Programming Assignment

Lesson 6

CISC 071

By

<Your Name>

Date: <mm/dd/yyyy>

Purpose

* To learn about loops

Rubric

* Correctness: 2 Points. Program should work as specified
* Input/Output: 2 Points. Show the inputs and outputs to the program. If Applicable multiple examples needed
* Coding style/Comments: 1 Points.

Project Assignment

* Use the setup function to set the serial monitor and print “Serial monitor setup done”
* Write code in the **setup** function
* Use two nested *for* loops (i.e. a *for* loop inside a *for* loop) to achieve the following to be printed out to the serial monitor.

0 1 2 3 4 5 6 7

0 1 2 3 4 5 6 7

0 1 2 3 4 5 6 7

0 1 2 3 4 5 6 7

* Leave the loop() function empty.

Learning Notes:

**The "for" loop.**

Whereas statements or code in the Arduino main loop will run continually and never exit the loop, the for loop allows us to loop through code a certain number of times before exiting the loop.

A common way to use the for loop is with the increment operator

The following sketch demonstrates the use of the for loop.

*void setup() {*

*int i;*

*Serial.begin(9600);*

*for (i = 0; i < 10; i++) {*

*Serial.print("i = ");*

*Serial.println(i);*

*}*

*}*

*void loop() {*

*}*

Can't see the video? [View on YouTube →](https://youtu.be/5a2im1PcVjc)

### How the *for* Loop Works

The image below shows the parts of the for loop.

[](https://startingelectronics.org/software/arduino/learn-to-program-course/07-for-loop/for-loop-parts.png)

The expressions between the parentheses are called the initialize expression, test expression and increment expression.

The body of the loop between the opening and closing braces **{}** contains statements that will run in the loop.

A variable must be defined to use in the three loop expressions. In the example sketch an integer variable called **i** is used.

The three loop expressions must appear in the order: initialize, test and increment. They are separated by semicolons (**;**). The increment expression does not end with a semicolon.

**Initialize Expression**

The initialize expression is only run once at the time that the loop starts. It initializes our **i** variable to zero (0) in the example sketch.

**Test Expression**

The test expression determines when we break out of the loop. It is used to set the number of times that the statements in the body of the loop are run.

When the test expression evaluates to **true**, the statements in the loop body will be run. When the test expression evaluates to **false** the loop will not be run again, but will be exited.

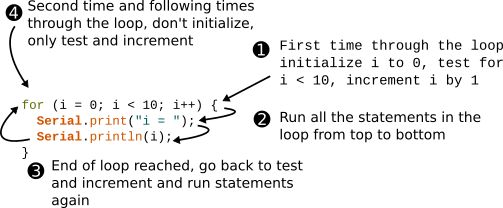
The test expression is evaluated every time that execution starts at the top of the loop.

**Increment Expression**

The increment expression is used to change the value that the **i** variable holds. It is run every time that execution starts at the top of the loop.

**Program Flow in the Loop**

The image below shows how program flow works in the for loop.

[](https://startingelectronics.org/software/arduino/learn-to-program-course/07-for-loop/for-loop-flow.png)

*for***Loop Program Flow**

**First Time Through the Loop**

The first time through the loop, **i** is initialized to 0, the test expression tests whether **i < 10 (0 < 10)** which is true, so the statements in the loop will run.

Because the post increment operator is used with the variable, **i** will only be incremented at the end of the loop. The statements in the loop run and print the value of **i** as 0 because it has not yet been incremented.

We therefore have this:

*i is initialized to 0*

*i contains 0*

*i < 10 evaluates to true or 1 because i is less than 10*

*The two statements in the loop run, print i as 0*

*At the end of the loop i is incremented so i == 1*

**Second Time Through the Loop**

The second time through the loop, **i** now contains 1 as it was incremented at the bottom of the loop. The test expression now tests whether **i < 10 (1 < 10)** which is true, so the statements in the loop will run again. The **i** variable will only be incremented to 2 at the bottom of the loop, so 1 is printed to the serial monitor window.

We now have this:

*i is not initialized again*

*i contains 1*

*i < 10 evaluates to true or 1 because i is less than 10*

*The two statements in the loop run, print i as 1*

*At the end of the loop i is incremented so i == 2*

**Last Time Through the Loop**

Execution of the loop will continue and **i** will be incremented every time.

The last time through the loop, we have this:

*i is not initialized again*

*i contains 9*

*i < 10 evaluates to true or 1 because i is less than 10*

*The two statements in the loop run, print i as 9*

*At the end of the loop i is incremented so i == 10*

Execution starts at the top of the loop again, the evaluation expression is tested.

We now have this:

*i is not initialized again*

*i contains 10*

*i < 10 evaluates to false or 0 because i is not less than 10 (it is equal to 10)*

*The statements in the loop are not run again*

*The loop is exited*

*The statement below the closing bracket of the loop will be run*

**A Loop Within a Loop**

The next sketch uses a **for** loop within the Arduino main loop.

*void setup() {*

*Serial.begin(9600);*

*}*

*void loop() {*

*for (int i = 0; i < 10; i++) {*

*Serial.print("i = ");*

*Serial.println(i);*

*}*

*delay(1000);*

*}*

The *for* loop works exactly the same as it did before, but now after it has been exited, the delay() function is run to give a 1 second delay. The end of the Arduino main loop **loop()** is reached, so the *for* loop is run again.

When the *for* loop is run again, **i** is initialized to 0 because the *for* loop is being started from the top again. It then runs again as previously described.

The *for* loop and delay() function will be run continually because the main Arduino loop never exits.

The next sketch uses a nested ***for*** loop inside a ***for*** loop.

*void setup() {*

*Serial.begin(9600);*

*for (int i = 0; i < 8; i++) {*

*for (int j = 0; j < 8; j++) {*

*Serial.print(j);*

*Serial.print(“ “);*

*}*

*Serial.println();*

*}*

*void loop() {*

*}*

**Notes on the *for* Loop Sketch Examples**

Note the following about the sketch examples.

**Initialize Expression**

The initialize expression in the *for* loop does not have to be initialized to zero (0), but can be initialized to any integer value, even a negative value.

**Increment Expression**

The increment expression is used to change the value of the variable that the test expression tests. This does not have to be an increment operator, but can be the decrement operator (subtracts 1 from the variable) or any other arithmetic expression.

The increment operator has been used in the example sketches to keep things simple at the beginning of the course, and because it is a common way of using the *for* loop. We will look at other ways to use the *for* loop later in the course.

**Counting from Zero (0)**

Note that in the example sketches, that the value that the **i** variable contains is initialized to 0 and not 1. We therefore print out a count value that starts at 0 and ends at 9.

The loop is actually run 10 times and not 9 times. This is because we are starting at 0 – 0 to 9 are 10 numbers, 1 to 9 are 9 numbers, 1 to 10 are 10 numbers.

This list shows the number of times through the loop on the left and the value of **i** on the right.

*i = 0*

*i = 1*

*i = 2*

*i = 3*

*i = 4*

*i = 5*

*i = 6*

*i = 7*

*i = 8*

*i = 9*

**The Arduino *while* and *do while* Loops**

The *while* loop is similar to the *for* loop. The main difference is that the *while* loop separates the elements of the *for* loop as will be shown.

Another loop called the *do while* loop is also covered. The *do while* loop is always run at least once before any tests are done that could break program execution out of the loop.

**The *while* Loop**

The sketch that follows does exactly the same as the *for* loop sketch, except that it uses the *while* loop so that we can see the similarities between the two loops.

*void setup() {*

*int i = 0;*

*Serial.begin(9600);*

*while (i < 10) {*

*Serial.print("i = ");*

*Serial.println(i);*

*i++;*

*}*

*}*

*void loop() {*

*}*

***while* Loop Structure**

The *while* loop has a structure as follows:

*while (loop test expression goes here) {*

*Statements that run in the loop go here*

*Statement 1*

*Statement 2*

*...*

*}*

The *while* loop starts with the **while** keyword followed by a test expression between opening and closing parentheses. Opening and closing braces denote the body of the loop.

***Test Expression***

As with the *for* loop, the *while* loop has a test expression that will determine whether the statements in the loop will run or not. If the test expression evaluates to **true**, the loop statements are run. If the test expression evaluates to **false**, the loop statements will not be run, but the statements that follow the closing brace of the loop will be run – i.e. execution continues outside and below the loop.

***Initialize Expression***

The *for* loop had an initialize expression as part of the loop. The *while* loop can use any variable from the sketch that contains a valid value. In the example sketch, the variable used in the loop (**i**) must be initialized when it is defined, otherwise it will contain any random value.

***Increment Expression***

An increment expression was used in the *for* loop examples in the previous part of this course. In the *while* loop example, the increment expression is placed inside the loop body.

**How the *while* Loop Example Works**

In the example sketch, the following happens:

1. *The variable****i****is initialized to 0 when the sketch starts running.*
2. *The*while*loop evaluates the test expression****(i < 10)****.*
3. *The test expression evaluates to****true****because****i****is less than 10.*
4. *Because the test expression is true, the statements in the loop run.*
5. *The current value of****i****is printed and then incremented.*
6. *When the bottom of the loop is reached, execution is started at the top of the loop again.*
7. *The test expression is evaluated again, it is true again, so the loop runs again.*

Only when the variable **i** has been incremented to 10, will the loop expression evaluate to **false** and the loop will be exited.

***while* Loop Example 2**

In the example sketch below, the *while* loop is used to count up to twenty-five in fives by adding five to a variable each time through the loop.

*void setup() {*

*int sum = 0;*

*Serial.begin(9600);*

*// count up to 25 in 5s*

*while (sum < 25) {*

*sum = sum + 5;*

*Serial.print("sum = ");*

*Serial.println(sum);*

*delay(500); // 500ms delay*

*}*

*}*

*void loop() {*

*}*

Can't see the video? [View on YouTube →](https://youtu.be/rx8DRG24Nvs)

In this sketch, a variable called **sum** is defined and initialized to 0. The test expression in the *while* loop tests if **sum** contains a value less than 25.

Inside the loop, the **sum** variable is incremented by 5 each time through the loop by the arithmetic expression:

*sum = sum + 5;*

This expression means "add 5 to the **sum** variable".

The value that the **sum** variable holds is then printed out, followed by a half-second delay.

Because the value of the variable is first incremented and then printed out, we see the value 5 printed first and not the value of 0 that it was initialized to.

Although the test expression will evaluate to false when **sum == 25**, 25 is still the last number that is printed. This is because the last time that the test expression evaluates to true is when sum == 20, but sum is then incremented to 25 and printed before the test expression is evaluated again.

***The do while Loop***

The *do while* loop works in the same way as the *while* loop, except that it always runs once even if the test expression evaluates to false.

***do while* Loop Structure**

The do while loop consists of two keywords **do** and **while**, as shown below.

*do {*

*Statements that run in the loop go here*

*Statement 1*

*Statement 2*

*...*

*} while (test expression goes here);*

The body of the *do while* loop falls between opening and closing braces and contains statements that are to be run in the loop.

The **while** keyword and test expression come after the body of the loop and are terminated by a semicolon (**;**).

### *do while* Loop Example

This example demonstrates the do while loop.

*void setup() {*

*int sum = 0;*

*Serial.begin(9600);*

*// count up to 25 in 5s*

*do {*

*sum = sum + 5;*

*Serial.print("sum = ");*

*Serial.println(sum);*

*delay(500); // 500ms delay*

*} while (sum < 25);*

*}*

*void loop() {*

*}*

All the statements are run in the loop body before the test expression is evaluated.

If **sum** is initialized to a value of 25 when it is defined, as shown in the sketch below, the loop will run once and 30 will be printed. The loop will then not run again because the test expression evaluates to false.

*void setup() {*

*int sum = 25;*

*Serial.begin(9600);*

*// count up to 25 in 5s*

*do {*

*sum = sum + 5;*

*Serial.print("sum = ");*

*Serial.println(sum);*

*delay(500); // 500ms delay*

*} while (sum < 25);*

*}*

*void loop() {*

*}*

Using the same sketch, but changing the *do while* loop to a *while* loop, as shown below, the statements in the loop body will never run. This is because the test expression is evaluated before executing statements in the loop body. The test expression immediately evaluates to false, so the loop statements will never run.

*void setup() {*

*int sum = 25;*

*Serial.begin(9600);*

*// count up to 25 in 5s*

*while (sum < 25) {*

*sum = sum + 5;*

*Serial.print("sum = ");*

*Serial.println(sum);*

*delay(500); // 500ms delay*

*}*

*}*

*void loop() {*

*}*

In the above example, no output will be seen in the serial monitor window when the sketch is run. The *while* loop evaluates to false and then execution drops straight into the empty main Arduino loop.

**For further details refer to the Arduino programming reference guide**

<https://playground.arduino.cc/uploads/Main/arduino_notebook_v1-1.pdf>

Program

void setup() {

Serial.begin(9600); //initalize

for(int col = 0; col < 3; col++) //outer loop that will establish columns

{

for(int row = 0; row < 8; row++) //inner loop that will create the numbered rows

{

Serial.print(row); //print current iteration of row

Serial.print(" "); //print space string

}

Serial.println();//end line

}

}

void loop() {

// put your main code here, to run repeatedly:

}

Inputs/Outputs

0 1 2 3 4 5 6 7

0 1 2 3 4 5 6 7

0 1 2 3 4 5 6 7