Challenge 1: PWM Control

```
#include "mbed.h"
PwmOut myled(p21);
float duty=0;

int main() {

  myled.period(1.0/10000);    //10 kHz

  while (1) {

   duty=duty+0.1;  // 0 - 1 value. 10% jumps
   if (duty>=1) (duty = 0); // if you get over 100%, zero it.
   myled.write(duty);
   wait(1); //wait a second before adding 10% to the duty cycle
  }
}
```

The above code controls the brightness of an LED using a PWM object named “myled” which is assigned to pin 21. Once created, the object is used to define a period of 100uS (corresponding to a frequency of 10kHz) and an infinite while loop is set up. Over each iteration of the loop the value of the ‘duty’ variable (representing duty cycle of the PWM) is incremented by 0.1 (10% increases) until it reaches the value of 1.0, where it is reset to 0. Finally, the myled object uses the PWMout function ‘write()’ to output the current duty cycle to the LED connected to pin 21.

```
#include "mbed.h"

PwmOut led(p5);

int main() {
    while(1) {
        led = led + 0.10;
        printf("LED is now %.2f\n", led.read());
        wait(0.2);

        if(led == 1.0) {
            led = 0;
        }
    }
}
```

A similar effect is produced by the above code only without defining a period and specific duty cycle variable. In this example, the value is directly assigned to the PwmOut object ‘led’ using the ‘=’ operator.

```
#include "mbed.h"

PwmOut speaker(p21);

void play_tone(float frequency, float volume, int interval, int rest) {
    speaker.period(1.0 / frequency);
    speaker = volume;
    wait(interval);
    speaker = 0.0;
    wait(rest);
}

int main()
{
    while(1) {
        play_tone(200.0, 0.5, 1, 0);
        play_tone(150.0, 0.5, 1, 0);
        play_tone(125.0, 0.5, 1, 2);
    }
}
```

This code uses PWM to control the volume of an audio signal output to a speaker. The program creates a function which defines a period that is specific to each individual frequency so that the ‘volume’ variable can act as the duty cycle and control the volume of each tone.