

# Tutorial 4

MATH10017

10-11 Nov 2022

## Contents

<b>1</b>	<b>Stay organized</b>	<b>1</b>
<b>2</b>	<b>Array printing function</b>	<b>2</b>
<b>3</b>	<b>Sum</b>	<b>2</b>
<b>4</b>	<b>Filling arrays</b>	<b>2</b>
<b>5</b>	<b>Copy</b>	<b>3</b>
<b>6</b>	<b>Reverse</b>	<b>3</b>
<b>7</b>	<b>Aggregating in place</b>	<b>4</b>
<b>8</b>	<b>Aggregating to a new array</b>	<b>4</b>
<b>9</b>	<b>Bad agg</b>	<b>5</b>

**Instructions.** Create a file tutorial4.c (use the .bat file from lab to open VS Code first).

- Don't copy and paste from the .pdf file, rewrite the code.
- Make sure your code compiles.
- Use comments to stay organized

## 1 Stay organized

Use comments and "helper functions" to keep your code neat.

```
#include <stdio.h>

// print header and newline, for test printing
void header(char func_name[]){
    printf("Test(s) for %s\n", func_name);
}

// Exercise 1: array printing function

int main(){

    // Test for print
    header("print");
    int arr1[] = // YOUR CODE HERE
    int len1 = // Set this to the length of arr1; this is clearer than putting a number
}
```

- The function `header` prints out newlines and some text, so we know what the output in the terminal is for.
- Comments are used to describe functions, and to divide the code into sections.
- Use descriptive variable names (but use short names for loop indices etc).

## 2 Array printing function

Write a function to print out arrays:

```
// print: print out the elements of an array on one line
void print(int a[], int len){
    ; // YOUR CODE HERE
}
```

- Make sure there is a space after each element of the array, and a newline at the end.

- In `main`, declare an array with the values `{1,2,3,4,5}` and print it using `print`.
- Remember to print a header and use comments to organize `main`.

### 3 Sum

- Write a function that takes an integer array (and its length) and returns the sum of its entries.
- Test this function in `main` by printing out the result of summing the array you defined in the first exercise.
- Hint: the declaration should be `int sum(int a[], int len)`

### 4 Filling arrays

Make two functions:

1. `fill_zeros` that sets all the elements of an array to zero
2. `fill_range` that sets the elements of the array to `1, 2, 3, \dots, n` (for the largest  $n$  that makes sense).

```
// ADD A DESCRIPTIVE COMMENT
void fill_zeros(int a[], int len){
    // YOUR CODE HERE
}
```

```
// NOW DEFINE fill_range
```

Test these functions in `main`.

- Remember the declaration for a blank array is `int arr[n];` where  $n$  is some number (**not** a variable).

### 5 Copy

Write a function to copy one array to another, using a for loop to set the value of each element of array `b` to the corresponding value in array `a`.

```
// copy array a to array b
void copy(int a[], int b[], int len){
    // YOUR CODE HERE
}
```

To test this, create two arrays of the same length: one with values assigned to it, and one blank.

## 6 Reverse

Write a function that reverses the elements of an array:

```
// reverse the elements
void reverse(int a[], int len){
    ;
}
```

Test this:

- define a new array (of even length)
- print the new array
- call **reverse**, then print again.
- do this again for an array of odd length.

Hints:

- use a for loop up to `len / 2`
- use a "temp" variable to swap element  $i$  with the element that is  $i$  spaces from the end.

## 7 Aggregating in place

Write a function that "aggregates" an array, by setting the  $i$ th element to the sum of the first  $i$  elements. For instance,

$$\{1, 2, 3, 4, 5\} \implies \{1, 3, 6, 10, 15\}$$

This function aggregates "in place", because it stores the values of the sum in the same array that has the original values.

```

// replace the i-th element of a with the sum of the first i elements
void agg_in_place(int a[], int len){
    for(int i = 0; i < len; i++){
        ; // YOUR CODE HERE
    }
}

```

Test this code.

## 8 Aggregating to a new array

Use `copy` and `agg_in_place` to make a function that puts the sum of the first  $i$  elements of an array into the  $i$ -th element of a different array.

```

void agg(int a[], int b[], int len){
    // COPY a to b
    // aggregate b in place
}

```

Test this code.

## 9 Bad agg

The following code tries to get away with not using `copy`. What do you think it will do?

```

void bad_agg(int a[], int b[], int len){
    b = a;
    agg_in_place(b, len);
}

```

Test this code.

- What happens? Is this different from what you expect?
- Next week we will know enough to explain what happened.