Programming Assignment

Lesson 5

CISC 071

By

Jeremy Reuwer

Date: 03/06/2019

Purpose

* To learn about Operators

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

* Define a variable called number\_counter. Initialize it to 0
* Use the setup function to set the serial monitor and print “Serial monitor setup done”
* Write code in the **loop** function
* Print the number\_counter
* Increment the variable number\_counter
* Add a 2500 ms (milli seconds) delay after every increment

Learning Notes:

**Relational operators**

Relational operators test the relationship between values, for example is the number 7 greater than the number 5; or is the value that a variable holds less than 10. The result of a comparison of two values by a relational operator will either be true or false, for example is the value contained in a variable equal to 8? The answer will either be true (it does hold the value 8) or false (it does not hold the value 8). In programming, true is represented by the value 1 and false is represented by the value 0. The example that follows will help to clarify this explanation of relational operators.

**Relational Operator Example**

The following sketch uses the greater than (>) relational operator and the less than (<) relational operator to compare two numbers.

*void setup() {*

*Serial.begin(9600);*

*Serial.print("Is 7 greater than 5? ");*

*Serial.println(7 > 5);*

*Serial.print("Is 7 less than 5? ");*

*Serial.println(7 < 5);*

*}*

*void loop() {*

*}*

The output of the above sketch is shown here.

*Is 7 greater than 5? 1*

*Is 7 less than 5? 0*

**Output Explanation**

The first relational operator tests whether 7 is greater than 5: 7 > 5

7 is greater than 5, so the output from this comparison is true or 1.

The second relational operator tests whether 7 is less than 5: 7 < 5

7 is not less than 5, so the output from the comparison is false or 0.

**Types of Relational Operators**

The Arduino programming language has 6 relational operators listed below.

**Greater Than >**

We have already seen the greater than relational operator working in the above example. Here is another example using the same operator to compare the contents of two variables.

*int a = 2;*

*int b = 3;*

*Serial.println(a > b);*

The relational operator asks the question "Is a greater than b?".

The result of the comparison will be 0 (false) in this example because the value that variable a contains is not greater than the value that variable b contains.

**Less Than <**

Here we use the less than relational operator to compare the contents of the same two variables.

*int a = 2;*

*int b = 3;*

*Serial.println(a < b);*

The relational operator asks the question "Is a less than b?".

The output of this sketch will be 1 (true) because the value that variable a contains is less than the value that variable b contains.

**Greater Than or Equal To >=**

This operator acts in the same way as the greater than operator, but will also evaluate to true (1) if the two values being compared are equal to each other.

**Less Than or Equal To <=**

This operator will evaluate to true (1) if the value on the left of the operator is less than the value on the right of the operator; or if the two values are equal.

**Equal To == (Note there are two equal signs)**

The equal to relational operator will only evaluate to true (1) if both values being compared are equal, otherwise it will evaluate to false (0).

It is easy to forget to use two equals signs (==) and use only one equals sign for the equal to relational operator. Using only one equals sign will cause the variable on the left of the operator to be assigned the value of the variable or constant on the right. If the value on the left is a constant, a compile error will occur if a single equals (=) is used instead of a double equals (==).

**Not Equal To !=**

The not equal to relational operator will only evaluate to true if the two values being compared to each other are not equal to each other, otherwise it will evaluate to false.

***Confusing = and ==***

As already noted, it is easy to forget the second equals sign when using the equal to operator. The sketch below shows what can happen with this mistake.

*void setup() {*

*int a = 2;*

*int b = 3;*

*Serial.begin(9600);*

*Serial.print("a == b: ");*

*Serial.println(a == b);*

*Serial.print("a = b: ");*

*Serial.println(a = b);*

*}*

*void loop() {*

*}*

The comparison of a and b in the expression a == b is correct and evaluates to 0 as expected.

If the second equals sign is left off as shown in the second expression a = b, the assignment operator (=) is accidentally used instead of the equal to operator (==).

This causes a to be assigned the value of b which is 3 and then 3 is printed out. A non-zero number will actually evaluate to true as we will see later in this course.

**Arduino Increment Operator**

The increment operator is an Arduino arithmetic operator that is used to increment an integer variable by a value of one.

The Arduino increment operator is used to increase the value that a variable holds by one. This is useful in certain types of loops as will be shown later in this course.

**Increment Operator Example 1**

The example below shows the increment operator being used to increment a value several times.

*void setup() {*

*int count = 0;*

*Serial.begin(9600);*

*Serial.println(count++);*

*Serial.println(count++);*

*Serial.println(count++);*

*Serial.println(count);*

*}*

*void loop() {*

*}*

Sketch Output

The output of the sketch in the Arduino serial monitor window is shown here.

*0*

*1*

*2*

*3*

In the above example, the variable count is defined and initialized to a value of 0 (zero).

It is important to initialize this variable before using the increment operator because if it is not initialized, it can contain any random value. Here we initialize it to 0, but could have used any other integer value.

**Printing and Incrementing**

Serial.println() is used to print the value of count, but in the same statement, the increment operator (++) is used to increase the value that the count variable holds.

The increment operation causes the value in count to increase from 0 to 1.

**Post Incrementing**

Placing the ++ after the variable name (count++) is a post increment operation. This means that the variable is used in the statement and only incremented after this.

Because of the post increment operation, the initial value of count (which is 0) is printed by the first println() function, and only incremented after this.

The next println() function prints the new incremented value (now 1) and then the increment operator in the statement increments the value of count after it has been printed – it is now incremented to 2.

The third println() statement prints out 2 and increments the value in count to 3.

The final println() statement prints out the value of count but does not use the increment operator. It prints out the value of 3 which the variable was previously incremented to.

**Increment Operator Example 2**

The following sketch does exactly the same as the previous sketch, but the increment operations and print operations have been separated into their own statements.

This example may help to clarify what is happening in the previous example. We can see that the first println() statements prints the value of count and only after this is the value in count incremented.

*void setup() {*

*int count = 0;*

*Serial.begin(9600);*

*Serial.println(count);*

*count++;*

*Serial.println(count);*

*count++;*

*Serial.println(count);*

*count++;*

*Serial.println(count);*

*}*

*void loop() {*

*}*

From these two examples, we can see that this single line of code:

*Serial.println(count++);*

Is the same as these two lines of code:

*Serial.println(count);*

*count++;*

The important thing to note is that the incrementing of the variable takes place after printing it, so the value that the variable holds is printed before it is updated to the new incremented value.

**Commenting Sketches**

Comments in programming are notes that are written in a program (or sketch) by the programmer. These notes are used to explain what the code is doing, or to add other useful human readable information to the code.

Commenting sketches allows you to write your own text notes along with the code of the sketch. The next sketch shows two ways of writing comments.

*/\**

*Sketch Name: comments*

*Purpose: Demonstrates how to use comments.*

*Increments and displays a number in the main loop.*

*Date: xx/yy/zz*

*Author: ABC*

*\*/*

*void setup() {*

*Serial.begin(9600); // use the serial port for printing the number*

*}*

*int count = 0; // the number to print*

*void loop() {*

*Serial.println(count++); // print and increment the number*

*delay(1000); // 1 second delay between printing*

*}*

This sketch uses the increment operator and can be seen running in the video below.

**Multi-line Comments**

Comments can be written on multiple lines when using the opening forward slash asterisk (/\*) and closing asterisk forward slash (\*/) as shown at the top of the sketch.

Any text comments can be written between the opening and closing of the comment and can span multiple lines. All text in the comment will be ignored by the compiler.

This style of commenting can also be used on a single line, but must always be closed with the asterisk and forward slash as shown below.

*delay(1000); /\* 1 second delay between printing \*/*

There is an easier way to comment a single line of code, by using the single line comment, explained next.

**Single Line Comments**

The double forward slash (//) is used to create a single line comment. All text to the right of the double forward slash on the same line will be a comment and will be ignored by the compiler.

The single line comment can be used on its own line or to the right of a program statement.

*// the delay slows down the printing of the numbers*

*delay(1000); // 1 second delay between printing*

**What Comments are Used For**

Comments are used to explain how parts of a sketch work, rather than explain the actual programming language that is being written.

When you come back to a program or sketch that you wrote a month or a year ago, you may not remember why you wrote some part of the sketch a particular way. By writing comments in the sketch, you are making notes for yourself to help explain what you were doing when you first wrote the sketch.

Comments can also be arranged as a header section or comment block, like the multi-line comment block at the top of the example sketch. The block contains the name of the sketch, description or purpose of the sketch, date that it was written and the authors name. Comments can also contain references to any books, websites or other resources that were used when writing the sketch.

In short, comments and comment blocks communicate helpful information about the sketch for yourself and for other programmers who may also use the sketch.

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

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

Program

float numberCounter = 0; //initialize number counter holder

void setup() {

Serial.begin(9600); //begin

Serial.println("Serial monitor setup done \n"); //setup complete

}

void loop() {

Serial.println(numberCounter); //print current iteration of numberCounter

numberCounter++; //increment counter

delay(2500); //delay 2500 milli seconds

}

Inputs/Outputs

Serial monitor setup done

“

0.00

1.00

2.00

3.00

4.00

5.00

6.00

7.00

8.00

9.00

10.00

11.00 ”

… etc.