# Programming Assignment #2

This is another short assignment to help you practice with arrays, functions, and iteration. The program will manipulate arrays in several different ways. The elements in all of the arrays are integers. There are five simple functions that you need to write to complete the assignment.

| Function | Description |
|---|---|
| `void reverse_array(int a[], int size);` | Given an array, reverse the order of the elements in the array. Do **not** create another array in the function. |
| `void add_arrays(const int a[], const int b[], int c[], int size);` | Given three arrays, add the elements of the first two arrays and put the sum in the third array. |
| `void scalar_multiply(int a[], int size, int multiplier);` | Given an array and a multiplier, multiply each element by the multiplier. |
| `int dot_product(const int a[], const int b[], int size);` | Given two arrays, determine the dot product (multiply each corresponding element and sum the products). Return the value. |
| `void cross_product(const int a[], const int b[], int c[]);` | Given three arrays, determine the cross product of the first two. The cross product is another array and will be placed into the third array. The size of all three arrays will always be 3. |

Each function provides the size of the array, with the exception of the cross product. (The cross product will always have arrays of size 3.) If there is more than one array, you can be sure that all of them are the same size.

**Examples:**

`reverse_array:`

Given array **a** with these values: `1 2 3 4 5 6 7 8`

After reversing, **a** looks like this: `8 7 6 5 4 3 2 1`

`add_arrays:`

Given array **a** with these values: `1 2 3 4 5`

Given array **b** with these values: `3 4 7 2 1`

Array **c** will look like this: `4 6 10 6 6`

`scalar_multiply:`

Given array **a** with these values: `1 2 3 4 5`

Given this multiplier: `8`

Array **a** will look like this: `8 16 24 32 40`

`dot_product:`

Given array **a** with these values: `1 2 3`

Given array **b** with these values: `3 4 7`

The return value is this: `32`

`cross_product:`

Given array **a** with these values: `1 2 3`

Given array **b** with these values: `3 4 7`

Array **c** will look like this: `2 2 -2`

**Details**
You are given a file called **`driver.c`**, which includes the main function. There are four functions prototyped in that file, one for each of the functions you are to write. As always, you must implement all of these functions *exactly as prototyped.* You will implement these functions in a file called **`array.c`**. A sample command line to compile this assignment looks like this:

```
gcc -O -Werror -Wall -Wextra -ansi -pedantic driver.c array.c PRNG.c -o arrays
```

This will compile and link both files and produce an executable file named **`arrays.exe`**. Please use this ***exact*** command line. Just copy and paste it into your command window.

This assignment will not require you to include any header files in your code so do not include anything. All of the code that you write must be in `array.c`, since you will not be turning in `driver.c`. **REPEAT**: **You will not turn in `driver.c`**, so any changes you make to `driver.c` will not be seen by me.

With the exception of the `cross product` function, all of the functions will be doing some sort of looping. You can use either a **for** loop, **while** loop, or **do** loop. You are **NOT** to create any arrays in any of the functions you write. All of the arrays that you need are given to you as inputs to the functions. No credit will be given for any function that creates an array. The website contains additional information on the dot product and cross product.

**What to submit**
You must submit the C file (**array.c**) in to the submission server.

| File | Description |
|---|---|
| array.c | The C file. This file contains all of the source code for the program, properly formatted and commented following the CS120 Style Guide. |

**Make sure your name and other info is on all documents.**