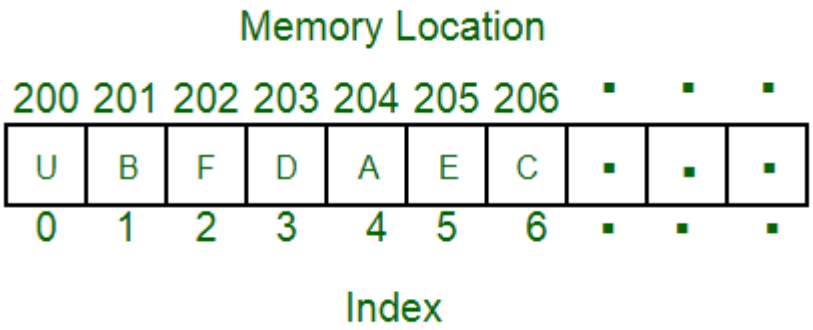
What is an Array?

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). 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 flight 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".

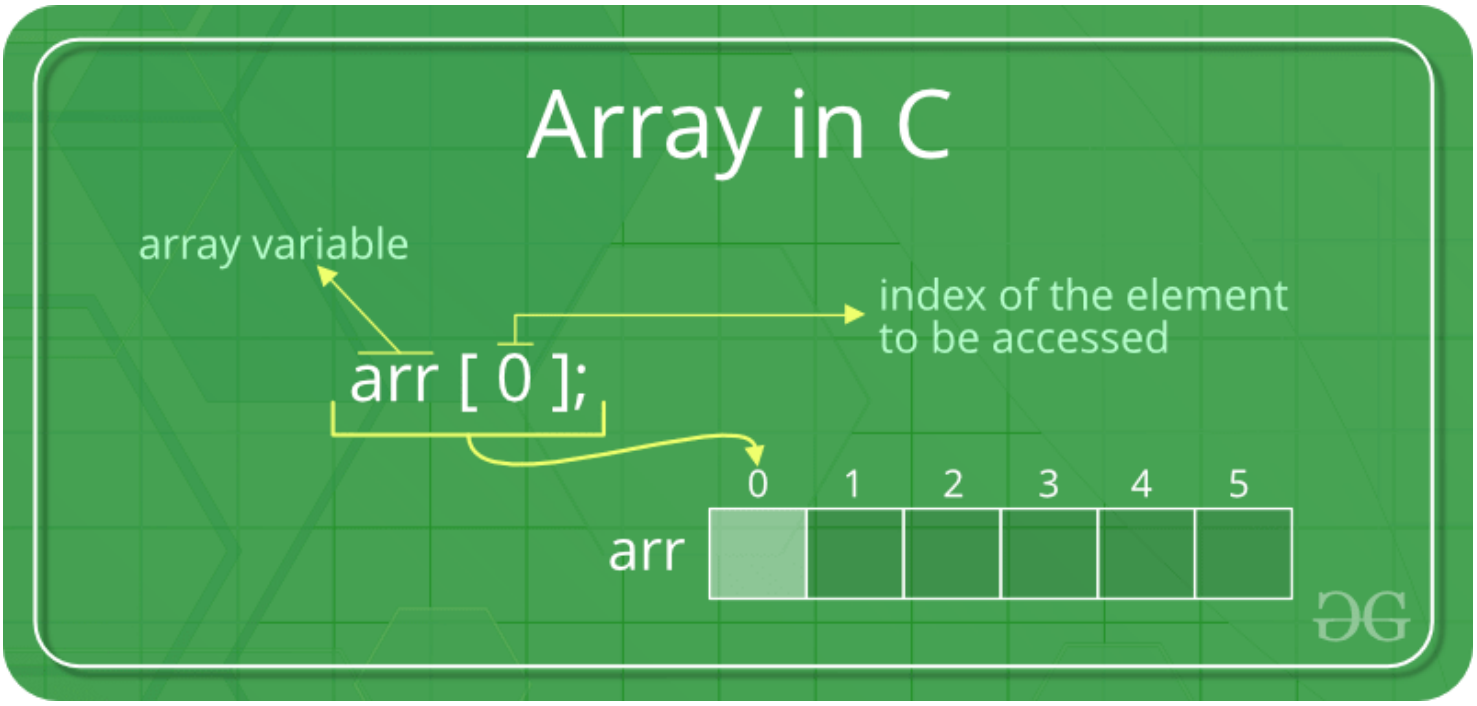


Is the array always of a fixed size?

In C language, the array has a fixed size meaning once the size is given to it, it cannot be changed i.e. you can't shrink it nor 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 for free. The shrinking will not work because the array, when declared, gets memory statically allocated, and thus compiler is the only one that 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.



How an Array is initialized?

By default the array is uninitialized, and no elements of the array are set to any value. However, for the proper working of the array, array initialization becomes important. Array initialization can be done by the following methods:

1. Passing no value within the initializer: One can initialize the array by defining the size of the array and passing no values within the initializer.

Syntax:

```
int arr[ 5 ] = {  };
```

2. By passing specific values within the initializer: One can initialize the array by defining the size of the array and passing specific values within the initializer.

Syntax:

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

Note: The count of elements within the "{ }", must be less than or equals to the size of the array
If the count of elements within the "{ }" is less than the size of the array, the remaining positions are considered to be '0'.

Syntax:

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

3. By passing specific values within the initializer but not declaring the size: One can initialize the array by passing specific values within the initializer and not particularly mentioning the size, the size is interpreted by the compiler.

Syntax:

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

4. Universal Initialization: After the adoption of **universal initialization** in C++, one can avoid using the equals sign between the declaration and the initializer.

Syntax:

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

To know more about array initialization, click here.

What are the different operations on the array?

Arrays allow random access to elements. This makes accessing elements by position faster. Hence operation like searching, insertion, and access becomes really efficient. Array elements can be accessed using the loops.

1. Insertion in Array:

We try to insert a value to a particular array index position, as the array provides random access it can be done easily using the assignment operator.

Pseudo Code:

// to insert a value= 10 at index position 2;

arr[2] = 10;

Time Complexity:

- O(1) to insert a single element
- O(N) to insert all the array elements [w

Here is the code for working in an array:

```
#include <iostream>

using namespace std;

int main()
{
    // initialize an integer array with three elements, all
    // of which are initially set to 0
    int arr[3] = { 0, 0, 0 };

    // set the first element to 1
    arr[0] = 1;

    // set the second element to 2
    arr[1] = 2;

    // set the third element to 3
    arr[2] = 3;

    // print the contents of the array to the console using
    // a for loop
    for (int i = 0; i < 3; i++) {
        cout << arr[i] << " ";
    }
```

```
    return 0;
}
```
Output
```
1 2 3
```

2. <u>Access elements in Array:</u>

Accessing array elements become extremely important, in order to perform operations on arrays.

Pseudo Code:

// to access array element at index position 2, we simply can write

return arr[2] ;

Time Complexity: O(1)

Here is the code for working in an array:

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

int main()
{

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

    // accessing element at index 2
    cout << arr[2] << endl;
    return 0;
}
```
Output
```
3
```

3. <u>Searching in Array:</u>

We try to find a particular value in the array, in order to do that we need to access all the array elements and look for the particular value.

Pseudo Code:

// searching for value 2 in the array;

Loop from i = 0 to 5:
  check if arr[i] = 2:
    return true;

Time Complexity: O(N), where N is the size of the array.

Here is the code for working with an array:

```
#include <stdio.h>

int main()
{

    int arr[4];
```

```
    arr[0] = 1;
    arr[1] = 2;
    arr[2] = 3;
    arr[3] = 4;
    for (int i = 0; i < 4; i++)
        printf("%d\n", arr[i]);

    return 0;
}
```

Output

```
1
2
3
4
```

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

Types of arrays :

1. One-dimensional array (1-D arrays)
2. Multidimensional array

```
// cpp program to display all elements of an initialised 2D
// array.
#include <iostream>
using namespace std;

int main()
{

    int i, j;
    int matrix[3][2] = { { 4, 5 }, { 34, 67 }, { 8, 9 } };

    // use of nested for loop
    // access rows of the array.
    for (i = 0; i < 3; i++) {
        // access columns of the array.
        for (j = 0; j < 2; j++) {
            cout << "matrix[" << i << "][" << j
                 << "]=" << matrix[i][j] << endl;
        }
    }
    return 0;
}
```

Output

```
matrix[0][0]=4
matrix[0][1]=5
matrix[1][0]=34
matrix[1][1]=67
matrix[2][0]=8
matrix[2][1]=9
```

To learn about the differences between One-dimensional and Multidimensional arrays, click here.

Space complexity: The space complexity of this code is O(1) because we are not creating any new variables or data structures that depend on the input size. We are simply using a constant amount of memory to store the 2D array and the loop variables.

Time complexity: The time complexity of the nested for loop used in this code is O(n^2), where n is the number of rows in the 2D array. This is because we are iterating over all the elements of the array, which requires n x m iterations (where m is the number of columns in the array), resulting in a time complexity of O(n^2).

Advantages of using arrays:

- Arrays allow random access to elements. This makes accessing elements by position faster.
- Arrays have better cache locality which makes a pretty big difference in performance.
- Arrays represent multiple data items of the same type using a single name.

Disadvantages of using arrays:

You can't change the size i.e. once you have declared the array you can't change its size because of static memory allocation. Here Insertion(s) and deletion(s) are difficult as the elements are stored in consecutive memory locations and the shifting operation is costly too.

Now if take an example of the implementation of data structure Stack using array there are some obvious flaws. Let's take the **POP** operation of the stack. The algorithm would go something like this.

1. Check for the stack underflow
2. Decrement the top by 1

What we are doing here is, that the pointer to the topmost element is decremented, which means we are just bounding our view, and actually that element stays there taking up the memory space. If you have an array (as a Stack) of any primitive data type then it might be ok. But in the case of an array of Objects, it would take a lot of memory.

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'
arr2[3] gives us 40
```

Usually, an array of characters is called a 'string', whereas an array of ints or floats is simply called an array.

Applications on Array

- Array stores data elements of the same data type.
- Arrays are used when the size of the data set is known.
- Used in solving matrix problems.

- Applied as a lookup table in computer.
- Databases records are also implemented by the array.
- Helps in implementing sorting algorithm.
- The different variables of the same type can be saved under one name.
- Arrays can be used for CPU scheduling.
- Used to Implement other data structures like Stacks, Queues, Heaps, Hash tables, etc.

Arrays in data structures help solve some high-level problems like the "longest consecutive subsequence" program or some easy tasks like arranging the same things in ascending order.The concept is to collect many objects of the same kind.

What Are Arrays in Data Structures?

An array is a linear data structure that collects elements of the same data type and stores them incontiguous and adjacent memory locations. Arrays work on an index system starting from 0 to (n-1), where n is the size of the array.

It is an array, but there is a reason that arrays came into the picture.

Why Do You Need an Array in Data Structures?

Let's suppose a class consists of ten students, and the class has to publish their results. If you had declared all ten variables individually, it would be challenging to manipulate and maintain thedata.

If more students were to join, it would become more difficult to declare all the variables and keeptrack of it. To overcome this problem, arrays came into the picture.

What Are the Types of Arrays?

There are majorly two types of arrays, they are:

One-Dimensional Arrays:

You can imagine a 1d array as a row, where elements are stored one after another.

Multi-Dimensional Arrays:

These multi-dimensional arrays are again of two types. They are:

Two-Dimensional Arrays:

You can imagine it like a table where each cell contains elements.

Three-Dimensional Arrays:

You can imagine it like a cuboid made up of smaller cuboids where each cuboid can contain anelement.

In this "arrays in data structures" tutorial, you will work around one-dimensional arrays.

How Do You Declare an Array?

Arrays are typically defined with square brackets with the size of the arrays as its argument.Here is the

syntax for arrays:

1D Arrays: int arr[n]; 2D

Arrays: int arr[m][n];

3D Arrays: int arr[m][n][o];

How Do You Initialize an Array?

You can initialize an array in four different ways:

- Method 1:

```
int a[6] = {2, 3, 5, 7, 11, 13};
```

- Method 2:

```
int arr[]= {2, 3, 5, 7, 11};
```

- Method 3:

```
int n;
```

```
scanf(“%d”,&n);

int arr[n];

for(int i=0;i<5;i++)

{

scanf(“%d”,&arr[i]);

}
```

- Method 4:

```
int arr[5];arr[0]=1;

arr[1]=2;

arr[2]=3;

arr[3]=4;

arr[4]=5;
```

How Can You Access Elements of Arrays in Data Structures?

You can access elements with the help of the index at which you stored them. Let's discuss itwith a code:

```
#include<stdio.h> int

main()

{

int a[5] = {2, 3, 5, 7, 11};

printf(“%d\n”,a[0]); // we are accessing

printf(“%d\n”,a[1]);
```

```
printf(“%d\n”,a[2]);

printf(“%d\n”,a[3]);

printf(“%d”,a[4]);

return 0;

}
```

In this “Arrays in Data structures” tutorial, you have seen the basics of array implementation, nowyou will perform operations on arrays.

<

What Operations Can You Perform on an Array?

- Traversal
- Insertion
- Deletion
- Searching
- Sorting

Traversing the Array:

Traversal in an array is a process of visiting each element once.Code:

```
#include<stdio.h> int

main()

{

int a[5] = {2, 3, 5, 7, 11};

for(int  i=0;i<5;i++)

{

//traversing ith element in the array

printf("%d\n",a[i]);

}
```

return 0;

}

Insertion:

Insertion in an array is the process of including one or more elements in an array.Insertion

of an element can be done:

- At the beginning
- At the end and
- At any given index of an array.

At the Beginning:

Code:

#include<stdio.h> int

main()

```
{
   int array[10], n,i, item;
   printf("Enter the size of array: ");
   scanf("%d", &n);
   printf("\nEnter Elements in array: ");
      for(i=0;i<n;i++)
      {
       scanf("%d", &array[i]);
      }
   printf("\n enter the element at the beginning");
   scanf("%d", &item);
   n++;
   for(i=n; i>1; i--)
   {
     array[i-1]=array[i-2];
   }
   array[0]=item;
   printf("resultant array element");
```

```
    for(i=0;i<n;i++)
    {
     printf("\n%d", array[i]);
    }
    getch();return
    0;
}
```

At the End:

Code:

```
#include<stdio.h>
#include<conio.h> int
main()
{
int array[10], i, values; printf("Enter
5 Array Elements: ");for(i=0; i<5;
i++)
     scanf("%d", &array[i]);
printf("\nEnter Element to Insert: ");
```

```
scanf("%d", &values);

array[i] = values;

printf("\nThe New Array is:\n");

for(i=0; i<6; i++)

    printf("%d ", array[i]);

getch();

return 0;

}
```

At a Specific Position:

Code:

```
#include <stdio.h>int

main()

{

 int array[100], pos, size, val;

 printf("Enter size of the array:");

 scanf("%d", &size);

 printf("\nEnter %d elements\n", size);for

 (int i = 0; i < size; i++)
```

```
 scanf("%d", &array[i]);

 printf("Enter the insertion location\n");

 scanf("%d", &pos);

 printf("Enter the value to insert\n");

 scanf("%d", &val);

 for (int i = size - 1; i >= pos - 1; i--)

  array[i+1] = array[i];

 array[pos-1] = val;

 printf("Resultant array is\n");for

 (int i = 0; i <= size; i++)

 printf("%d\n", array[i]);

 return 0;

}
```

Deletion:
Deletion of an element is the process of removing the desired element and re-organizing it.You can also do deletion in different ways:

- At the beginning
- At the end

At the Beginning:

```
#include<stdio.h> int

main()
```

```
{
    int n,array[10];
    printf("enter the size of an array");
    scanf("%d" ,&n);
    printf("enter elements in an array");for(int
    i=0;i<n;i++)
        scanf("%d", &array[i]);
    n--;
    for(int i=0;i<n;i++)
```

```
    array[i]=array[i+1];
    printf("\nafter deletion ");
    for(int i=0;i<n;i++)
    printf("\n%d" , array[i]);
}
```

At the End:

```
#include<stdio.h> int
main()
{
int n,array[10];
printf("enter the size of an array");
scanf("%d" ,&n);
printf("enter elements in an array");
for(int i=0;i<n;i++)
scanf("%d", &array[i]);
printf("\nafter deletion array elements are");for(int
i=0;i<n-1;i++)
printf("\n%d" , array[i]);
```

}

Searching for an Element

The method of searching for a specific value in an array is known as searching.There

are two ways we can search in an array, they are:

- Linear search
- Binary search

Linear Search:

Code:

```
#include <stdio.h>

int linear(int a[], int n, int x)

{
```

```
for (int i = 0; i < n; i++)

if (a[i] == x)

return i;

return -1;

}

int main(void)

{

int a[] = { 2, 3, 4, 10, 40 };

int x = 10;

int n = sizeof(a) / sizeof(a[0]);

// Function call

int result = linear(a, n, x);

if(result == -1)

{

printf("Element is not present in array");

}

else

{

printf("Element found at index: %d", result);
```

```
}

return 0;

}
```

Binary Search:

```
#include <stdio.h>

int binary(int a[], int lt, int rt, int k)

{

if (rt >= lt) {

int mid = lt + (rt - l) / 2;

// check If element is at the middleif

(a[mid] == k)

return mid;

//check if element is at the left side of midif

(a[mid] > x)

return binary(a, lt, mid - 1, k);

// Else element is at the right side of mid

return binary(a, mid + 1, rt, k);

}
```

```
// if element not foundreturn -

1;

}

int main(void)

{

int a[] = { 2, 3, 5, 7, 11 };

int n = sizeof(a) / sizeof(a[0]);int

k = 11;

int res = binary(arr, 0, n - 1, k);

if(res == -1)

{

printf("Element is not found")

}

else

{

printf("Element found at index %d",res);

}

return 0;
```

}

Sorting:

Sorting in an array is the process in which it sorts elements in a user-defined order.Code:

```
#include <stdio.h>

void main()

{

    int temp, size, array[100];

    printf("Enter the size \n");

    scanf("%d", &size); printf("Enter

    the numbers \n");
```

```
    for (int i = 0; i < size; ++i)
    scanf("%d", &array[i]);
    for (int i = 0; i < size; ++i)
    {
      for (int j = i + 1; j < size; ++j)
     {
        if (array[i] > array[j])
     {
            temp = array[i];
            array[i] = array[j];
            array[j] = temp;
       }
     }
    }
  printf("sorted array:\n"); for
   (int i = 0; i < size; ++i)
   printf("%d\n", array[i]);
}
```

What Are the Advantages of Arrays in Data Structures?

- Arrays store multiple elements of the same type with the same name.
- You can randomly access elements in the array using an index number.
- Array memory is predefined, so there is no extra memory loss.
- Arrays avoid memory overflow.
- 2D arrays can efficiently represent the tabular data.

What Are the Disadvantages of Arrays in Data Structures?

- The number of elements in an array should be predefined
- An array is static. It cannot alter its size after declaration.
- Insertion and deletion operation in an array is quite tricky as the array stores elements in continuous form. Allocating excess memory than required may lead to memory wastage.