ENGG1003 - PASS Session 5

Mitchell Deltoer

Monday	**14-15**	**ES238**
Wednesday	**12-13**	**ES238**
Thursday	**10-11**	**MCLG42**

Arrays

Arrays can be thought of as a structure that stores a fixed-size collection of variables of the same data type. Instead of declaring individual variables, you declare one array variable and each individual variable is identified with its *index*. **Array indexes begin at zero**, i.e. the first element in an array would be `array[0]`.

An array is declared as follows

```
data_type array_name[array_size];
```

`data_type` can be any valid C data type, `array_size` needs to be an integer greater than zero and the `array_name` follows the same rules as any other C variable name.

The following example shows an example of initialising an array in one line

```
float array[5] = {1000.0, 2.0, 3.4, 7.0, 50.0};
```

The number of values between the braces { } cannot be larger than the number of elements between the square brackets [ ]. If you omit the size of the array, an array just big enough to hold the initialization is created. Single values of an array can also be initialised individually using

```
array[3] = 7.0;
```

Arrays and Functions (Intro to pointers)

A *pointer* is a variable whose value is the address of another variable. The syntax for declaring a pointer variable is

```
data_type *var_ptr;
```

where `data_type` is the base data type which the pointer is pointing at and `var_ptr` is the name of the pointer variable.

You have already seen *pointers* before. Using a variable name with an ampersand (`&`) before it accesses the address of the variable rather than the value that it holds. So an example pointer declaration would look something like

```
int var;              // declaring a integer variable
int *var_ptr = &var; // initialising a pointer-to-int with the address of var
```

This relates to arrays in that **the name of an array without an index is a *pointer* to the first value in the array**. That is, the following two `printf` statements of code are equivalent:

```
int array[2];      // declaring an array of int, length 2
printf("This is the address of array[0]: %lu\n", array);
printf("This is the address of array[0]: %lu\n", &array[0]);
```

When using arrays in functions, a pointer to the first value in the array and the array size are both required as separate arguments (inputs) to the function. Because **the function is using the address of the original array, when values in the array are changed in the function they are also updated in the main function**.

```
// either one of these is a valid function prototype
void example_function(int *array_name, int array_size)
void example_function(int array_name[], int array_size)
```

Practice Programming

Some things to consider when using functions within your program:

- What *arguments* (inputs) are there to the function? What are their data types?
- What is the *return value* (output) of the function? What is its data type?
- Have I remembered to (correctly) include the *function prototype* and the *function definition*?

Task 1: Set array values to zero

1. Write a C program to set all the values within an array to zero. Print the entire array to confirm your result.
2. Modify the previous program to instead use a *function* to set all the values to zero. The array should still be declared from within the main function.

Task 2: Array copying

1. Write a C program that copies the values of one array into another. Print both arrays to confirm your result.
2. Modify the previous program to instead use a *function* to do the copying. The arrays should still be declared from within the main function.

Task 3: `scanf` values in to array

1. Write a C program to read in **up to 10** values into an array using standard input (`scanf`). Print the entire array to confirm your result.
   Hint: First read in an integer which is the number of variables they want to read into the array.
2. Modify the previous program to instead use a *function* to read in the values. The array should still be declared from within the main function.

Task 4: Array arithmetic

1. Write a C program to calculate the average of all the values in an array.
2. Modify the previous program to instead use a *function* to perform the calculation and return the result to the main. The array should still be declared from within the main function.

Task 5: Challenge: Duplicate counting

Write a C program that reads in up to 10 numbers into an array. The program should count the number of duplicate elements within that array and print the result. Example test data and expected output is given below:

```
Test Data:  [2,1,1,2,1,3,4]
Total number of duplicate elements found in the array is:  3
```