# 202-A03 Pointers, Arrays and Parameters

This exercise is different from our usual exercises. You don't have so much a problem to solve by creating a program but rather some things to understand about the programming language you are learning. To do so, we are going to write little programs which illustrate the relation between what we write and how it executes.

## Swapping two integer arrays, the basic way

Here is the main function of a little program which is going to declare two arrays of integers (same size) and which is going to call a function to swap their respective contents.

```
#define SIZE 3
void swap_arrays_1 ( int p1[] , int p2[] , int size);

int main ( )
{

    int  a1[ SIZE ] = {0,0,0} ;
    int  a2[ SIZE ] = {1,1,1} ;
    swap_arrays_1 ( a1 , a2 , SIZE  ) ;

}

void swap_arrays_1 ( int p1[] , int p2[] , int size)
{

    int i , tmp ;
    for  ( i = 0 ; i < size ; i++ ) {
            tmp = p1[i] ;
            p1[i] = p2[i] ;
            p2[i] = tmp ;
    }

}
```

Complete the above code with a function which will display a message (e.g. "Before swapping", "After swapping") followed by the contents of both arrays. You will use it to make sure that our *swap_arrays* function is working properly. This function should have the following prototype;

```
void display_arrays(char* message, int data1[], int data2[], int size);
```

The output of your code should look as follows;

```
Initial values;
        Array #1: [0][0][0]
        Array #2: [1][1][1]

Swapped values;
        Array #1: [1][1][1]
        Array #2: [0][0][0]
```

Why does it work?

## Swapping arrays with pointers

What about switching the parameters' types to pointers on *int*?

```
void swap_arrays_two ( int* p1 , int* p2 , int size )
{

    int i , tmp ;
    for ( i = 0 ; i < size ; i++ ) {
            tmp = p1[i] ;
            p1[i] = p2[i] ;
            p2[i] = tmp ;
    }

}
```

This swap function is working, however we didn't use & when calling it with the a1 and a2 parameters. Why does it work then?

Oddly enough, it seems that C lets us declare our parameters as pointers on *int* or arrays of *int* (without size specified). Let's see if this holds true even outside of the context of functions parameters. Add to your main the following code;

```
int another[];
```

What is the compiler error (cut and paste it below)? Does this mean we are allowed to define arrays without size specification when they are not parameters?

?

## Swap Arrays, the third

Let's now try to implement an integer arrays swap function which does not copy each element. We will attempt to swap the arrays by swapping their start address instead.

After all, the name of an array is the same thing as the address of its first element, right?

So why don't we simply write a function which, given *a1* and *a2* as parameters, will make sure that after its execution *a1* and *a2* will now refers to each others' contents.

Since we are allowed, for functions parameters only, to declare our arrays parameters as pointers on *int* let's use this to our advantage and treat these arrays as pointers.

```
void swap_arrays_3a ( int* p1 , int* p2 )
{

    int* tmp ;
    tmp = p1 ;
    p1 = p2 ;
    p2 = tmp ;

}
```

Implement this function, call it from the main with our two arrays *a1* and *a2* as parameters then display the arrays' contents before / after the function call to check if it's working.

Does it work?

?

Why yes / no? Draw a stack diagram on a separate sheet of paper to come up with a real understanding of the situation.

?

Let’s try for something different. What if we declare our arrays parameters as arrays?

```
void swap_arrays_3b ( int p1[] , int p2[] )
{
    int* tmp ;
    tmp = p1 ;
    p1 = p2 ;
    p2 = tmp ;
}
```

Does it compile? What does this tell you about whether C allows you to assign value to an array directly?

?

Does it work? Compare the reasons to the previous version (3a)

?

Wait a minute, I thought we couldn’t assign values to variable arrays. We know that some things are allowed for array parameters but not for all array variables (cf. above the definition of *int another[];* in the main). Let’s verify that treating array names, used as parameters, as pointers is or is not a general rule also applying to array variables by adding the following to the main;

```
int another[3];
int * p;
p = another;
another = p;
```

What is the output of the compiler on this code (cut and paste it below)? What are the implications on whether C allows us to assign a value to an array variable?

?

## Fast swapping (fourth version)

We are going to modify the main of our program so that, instead of trying to swap around the two arrays using their variables names (remember, we can't anyway assign anything to array variables names), we try to obtain a fast swap by working on two pointers on the arrays rather than the arrays themselves.

```
#define SIZE 3
void swap_arrays_3a ( int *p1 , int* p2 );

int main ( )
{

    int  a1[ SIZE ] = {0,0,0} ;
    int  a2[ SIZE ] = {1,1,1} ;

    int *p1, *p2;

    p1 = a1;
    p2 = a2;

    swap_arrays_3a ( &p1 , &p2 ) ;
}
```

Let's try first to apply again our 3rd version of the swap function.

```
void swap_arrays_3a ( int* p1 , int* p2 )
{

    int* tmp ;
    tmp = p1 ;
    p1 = p2 ;
    p2 = tmp ;

}
```

The result is expectable. Make sure you understand why calling the function with pointers variables containing the address of the arrays didn't change anything to the situation (draw a stack diagram to clarify this).

?

Now, let's try for something different. First, you need to understand how to write a function which swaps the contents of two integer variables by using pointers. You want

your function to be able to modify a variable parameter by reference, you use pointers on the type of this variable and pass its address by value during the call.

Well, here, we'd like to enable our swap function to change the contents of the two pointers *p1* and *p2* (local variables from the main) which are its parameters during the call. To be able to modify an *int* you need pointer on int parameters. To modify a pointer on *int* you need pointers on pointers on *int* parameters. Similarly, you call the function with the address of an *int* variable you want it to be able to modify. Here, you'll call our function with the address of the pointer on *int* you want to modify.

Implement the following and use it from the main;

```
void swap_arrays_4a ( int** p1 , int** p2 )
{
    int* tmp ;
    tmp = *p1 ;
    *p1 = *p2 ;
    *p2 = tmp ;
}
```

Does it work?

?

Explain, in a detailed manner by using stack diagrams, what happens during the call of this function.

?