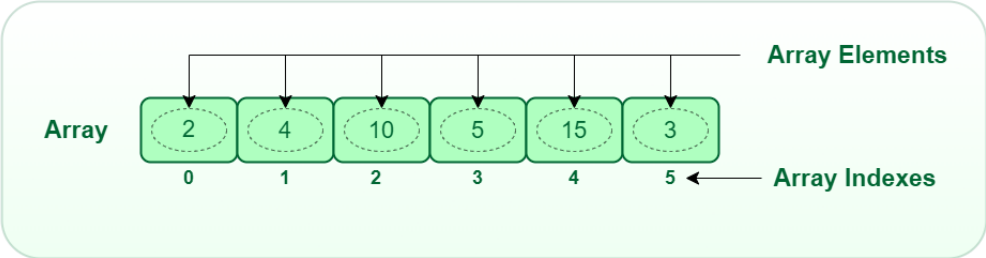
Chapter 2 Part 1: Arrays – the basics

Introduction



An array is a fixed-dimensional and indexed sequence of fix number of elements of a single type. Every aspect of this definition is important. First, an array has dimensions. So we can create 1D arrays, 2D arrays, 3D arrays, and so on. The simplest form is the one-dimensional array. Second, an array is an indexed collection. Indexed means each element has a designated position. Hence, you can retrieve an element by its position. For example, if you ask what is the value at position 2 in the above picture then the answer is 10. Third, an array has a fixed number of elements along each dimension. This count of elements along a dimension is called the length of the dimension. When people deal with a 1D array, they typically use the term length of the array to indicate the length of that sole dimension. Note that this does not mean you have to use a constant in your code as an array dimension length in your source code. It means that when you create an array in the program, you have to specify the length of each of its dimensions that you cannot change later. Finally, an array can hold elements of only a single type. That is, you cannot mix strings with numbers in an array – not even mix integer numbers with fractional numbers. All these restrictions are essential for efficient operations. In fact, arrays are by far the fastest data structures that computers can process. There is a reason for this efficiency. A computer's own memory is a 1D array of fixed-sized memory cells. Consequently, the notion of a programming language array fits directly with how computer stores and access data from memory.

The sit arrangement in a classroom can be described using a 2D array. In a multidimensional array, each array element has an index for each of the array dimensions. So a 2D array of classroom sits has two indexes, a row index and a column index. The value to store in the index depends on array type. For example, you may be interested to know which sits are occupied and what are not. Then each element can be a Boolean value. However, if you want to know who sit where, then each element of the array will be a student object.

A real life example of a 2D array

Accessing an Array Element: If you consider how the CPU of your computer read/write data item from the computer memory (aka the RAM) for an array, you will understand why array access is so efficient. The CPU is given the starting address of the array in memory and the index of the element program needs to access. As the size of the object an array hold can be larger than the size of a single memory cell, CPU cannot just add the index with the starting address to locate the element in the searched index. However, we know that the elements are all of the same type. So there size is fixed. Therefore, the CPU just multiply the index by the fixed object size and add that result with the starting address to determine the address of the searched element. Then it needs a single memory load/store operation to read/write the element. So accessing element by index from an array is a constant-time operation.

This also tells you why it is important to have the length of each dimension of an array immutable (means you cannot change). Because only then the language can decide how any consecutive memory cells it should reserve from the hardware memory to store the array.

Operations on Array

The most common operations that you do on an array are writing/reading a data item to/from an index. Suppose you have a 1D array named *student_names*. If you want get the 5th student from the array then you write:

fifth_student = student_names[5]

To change the value at 5th index and set it to 'John Doe', you write:

student_names[5] = 'John Doe'

Typically, you do not shift the positions of elements of an array, unless you are swapping the positions of two elements (often needed for sorting the array in increasing or decreasing order of element values). Similarly inserting a new element at a particular index or removing an element from that index is also atypical as that involves shifting the positions of other elements. You avoid such operations on an array as they are costly. However, it is common to create a new array from an existing array where the new array has a filtered subset of elements from the old array or have the same elements but in a different order. There is another basic data structure that is highly efficient for that, it is called a *List*.

Determining the Length of an Array: it is a curious question, how do you know the dimension lengths of an array. Languages that are focused on the best performance say that the programmers should store the length of each dimension in some other variable. This makes sense as you provide the dimension lengths when you create the array. This means that when you write something like *array[i]* to access the i^{th} element of the array, the language does not check if *i* is crossing the length of the array. In other word, there is no checking if the index you gave is valid or not. If you give an invalid index, you get invalid data. Examples of such languages are C, Fortran, and Golang.

On the other hand, some other languages, particularly object oriented languages, store the length information you supplied in the first memory cell of the array and you cannot access that location from

your program. When you write *array[i]* in such a language, the generated code put the array access inside a conditional if-else block. The if-else block check if *i* is greater than equal to 0 and less than the length of the array. If that is not true then you get an error when your program executes the statement containing array[i]. Examples of such languages are Java and C#.

The tradeoff here is that accessing an array element in Java/C# is two to three times slower than accessing an array element in C/Fortran/Golang. However, if you are a careless programmer then Java/C# gives better protection that you will not make errors. This efficiency of array access is one of the main reasons critical system software such as the operating systems and compilers are written in C.