Arrays

By: **Annamalai A**

An array is a like a container that stores multiple values of similar data type. They are stored in contiguous memory locations .i.e. The elements of an array stored in consecutive memory addresses. Let's see how this can used for our advantage later.

The elements of an array accessed using the name of the array and specifying the position of the element, called **index**.

Uses of Array

So where can arrays be useful exactly? Let's say we want to store data of similar type and which represents similar information, such as marks of students or time I spend on phone everyday in minutes.

Now, when giving inputs, we consider `n` data, or `n` cases, such as marks of 10 students or the time I spent on phone for the past 30 days. Imagine taking input for each case in separate variable. It would look something like this:

```
int day1 = 245
int day2 = 228
int day3 = 189
...
int day28 = 195
int day29 = 200
int day30 = 167
```

The amount of time we spend on typing out these variables is going to be a lot, especially for large data. This is where arrays come become useful. The same code with arrays can be written like this:

```
int days[30] = {245, 228, 189, ..., 195, 200, 167};
```

Boom! All data written in just one line!

Creating and Accessing Arrays

The above code snippet also shows us how to create an array. The common way to create an array is done like this:

```
type array_name[size] = {element_1, element_2, ..., element_n};
```

The size of the array must be mentioned while creating it so that the compiler knows how many elements are going to be stored in it. The size cannot be changed later in the program. Why? Think about it. It has something to do with memory.

Now, we have an array with data. How do I access it? We can access the elements by specifying the array and position (index). The positioning of elements start from `0` and go upto `n - 1` from left to right where `n` is the number of elements in the array or the length of the array.

The specify the position, follow this syntax: `array_name[index]`. Passing an index that does not lie within the range will throw an error. Here is an example, where we want to access the number of minutes used on day 28.

```
days[27];
```

Its that simple!

Traversing

Traversing means going through the elements of the array one by one. It can be easily done using a `for` loop!

```
for (int i = 0; i < 30; i++) {
    printf("%d", days[i]);
}
```

Now, for each value of `i`, the compiler prints the element in that position.

Now, there is another way in which arrays can be created. We first initialize the array (it's name and size) and pass the values by traversing. Here is an example:

```
int arr[10];
for (int i = 0; i < 10; i++) {
    arr[i] = i;
}
```

It works best when elements have some common relation. In this case, it is the first 10 whole numbers.

You can also do this by specifying a variable that holds the size of the array. This works best when we are changing the size while testing and we don't want to change the value in multiple areas.

```
int n = 10;
int arr[n];
for (int i = 0; i < n; i++) {
    arr[i] = i;
}
```

Now when changing the value of `n`, both the size and condition in the loop will change accordingly.

The `sizeof()` Function

This function gives the number of bytes a variable or a data type holds. When talking about arrays, it gives the number of bytes all the elements occupy together. It does not give the length of the array directly.

Suppose, we have an array of 5 values of type `int`. A variable of type `int` occupies 4 bytes. So this array occupies a total of 20 bytes, because each value in the array occupies 4 bytes, and there are 5 elements.

To use the function, call it and pass the variable or array for which we want to measure the size:

```
int n, arr[5];
sizeof(n); // Returns 4
sizeof(arr); // Returns 20
```

So, to get the number of elements or the length of array, we can divide the number of bytes occupied by the array by the size of a value or its data type in

the array.

```
int len = sizeof(arr) / sizeof(int); // Length of the array 'arr'
```

Memory Representation

So remember the question I asked earlier? Why can't we change the length of array? The answer is how memory is allocated to arrays.

In languages like `Python`, the list (array) is allocated a special memory address, and each element inside it also has a memory address but it is given randomly, and it has nothing to do with the address of the array it is in. In fact, the address of each element is random, and does not depend on the others'.

But in languages like `C`, The array actually has no memory. It is the elements that has a memory address and act like arrays. The elements take up memory in a contiguous manner, and hence their address depends on the previous elements, and the hence the first element.

Let's say we have an array `arr` and it's first element `arr[0]` takes 4 bytes, and it's memory address is `1000`. Then the order goes like this:

Index of Elemenet	Address
0	1000
1	1004
2	1008
...	...
n	$1000 + 4n$

And there is our answer! Now if we create any more variables after this, it is going to occupy the address after the last element. If we change the length later, then we need to give enough space for the new elements and still make sure that the address is occupied in a contiguous manner. But may not happen because another variable would've already occupied that space.

Arrays in Functions

When passing arrays to functions, it is important to note that we do not use call by value method. It is always done using call by reference method.

```
void functionName(int arr[], int size); // Method 1
void functionName(int *arr, int size); // Method 2
```

```
#include <stdio.h>

// Function to display array elements
void display(int arr[], int n) {
   int i;
   for(i = 0; i < n; i++)
      printf("%d ", arr[i]);
}

int main() {
   int data[] = {10, 20, 30, 40, 50};
   int size = 5;
   display(data, size); // passing array to function
   return 0;
}
```

Here is how it can be done using address-of operator.

```
#include <stdio.h>

// Function to display array elements
void display(int *arr, int n) {
   int i;
   for(i = 0; i < n; i++)
      printf("%d ", *(arr + i));
}

int main() {
   int data[] = {10, 20, 30, 40, 50};
   int size = 5;
   display(&data[0], size); // passing address of first element of array to fun
ction
```

```
    return 0;
}
```

Common Errors

There are some common errors that can possibly be made while working with arrays:

- **Accessing invalid index**:

  ```
  int arr[5] = {1, 2, 3, 4, 5};
  printf("%d", arr[5]); // Invalid: last index is 4
  ```

- **Using uninitialized elements**

  ```
  int arr[5];
  printf("%d", arr[2]); // Value is undefined
  ```

- **Forgetting array size**: It does not throw an error, but causes logical problems

  ```
  int arr[10];
  int n = sizeof(arr); // gives total bytes, not element count
  ```