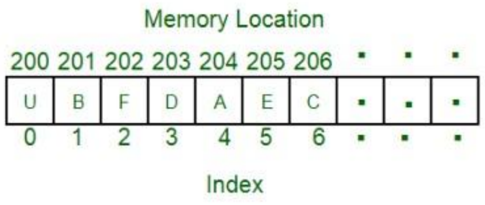
Introduction To arrays

An array is a collection of items of same data type stored at contiguous memory locations. This makes it easier to calculate the position of each element by simply adding an offset to a base value, i.e., the memory location of the first element of the array (generally denoted by the name of the array).

For simplicity, we can think of an array a fleet of stairs where on each step is placed a value (let's say one of your friends). Here, you can identify the location of any of your friends by simply knowing the count of the step they are on.

Remember: "Location of next index depends on the data type we use".



The above image can be looked as a top-level view of a staircase where you are at the base of staircase. Each element can be uniquely identified by their index in the array (in a similar way as you could identify your friends by the step on which they were on in the above example).

Defining an Array: Array definition are similar to defining any other variable. There are two things need to be kept in mind, **data type of the array elements** and the **size** of the array. The size of the array is fixed and the memory for an array needs to be allocated before use, size of an array cannot be increased or decreased dynamically.

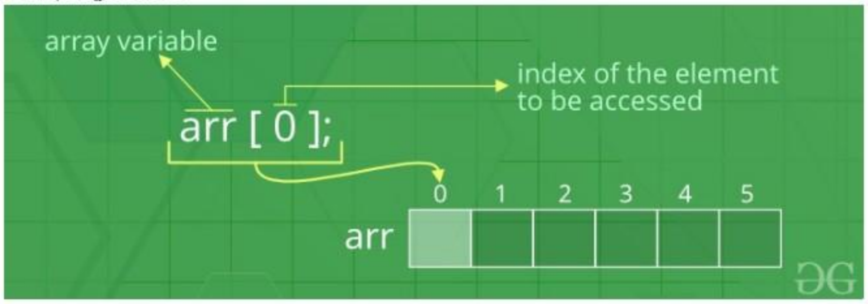
Generally, arrays are declared as:

```
dataType arrayName[arraySize];

An array is distinguished from a normal variable
by brackets [ and ].
```

Accessing array elements: Arrays allows to access elements randomly. Elements in an array can be accessed using indexes. Suppose an array named **arr** stores N elements. Indexes in an array are in the range of **0 to N-1**, where the first element is present at 0-th index and consecutive elements are placed at consecutive indexes. Element present at ith index in the array **arr[]** can be accessed as arr[i].

Below image shows an array **arr[]** of size 5:



Advantages of using arrays:

- Arrays allow random access of elements. This makes accessing elements by position faster.
- Arrays have better cache locality that can make a pretty big difference in performance.

Examples:

```
// A character array in C/C++/Java

char arr1[] = {'g', 'e', 'e', 'k', 's'};



// An Integer array in C/C++/Java

int arr2[] = {10, 20, 30, 40, 50};



// Item at i'th index in array is typically accessed

// as "arr[i]".  For example arr1[0] gives us 'g'

// and arr2[3] gives us 40.
```

Searching in an Array

Searching an element in an array means to check if a given element is present in an array or not. This can be done by accessing elements of the array one by one starting from the first element and checking if any of the element matches with the given element.

We can use loops to perform the above operation of array traversal and access the elements using indexes.

Suppose the array is named **arr[]** with size **N** and the element to be searched is referred as **key**. Below is the algorithm to perform to the search operation in the given array.

```
for(i = 0; i < N; i++)
{
    if(arr[i] == key)
    {
        print "Element Found";
    }
    else
    {
        print "Element not Found";
    }
}
```

Time Complexity of this search operation will be O(N) in the worst case as we are checking every element of the array from 1st to last, so the number of operations is N.