# Session Three

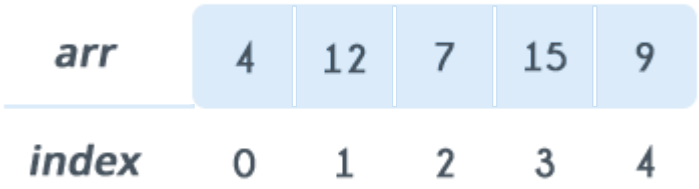
## Agenda

- **Array**
- **multi-dimensional array**
- **String**
- **Built in function (sort, reverse)**
- **Frequency Array**

## What is Array ?

**An array is a sequential collection of elements of same data type and stores data elements in a continuous memory location. The elements of an array are accessed by using an index. The index of an array of size `N` can range from `0` to `N-1`. For example, if your array size is `5` , then your index will range from `0` to `4 (5-1)`. Each element of an array can be accessed by using `arr[index]`.**

**Consider following a rray. The size of this array is`5`. If you want to access `12`, then you can access it by using `arr[ 1 ]` i.e. `12`.**




## Array declaration

**Declaring an array is language-specific.**

**For example, in C/C++, to declare an array, you must specify, the following:**

- **Size of the array: This defines the number of elements that can be stored in the array.**

- **Type of array: This defines the type of each element i.e. number, character, or any other data type.**

## A C++ example would be:

```
    int arr[5];
```

**This is a static array and the other kind is dynamic array, where type is just enough for declaration. In dynamic arrays, size increases as more elements are added to the array.**

**Array Initialization:**

**Array can be initialized either at the time of declaration or after that.**

**The sample format if an array is initialized at the time of declaration is**

```
type arr[size] = {elements}
```

**The sample format of an array that is initialized in C++, is**

```
int arr[5] = {4, 12, 7, 15, 9};
```

**An array can be initialized after declaration by assigning values to each index of the array as follows**

```
 type arr[size]

 arr[index] = 12
```

C++ example:

```
int arr[5];
arr[0] = 4;
arr[1] = 12;
```

## Processing an Array:

The most basic form of processing an array is to loop over the array and print all its elements.

A sample of processing an array by looping over the array and printing its elements is as follows:

C++ example:

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

int main()
{
    // Array declaration and initialization
    int arr[5] = {4, 12, 7, 15, 9};
    // Iterate over the array
    for(int idx=0; idx<5; idx++)
    {
        // Print out each element in a new line
        cout << arr[idx] << endl;
    }
    return 0;
}
```

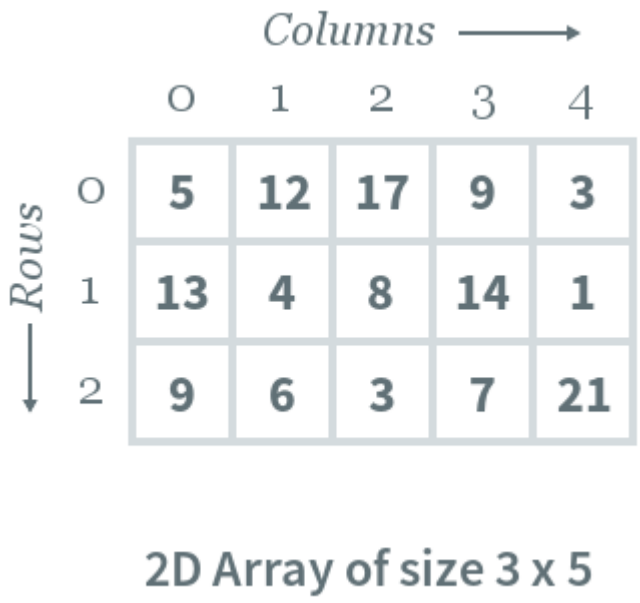
**A practice problem is be a great way to apply these concepts.**

# Multi-dimensional array.

### What is multi-dimensional array ?

A multi-dimensional array is an array of arrays. 2-dimensional arrays are the most commonly used. They are used to store data in a tabular manner.

Consider following 2D array, which is of the size `3 * 5`. For an array of size `N * M`, the rows and columns are numbered from `0` to `N - 1` and columns are numbered from `0` to `M - 1`, respectively. Any element of the array can be accessed by `arr[i][j]` where `0 ≤ i < N` and `0 ≤ j < M`. For example, in the following array, the value stored at `arr[1][3]` is `14`.




2D Array of size 3 x 5

## 2D array declaration:

**To declare a 2D array, you must specify the following:***

**Row-size: Defines the number of rows**

**Column-size: Defines the number of columns**

**Type of array: Defines the type of elements to be stored in the array i.e. either a number, character, or other such datatype. A sample form of declaration is as follows:**

```
type arr[row_size][column_size]
```

**A sample C++ array is declared as follows:**

```
int arr[3][5];
```

## 2D array initialization:

An array can either be initialized during or after declaration. The format of initializing an array during declaration is as follows:

```
type arr[row_size][column_size] = {{elements}, {elements} ... }
```

## An example in C++ is given below:

```
int arr[3][5] = {{5, 12, 17, 9, 3}, {13, 4, 8, 14, 1}, {9, 6, 3, 7, 21}};
```

**Initializing an array after declaration can be done by assigning values to each cell of 2D array, as follows.**

```
type arr[row_size][column_size]
arr[i][j] = 14
```

## A C++ example of initializing an array after declaration by assigning values to each cell of a 2D array is as follows:

```
int arr[3][5];
arr[0][0] = 5;
arr[1][3] = 14;
```

## Processing 2D arrays:

**The most basic form of processing is to loop over the array and print all its elements, which can be done as follows:**

**A C++ example of looping over the array and printing all its elements is as follows:**

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

```
int main()
{
    // Array declaration and initialization
    int arr[3][5] = {{5, 12, 17, 9, 3}, {13, 4, 8, 14, 1}, {9, 6, 3, 7,
21}};
    // Iterate over the array
    for(int i=0; i<3; i++)
    {
        for(int j=0; j<5; j++)
        {
            // Print out each element
            cout << arr[i][j];
        }
        // Print new line character after the row is printed in above loop
        cout << endl;
    }
    return 0;
}
```

# String.

## The C-style character string originated within the C language and continues to be supported within C++. This string is actually a one-dimensional array of characters which is terminated by a null character '\0'. Thus a null-terminated string contains the characters that comprise the string followed by a null.

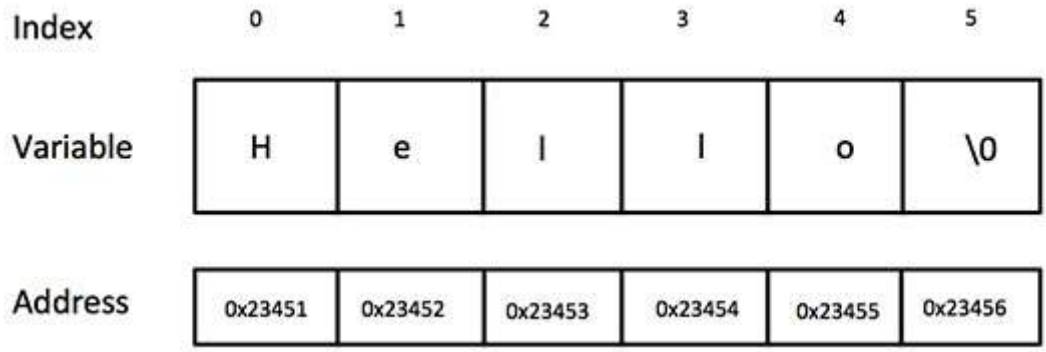
## The following declaration and initialization create a string consisting of the word "Hello". To hold the null character at the end of the array, the size of the character array containing the string is one more than the number of characters in the word "Hello."

```
char greeting[6] = {'H', 'e', 'l', 'l', 'o', '\0'};
```

## If you follow the rule of array initialization, then you can write the above statement as follows −

```
char greeting[] = "Hello";
```

**Following is the memory presentation of above defined string in C/C++ −**




**Actually, you do not place the null character at the end of a string constant. The C++ compiler automatically places the '\0' at the end of the string when it initializes the array. Let us try to print above-mentioned string −**

```
#include <iostream>

using namespace std;

int main () {

   char greeting[6] = {'H', 'e', 'l', 'l', 'o', '\0'};

   cout << "Greeting message: ";
   cout << greeting << endl;

   return 0;
}
```

## When the above code is compiled and executed, it produces the following result −

```
Greeting message: Hello
```

## Following example makes use of few of the above-mentioned functions −

```
#include <iostream>
#include <cstring>

using namespace std;

int main () {

   char str1[10] = "Hello";
   char str2[10] = "World";
   char str3[10];
   int  len ;

   // copy str1 into str3
   strcpy( str3, str1);
   cout << "strcpy( str3, str1) : " << str3 << endl;

   // concatenates str1 and str2
   strcat( str1, str2);
   cout << "strcat( str1, str2): " << str1 << endl;

   // total lenghth of str1 after concatenation
   len = strlen(str1);
   cout << "strlen(str1) : " << len << endl;

   return 0;
}
```

### When the above code is compiled and executed, it produces result something as follows −

```
strcat( str1, str2): HelloWorld
strlen(str1) : 10
```

## The String Class in C++

### The standard C++ library provides a string class type that supports all the operations mentioned above, additionally much more functionality. Let us check the following example −

```
#include <iostream>
#include <string>

using namespace std;

int main () {

   string str1 = "Hello";
   string str2 = "World";
   string str3;
   int  len ;

   // copy str1 into str3
   str3 = str1;
   cout << "str3 : " << str3 << endl;

   // concatenates str1 and str2
   str3 = str1 + str2;
   cout << "str1 + str2 : " << str3 << endl;

   // total length of str3 after concatenation
   len = str3.size();
   cout << "str3.size() :  " << len << endl;

   return 0;
}
```

## When the above code is compiled and executed, it produces result something as follows –

```
str1 + str2 : HelloWorld
str3.size() :  10
```

## Sort.

### By default, the sort() function sorts the elements in ascending order.

#### Below is a simple program to show the working of sort().

Line

```
#include <bits/stdc++.h>
```

---

```
#include <bits/stdc++.h>
using namespace std;

int main()
{
    int arr[] = { 1, 5, 8, 9, 6, 7, 3, 4, 2, 0 };
    int n = sizeof(arr) / sizeof(arr[0]);

    /*Here we take two parameters, the beginning of the
    array and the length n upto which we want the array to
    be sorted*/
    sort(arr, arr + n);

    cout << "\nArray after sorting using "
            "default sort is : \n";
    for (int i = 0; i < n; ++i)
        cout << arr[i] << " ";

    return 0;
}
```

Output

```
Array after sorting using default sort is :
0 1 2 3 4 5 6 7 8 9
```

## Using reverse() function in C++

The built-in reverse function reverse() in C++ directly reverses a string. Given that both bidirectional begin and end iterators are passed as arguments.

This function is defined in the algorithm header file. The code given below describes the use of reverse() function,

```
#include <algorithm>
#include<iostream>
#include<string>
using namespace std;
int main()
{
    string str = "Journal Dev reverse example";
    reverse(str.begin(), str.end());
    cout<<"\n"<<str;
    return 0;
}
```