

Objectives: This laboratory aims to practice using array pointers in C (with the Arduino platform).

Learning outcomes:

- Declare and use array pointers
- Use array pointers and arrays with subscripting interchangeably

Lab instructions

1. The lab is for individuals working alone and must be completed during the lab session
2. Create a new sketch for each major section in the lab.
3. Before you leave the lab, call the lab demonstrator to check what you have done for all the sections (this is why separate sketches are important). *Anything not checked by the demonstrator during lab will have to be assigned zero marks.*
4. Create a single plain text submission file (.txt) for the lab using a plain text editor (e.g. NotePad++) and submit only this text file. Copy all the sketches you write and any other answers required for the lab into the submission text file. *Name the file "108_Lab7_firstname_surname.txt", using your actual name. Include your name and lab number at the top of the submission file also and clearly label everything in the file. Unclear submissions will have to be marked down or (in the worst case) not marked at all.*

Marking for lab/assignment

Most of the lab and assignment will be marked during lab sessions. It is essential that you get the demonstrator to confirm your progress during the lab (for the lab portion) and at the start of the next lab (for the assignment portion).

For all code sections, marks will be deducted for bad communication and style (e.g. missing or mismatching comments, poor variable names, bad indentation, inappropriate use of global variables, unnecessary code repetition, etc.) and incorrect behaviour, or failure to follow the requirements of the question.

General marks will also be lost if the submission document instructions are not followed.

1 Basic array pointers – save sketch as “Lab7_BasicArrayPointers”

Background: First review notes 2.21 Array Pointers. In particular slide 15 shows 4 functions which have exactly the same effect but are implemented differently. There are 2 differences to look for:

- The first parameter to the function is an output parameter which identifies the array to be modified. In 2 of the functions this parameter is an array, in the others it is an array pointer.
- The other difference is internal to the functions. Two of the functions use an index variable and array subscripting to iterate over the array. The others use a pointer and pointer arithmetic to iterate over the array.

You should notice that it is possible to mix and match the type of parameter passed in and the internal implementation. E.g. using a pointer parameter does not force you to use pointer arithmetic in the function (although typically you would).

Finally, it is important to note that all 4 functions can be called exactly the same way.

The worked example starting on slide 19 provides further examples of defining functions that use array pointers and pointer arithmetic and how to call such functions.

The objective of the first lab exercise is to modify some starter code that currently uses arrays and array subscripting so that it uses array pointers and pointer arithmetic instead. The supplied sketch just fills in an array with random values and then prints the array contents whenever the button is clicked. The `randomizeArray` function fills the array with random data and the `printArray` function prints the contents of the array. In the supplied starter code, both functions use array parameters to the functions and array subscripting within the functions to access to elements of the array.

Lab7_BasicArrayPointers requirements:


- First download the starter code, `Lab7_BasicArraySubscripting`, from moodle and extract it into your sketchbook. Save the starter code with the correct sketch name for part 1 of the lab.
- Modify the `randomizeArray` function to use a pointer parameter for the array and internally to use pointer arithmetic to iterate over the array. This does not require any changes to the `loop()` function. (Test that this works).
- Modify the `printArray` function to use a pointer parameter for the array and internally to use pointer arithmetic to iterate over the array. Again, you do not have to change the `loop()` function at this point. (Test that it works as before).
- Finally, show that the `printArray` function can be used with an array slice just as easily as an array. Add a line to the `loop` function to call the `printArray` function correctly so that it only prints out array elements at indexes 1, 2, and 3. Note you must not modify the `printArray` function to achieve this. You must not modify the array in the `loop` function or make a copy of it either. All you need to do is call `printArray` with the appropriate parameters.

Hint: see notes 2.21 slide 17 for example code using an array slice

- The output of the final version of the sketch should be something like:

```
Press SW1 to restart
Array has been filled with random values
printArray {45, 31, 12, 5, 36, 57}
Array slice 1..3
printArray {31, 12, 5}
```

Reminder: Put a comment at the top of each sketch as in lab 1. Do this for every sketch in the lab. Don't forget to use good coding practice, e.g. prefer constants to literal values.

 Copy the sketch into your answer document and demonstrate it.

2 More array pointers – save sketch as “Lab7_UpdateArrayPointerArithmetic”

Background: in this second exercise you will use pointer arithmetic inside a function to modify an array element based on the value of its neighbouring elements. Unlike the previous exercise where you were iterating over an entire array (or slice) in this case you will only be modifying a single element of the array. However the modification will depend both on the value of the element to be modified and the value of the following element in the array.

Lab6_UpdateArrayPointerArithmetic requirements:


- Create a local array variable in the loop function, initialized with 10 numbers: 100, 110, 120, 130, 140, 150, 160, 170, 180, and 190. The array must remember its values from one execution of the loop function to the next.
- Define a function called `updateElement` which takes exactly one parameter: a pointer parameter that points to the element to be modified.
 - Inside `updateElement`, modify the element by setting its value equal to the value of following element + 1. You must use pointer arithmetic to access the following element based on the pointer you have to the element to be modified.

*Example: Assume an array whose elements are { 100, 110, 120, 130, 140 ... }. Suppose the pointer passed into `updateElement` refers to the element at position 2. Then `updateElement` should modify the element such that the array becomes { 100, 110, **131**, 130, 140 ... } because the element at position 2 (120) gets overwritten with 1 plus the value of the element at following position, position 3 (130).*

- Whenever SW1 is clicked
 - Choose a random position between 0 and length of array minus 2 and print this value.
 - Call `updateElement` to modify the array at that position.
 - Finally `printArray` to print the updated array contents.
 - The program output should look like:

```
Lab7_UpdateArrayPointerArithmetic starting...  
Click SW1 to start/restart
```

```
Initial data array contents: { 100, 110, 120, 130, 140, 150, 160, 170, 180, 190 }  
Chosen offset into array was 4  
Modified data array contents: { 100, 110, 120, 130, 151, 150, 160, 170, 180, 190 }
```

 Copy the sketch into your answer document and demonstrate it.