25 beats
    Reverse 2
    TurnLeft 130
    move forward at power power %
else
    turn on all light with color red 0 green 20 blue 0
```

## Default Program 5: Maze Solving

Time required: 120 minutes

### Requirements

Example: https://youtu.be/ZhRuerC0Xgw

- Button A: Remote Control
- Button B: Smart Obstacle Avoidance
- Button C: Smooth Line Following
- Button D: Stay Inside the Line
- Button E: Follow the Line and Avoid an Obstacle

## Default Program 6: Shake that Bot!

Time required: 120 minutes

Please read all the directions carefully before beginning the assignment.

1. Comment your code as shown in the tutorials and other code examples.
2. Follow all directions carefully and accurately.
3. Think of the directions as minimum requirements.

### Understanding

Demonstrate understanding of:

**libraries, functions**

The mBot can appear to be playing a song and moving at the same time. It looks like the mBot is dancing to the music. You will want a catchy song.

Here is a web site to get you started with a known song. https://noobnotes.net

### Requirements

Example: https://youtu.be/ZhRuerC0Xgw

- Button A: Remote Control
- Button B: Smart Obstacle Avoidance
- Button C: Smooth Line Following
- Button D: Stay Inside the Line
- Button E: Follow the Line and Avoid an Obstacle
- Button F: Shake that mBot!
- Add this to your Arduino Default Program. Save it as **DefaultProgramDanceParty**
- Use **notes.h** and M**ovement.h**
- Spin in circles, wiggle back and forth, make turns, move forward and backward, etc.
- **Movement function without a parameter:** The music will keep playing while the mBot moves. The mBot will keep moving until you change to another movement or stop.

- **Movement function with a parameter:** The music will stop and wait until the movement is complete.
- **Slow song:** If you use a slow song, it can have more movements changes per number of notes played.
- **Fast song:** If you use a fast song, it would have less movement changes per the number of notes played.
- You do not have to do the whole song, just a part of it.
- The music dance party should last a minimum of 15 seconds.
- Comment your code. Please put the name of the song in the comments.

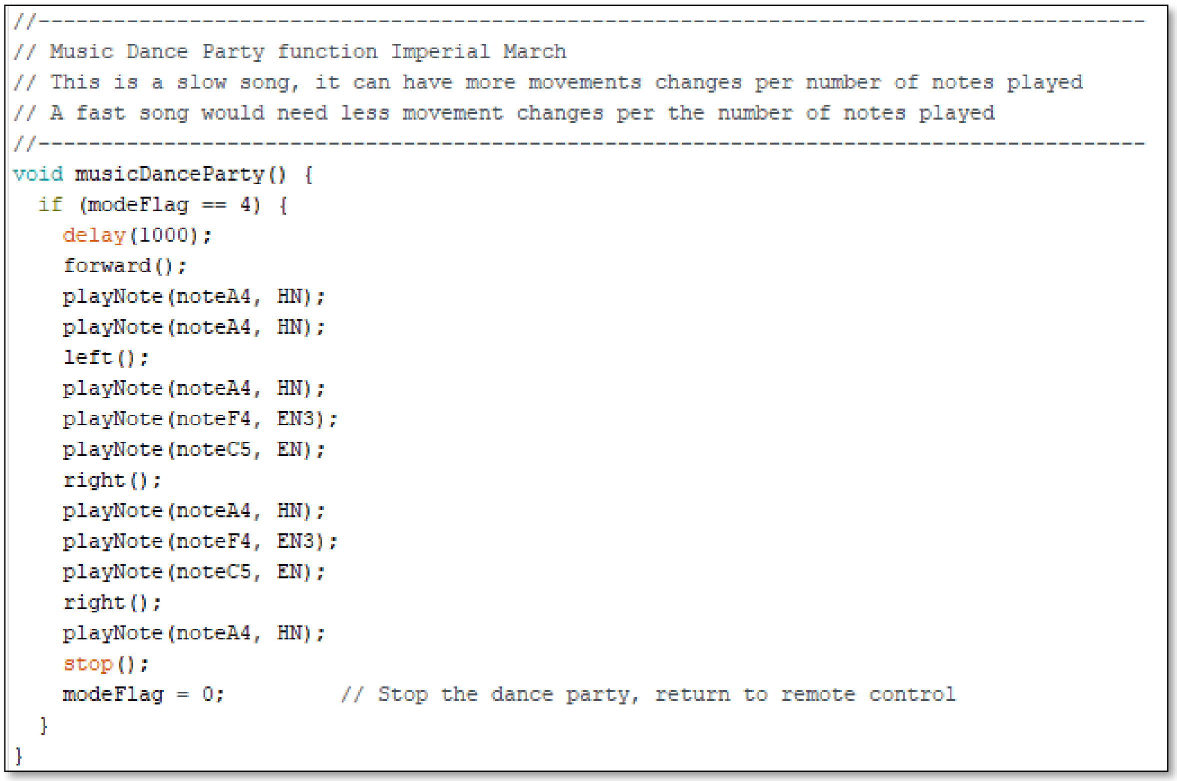
## Assignment Submission

1. All students: Zip up the sketch folder. Attach the zip file to the assignment in Blackboard.
2. The assignment is demonstrated in class.
3. Online students: A link to a YouTube video recording showing your robot going through its motions is placed in the submission area in BlackBoard.

## Examples

Be creative: Find your own song and your own path!

- mBot Imperial March
- 11/19/18 I wish you a merry mBot Christmas from Andrew
- 11/19/18 4 mBots moving and playing Mario!



```
//----------------------------------------------------------------------------------------
// Music Dance Party function Imperial March
// This is a slow song, it can have more movements changes per number of notes played
// A fast song would need less movement changes per the number of notes played
//----------------------------------------------------------------------------------------
void musicDanceParty() {
  if (modeFlag == 4) {
    delay(1000);
    forward();
    playNote(noteA4, HN);
    playNote(noteA4, HN);
    left();
    playNote(noteA4, HN);
    playNote(noteF4, EN3);
    playNote(noteC5, EN);
    right();
    playNote(noteA4, HN);
    playNote(noteF4, EN3);
    playNote(noteC5, EN);
    right();
    playNote(noteA4, HN);
    stop();
    modeFlag = 0;         // Stop the dance party, return to remote control
  }
}
```

## SOS (LED's and Arrays)

Time required: 30 minutes

Please read all the directions carefully before beginning the assignment.

1. Comment your code as shown in the tutorials and other code examples.
2. Follow all directions carefully and accurately.
3. Think of the directions as minimum requirements.

### Understanding

Demonstrate understanding of:

**arrays**

A variable stores only a single value. Arrays store more than one instance of a variable, they store a list of variables/values. Please go to the following web site to learn more about arrays.

https://startingelectronics.org/software/arduino/learn-to-program-course/17-arrays/

### Tutorial Assignment

1. Start the Arduino IDE. Save the sketch as **SOS**.
2. Complete and test the program as pictured with the requirements listed.

### Requirements

1. Complete and successfully run the program as displayed.
2. Comment your code.

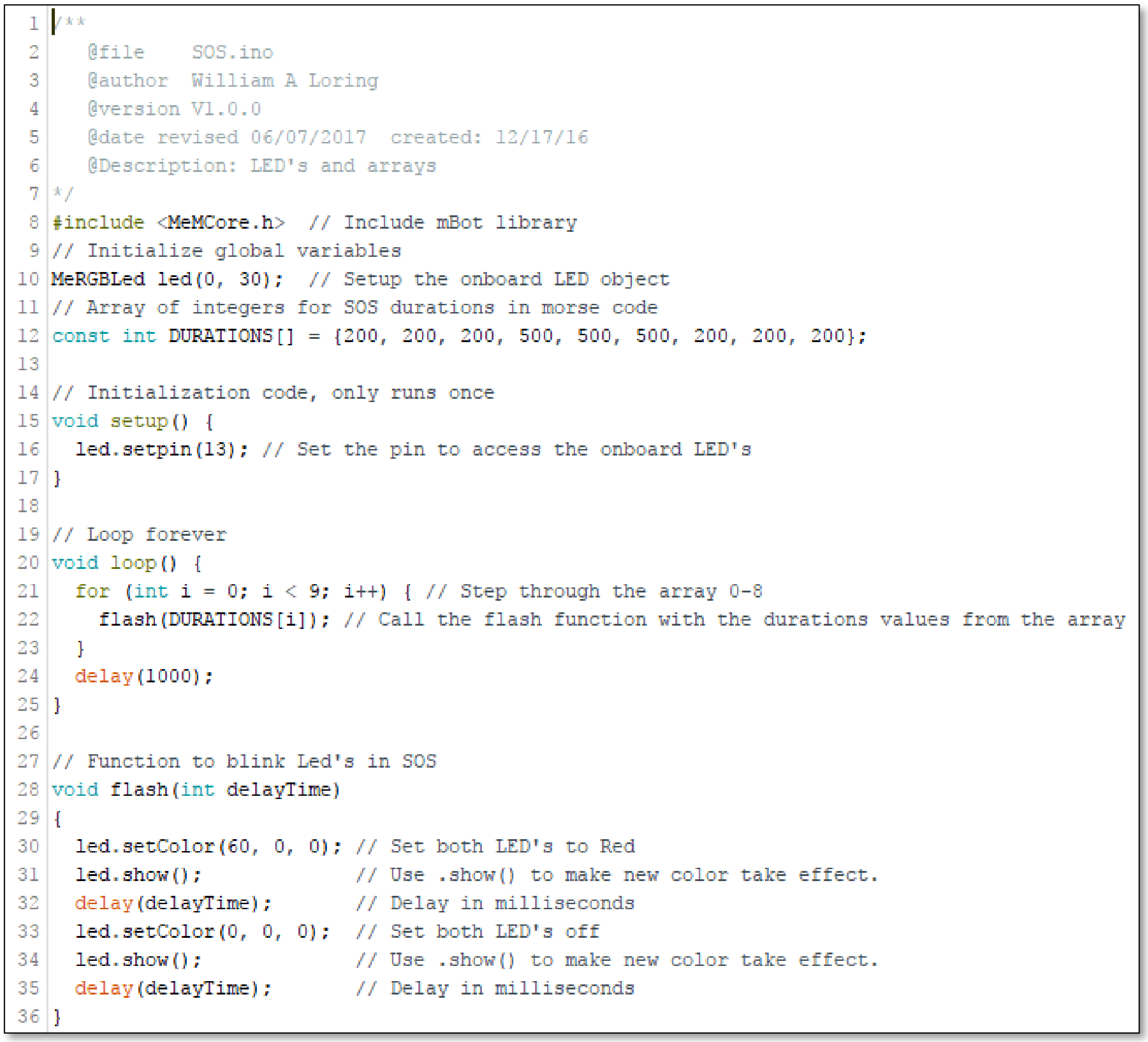
### Challenge

1. Add or just use the buzzer to send an SOS.

### Assignment Submission

1. All students: Zip up the sketch folder. Attach the zip file to the assignment in Blackboard.
2. The assignment is demonstrated in class.

3. Online students: A link to a YouTube video recording showing your robot going through its motions is placed in the submission area in BlackBoard.



```
/**
   @file    SOS.ino
   @author  William A Loring
   @version V1.0.0
   @date revised 06/07/2017  created: 12/17/16
   @Description: LED's and arrays
*/
#include <MeMCore.h>  // Include mBot library
// Initialize global variables
MeRGBLed led(0, 30);  // Setup the onboard LED object
// Array of integers for SOS durations in morse code
const int DURATIONS[] = {200, 200, 200, 500, 500, 500, 200, 200, 200};

// Initialization code, only runs once
void setup() {
  led.setpin(13); // Set the pin to access the onboard LED's
}

// Loop forever
void loop() {
  for (int i = 0; i < 9; i++) { // Step through the array 0-8
    flash(DURATIONS[i]); // Call the flash function with the durations values from the array
  }
  delay(1000);
}

// Function to blink Led's in SOS
void flash(int delayTime)
{
  led.setColor(60, 0, 0); // Set both LED's to Red
  led.show();             // Use .show() to make new color take effect.
  delay(delayTime);       // Delay in milliseconds
  led.setColor(0, 0, 0);  // Set both LED's off
  led.show();             // Use .show() to make new color take effect.
  delay(delayTime);       // Delay in milliseconds
}
```

## mBot Synchronized Dance Party!

Working in teams of two or more, create a synchronized robot dance!

### Requirements

- Use dead reckoning to create synchronized movement.
- Use Calibrate Movement get squared away.
- Break the movements into functions for easier troubleshooting.

### Assignment Submission

1. All students: Zip up the sketch folder. Attach the zip file to the assignment in Blackboard.
2. The assignment is demonstrated in class.
3. Online students: A link to a YouTube video recording showing your robot going through its motions is placed in the submission area in BlackBoard.

## Obstacle Course

Time required: 60 minutes

Please read all the directions carefully before beginning the assignment.

1. Comment your code as shown in the tutorials and other code examples.
2. Follow all directions carefully and accurately.
3. Think of the directions as minimum requirements.

Use Arduino programming to successfully navigate an obstacle course by dead reckoning. Dead reckoning means to navigate without any outside input, like a sensor. A simple example: place an object, start the robot from a certain point, go around the object and return to the starting point.

Add this program to your last Default Program to navigate the Obstacle Course. This program includes the blocks that will drive your robot a certain distance, and turn at a specific angle. You may want to include an if statement like the remote control block to control when it starts, and stops.

### Requirements

1. Use CalibrateMovement to calibrate your robot.
2. Measure the distances and angles in the obstacle course.
3. Navigate without sensors.
4. There must be a minimum of three obstacles to navigate around and between.

### Assignment Submission

- All students attach the finished program to the assignment in Blackboard.
- Each assignment can be demonstrated in class.
- For online students, a link to a YouTube video recording showing the assignment can be placed in the submission area in BlackBoard.