ME 120: `for` Loops in Arduino

Gerald Recktenwald
gerry@pdx.edu

October 24, 2021

1 Overview

This document describes the `for` loop structure used in Arduino sketches. The syntax of a `for` loop is described and a series of simple example programs is used to demonstrate how `for` loops work. The article ends with a practical example of using a `for` loop to average a sensor reading.

Your Arduino code can have as many (or as few) loops as you need. Remember that the `loop` function is itself executed repeatedly. When using a `for` loop structure you are introducing an additional form of repetition.

1.1 A Motivating Example

There are many computing tasks that require repetition.

A `for` loop is an iteration structure that is most often used when there is a known number of repetitions to be performed. For example, if you want to blink an LED three times, you would (most likely) create a `for` loop that repeats three times. The body of a loop would contain the sequence necessary to blink the LED just once.

Another use of `for` loops is to evaluate of a mathematical formula with many terms that are expressible in a pattern. For example, to add up a list of numbers when the length of the list is known, use a `for` loop. A somewhat contrived example of this idea is the computation

$$S = \sum_{i=1}^{n} i$$

where the value of n is not known until the program executes. In this case the list of numbers to be added is just the sequence of integers, $1, 2, \ldots, n$. The translation of that formula into code involves a loop, as in the following code snippet.

```
int i, n, sum;

n = ...    //   The value of n is determined somehow

sum = 0;
for ( i=1; i<=n; i++) {
  sum = sum + i;
}
```

The summation is implemented with the loop structure that begins with the `for` statement. The calculations to be repeated are contained inside the curly brackets $\{ \ldots \}$. This simple example is intended to preview the discussion of `for` loop syntax, which is described in the next section.

A less contrived example is the averaging of a series of analog input measurements. That example is described in § 5.1 beginning on page 7.

1.2 Our Plan

Like any code feature, we need to consider both the syntax and application of `for` loops. We start with the syntax, which introduces some new notation and also uses the logical expressions found in

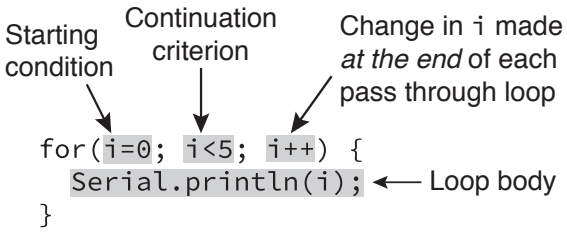

Figure 1: Components of the counter specification in a `for` loop.

`if` constructs. Understanding the syntax of `for` loops is necessary. Understanding how a `for` loop is used to implement an algorithm allows us to put that syntax to good use.

2 Syntax of a `for` Loop

In this section we describe the syntax of the the `for` loop structure. As a preview, consider Figure 1 which shows a `for` loop that prints out the first five integers, beginning with zero. The generic structure of a for loop is

```
for ( Starting condition; Continuation criterion; increment/decrement ) {
    Loop body
}
```

The *`Starting condition`*, *`Continuation criterion`* and the *`increment/decrement`* rule are expressions that the control how the *loop counter* changes on each pass through the loop. A loop counter, is a C/C++ variable. Often the loop counter is an integer variable, and often it is a single letter label like `i`, `j`, or `k`. However, the loop counter, starting condition, continuation criterion and increment/decrement can use any numeric variable type including `int`, `long`, and `float`.

The *`Starting condition`* assigns the initial value of the loop counter. The *`Continuation criterion`* is a logical statement that compares the loop counter to a value. The loop body is executed only if the continuation criterion is met. The *`increment/decrement`* rule is the formula for changing the loop counter on each pass through the loop. Table 1 gives some examples of common combinations of starting conditions, continuation criteria, and increment/decrement rules. Table 2 lists common examples of just the increment/decrement expression.

Table 1: Examples of rules for counters in a `for` loop. There are many possible different combinations of starting conditions, continuation criteria, and increment/decrement rules. The expressions in this table are common patterns.

Starting condition	Continuation criterion	Increment decrement rule	Description
`i = 0;`	`i < 5;`	`i++`	Start with `i=0`. Continue as long as `i` is less than 5. Increment `i` by one at the end of each pass through the loop.
`i = 1;`	`i <= n;`	`i+=1`	Start with `i=1`. Continue as long as `i` is less than or equal to the value stored in the variable `n`. Increment `i` by one at the end of each pass through the loop.
`i = 10;`	`i > 0;`	`i--`	Start with `i=10`. Continue as long as `i` is greater than 0. Decrement `i` by one at the end of each pass through the loop.
`i = 0;`	`i < 8;`	`i+=2`	Start with `i=0`. Continue as long as `i` is less than 8. Increment `i` by two at the end of each pass through the loop.

Table 2: Common increment/decrement expressions used in `for` loops.

Expression	Description
`i++`	Increment by 1 at the end of the loop
`i--`	Decrement by 1 at the end of the loop
`i+=1`	Same as `i++`
`i-=1`	Same as `i--`
`i+=2`	Increment by 2 at the end of the loop
`i-=2`	Decrement by 2 at the end of the loop
`++i`	Increment by 1 at the beginning of the loop
`--i`	Decrement by 1 at the beginning of the loop

3 Usage Cases for `for` Loops

The fundament purpose of a loop is to repeat an operation. The syntax of a `for` loop makes it the obvious choice with an operation needs to be repeated a known number of times[1]. A `for` loop is useful in the following situations

1. Repetition
   - Compute the average of sensor input to reduce the effect of noise.
   - Blink an indicator a specified number of times.
   - Apply an actuator multiple times (think hammering a nail)
2. Striding through a list
   - Turn on or off a set of switches or LEDs.
   - Sample a sequence of analog inputs.
   - Find the maximum (or minimum) value in an array.
   - Sweep a servo through a sequence of positions.

4 Examples to Test Your Understanding

This section provides a series of very simple Arduino sketches to demonstrate how `for` loops work. The sketches are designed to show how loops can be structured and are not meant to perform useful functions. Study these examples to see how the body of the loop is controlled by the loop counter expressions in the parentheses after the `for` key word. These example sketches also show how a `for` loop inside the `loop` function interacts with the repeated execution of the `loop` function. Refer to Section 5.1 for a scenario where a `for` loop performs a useful task.

> **Note to instructors and to students using these notes for self-study:**
> Explanations are given here to support instructors, or students doing self-study. It would be best for students to study these codes, and predict the outcome of running the code *before* consulting the notes in the right-hand column and *before* they use an Arduino to test their understanding.
>
> For students, I recommend using sticky notes to cover up the answers given for each exercise. Predict the behavior of the `for` loop and then remove the sticky note to check your understanding.
>
> For students, I also recommend reviewing the examples multiple times over the course of one or two or three weeks. For example, you can study the examples once or twice per week, each time with the answer initially covered with the sticky notes.
>
> Another study recommendation is to copy this code into the Arduino IDE. Manually typing each sketch instead of using copy/paste will also help you learn better as well as giving you practice fixing the small bugs that inevitably show up in code we write. During each study session – remember to space them throughout one or two weeks – open the code in the Arduino IDE and predict the output *before running the sketch*.
>
> And of course you can alternate the study techniques to reading and predicted the results on paper or with the Arduino IDE. There are two key ideas in this recommended study technique. First, make an honest effort predicting the output before seeing the answer. Second, space the study sessions over time, i.e., don't cram your studying into one long session.

[1] In contrast, a `while` loop is better suiting to the case where the number of repetitions is not known in advance.

Loop 1

```
void setup() {
  Serial.begin(9600);
}

void loop() {
  int i = 0;
  i = i + 1;
  Serial.println(i);
}
```

What is the output of the code to the left?

Answer: This code continuously prints "1" to the Serial Monitor. That is not likely to be the intent of the code developer. The declaration `int i = 0` resets the value of `i` every time the `loop` function is executed.

Loop 2

```
int i = 0;

void setup() {
  Serial.begin(9600);
}

void loop() {
  i = i + 1;
  Serial.println(i);
}
```

What is the output of the code to the left?

Answer: This code prints the integers 0, 1, 2, ... and continues until the reset button in pushed or the Arduino is disconnected from its power supply. Each integer is on a separate line because the `println` method of the `Serial` object is used.

If the reset button is pushed, the code begins executing again by running `setup` once, and then calling `loop` indefinitely. This causes the integers to be printed again, starting with 0, 1, 2, etc.

If the power to the Arduino is disconnected, the program stops running. The next time the power is restored, the program resumes as if the reset button had been pushed.

Loop 3

```
void setup() {
  Serial.begin(9600);
}

void loop() {
  int i;
  for ( i=0; i<5; i++ ) {
    Serial.println(i);
  }
}
```

What is the output of the code to the left?

Answer: The integers from 0 through 4 are printed in a repeating pattern, i.e., 0, 1, 2, 3, 4, 0, 1, 2, 3, 4, 0, 1, 2, 3, 4.... Each integer is printed on a separate line.

Loop 4

```
void setup() {
  Serial.begin(9600);
}

void loop() {
  int i;
  for ( i=0; i<5; i+=2 ) {
    Serial.println(i);
  }
}
```

What is the output of the code to the left?

Answer: The integers 0, 2, and 4 are printed in a repeating pattern i.e., 0, 2, 4, 0, 2, 4, 0, 2, 4,.... Each integer is printed on a separate line.

Loop 5

```
void setup() {
  Serial.begin(9600);
}

void loop() {
  int i;
  for ( i=0; i<10; i++) {
    i = 5;
    Serial.println(i);
  }
  Serial.println("for loop over\n");
}
```

What is the output of the code to the left?

Answer: This code prints the number 5 indefinitely. Each copy of 5 is on its own line. The message "`for loop over`" is never printed because the `for` loop never ends.

This code contains a programming bug: the loop index `i` is changed in the body of the loop. In a `while` loop it is often *necessary* to change the loop counter in the body of the loop. In a `for` loop, the loop index should only be changed inside the parentheses of the `for (...)` statement.

Never change the loop index inside the *body* of a `for` loop.

Loop 6

```
void setup() {
  Serial.begin(9600);
}

void loop() {
  int i;
  for ( i=0; i<10; i++) {
    Serial.println(i);
    delay(100);
  }
  Serial.print("\nfor loop over:  ");
  Serial.print("i = ");
  Serial.print(i);
  delay(2000);
}
```

What is the output of the code to the left?

Answer: The integers 0, 1, 2, ... 9 are printed, followed by the message "`for loop over:  i = 10`". Each integer, and the final message are printed on a separate line. The pattern of integers, followed by the text message, is repeated indefinitely. The calls to the `delay` function slow the execution of the sketch to make the output to the Serial Monitor easier to read.

Loop 7

```
void setup() {
  Serial.begin(9600);
}

void loop() {
  int i,n=3;
  for ( i=0; i<=n; i++ ) {
    Serial.print(i);
    Serial.print("  ");
  }
  for ( i=n-1; i>0; i-- ) {
    Serial.print(i);
    Serial.print("  ");
  }
  Serial.println("Both loops finished");
  delay(2000);
}
```

What is the output of the code to the left?

Answer: The pattern `0 1 2 3 2 1` pattern is printed repeatedly, with `Serial.print` statements that add text to a single line of output.
The `Serial.println("\nBoth loops finished")` statement prints the message to make it clear that the body of the `loop` function has finished. The `\n` newline character advances the output to the next line before printing the message.

Notice the combination of loop continuation criterion and indexing rule for the second loop.

```
for (i=n-1; i>0; i--) { ... }
```

The (second) loop starts with the value of `i = 2` (because `n = 3` and stops when `i = 1`.

5 Application Examples

In this section `for` loops are used in more practical examples.

5.1 Average n Readings on an Analog Input Channel

In many practical situations, the voltage value on an analog input channel fluctuates. Sometimes the fluctuations are large enough to cause trouble when the voltage reading is used in a subsequent calculation or control decision.

For example, suppose a temperature sensor is influenced by air currents, and the reading of the sensor is used to determine whether or not to turn on a heater. The fluctuations in the apparent temperature reading are caused by the environment, and we want the decision to turn the heater on and off to be made on the average temperature and not the fluctuations in temperature. This is a situation where averaging the readings of the analog input channel would help[2].

The `averagen_demo` sketch shows how a user-defined function called `average_reading` make an user-specified number of readings on an user-specified analog input channel. The `for` loop in the body of the `average_reading` function has a flexible continuation criterion, `i<=nave`, where `nave` is the number of readings to be averaged.

[2]Of course, this also applies to a temperature sensor that has a digital interface such as I2C.

```
//   File:  averagen_demo.ino
//
//   Demonstrate a user-defined function to average N readings on an
//   analog input pin

// ------------------------------------------------
void setup() {
  Serial.begin(9600);
}

// ------------------------------------------------
void loop() {
  int n=15, photo_pin=1;
  float reading;

  reading = average_reading(photo_pin,n);  //  Average on analog input channel
  Serial.println(reading);                 //  Print the averaged reading
}

// ------------------------------------------------
//  Return the average of nave readings on a user-specficied analog input pin
float average_reading(int sensor_pin, int nave) {
  int   i;
  float ave,sum;

  sum = 0.0;    // initial value of sum
  for ( i=1; i<=nave; i++ ) {
    sum = sum + analogRead(sensor_pin);
  }
  ave = sum/float(nave);
  return(ave);
}
```

Listing 1: Arduino sketch to demonstrate how to average N readings.

```
//   File:  average_compare.ino
//
//   Compare average readings of two different sensors

// ----------------------------------------------------
void setup() {
  Serial.begin(9600);
}

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

  int n=15, photo_room_A=1, photo_room_B=1;
  float reading_A, reading_B;

  reading_A = average_reading(photo_room_A,n);  //  Average of photoresistor in room A
  reading_B = average_reading(photo_room_B,n);  //   and room B

  if ( reading_A > reading_B ) {
    Serial.print("Room A is brighter than Room B: ");
  }
  Serial.print(reading_A);
  Serial.print("\t");
  Serial.println(reading_B);
}

// ----------------------------------------------------
//  Return the average of nave readings on a user-specficied analog input pin
float average_reading(int sensor_pin, int nave) {

  int   i;
  float ave,sum;

  sum = 0.0;    // initial value of sum
  for ( i=1; i<=nave; i++ ) {
    sum = sum + analogRead(sensor_pin);
  }
  ave = sum/float(nave);
  return(ave);
}
```

Listing 2: Arduino sketch to compare two sensors from average readings on each.

5.2 Reuse the Code

When we use more than one sensor, the `average_reading` function can be used for both sensors without any change to the `average_reading` function. This example of code reuse shows the benefit of writing general purpose code.

Consider an experiment (or perhaps a building control system) where the brightness of lighting in two different rooms needs to be compared. Maybe the brightness is affected by the sun shining through south-facing windows and either the window-shades or the air-conditioning controls need to be adjusted. The code in the `average_compare` sketch shows how the `average_reading` function is reused. Note that the loop inside `average_reading` is flexible enough that the average value for each sensor could be made with a different number of readings, i.e., a different `n`.

```
//  File:  blinkn.ino
//
//  Demonstrate how to blink an LED a variable number of times

int LEDred = 5;
int LEDyellow = 6;

// ---------------------------------------------------------------------
void setup() {
  pinMode(LEDred, OUTPUT);
  pinMode(LEDyellow, OUTPUT);
}

// ---------------------------------------------------------------------
void loop() {
  int  nRed=3, nYellow=2;         // number of blinks for red and yellow LEDs
  int  dtRed=200, dtYellow=600;   // duration of blink cycles for red and yellow

  blinkLED(LEDred, dtRed, nRed);
  blinkLED(LEDyellow, dtYellow, nYellow);

}

// ------------------------------------------------------------
void blinkLED(int pin, int duration, int nrep) {

  for (int i=1; i<=nrep; i++ ) {
    digitalWrite(pin, HIGH);
    delay(duration);
    digitalWrite(pin, LOW);
    delay(duration);
  }
}
```

Listing 3: Demonstrate function that blinks an LED N times.

5.3 Variable Blinking

A blinking LED can be used to indicate the operating state of a sketch. The code to blink an LED is well known, it would not be hard to incorporate that code into a sketch. However, consider the case where either more than one LED or number of blinks are is used to indicate the state of the system. In this case, a general-purpose function to blink an LED would be useful.

The `blinkn` sketch in Listing 3 contains a `blinkLED` function that blinks an LED a user-specified number of times and at a user-specified rate.

5.4 A Countdown Timer with Blinking Warning Lights

Suppose you want to create a countdown timer for a game[3]. Each game (or maybe each round of the game) has a time limit. You also want to include a warning that the countdown timer is about to expire. The `gameTimer` sketch in Listing 4 shows one way to implement a 20 second countdown timer (`timeCounter`). A yellow LED is flashed when the timer is 5 seconds (`timeWarn`) from expiring. A red LED is flashed when the timer is 1 second (`timeFast`) from expiring. Both LEDs are turned on when the timer has expired completely.

[3] An obvious variation to the approach used here would be to show the time remaining on a clock display.

```
//  File:  gameTimer.ino
//
//  Simulate a countdown time for a game.  Push the reset button to
//  reset the timer.  Flash a yellow LED for the first warning.  Flash
//  a Red LED when the timer is very close to expiring.

int LEDred = 5;
int LEDyellow = 6;

unsigned long start_time;

// ---------------------------------------------------------------
void setup() {
  pinMode(LEDred, OUTPUT);
  pinMode(LEDyellow, OUTPUT);

  start_time = millis();      //  Clock reading at start
}

// ---------------------------------------------------------------
void loop() {
  int timeLeft, timeNow;  // Time remaining and current time
  int timeCounter=20000;  // The timer cycle
  int timeWarn=5000;      // Time left when yellow LED flashes
  int timeFast=1000;      // Time left when red LED flashes

  timeNow  = millis() - start_time;  //  Current time
  timeLeft = timeCounter - timeNow;  //  Time left on the counter

  if ( timeLeft <= 0 ) {             //  Both lights on when timer expires
    digitalWrite(LEDyellow, HIGH);
    digitalWrite(LEDred, HIGH);
  } else if ( timeLeft<timeFast ) {  //  Almost done, flash red
    blinkLED(LEDred, 100, 1);
  } else if ( timeLeft<timeWarn ) {  //  Warning, flash yellow
    blinkLED(LEDyellow, 100, 2);
  }
}

// ------------------------------------------------------------
void blinkLED(int pin, int duration, int nrep) {

  for (int i=1; i<=nrep; i++ ) {
    digitalWrite(pin, HIGH);
    delay(duration);
    digitalWrite(pin, LOW);
    delay(duration);
  }
}
```

Listing 4: Demonstrate a count-down timer that blinks yellow and then red LEDs as the timer is about to expire.