Programming Assignment

Lesson 8

CISC 071

By

<Your Name>

Date: <mm/dd/yyyy>

Purpose

* To learn about ***arrays***

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
* Define an array of 10 integers. The array holds the following numbers in this sequence 8, 99, 87, 2, 5, 34, 23, 68, 90, 1.
* Use a *for* loop to go through all the elements of the array and print the number only if it is less than 50.
* Cut and paste your program and serial monitor in this document.
* Leave the **loop** function empty.

Learning Notes:

**Arrays**

Arrays are groups of the same kind of data that are placed consecutively in memory. For example, we can have an array of integers (type *int*) which is two or more integer numbers occurring one after the other.

The key here is that each element in an array is placed directly after the previous element which allows us to access each element in turn using a loop. Elements in an array are placed sequentially in the memory of the computer.

An element in an array refers to each value in the array. If we have an array of integers, then each individual integer is referred to as an element of the array. In an array of bytes, each element is a byte (of the Arduino *byte* type).

**Using Arrays**

The sketch below shows the basic use of an array.

*void setup() {*

*int my\_array[5]; // an array with 5 integer elements*

*int i; // used as an index into the array*

*Serial.begin(9600);*

*my\_array[0] = 23;     // assign a value of 23 to the 1st element*

*my\_array[1] = 1001;   // assign a value of 1001 to the 2nd element, etc.*

*my\_array[2] = 9;*

*my\_array[3] = 1234;*

*my\_array[4] = 987;*

*// display each number from the array in the serial monitor window*

*for (i = 0; i < 5; i++) {*

*Serial.println(my\_array[i]);*

*}*

*}*

*void loop() {*

*}*

Can't see the video? [View on YouTube →](https://youtu.be/8UeyqypsaG0)

**Defining the Array**

In this sketch, an array of 5 elements is defined. This makes space in memory for 5 integers that are put in the memory one after the other. The values that each element contains after the array is defined can contain any random data – whatever happens to be in the memory at the time. It is also possible that the compiler is set to make the values zero, but we can not rely on this.

Defining an array of 5 integers:

*int my\_array[5]; // an array with 5 integer elements*

In this example the array is of type int, but could be a float, byte, etc.

The array has a name which is my\_array in the example.

The array has a length [5] which means that space for 5 consecutive integers is made in memory.

**Assigning Values to Elements in the Array**

Each element is assigned an integer value by referencing it using square brackets [] with the number of the element to access in the brackets.

*my\_array[0] = 23;*

*my\_array[1] = 1001;*

*my\_array[2] = 9;*

*my\_array[3] = 1234;*

*my\_array[4] = 987;*

Note that the element numbering starts from zero [0] and not one [1], so the first element in the array is element 0.

In the same way, the last element in the array is numbered one less than the size of the array. In the example, the size of the array is 5, so the number of the last element is 4 – again this is because we are numbering the elements starting with 0.

**Accessing an Array in a Loop**

A *for* loop is used to get the contents of each element in the array in turn and print the values to the Serial Monitor window.

*for (i = 0; i < 5; i++) {*

*Serial.println(my\_array[i]);*

*}*

The variable i is used in the for loop as an index into the array to access each element of the array.

In the loop, *i* is initialized to 0 and then incremented by one each time through the loop so that it counts from 0 to 4. The loop is exite d when *i* becomes 5.

The *i* variable is used in the array to get the value that the array element is holding starting with element 0 and ending with 4.

*(my\_array[i]);*

In the above code snippet, when *i*is 0, the first element of the array is accessed and we can then get the value that it contains which is 23 in the example sketch.

**More on Initializing Arrays**

Instead of initializing each element in the array individually, the array can be defined and initialized in one line as shown in this code.

This sketch does exactly the same as the previous sketch.

*void setup() {*

*int my\_array[5] = {23, 1001, 9, 1234, 987};*

*int i;*

*Serial.begin(9600);*

*// display each number from the array in the serial monitor window*

*for (i = 0; i < 5; i++) {*

*Serial.println(my\_array[i]);*

*}*

*}*

*void loop() {*

*}*

The values to initialize each element with are placed between braces {} after the assignment operator (the equals sign =). The first value between the braces will be assigned to the first element in the array (element number 0), the second number between braces will be assigned to the second element in the array (element number 1), etc.

The code that does the defining and initializing can also be written without the number of elements in the array between the square brackets:

*int my\_array[] = {23, 1001, 9, 1234, 987};*

In this case, the compiler will work out how many elements the array must have based on the number of values that are used to initialize it.

**Uses of Arrays**

This part of the course shows that arrays can store data variables of the same type consecutively in memory which allows easy access using a loop.

There are many uses for arrays in programming, for example, arrays can store data that is being logged, such as temperatures.

**Array Bounds**

When working with strings and arrays, it is very important to work within the bounds of the string or array. In the example sketch an array was created that was 40 characters long in order to allocate memory that could be used to manipulate strings.

If the array was made too small and we tried to copy a string that is bigger than the array to it, the string would be copied over the end of the array. The memory beyond the end of the array could contain other important data used in the sketch which would then be overwritten by our string. If the memory beyond the end of the string is overrun, it could crash the sketch or cause unexpected behavior.

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

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

Program

void setup() {

int lotsOfNumbers[10] = {8, 99, 87, 2, 5, 34, 23, 68, 90, 1};

Serial.begin(9600);

for(int l = 0; l < 9; l++)

{

if(lotsOfNumbers[l] < 50)

Serial.println(lotsOfNumbers[l]);

}

}

void loop() {

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

}

Inputs/Outputs

8

2

5

34

23