EAS 199
Fall 2011
v: October 20, 2011

Arduino Programs for a Breathing LED

Gerald Recktenwald
gerry@me.pdx.edu

1 Goal: A Breathing LED Indicator

When the lid of an Apple Macintosh laptop is closed, an LED indicator light pulses with the rhythm of human breathing. On December 2, 2003, Apple was awarded US Patent number 6,658,577 for an *Breathing status LED indicator*. The goal of this exercise is to develop an Arduino program to imitate the Apple LED indicator.

1.1 Learning Objectives

These notes present a series of Arduino programming snippets that implement aspects of the breathing LED. The programs start simple and become more complex. The first program creates a repeating pattern of three constant brightness levels. Alternative methods of obtaining the constant brightness levels are presented. Next, the lightness level is varied linearly during the inhale and exhale portions of the breathing pattern. Your are expected to make the final modifications to produce a nonlinear variation during the inhale and exhale portions.

The learning objectives for this exercise are

1. Be able to use the `analogWrite` and `delay` functions to create a repeating pattern of three constant LED brightness levels.
2. Be able to use loop structures to achieve the same repeating pattern of three constant LED brightness levels.
3. Be able use loop structures to create a repeating pattern of linearly increasing, constant, and linearly decreasing LED brightness levels.
4. Be able to use a single loop structure and `if` statements to achieve the same pattern of linearly increasing, constant, and linearly decreasing intensity.

As a bonus objective for students rapidly progressing through the material

5. Be able to use the `millis` command to more precisely control the timing of the linearly varying pattern of LED brightness patterns.

These goals reflect the larger objective of learning programming patterns with the Arduino platform. Furthermore, by achieving these objectives, students will be well-prepared to complete the assignment of achieving a breathing LED that varies in a more natural pattern than the constant levels (objectives 1 and 2) or linearly varying levels (objectives 3 and 4).

Following the exercises will give you a series of codes of increasing complexity. We strongly recommend that you save each code as a separate sketch rather than continuously modifying the same sketch. By saving the code you will be able to revisit these notes and study your own intermediate steps. You will also have working code to revert to if, as you add new features, you find yourself with a severely broken code. In that case you can discard broken code and start over from an earlier, saved version.

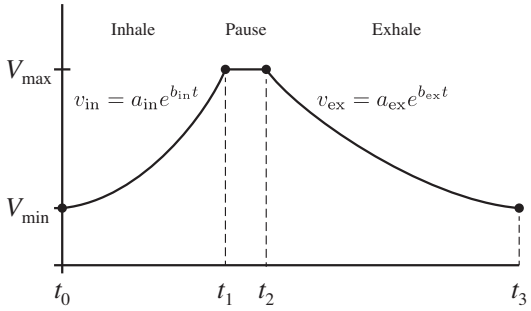
These exercises presume that you already understand

- Basic Arduino program structure

- Syntax of `for` loops and `while` loops
- Syntax of `if` constructs
- How to build a circuit for blinking an LED with Arduino
- Use of PWM to control an LED

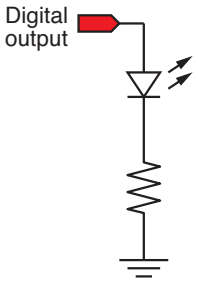
1.2 Pattern of the Breathing LED

The diagram to the right represents the pattern of light intensity for the final implementation of the Arduino program. The breathing pattern has three phases: inhale, pause, and exhale. The inhale and exhale phases are modeled with exponentially increasing and exponentially decreasing functions of time. The pattern repeats every $t_3 - t_0$ units of time. The inhale phase ends at t_1, and the exhale phase begins at t_2.



1.3 Circuit for the Breathing LED

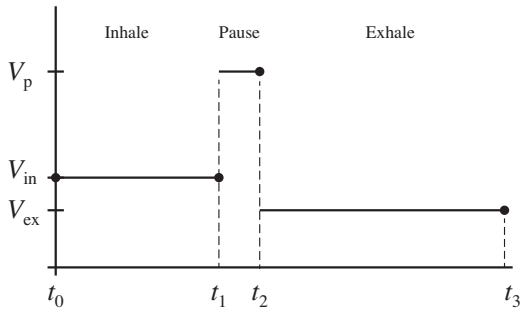
The circuit for this project is depicted in the sketch to the right. One of the PWM output pins (digital outputs 3, 5, 6, 9, 10 or 11 on an Arduino Uno or Duemilanove) is connected to the LED, which is tied to ground by a current-limiting resistor. The resistor can be between 330 Ω and 10 kΩ. Smaller resistors result in a brighter LED for a given PWM output.



2 Constant Brightness Levels

To establish a starting code structure, and to verify that the electrical circuit is working correctly, we first develop a very simple code that uses the `delay` function to control the duration of a constant brightness level. As depicted in the figure to the right, the light intensity of the LED changes from V_{in} to V_p and then to V_{ex} during each cycle.

The value of V can be thought of as a voltage. However, for convenience we never convert the V values to voltage. Instead, we use the 8-bit scale ($0 \leq V \leq 255$) of the PWM output channel.



2.1 Basic Timing with the `delay` Function

To implement the three step brightness pattern, enter the code to the right in the `loop` function of an Arduino sketch. The three brightness levels mark the three phases of the breath cycle, and are only meant to establish the proper timing. The PWM outputs for those three phases have the arbitrary numerical values `Vin`, `Vpause`, and `Vex`.

You should experiment with the values of `Vin`, `Vpause`, and `Vex` to develop a feel for the brightness levels obtained with different values of the PWM duty cycle. The perceived brightness is not linearly related to the value of the duty cycle.

```
int Vin = ...;
int Vpause = ...;
int Vex = ...;

analogWrite(LED_pin, Vin);
delay(2000);

analogWrite(LED_pin, Vpause);
delay(500);

analogWrite(LED_pin, Vex);
delay(2500);
```

Note that the values of `Vin`, `Vpause`, and `Vex` must be in the range $0 \leq \texttt{V} \leq 255$ because the second argument of the `analogWrite` function is an 8-bit value. The argument of the `delay` function is the time to pause in milliseconds.

2.2 Alternative Timing with Loops

The next step is to rewrite the `LED_three_levels` sketch so that the timing for each phase of LED brightness is controlled by a loop that repeatedly calls the `analogWrite` and `delay` function a pre-determined number of times. This alternative version of the code does not yield an immediate benefit – the `LED_three_levels` code is actually simpler – but it sets the stage for more complex control of the LED brightness pattern.

If the time delay for each phase of the brightness pattern is contained in a loop, the total duration of each phase is approximately equal to the product of time delay per loop and the number of loop repetitions.

$$\begin{array}{c}\text{Time duration}\\ \text{per phase}\end{array} \approx \begin{array}{c}\text{Time delay}\\ \text{per loop}\end{array} \times \begin{array}{c}\text{Number of}\\ \text{loop repetitions}\end{array}$$

The approximate equality is due to the nonzero time taken by other operations in the loop. The actual time taken for execution of each loop will be greater than the time spent waiting for the `delay(dtwait)` function to execute.

To implement the loop-based delay, enter the code to the right in the `loop` function of an Arduino sketch. `dtwait` is the time delay added to each loop. `nin`, `npause` and `nex` are the number of loop repetitions for the inhale, pause, and exhale phases, respectively.

Inexperienced Arduino programmers can be confused by the use of loops inside the `loop` function. First, remember that loops can be contained inside of other loops, so the loops inside the loop function are perfectly normal programming practice. Second, the `loop` function is not really a loop anyway, it is just a function with the *name* "`loop`". The `loop` function is the heartbeat of any Arduino sketch. The `loop` function is repeated continuously until either the reset button is pushed or the power to the board is removed.

```
int dtwait = 500;
int i, nin=4, npause=1, nex=5;

for ( i=1; i<=nin; i++ ) {
  analogWrite(LED_pin, Vin);
  delay(dtwait);
}

for ( i=1; i<=npause; i++ ) {
  analogWrite(LED_pin, Vpause);
  delay(dtwait);
}

for ( i=1; i<=nex; i++ ) {
  analogWrite(LED_pin, Vex);
  delay(dtwait);
}
```

Many combinations of the time delay per loop and number of loop repetitions are feasible. For

example, the inhale phase can be achieved by either of these two combinations.

$$400 \text{ millisecond delay} \times 5 \text{ loop repetitions} \approx 2000 \text{ milliseconds}$$

$$5 \text{ millisecond delay} \times 400 \text{ loop repetitions} \approx 2000 \text{ milliseconds}$$

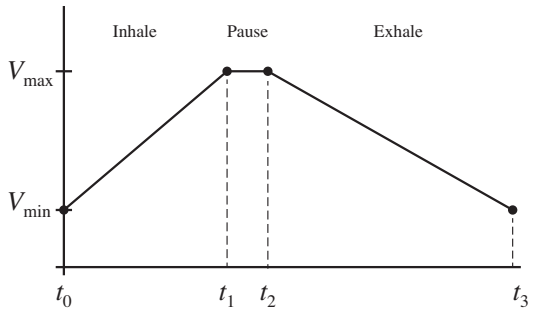
What happens when the timing constants are defined as follows:

```
int dtwait = 5;
int i, nin=400, npause=100, nex=500;
```

Answer: There is no perceptible change in the brightness pattern for the three constant brightness levels. However, for other intensity versus time functions, we want the inner loops to execute quickly so that the LED level is updated more frequently. In other words, a small value of `dtwait` and correspondingly larger values of `nin`, `npause`, and `nex` allow for faster updates to the voltage output, and hence a smoother variation in brightness as a function of time.

3 Linear Ramps for Intensity Levels

We now introduce linear variation of PWM output during the inhale and exhale phases of the breathing pattern. The brightness of the LED will increase and decrease, but the intensity pattern is not quite as organic as breathing. However, once the programming pattern is established with linearly increasing and linearly decreasing PWM output, it is relatively simple to replace the linear functions with other time varying functions.



The figure to the right shows the shape of the PWM output curve for linearly varying inhale and exhale phases. The magnitudes of the PWM output are specified with only two parameters, `Vmin` and `Vmax`. Remember that

The equations for the inhale and exhale output are

$$v = a_{\text{in}}t + b_{\text{in}} \qquad v = a_{\text{out}}t + b_{\text{out}} \tag{1}$$

where v is the value sent to the PWM output, a_{in} and b_{in} are the slope and intercept of the inhale function, and a_{ex} and b_{ex} are the slope and intercept of the exhale function. Remember that the values used in the `analogWrite` function must be limited to the range $0 \leq v \leq 255$.

3.1 Implementation with Separate Loops

A linear variation in PWM output to the LED can be implemented by modifying the code from Section 2.2. The simplest solution is to use separate loops for the inhale, pause and exhale phases. After showing how to use that solution with separate loops, an improved version of the code is developed, one that uses only a single loop.

The code except at the right shows the basic ideas. A complete sketch called `LED_linear_levels.pde` is given in Section 5 at the end of this document

The `ain` and `bin` coefficients are the slope and intercept of the PMW output function for the inhale phase. `aex` and `bex` are the slope and intercept for the exhale phase. The values of these coefficients are specified by the user. In the code to the right, the user-supplied values are represented by `...`. These are placeholders and the sketch will not compile if you literally enter `...` into the code.

```
//  Slopes and intercepts of ramps
double ain, bin, aex, bex;
double dt, t;
int    v;

ain = ...;    //  Slope of inhale curve
bin = ...;    //  Intercept for inhale curve
aex = ...;    //  Slope of exhale curve
bex = ...;    //  Intercept for exhale curve

t = 0.0;
for ( i=1; i<=nin; i++ ) {
  t += dt;
  v = int( ain*t + bin );
  analogWrite(LED_pin, v);
  delay(dtwait);
}

for ( i=1; i<=npause; i++ ) {
  analogWrite(LED_pin, Vpause);
  delay(dtwait);
}

for ( i=1; i<=nex; i++ ) {
  t += dt;
  v = int( aex*t + bex );
  analogWrite(LED_pin, v);
  delay(dtwait);
}
```

3.2 Alternative Implementation with a Single Loop

The use of three separate loops is clumsy. The code can be refactored so that there is a single loop for the inhale-pause-exhale cycle. Instead of counting cycles, the appropriate PWM output function is selected by comparing the time (computed from the loop counter) to the times at which the output functions change.

The code excerpt at the right shows how the two linear ramps can be evaluated in a single loop. A complete sketch called `LED_linear_levels_if.pde` is given in Section 5 at the end of this document

Note that the PWM output function is called only once at the end of the `if...else` structure. In other words, the only purpose of the `if...else` structure is to specify the correct value of `v` depending on the time.

```
int nstep = ...;
double t1 = ...;
double t2 = ...;
double dt = ...;

for ( i=1; i<=nstep; i++) {
  t = i*dt;
  if ( t <= t1 ) {
    v = int( ain*t + bin );
  }  else if ( t <= t2 ) {
    v = vpause;
  }  else  {
    v = int( aex*t + bex );
  }

  analogWrite(LED_pin, Vpause);
  delay(dt);
}
```

3.3 Alternative Implementation with the `fade` Example

The standard Arduino installation includes the `Fade.pde` sketch which provides a PWM output in the form of a slow triangle wave. If the output of the `Fade` sketch is connected to an LED, the brightness of an LED will increase and decrease indefinitely. To view the code, make the following menu selections from an open Arduino window

File → Examples → Basics → Fade

3.4 Alternative Implementation with `millis`

The preceding Arduino programs use the `delay` function to control the timing of the breathing LED algorithm. For some applications, more precise control of time is necessary. In this section the `millis` function is introduced to control the timing of the breathing LED. The precision is not necessary, but using `millis` to measure the time does not introduce substantial complexity.

Advantages of measuring time instead of using `delay`:

- No need to figure out the values of delay to achieve a desired timing of the program.
- Elimination of the error in timing due to execution of other program steps.

Disadvantages:

- Code is slightly more complex
- There is a potential subtle problem: `millis()` rolls over every 50 days (?). This can be overcome with a simple hack: test for `t < 0`

The key advantage of using the on-board clock is that it makes timing of the breathing computations independent of any other operations you may wish to have the Arduino perform. This also makes the breathing computations work without changes on any Arduino platform, regardless of the clock speed.

The code excerpt at the right shows how to read the on-board clock with the `millis` function, and how to use that time value to set the pace of the breathing algorithm. A complete sketch called `LED_linear_levels_millis.pde` is given in Section 5 at the end of this document

Note that a `while` loop is used instead of a `for` loop. This is more than a question of style. The `while` loop is more natural when the timing is not controlled by inserting calles to `delay(dt)`. The stopping condition in the `while` loop does not keep track of the number of time steps. Rather, the stopping condition in the `while` statement uses the current value of `t`, however that value is obtained. In this code, the value of `t` is determined by reading the on-board clock.

```
int  v;
unsigned long t, tstart;

//  save tstart so that t = millis() - tstart
//  is zero at beginning of loop function
tstart = millis();
t = 0;

while ( t < tcycle ) {

  if ( t <= dtin ) {
    v = int( ain*t + bin );
  } else if ( t <= t2 ) {
    v = Vmax;
  } else {
    v = int( aex*t + bex );
  }
  analogWrite(LED_pin, v);

  //  Get ready for next pass through loop
  t = millis() - tstart;
  if ( t<0 ) break;       //  Fix roll-over
}
```

4 Non-linear Variation During Inhale and Exhale

To create a more brightness pattern that is more natural, replace the linear ramps in intensity with other functions. For example, in an earlier lecture the following functions were fit to the inhale and exhale phases.

$$v_{\text{in}} = a_{\text{in}} e^{b_{\text{in}} t} \qquad\qquad v_{\text{ex}} = a_{\text{ex}} e^{b_{\text{ex}} t}$$

It is relatively straightforward to replace the linear segments with these or other formulas for PWM output versus time.

5 Answers

The following code listings show the complete Arduino sketches for the exercises described in the preceding sections.

Table 1: List of Arduino sketches for demonstrating aspects of the breathing LED code.

Arduino Code	Description
`LED_three_levels.pde`	Pattern of three constant brightness levels with the duration controlled by the `delay` function.
`LED_three_level_loops.pde`	Pattern of three constant brightness levels with the duration controlled by `for` loops and short time delays with the `delay` function.
`LED_three_level_loops_alt.pde`	Same as `LED_three_level_loops.pde` except for alternative timing variables.
`LED_linear_levels.pde`	PWM output to the LED is linearly increasing during inhale and linearly decreasing during exhale.
`LED_linear_levels_if.pde`	Same as `LED_linear_levels.pde` except that the three phases are executed within a single loop instead of three separate loops. The phases are determined by checking the time in an `if...else if` code structure.
`LED_linear_levels_if_global.pde`	Same as `LED_linear_levels_if.pde` except that parameters controlling the shape of the brightness function are global variables that are computed only once, instead of redundantly being computed on each pass through the `loop` function.
`LED_linear_levels_millis.pde`	Use the `millis` function to read the internal clock to control timing of the three phases. Retain the use of global variables from `LED_linear_levels_if_global.pde`. All timing parameters are now in milliseconds.

```
//   File:  LED_three_levels.pde
//
//   Use PWM to control the brightness of an LED
//   Repeat a pattern of three brightness levels
//
//   Gerald Recktenwald,  gerry@me.pdx.edu,  20 August 2011

int  LED_pin = 11;                  // Use pin 3, 5, 6, 9, 10 or 11 for PWM

void setup() {
  pinMode(LED_pin, OUTPUT);         // Initialize pin for output
}

void loop() {

  int  Vin=20, Vpause=220, Vex=80;  //  8-bit output values for PWM duty cycle

  analogWrite(LED_pin, Vin);        //  Inhale
  delay(2000);

  analogWrite(LED_pin, Vpause);     //  Pause
  delay(500);

  analogWrite(LED_pin, Vex);        //  Exhale
  delay(2500);
}
```

```
//   File:  LED_three_level_loops.pde
//
//   Use PWM to control the brightness of an LED.
//   Repeat a pattern of three brightness levels where the time delay
//   for each brightness is controlled by a loop.
//
//   Gerald Recktenwald,  gerry@me.pdx.edu,  20 August 2011

int  LED_pin = 11;                  //  must be one of 3, 5, 6, 9, 10 or 11

void setup() {
  pinMode(LED_pin, OUTPUT);         // Initialize pin for output
}

void loop() {

  int  dtwait=500;                  //  Time delay during each loop
  int  i, nin=4, npause=1, nex=5;   //  Index (i) and number of repetitions for each loop
  int  Vin=20, Vpause=220, Vex=80;  //  8-bit output values for PWM duty cycle

  for ( i=1; i<=nin; i++ ) {        //  Inhale
    analogWrite(LED_pin, Vin);
    delay(dtwait);
  }

  for ( i=1; i<=npause; i++ ) {     //  Pause
    analogWrite(LED_pin, Vpause);
    delay(dtwait);
  }

  for ( i=1; i<=nex; i++ ) {        //  Exhale
    analogWrite(LED_pin, Vex);
    delay(dtwait);
  }
}
```

```
//   File:  LED_three_level_loops_alt.pde
//
//   Use PWM to control the brightness of an LED.
//   Repeat a pattern of three brightness levels where the time delay
//   for each brightness is controlled by a loop.  Alternate timing version
//
//   Gerald Recktenwald,  gerry@me.pdx.edu,  20 August 2011

int  LED_pin = 11;                      //  must be one of 3, 5, 6, 9, 10 or 11

void setup() {
  pinMode(LED_pin, OUTPUT);             // Initialize pin for output
}

void loop() {

  int  dtwait=5;                        //  Time delay during each loop
  int  i, nin=400, npause=100, nex=500;  //  Index (i) and number of repetitions for each loop
  int  Vin=20, Vpause=220, Vex=80;      //  8-bit output values for PWM duty cycle

  for ( i=1; i<=nin; i++ ) {            //  Inhale
    analogWrite(LED_pin, Vin);
    delay(dtwait);
  }

  for ( i=1; i<=npause; i++ ) {         //  Pause
    analogWrite(LED_pin, Vpause);
    delay(dtwait);
  }

  for ( i=1; i<=nex; i++ ) {            //  Exhale
    analogWrite(LED_pin, Vex);
    delay(dtwait);
  }
}
```

```
//   File:  LED_linear_levels.pde
//
//   Use PWM to control the brightness of an LED.
//   Pattern is a linear ramp up to a constant, followed by a linear decrease
//   back to the starting intensity.  Repeat indefinitely.  Timing for each
//   phase is obtained with a separate loop.  Timing is imprecise because the
//   use of loop delays ignores time spent executing commands other than delay()
//
//   Gerald Recktenwald,  gerry@me.pdx.edu,  20 August 2011

int  LED_pin = 11;               //  must be one of 3, 5, 6, 9, 10 or 11

void setup() {
  pinMode(LED_pin, OUTPUT);      // Initialize pin for output
}

void loop() {

  int  i, nin,  npause, nex, dtwait;   //  Index, repetitions for each loop, and loop delay
  int  v, Vmin=20, Vmax=220;           //  PWM output (v) and min and max values of ramps
  double  ain, bin, aex, bex;          //  Slopes and intercepts of linear output functions
  double dt, dtin, dtpause, dtex, t;   //  Timing parameters

  dt = 0.01;                       //  Time step (seconds) don't make this smaller than 10 msec
  dtwait = dt*1000;                //  Loop delay (milliseconds) corresponding to dt
  dtin = 2.0;                      //  Time interval for inhale (seconds)
  dtpause = 0.5;                   //  Time interval for pause after inhale (seconds)
  dtex = 2.5;                      //  Time intervalfor exhale (seconds)
  nin = int( dtin/dt );            //  Number of time steps during inhale
  npause = int( dtpause/dt );      //  Number of time steps during pause
  nex = int( dtex/dt );            //  Number of time steps during exhale

  // -- Use other time interval and range parameters to compute slopes and intercepts of v(t)
  ain = double(Vmax - Vmin)/dtin;             //  Slope during inhale
  bin = double(Vmin);                         //  Intercept during inhale
  aex = double(Vmin - Vmax)/dtex;             //  Slope during exhale
  bex = double(Vmax) - aex*(dtin + dtpause);  //  Intercept during exhale

  t = 0.0;
  for ( i=1; i<=nin; i++ ) {                  //   Inhale
    t += dt;
    v = int( ain*t + bin );
    analogWrite(LED_pin, v);
    delay(dtwait);
  }

  for ( i=1; i<=npause; i++ ) {               //   Pause
    t += dt;
    analogWrite(LED_pin, Vmax);
    delay(dtwait);
  }

  for ( i=1; i<=nex; i++ ) {                 //   Exhale
    t += dt;
    v = int( aex*t + bex );
    analogWrite(LED_pin, v);
    delay(dtwait);
  }
}
```

```
//   File:  LED_linear_levels_if.pde
//
//   Use PWM to control the brightness of an LED.
//   Pattern is a linear ramp up to a constant, followed by a linear decrease
//   back to the starting intensity.  Repeat indefinitely.  Timing is controlled
//   by a single loop.  Switching between phases is determined with "if" statements
//   that check the current (estimate of) time.  Timing is still imprecise because the
//   use of loop delays ignores time spent executing commands other than delay()
//
//   Gerald Recktenwald,  gerry@me.pdx.edu,  20 August 2011

int  LED_pin = 11;               //  must be one of 3, 5, 6, 9, 10 or 11

// -------------
void setup() {
  pinMode(LED_pin, OUTPUT);      // Initialize pin for output
}

// -------------
void loop() {

  int  i, ncycle, dtwait;                  //  Index, total steps for all three phases, loop delay
  int  v, Vmin=20, Vmax=220;               //  PWM output (v) and min and max values of ramps
  double  ain, bin, aex, bex;              //  Slopes and intercepts of linear output functions
  double dt, dtin, dtpause, dtex, t, t3;   //  Timing parameters

  dt = 0.01;                               //  Time step (seconds). Should be >= 10 milliseconds
  dtwait = dt*1000;                        //  Loop delay (milliseconds) corresponding to dt
  dtin = 2.0;                              //  Time interval for inhale (seconds)
  dtpause = 0.5;                           //  Time interval for pause after inhale (seconds)
  dtex = 2.5;                              //  Time intervalfor exhale (seconds)
  t3 = dtin + dtpause;                     //  Time at end of the pause (seconds)
  ncycle = ( dtin + dtpause + dtex ) / dt;  //  Total time steps in a cycle

  // -- Use other time interval and range parameters to compute slopes and intercepts of v(t)
  ain = double(Vmax - Vmin)/dtin;          //  Slope during inhale
  bin = double(Vmin);                      //  Intercept during inhale
  aex = double(Vmin - Vmax)/dtex;          //  Slope during exhale
  bex = double(Vmax) - aex*t3;             //  Intercept during exhale

  t = 0.0;
  for ( i=1; i<=ncycle; i++ ) {
    t += dt;
    if ( t <= dtin ) {
      v = int( ain*t + bin );                //  Inhale
    } else if ( t <= t3 ) {
      v = Vmax;                              //  Pause
    } else {
      v = int( aex*t + bex );                //  Exhale
    }
    analogWrite(LED_pin, v);
    delay(dtwait);
  }
}
```

```
//   File:  LED_linear_levels_if_global.pde
//
//   Use PWM to control the brightness of an LED.
//   Pattern is a linear ramp up to a constant, followed by a linear decrease
//   back to the starting intensity.  Repeat indefinitely.  Timing is controlled
//   by a single loop.  Switching between phases is determined with "if" statements
//   that check the current (estimate of) time.  Timing is still imprecise because the
//   use of loop delays ignores time spent executing commands other than delay().
//   Timing and ramp parameters are declared as gobal variables so they can be computed
//   once at startup.  This removes unecessary computation from the loop function.
//
//   Gerald Recktenwald,  gerry@me.pdx.edu,  20 August 2011

int  LED_pin = 11;               //  must be one of 3, 5, 6, 9, 10 or 11

int    Vmin=20, Vmax=220, ncycle, dtwait;   //  Min & max of ramps, total cycles, loop delay
double  ain, bin, aex, bex;                 //  Slopes and intercepts of linear output functions
double dt, dtin, dtpause, dtex, t, t3;      //  Timing parameters

// -------------
void setup() {
  pinMode(LED_pin, OUTPUT);                 // Initialize pin for output

  dt = 0.01;                                //  Time step (seconds). Should be >= 10 milliseconds
  dtwait = dt*1000;                         //  Loop delay (milliseconds) corresponding to dt
  dtin = 2.0;                               //  Time interval for inhale (seconds)
  dtpause = 0.5;                            //  Time interval for pause after inhale (seconds)
  dtex = 2.5;                               //  Time intervalfor exhale (seconds)
  t3 = dtin + dtpause;                      //  Time at end of the pause (seconds)
  ncycle = ( dtin + dtpause + dtex ) / dt;  //  Total time steps in a cycle

  // -- Use other time interval and range parameters to compute slopes and intercepts of v(t)
  ain = double(Vmax - Vmin)/dtin;           //  Slope during inhale
  bin = double(Vmin);                       //  Intercept during inhale
  aex = double(Vmin - Vmax)/dtex;           //  Slope during exhale
  bex = double(Vmax) - aex*t3;              //  Intercept during exhale
}

// -------------
void loop() {

  int  i, v;
  double t;

  t = 0.0;
  for ( i=1; i<=ncycle; i++ ) {
    t += dt;
    if ( t <= dtin ) {
      v = int( ain*t + bin );
    } else if ( t <= t3 ) {
      v = Vmax;
    } else {
      v = int( aex*t + bex );
    }
    analogWrite(LED_pin, v);
    delay(dtwait);
  }
}
```

```
//   File:  LED_linear_levels_millis.pde
//
//   Use PWM to control the brightness of an LED.
//   Pattern is a linear ramp up to a constant, followed by a linear decrease back to the
//   starting intensity.  Repeat indefinitely.  Timing is controlled  with reference to
//   the internal clock via millis().  Switching between phases is determined with "if"
//   statements that check the current time.  Timing and ramp parameters are gobal
//   variables that are computed once at startup.  Note that *all* times are in milliseconds
//
//   Gerald Recktenwald,  gerry@me.pdx.edu,  21 August 2011

int  LED_pin = 11;                          //  must be one of 3, 5, 6, 9, 10 or 11

int   Vmin=20, Vmax=220;                    //  Min & max of ramp output
int   dtin, dtpause, dtex, t3, tcycle;      //  Timing parameters, all in milliseconds
double  ain, bin, aex, bex;                 //  Slopes and intercepts of linear output functions

// -------------
void setup() {
  pinMode(LED_pin, OUTPUT);                 // Initialize pin for output

  dtin = 2000;                              //  Time interval for inhale (milliseconds)
  dtpause = 500;                            //  Time interval for pause after inhale (milliseconds)
  dtex = 2500;                              //  Time interval for exhale (milliseconds)
  t3 = dtin + dtpause;                      //  Time at end of the pause (milliseconds)
  tcycle = dtin + dtpause + dtex;           //  Total time for one cycle (milliseconds)

  // -- Use other time interval and range parameters to compute slopes and intercepts of v(t)
  ain = double(Vmax - Vmin)/double(dtin);   //  Slope during inhale
  bin = double(Vmin);                       //  Intercept during inhale
  aex = double(Vmin - Vmax)/double(dtex);   //  Slope during exhale
  bex = double(Vmax) - aex*double(t3);      //  Intercept during exhale
}

// -------------
void loop() {

  int  v;
  unsigned long t, tstart;

  //  save tstart, so that t = millis() - tstart is zero at beginning of loop function
  tstart = millis();
  t = 0;

  while ( t < tcycle ) {

    if ( t <= dtin ) {
      v = int( ain*double(t) + bin );
    } else if ( t <= t3 ) {
      v = Vmax;
    } else {
      v = int( aex*double(t) + bex );
    }
    analogWrite(LED_pin, v);

    //  Get ready for next pass through the loop
    t = millis() - tstart;
    if ( t<0 ) break;       //  Fix roll-over every 50 days or so
  }
}
```