Programming Fundamentals

Muhammad Ateeq

[Updated: 25-03-2023]

Arrays

Basics of Arrays

What are arrays and why do we need them?
Arrays are a data structure that allows you to store a collection of elements of the same type in a contiguous block of memory. They are an essential part of computer programming and are used extensively in various applications. Arrays are necessary because they provide a way to store and manipulate a collection of related data in a structured and efficient manner. Instead of using individual variables to store each item, you can use an array to store them all together. Here's an example to illustrate how arrays work. Let's say you need to store the marks of 10 students in a class and you create ten variables to store these.

```
int marks1 = 72;
int marks2 = 77;
int marks3  = 63;
int marks4 = 85;
int marks5 = 91;
int marks6 = 70;
int marks7 = 64;
int marks8 = 82;
int marks9 = 59;
int marks10 = 86;
```

Handling such data by creating numerous variables and handling it without problems is hard. As the amount of data grows and variables needed to store this data increase, program becomes harder and harder to manage. So what can be done? In this case, instead of creating 10 individual variables to store each student's mark, you can create an array that can store all the marks.

```
int marks[10];
```

In this example, marks is an array of **10** integers. You can access each element of the array using its **index**, which starts from **0**. For example, **marks[0]** will give you the mark of the first student, **marks[1]** will give you the mark of the second student, and so on.

Let’s see some advantages with the arrays that we don’t have with regular variables. You can use loops to iterate over all the elements of the array and perform some operation on each element. These operation may include printing the values, calculating percentage or average, etc. For example, the following loop will calculate the average mark of all the students:

```
int sum = 0;
for (int i = 0; i < 10; i++) {
    sum += marks[i];
}
float avg = sum / 10.0;
```

To do the same thing without arrays, while using ten variables, the loop could not be used. Therefore, it becomes more manageable, tidy, cleaner, compact, and shorter to use arrays.

How are arrays created, initialized, and accessed?
In C++, arrays are created and initialized using a simple syntax, and accessed using the square bracket notation. Here are examples of how to create, initialize, and access arrays in C++:

Creating an Array:
To create an array in C++, you specify the type of the elements, followed by the name of the array and the number of elements in square brackets. For example, to create an array of integers with five elements, you would write:

```
int arr[5];
```

This creates an array of integers called **arr** with five uninitialized elements. 5 in square brackets specifies the size of array (i.e., the number of variables in a way) and must be specified while declaring/creating an array.

Initializing an Array:
To initialize an array in C++, you can use the initializer list syntax. For example, to initialize the array **arr** with the values 1, 2, 3, 4, and 5, you would write:

```
int arr[5] = {1, 2, 3, 4, 5};
```

This initializes the array with the specified values where arr[0] corresponds to 1, arr[1] corresponds to 2, and so on. You may leave the size empty if initializing the array while declaring:

```
int arr[] = {1, 2, 3, 4, 5};
```

Accessing Elements of an Array:
You can access (individual) elements of an array using the square bracket notation. For example, to access the third element of the array **arr**, you would write:

```
int x = arr[2];
```

This sets the variable x to the value of the third element of the array. You cannot access the elements of an array all at once just by typing the name of the array without brackets and index because this will give you the address of the array in memory. We shall cover details about addresses later when we cover pointers. In short, you can only access a single item of an array at a time by specifying its index.

Here are some more examples that combine these concepts:

```
// Create an array of 10 integers, initialized to zero
int arr[10] = {0};

// Create an array of doubles with 3 elements
double vals[3] = {1.2, 2.3, 3.4};

// Access the second element of the array
double y = vals[1];

// Change the value of the fourth element of the array
arr[3] = 42;
```

In these examples, we create arrays of different sizes and types, initialize them with different values, and access and modify their elements using the square bracket notation.

So arrays make it easy to manipulate a lot of data! How?

Here's a C++ program that prompts the user to input marks in 10 subjects, calculates their average, and prints both the marks and the average:

```
#include <iostream>
using namespace std;

int main() {
  int marks[10];
  int total = 0;

  // Prompt the user to input marks for 10 subjects
  for (int i = 0; i < 10; i++) {
    cout << "Enter marks for subject " << i+1 << ": ";
    cin >> marks[i];
    total += marks[i];
  }

  // Calculate the average marks
  double average = static_cast<double>(total) / 10;

  // Print the marks and the average
  cout << "Marks: ";
  for (int i = 0; i < 10; i++) {
    cout << marks[i] << " ";
  }
  cout << endl;
  cout << "Average: " << average << endl;

  return 0;
}
```

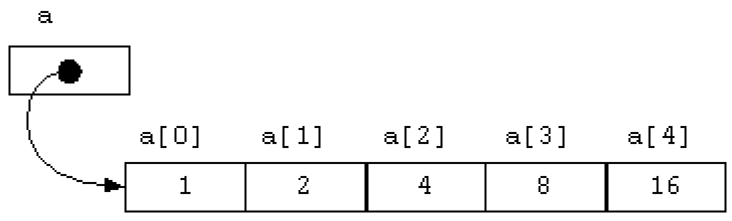
In this program, we declare an array named **marks** of integers with size 10, which will store the marks entered by the user. We then use a for loop to prompt the user to input marks for each subject, and store them in the array marks. We also keep track of the total marks entered in a variable called total. Next, we calculate the average marks by dividing the total by the number of subjects, and store the result in a variable called average. Finally, we use another for loop to print the marks entered by the user, and then print the average. Note that we use **static_cast** to convert total to a double before dividing it by 10, in order to ensure that the result is a floating-point number.

How are arrays stored in memory (RAM)?

In C++, arrays are stored in contiguous blocks of memory, with each element occupying a fixed amount of space determined by its data type. For example, if you declare an array of integers in C++ as follows:

```
int a[5] = {1, 2, 4, 8, 16};
```

The memory layout of the array would look like this:



0x100 0x104 0x108 0x10C 0x110

In this example, each integer element occupies 4 bytes of memory, so the first element arr[0] is stored at memory address 0x100, the second element arr[1] is stored at address 0x104, and so on. Notice that **0x** means that the number systems used to represent addresses is hexadecimal, and not decimal or binary. Hexadecimal is used in computers to represent memory address to shrink the length because in binary form the addresses are very long.

Note that C++ does not perform any bounds checking on arrays, so it's important to ensure that you don't access elements outside of the array bounds, as doing so can lead to undefined behavior and memory errors.

What is the size of the array and how does it relate to a normal variable?
C++ has an operator sizeof that can be used to determine the size of any object. For example, to determine the size of an int in c++:

```
int num = 42;
cout << "Size of integer: " << sizeof(num) << " bytes" << endl;
```

This would output something like:

```
Size of integer: 4 bytes
```

Similarly, the size of an array is the sum of the sizes of all element in it. For example, the size of an array of type int and size 10 is 10*4 = 40 bytes, and can be figured out as:

```
int arr[10];
cout << "Size of array: " << sizeof(arr) << " bytes" << endl;
```

This would output something like:

```
Size of array: 40 bytes
```

In this example, the sizeof operator is used to determine the size of the arr array in bytes, which is 40 bytes on most systems (4 bytes per integer * 10 integers). To determine the size of one element in array following can be used:

```
Cout<< "Size of one element in array: "<<sizeof(arr)/sizeof(arr[0]);
```

Here we get the size of whole array first and divide it with the size of one element in the array to get the size of a single element in the array.