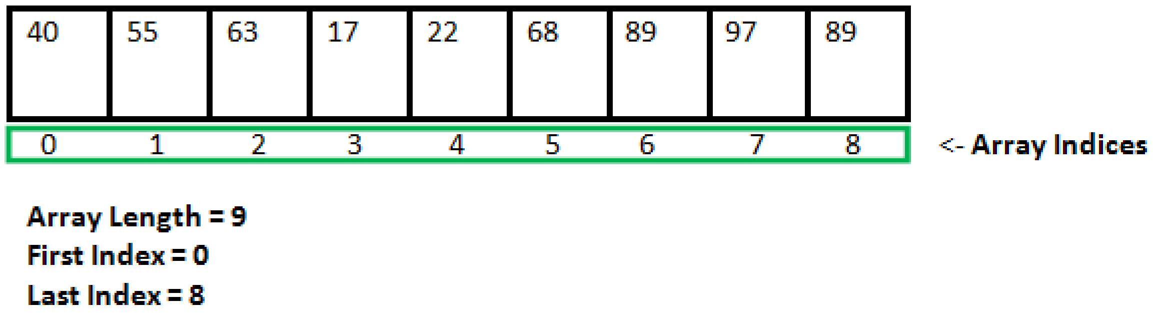
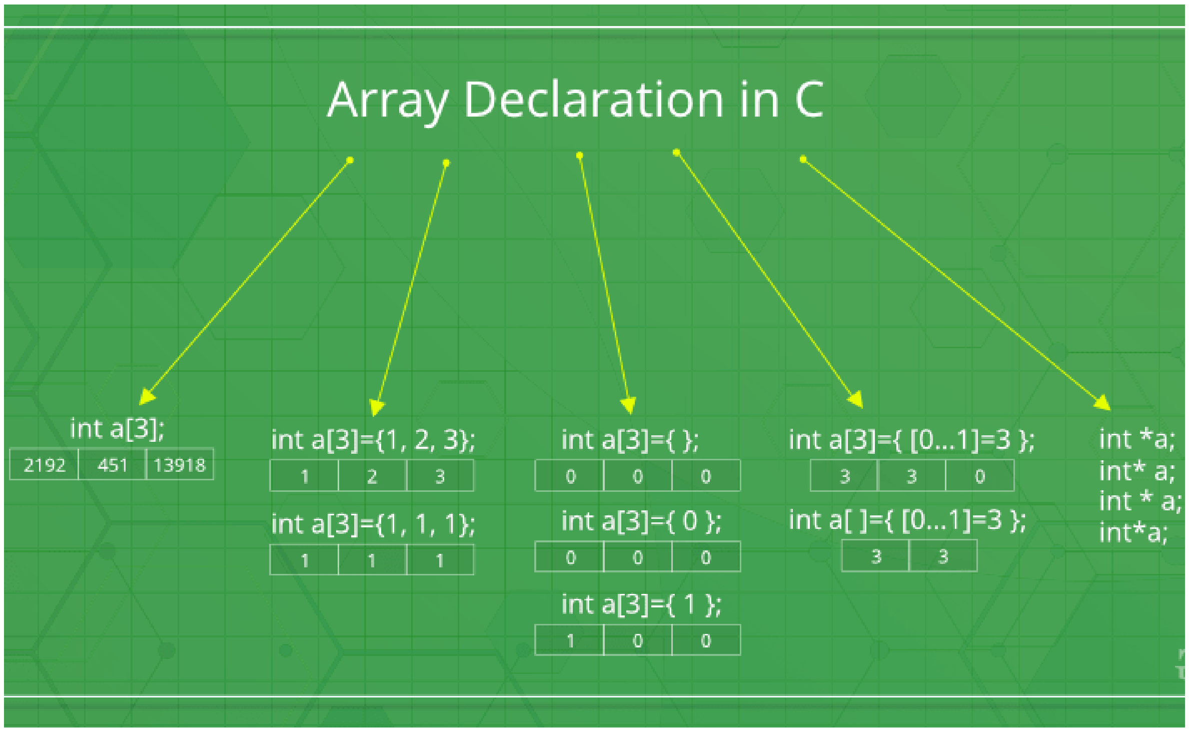
Arrays in C

An array in C or C++ is a collection of items stored at contiguous memory locations and elements can be accessed randomly using indices of an array. They are used to store similar type of elements as in the data type must be the same for all elements. They can be used to store collection of primitive data types such as int, float, double, char, etc of any particular type. To add to it, an array in C or C++ can store derived data types such as the structures, pointers etc. Given below is the picturesque representation of an array.



Why do we need arrays?
We can use normal variables (v1, v2, v3, ..) when we have a small number of objects, but if we want to store a large number of instances, it becomes difficult to manage them with normal variables. The idea of an array is to represent many instances in one variable.
Array declaration in C/C++:



There are various ways in which we can declare an array. It can be done by specifying its type and size, by initializing it or both.

1. **Array declaration by specifying size**
```
// Array declaration by specifying size
int arr1[10];

// With recent C/C++ versions, we can also
// declare an array of user specified size
int n = 10;
int arr2[n];
```
2. **Array declaration by initializing elements**
```
// Array declaration by initializing elements
int arr[] = { 10, 20, 30, 40 }

// Compiler creates an array of size 4.
// above is same as  "int arr[4] = {10, 20, 30, 40}"
```
3. **Array declaration by specifying size and initializing elements**
```
// Array declaration by specifying size and initializing
// elements
int arr[6] = { 10, 20, 30, 40 }

// Compiler creates an array of size 6, initializes first
// 4 elements as specified by user and rest two elements as 0.
// above is same as  "int arr[] = {10, 20, 30, 40, 0, 0}"
```

Advantages of an Array in C/C++:

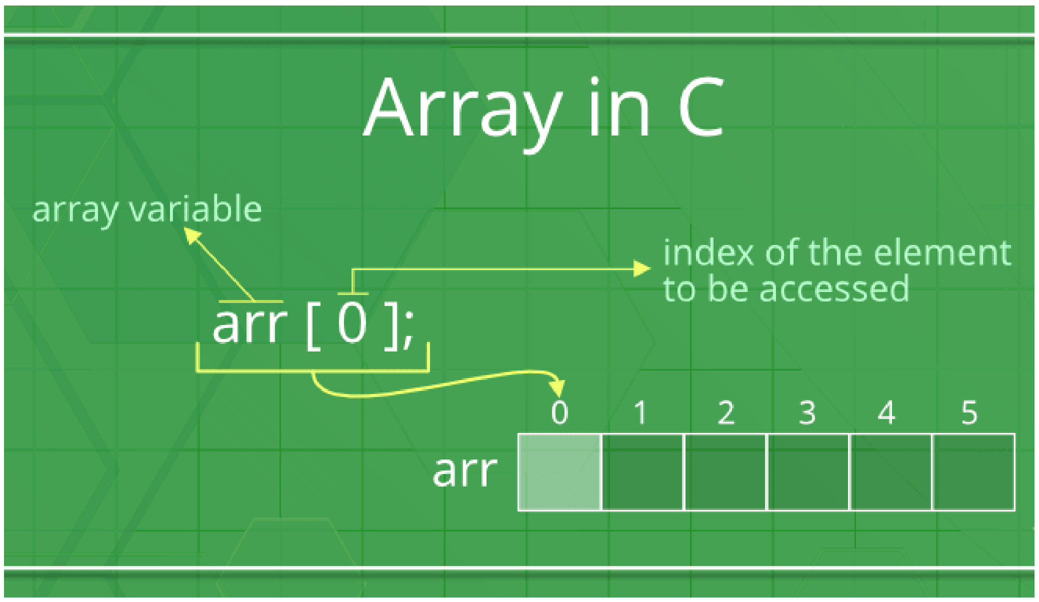
1. Random access of elements using array index.
2. Use of less line of code as it creates a single array of multiple elements.
3. Easy access to all the elements.
4. Traversal through the array becomes easy using a single loop.
5. Sorting becomes easy as it can be accomplished by writing less line of code.

Disadvantages of an Array in C/C++:

1. Allows a fixed number of elements to be entered which is decided at the time of declaration. Unlike a linked list, an array in C is not dynamic.
2. Insertion and deletion of elements can be costly since the elements are needed to be managed in accordance with the new memory allocation.

Facts about Array in C/C++:

- **Accessing Array Elements:**
  Array elements are accessed by using an integer index. Array index starts with 0 and goes till size of array minus 1.



-

Example:

```
#include <stdio.h>

int main()
{
    int arr[5];
    arr[0] = 5;
    arr[2] = -10;
    arr[3 / 2] = 2; // this is same as
arr[1] = 2
    arr[3] = arr[0];

    printf("%d %d %d %d", arr[0],
arr[1], arr[2], arr[3]);

    return 0;
}
```

Output:

```
5 2 -10 5
```

- **No Index Out of bound Checking:**
  There is no index out of bounds checking in C/C++, for example, the following program compiles fine but may produce unexpected output when run.

```
// This C program compiles fine
// as index out of bound
// is not checked in C.

#include <stdio.h>

int main()
{
    int arr[2];

    printf("%d ", arr[3]);
    printf("%d ", arr[-2]);
    return 0;
}
```

Output:

```
2008101287 4195777
```

- In C, it is not compiler error to initialize an array with more elements than the specified size. For example, the below program compiles fine and shows just Warning.

```
#include <stdio.h>
int main()
{

    // Array declaration by initializing it with more
    // elements than specified size.
    int arr[2] = { 10, 20, 30, 40, 50 };

    return 0;
}
```

Warnings:

```
prog.c: In function 'main':

prog.c:7:25: warning: excess elements in array initializer

  int arr[2] = { 10, 20, 30, 40, 50 };

                         ^

prog.c:7:25: note: (near initialization for 'arr')

prog.c:7:29: warning: excess elements in array initializer

  int arr[2] = { 10, 20, 30, 40, 50 };

                             ^

prog.c:7:29: note: (near initialization for 'arr')

prog.c:7:33: warning: excess elements in array initializer

  int arr[2] = { 10, 20, 30, 40, 50 };

                                 ^

prog.c:7:33: note: (near initialization for 'arr')
```

Note: The program won't compile in C++. If we save the above program as a .cpp, the program generates compiler error *"error: too many initializers for 'int [2]'"*.

- **The elements are stored at contiguous memory locations**
  Example:

```
// C program to demonstrate that array
elements are stored
// contiguous locations
```

```
#include <stdio.h>
int main()
{
    // an array of 10 integers.  If
arr[0] is stored at
    // address x, then arr[1] is stored
at x + sizeof(int)
    // arr[2] is stored at x +
sizeof(int) + sizeof(int)
    // and so on.
    int arr[5], i;

    printf("Size of integer in this
compiler is %lu\n", sizeof(int));

    for (i = 0; i < 5; i++)
        // The use of '&' before a
variable name, yields
        // address of variable.
        printf("Address arr[%d] is %p\n",
i, &arr[i]);

    return 0;
}
```

Output:

```
Size of integer in this compiler is 4

Address arr[0] is 0x7ffd636b4260

Address arr[1] is 0x7ffd636b4264

Address arr[2] is 0x7ffd636b4268

Address arr[3] is 0x7ffd636b426c

Address arr[4] is 0x7ffd636b4270
```

Array vs Pointers

Arrays and pointer are two different things (we can check by applying sizeof). The confusion happens because array name indicates the address of first element and arrays are always passed as pointers (even if we use square bracket). Please see Difference between pointer and array in C? for more details.