Goals of this Tutorial

The goal of this tutorial is to reinforce the following material:

- Arrays
- Pointer Arithmetic
- Debugging

Arrays

Arrays can be thought of as pointers to a block of memory. They can be used to store a **fixed number** of elements of the **same type**.

Example of array syntax:

```c
int my_array[3] = { 1, 2, 3 };  
int x = my_array[0]; // x = 1
```

In the above code, `my_array` is a pointer to the first element of the array. The `[i]` syntax **dereferences** the `(i + 1)`th element of the array.

The syntax `a[i]` is shorthand for the equivalent expression `*(a+i)`.

Array Initialization

There are several ways to define an array:

```c
int a[3]; // array is not fully defined

int b[3] = { 1, 2, 3 }; // array length is explicit

int c[] = { 1, 2, 3 }; // array length is inferred  
// c has length 3, and we cannot change it later

int d[3] = {0}; // array of length 3, filled with zeros

int e[8] = { 7, 4, 1 };  
// {7, 4, 1, 0, 0, 0, 0, 0}
```
A6 Hint

// find_position(pos, x, y) determines the x and y coordinates for pos on the chess board.
// e.g. When passed pos = 14, (and 2 valid pointers)
// find_position would set x = 6, y = 1
// representing the second row from the top,
// sixth column from the left
// requires: 0 <= pos < 64
// x & y are non-NULL
// effects: modifies x and y
void find_position(int pos, int *x, int *y);

Array Exercise

// reverse(arr, len) reverses arr, an array of len integers.
// requires: arr != NULL, len > 0,
// arr is an array of length len
// effects: mutates arr
void reverse(int arr[], int len);

Pointer Arithmetic

Certain arithmetic operations can be performed on pointers. An integer can be added or subtracted to a pointer, and pointers of the same type can be subtracted from one another.

int a[10];
int *p = a;    // a is a pointer to first element
int *q = &a[9];  // address of 10th element
q = a + 9;    // equivalent
a[2] = q - p;  // set the value of 3rd element as 9
q = p + 1;    // now q == &a[1]

Addition of pointers is not allowed.
Array Notation vs Pointer Notation

If you ask the right people, you can start heated debates on whether to use `int a[]` or `int *a` in a parameter declaration.

They are equivalent.

Some people will argue for the first style because it is clear that the function takes an array.

Other people will argue for the second style because they believe the first is misleading, as the array isn’t actually passed by value.

You should be familiar with both of them.

---

Pointer Arithmetic: Example

Implement the following function using only pointer notation.

```c
// ptr_product(arr, len) returns the product of len integers in array arr (using pointer arithmetic).
// requires: arr != NULL, len > 0,
// arr is an array of length len
int ptr_product(const int *arr, int len);
```

Now implement the function again using array notation instead.

```c
int arr_product(const int arr[], int len);
```

---

Debugging Example

A sample module has been provided on Seashell that contains a number of errors. Use Seashell error messages and your C knowledge to return the program to functioning order.
You should now be comfortable with the following array-related concepts:

- Defining and initializing arrays
- Pointer arithmetic, and converting array notation to its pointer arithmetic equivalent
- Using loops to manipulate arrays
- Basic debugging strategies