Passing Arrays as Parameters

In general, when we want to pass an array as a parameter, we will want to pass a pointer to the array, as well as an integer specifying how many elements are in the array, such as this:

```
3

1

2

}

int myFunction(int * myArray, int size) {

  // whatever code...
```

There is no way to get the size of an array in C, other than passing along that information explicitly, and we often want to make functions which are generic in the size of the array they can operate on (*i.e.*, we do not want to hardcode a function to only work on an array of a particular size). If we wanted, we could make a struct which puts the array and its size together, as one piece of data—then pass that struct around.

When we pass a pointer that actually points at an array, we can index it like an array (because it *is* an array—remember the name of an array variable is just a pointer), and perform pointer arithmetic on it. Such pointer arithmetic will be well defined, as long as the resulting pointer remains within the bounds of the array, as we are guaranteed that the array elements are sequential in memory.

We can also pass an array as a parameter with the square bracket syntax:

```
3

1

2

}

int myFunction(int myArray[], int size) {

  // whatever code...
```

This definition is functionally equivalent to the one we saw before with the pointer syntax. Some people prefer it, as it indicates more explicitly that *myArray* is intended to be an array. We can also write a size in the{}, however, the compiler will not make any attempt to check if that size is actually correct, thus it is easy to write something incorrect there (or something that becomes incorrect as you change your code), and may confuse a reader.

When you call a function that takes an array, you can pass any expression which evaluates to a pointer to a sequence of elements. Typically, this expression is just the name of the array (which is a pointer to the first element). However, we could perform pointer arithmetic on an array (to get a pointer to an element in the middle of it—which is a valid, but shorter array), or we might retrieve the pointer out of some other variable—it might be a field in a struct, or even an element in another array.

Completed