ME 120: What's this `void loop` thing?

Gerald Recktenwald
gerry@pdx.edu

October 20, 2019

1 Overview

This document aims to explain two kinds of loops: the `loop` function that is a required component of all Arduino sketches, and the standard `for` and `while` loops that are used whenever iteration within a function is needed.

Beginning Arduino programmers can be confused by the semantics and logic of the "`void loop`" function that is required by all Arduino sketches. The `loop` function is repeated continuously because it is called by a hidden loop that controls the execution of the Arduino program. In other words, the `loop` function is the *body* of a loop that is running in a master (or *main*) program. The master program is added automatically to the compiled version of the user sketch before the compiled code is downloaded to the microcontroller.

It's possible, and often necessary, to include `for` loops and `while` loops in the body of the `loop` function. The beginning programmer can get twisted around by the idea that there are loops running inside a function called *loop*. These are not hard concepts to grasp once you understand how each part of the program is being repeated, and how often each repetition is performed. The goal of this document is to expose and clarify those concepts.

The body of the paper covers three main concepts of Arduino programming:

- Basic structure of an Arduino sketch.
- Review of the syntax and use of Arduino functions, with special attention to the `setup` and `loop` functions.
- The relationship between the `loop` function and `for` loops and `while` loops that are contained inside the `loop` function.

We begin with a review of the basic program structure of a sketch. We show that the `loop` and `setup` functions are just ordinary Arduino functions that happen to have special names. Next we review the syntax of `for` and `while` loops. The paper ends with a series of example programs that demonstrate the interaction between the `loop` function and loops iterating inside the `loop` function.

2 Structure of an Arduino Sketch

All Arduino sketches *must have* two functions: `setup` and `loop`. The `setup` function is executed only once when the Arduino board is first turned on, or when the reset button is pressed. The `loop` function is repeated indefinitely after the `setup` function is finished.

2.1 `setup` and `loop` Are Required

Figure 1 is an annotated version of the basic `blink` sketch[1]. The sketch has the required `setup` and `loop` functions, along with an optional header at the top of the file. Most substantial sketches have a header that consists of comment statements and the definitions of global variables.

Global variables defined in the header are shared by all functions in the sketch. The `blink` sketch in Figure 1 has one global variable, `LED_pin`. The value of `LED_pin` is available to all functions in

[1] This is a slightly modified version of the `blink.ino` file that is distributed with the Arduino IDE.

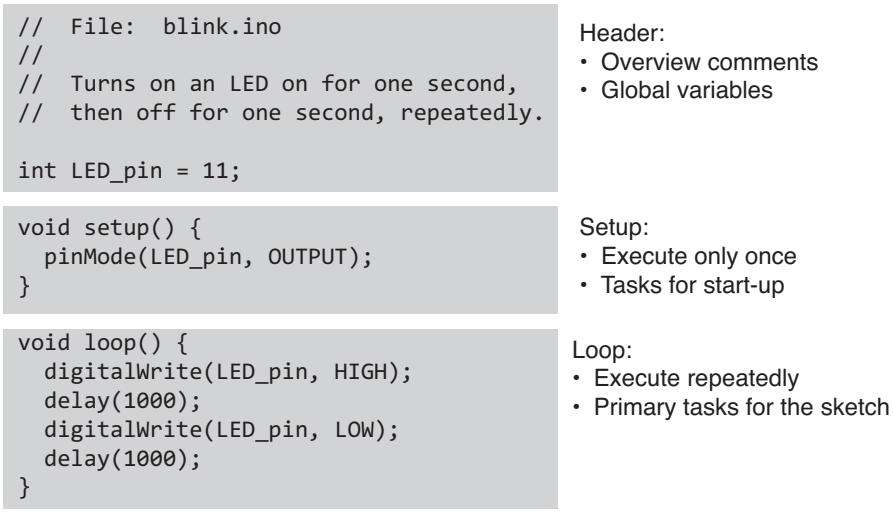

Figure 1: The header, `setup`, and `loop` components of a basic Arduino sketch.

the `blink.ino` file. The value 11 is assigned to `LED_pin` in the header. The `LED_pin` variable is used once in `setup` and twice in `loop`.

The `setup` function in the `blink` sketch performs only one task: configuring the digital pin for output. The `loop` function turns the LED on, waits one second, turns the LED off, and waits one more second. That pattern is repeated indefinitely, causing the LED to blink. Note that the blinking is achieved without the need to write an explicit loop inside the body of the `loop` function. That's because the `loop` function is called by the invisible loop in the invisible main program that is added after the sketch is compiled and before it is downloaded to the AVR microcontroller on the Arduino board.

2.2 `setup` and `loop` Have No Inputs and No Outputs

Figure 2 shows a skeletal `loop` function. The function definition statement begins with `void`, which declares that the `loop` function is *not* going to return a value. The `void` keyword is not optional: all functions must indicate the kind of value that they return to the calling program. Some other

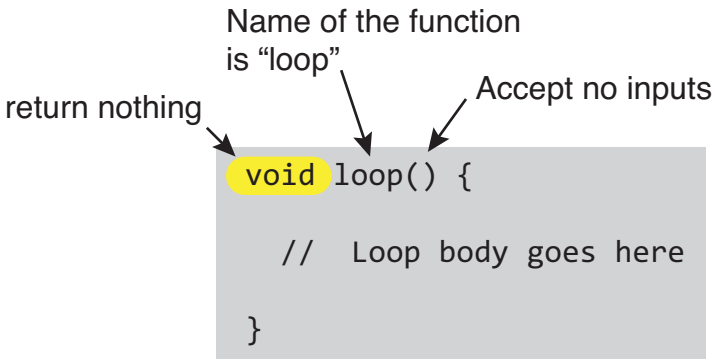

Figure 2: The `loop` function accepts no inputs and returns no values.

types of return values are `int`, `float`, `double` and `char`. More generally, return values can be any type of Arduino variable[2]. To indicate that the function returns no value, the function *must* be declared to have a `void` return value.

As shown in Figure 2, the `loop` function also has no input parameters. The empty parentheses are required: you cannot simply drop the parentheses for a function that has no input.

The `setup` function has a different purpose from the `loop` function. However, it is similar to `loop` in that it does not return a value, and it has no input parameters. In summary,

- `setup` and `loop` have no return values. Hence the function definitions begin with `void setup` and `void loop`.
- `setup` and `loop` have no input parameters. Hence the function definitions end with empty input parameter lists, i.e., `()`.

3 User-defined Functions in a Sketch

An Arduino sketch can contain user-defined functions designed to perform sub-tasks. Programming statements are *encapsulated* in the code block that forms the body of the function. This code block is isolated from other code blocks (i.e., from the code in other functions), and this isolation is an important mechanism for keeping code blocks from adversely interacting with each other.

Of course, programs also needs a mechanism to get data into a function and to obtain a result from a function. Input and output from a function is implemented with *input parameters* and *return values*. The programmer decides what values are needed (what inputs) to perform a computation. The programmer also decides what results (what outputs) need to be returned to the part of the program that invoked or *called* the function.

Functions allow programs to build reusable chunks of code for tasks that appear more than once in a sketch. Functions also provide a way to use the same code in other sketches.

Consider the function defined below that accepts any value, `x`, and returns the tangent of `x`.

```
double tangent( double x ) {
  double t = sin(x)/cos(x);
  return(t);
}
```

The function definition for `tangent` indicates that a value of type `double` is returned to the calling function when `tangent` terminates. The function definition also specifies that an input value of type `double` must be supplied by the calling function. Inside the body of the `tangent` function, the input value is stored in the variable `x`.

When `tangent` is executed, a temporary `double` variable `t` is created to store the result of the computation `sin(x)/cos(x)`. The value stored in `t` is returned to the calling function as a `double`, as specified by the function definition of `tangent`. The value of `t` is hidden from other functions (including `setup` and `loop`) in the sketch.

The `tangent` function is is a contrived example because the Arduino IDE provides a `tan` function for computing the tangent. One could imagine other computations that are more practical, for example to read 20 values from a photoresistor and return the average reading. We'll show how to do that after we describe `for` and `while` loops.

All Arduino functions have the potential for multiple input parameters and a single return value. The inputs and return values are specified in the one-line function definition statement that is required at the start of any function. A complete exposition on the design and use of functions is beyond the scope of this document[3]. For our immediate purposes, the important points are

[2] See, e.g., `http://www.arduino.cc/en/Reference/HomePage`

[3] For more information, see `http://www.arduino.cc/en/Reference/FunctionDeclaration`

- The `setup` and `loop` are ordinary Arduino functions. The names `setup` and `loop` are special because they are required by other parts of the Arduino software architecture.
- A sketch can contain any number (including zero) of user-defined functions.
- User-defined functions can have multiple inputs. Each input must be given a type, for example specifying `double` in the input argument `( double x )` for the `tangent` function defined above. Multiple inputs can have different types. When multiple inputs are provided, they must be separated by a comma. An example of multiple inputs is provided in a later section of this paper.

4 Who Calls `loop`?

The `setup` and `loop` functions are ordinary Arduino functions that happen to accept no input parameters and return no values. But that begs two questions. Where would the input parameters come from if there were any? And where would the return value go if the was one? In other words, what (or who) calls `setup` and `loop`?

The Arduino Integrated Development Environment (IDE) performs some hidden (and routine) work that simplifies the writing of code[4]. One of those simplifications is the way that the code written in a sketch is *compiled* and then converted to a program that the AVR microcontroller on the Arduino board can run. When you click the compile button, the IDE first examines the syntax of your sketch to make sure you haven't made any obvious, language-violating errors. It then converts the C code that *you* can read into binary machine code that the *microcontroller* can read and execute. Next, the IDE combines the binary code generated from your sketch with additional binary code to form a single, executable program that is downloaded to the Arduino board. The net effect of this *build process* is that the IDE adds a hidden *master* or *main* program that calls your `setup` and `loop` functions.

[4]See `http://arduino.cc/en/Hacking/BuildProcess`

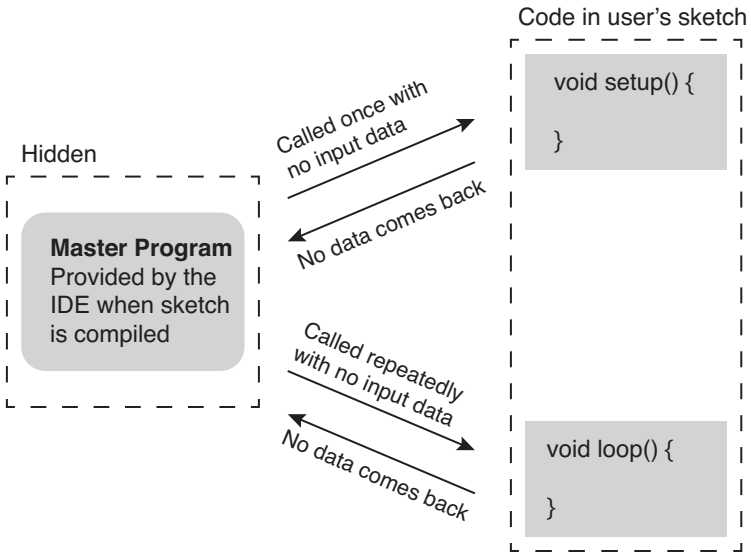

Figure 3: The hidden *master* function calls `setup` once and repeatedly calls `loop`.

The master program is hidden in the sense that you do not have to explicitly add it to your sketch or make any adjustments to it based on the features in your sketch. The hidden program expects to find the `setup` and `loop` functions defined in the appropriate way, i.e., as returning `void` and expecting no inputs.

The interaction of the master program with the `setup` and `loop` functions is depicted in Figure 3. The existence of the hidden master program helps to explain why the `loop` function is called "`loop`". The designers of the Arduino IDE could have called it `main_event_loop` or `user_function_that_repeats`. Instead they simply called it `loop`, a name that is descriptive of its operation, and that also reflects the truth that the `loop` function is called by a loop in the hidden master program.

The `loop` function is like the heartbeat of your program: it repeats indefinitely. The interesting work of your program occurs during each heartbeat, i.e., during each execution of the `loop` function.

5 Using `for` and `while` Loops inside `loop`

Since the `loop` function is just an ordinary Arduino function, it can contain all the normal parts of the Arduino programming language, including other loops! In this section we present some simple, though not very useful, codes that demonstrate the interaction between the `loop` function and `for` and `while` loops that are contained within it. Your Arduino code can have as many (or as few) loops as you need. Just remember that the `loop` function is itself inside an invisible loop that is executing inside the invisible master program.

Read the sketches and predict their behavior *before* you read the explanation provided here.

Note to instructors and to students using these notes for self-study:
It would be good for students to figure this code out for themselves without the explanations given below. Each program is short enough that students could enter it manually.

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 the Arduino to show how the code actually works.

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 3

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

void loop() {
  int i=0;
  while ( i<5 ) {
    Serial.println(i);
    i = i + 2;
  }
  Serial.println("... end while\n");
}
```

What is the output of the code to the left?

Answer: This code compiles and runs. Every time the `loop` function is executed, the values 0, 2, 4 are printed on separate lines, followed by the `... end while` message.
Extra question: what happens if the test for continuing is changed to `while ( i<=5 )`? In other words, what happens when `i<5` is changed to `i<=5` in the conditional test of the loop?

Loop 4

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

void loop() {
  int i=0, n=7;
  while ( i<n ) {
    Serial.print(i);  Serial.print("  ");
    i = i + 1;
  }
  while ( i>=0 ) {
    Serial.print(i);  Serial.print("  ");
    i = i - 1;
  }

  Serial.println("");
}
```

What is the output of the code to the left?

Answer: This code compiles and runs. Every time the `loop` function is executed, the values 0, 1, 2, ..., 8, 7, ..., 2, 1, 0 are printed on separate lines.