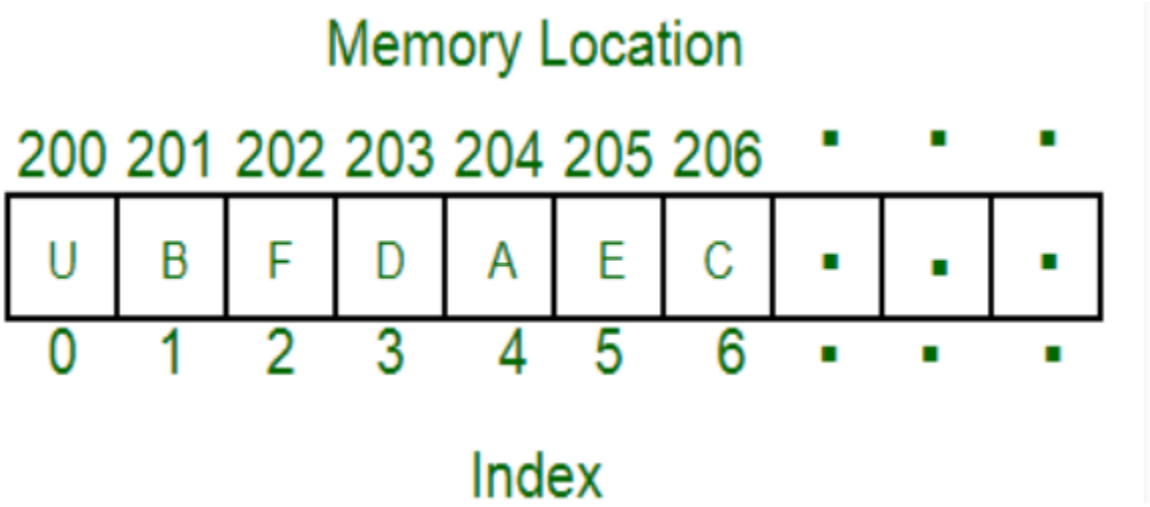
ARRAY AND STRING

An array is a collection of items stored at contiguous memory locations. The idea is to store multiple items of the same type together. 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). The base value is index 0 and the difference between the two indexes is the offset.

For simplicity, we can think of an array as 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".



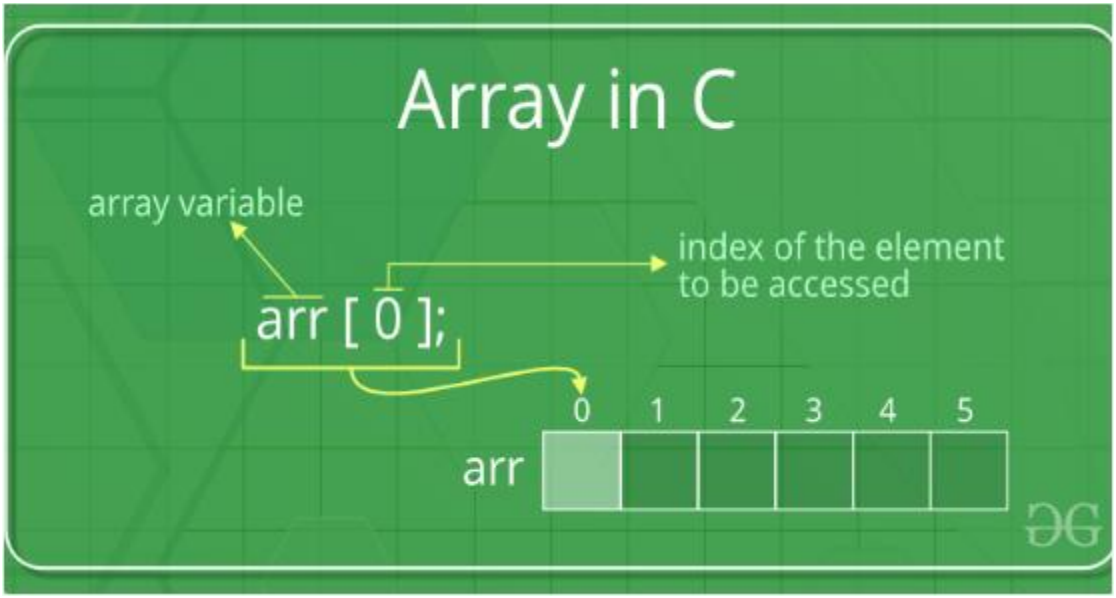
The above image can be looked at as a top-level view of a staircase where you are at the base of the staircase. Each element can be uniquely identified by its index in the array (in a similar way as you could identify your friends by the step on which they were on in the above example).

Array's size

In C language, array has a fixed size meaning once the size is given to it, it cannot be changed i.e. you can't shrink it neither can you expand it. The reason was that for expanding, if we change the size we can't be sure (it's not possible every time) that we get the next memory location to us as free. The shrinking will not work because the array, when declared, gets memory statically allocated, and thus compiler is the only one can destroy it.

Types of indexing in an array:

- 0 (zero-based indexing): The first element of the array is indexed by a subscript of 0.
- 1 (one-based indexing): The first element of the array is indexed by the subscript of 1.
- n (n-based indexing): The base index of an array can be freely chosen. Usually, programming languages allowing n-based indexing also allow negative index values, and other scalar data types like enumerations, or characters may be used as an array index.




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

int main()
{
   // Creating an integer array
   // named arr of size 10.
   int arr[10];
   // accessing element at 0 index
   // and setting its value to 5.
   arr[0] = 5;
   // access and print value at 0
   // index we get the output as 5.
   cout << arr[0];
   return 0;
}
```

Output

5

Here the value 5 is printed because the first element has index zero and at zeroth index we already assigned the value 5.

One Dimensional Array in C:

One dimensional array is an array that has only one subscript specification that is needed to specify a particular element of an array. A one-dimensional array is a structured collection of components (often called array elements) that can be accessed individually by specifying the position of a component with a single index value.

Rules for Declaring One Dimensional Array

1. An array variable must be declared before being used in a program.
2. The declaration must have a data type(int, float, char, double, etc.), variable name, and subscript.

3. The subscript represents the size of the array. If the size is declared as 10, programmers can store 10 elements.
4. An array index always starts from 0. For example, if an array variable is declared as s[10], then it ranges from 0 to 9.
5. Each array element is stored in a separate memory location.

Initialization of One-Dimensional Array in C

An array can be initialized at the following states:

1. At compiling time (static initialization)
2. Dynamic Initialization

Compiling time initialization:

The compile-time initialization means the array of the elements is initialized at the time the program is written or array declaration.

Syntax: data_type array_name [array_size]=(list of elements of an array);

Example: int n[5]={0, 1, 2, 3, 4};

Program:

```
#include<stdio.h>

int main()

{

   int n[5]={0, 1, 2, 3, 4};

   printf("%d", n[0]);

   printf("%d", n[1]);

   printf("%d", n[2]);

   printf("%d", n[3]);

   printf("%d", n[4]);

}
```

Output: 0 1 2 3 4

Introduction and Example of String

Strings can be passed to a function in a similar way as arrays. Learn more about passing arrays to a function.

Example : Passing string to a Function

```
#include <stdio.h>
void displayString(char str[]);
int main()
{
   char str[50];
   printf("Enter string: ");
   fgets(str, sizeof(str), stdin);
   displayString(str);    // Passing string to a function.
   return 0;
}
void displayString(char str[])
{
   printf("String Output: ");
   puts(str);
}
```

Strings and Pointers

Similar like arrays, string names are "decayed" to pointers. Hence, you can use pointers to manipulate elements of the string. We recommended you to check C Arrays and Pointers before you check this example.

Example : Strings and Pointers

```
#include <stdio.h>
int main(void) {
  char name[] = "Harry Potter";
  printf("%c", *name);    // Output: H
  printf("%c", *(name+1));   // Output: a
  printf("%c", *(name+7));   // Output: o
```

```
  char *namePtr;
  namePtr = name;
  printf("%c", *namePtr);     // Output: H
  printf("%c", *(namePtr+1));   // Output: a
  printf("%c", *(namePtr+7));   // Output: o
}
```