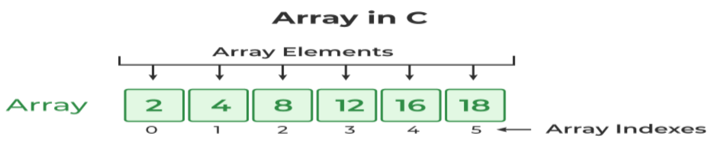
Arrays and Strings

What is array: An array is defined as the collection of similar type of data items stored at contiguous memory locations. Arrays are the derived data type in C programming language which can store the primitive type of data such as int, char, double, float, etc. It also has the capability to store the collection of derived data types, such as pointers, structure, etc. The array is the simplest data structure where each data element can be randomly accessed by using its index number.

Array indexing:An array is a fixed-size collection of similar data items stored in contiguous memory locations. It can be used to store the collection of primitive data types such as int, char, float, etc., and also derived and user-defined data types such as pointers, structures, etc.

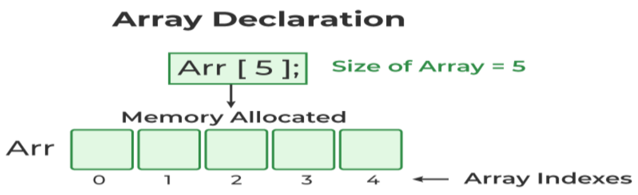


Array Declaration

we have to declare the array like any other variable before using it. We can declare an array by specifying its name, the type of its elements, and the size of its dimensions. When we declare an array in C, the compiler allocates the memory block of the specified size to the array name.

Syntax of Array Declaration

data_type array_name [size];
or
data_type array_name [*size1*] [*size2*]...[*sizeN*];



Example of Array Declaration

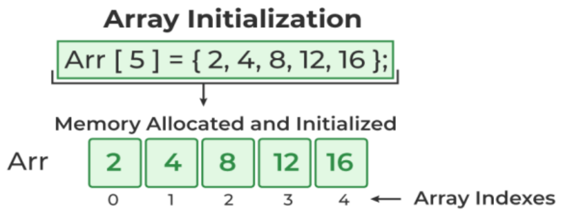
```
#include <stdio.h>
 int main()
{
    int arr_int[5];
    char arr_char[5];
   return 0;
}
```

Array Initialization

Initialization in C is the process to assign some initial value to the variable. When the array is declared or allocated memory, the elements of the array contain some garbage value. So, we need to initialize the array to some meaningful value. There are multiple ways in which we can initialize an array in C.

1. Array Initialization with Declaration

In this method, we initialize the array along with its declaration. We use an initializer list to initialize multiple elements of the array. An initializer list is the list of values enclosed within braces **{ }** separated b a comma.
Syntax:
data_type array_name [*size*] = {value1, value2, ... valueN};


Array Initialization
Arr [5] = { 2, 4, 8, 12, 16 };
Memory Allocated and Initialized
Arr
2
4
8
12
16
0
1
2
3
4
Array Indexes

2. Array Initialization with Declaration without Size

If we initialize an array using an initializer list, we can skip declaring the size of the array as the compiler can automatically deduce the size of the array in these cases. The size of the array in these cases is equal to the number of elements present in the initializer list as the compiler can automatically deduce the size of the array.

data_type array_name[] = {1,2,3,4,5};

The size of the above arrays is 5 which is automatically deduced by the compiler.

3. Array Initialization after Declaration (Using Loops)

We initialize the array after the declaration by assigning the initial value to each element individually. We can use for loop, while loop, or do-while loop to assign the value to each element of the array.

```
for (int i = 0; i < N; i++) {
array_name[i] = valuei;
}
```

Example of Array Initialization

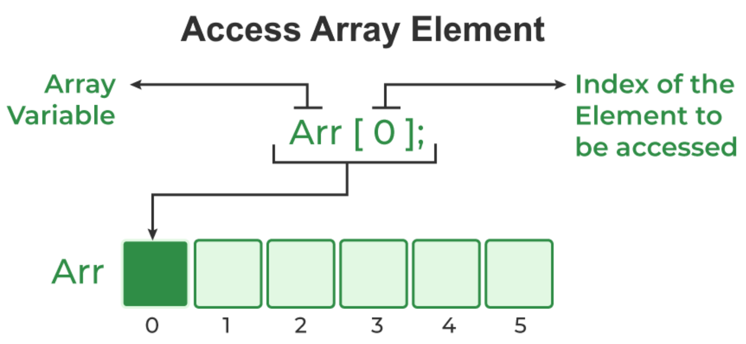
```
#include <stdio.h>

int main()
{
    int arr[5] = { 10, 20, 30, 40, 50 };
    int arr1[] = { 1, 2, 3, 4, 5 };
    float arr2[5];
    for (int i = 0; i < 5; i++) {
        arr2[i] = (float)i * 2.1;
    }
    return 0;
}
```

Access Array Elements

We can access any element of an array in C using the array subscript operator **[ ]** and the index value ***i*** of the element.

array_name [*index*];

One thing to note is that the indexing in the array always starts with 0, i.e., the **first element** is at index **0** and the **last element** is at **N – 1** where **N** is the number of elements in the array.



Example of Accessing Array Elements using Array Subscript Operator

```
#include <stdio.h>
 int main()
{
    int arr[5] = { 15, 25, 35, 45, 55 };
    printf("Element at arr[2]: %d\n", arr[2]);
    printf("Element at arr[4]: %d\n", arr[4]);
    printf("Element at arr[0]: %d", arr[0]);
   return 0;
}
```

Output

```
Element at arr[2]: 35

Element at arr[4]: 55

Element at arr[0]: 15
```

Programs with arrays of integers:

Array Input/Output

```
// Program to take 5 values from the user and store them in an array
// Print the elements stored in the array

#include<stdio.h>

intmain(){

int values[5];

printf("Enter 5 integers: ");

// taking input and storing it in an array
for(int i = 0; i <5; ++i) {
scanf("%d", &values[i]);
  }

printf("Displaying integers: ");

// printing elements of an array
for(int i = 0; i <5; ++i) {
printf("\n%d", values[i]);
```

```
  }
return0;
}
```

```
Output:
Enter 5 integers: 10 11 12 13 14
Displaying integers:
10
11
12
13
14
```

Types of arrays

Arrays are broadly classified into three types. They are as follows −

- One – dimensional arrays
- Two – dimensional arrays
- Multi – dimensional arrays

One – dimensional array

The Syntax is as follows −

datatype array name [size]

For example, int a[5]

Initialization

An array can be initialized in two ways, which are as follows −

- Compile time initialization
- Runtime initialization

Example

Following is the C program on compile time initialization −

```
#include<stdio.h>
void main(){
  //Declaring array with compile time initialization//
  int array[5]={1,2,3,4,5};
  //Declaring variables//
  int i;
  //Printing O/p using for loop//
  printf("Displaying array of elements :");
  for(i=0;i<5;i++){
    printf("%d ",array[i]);
  }
}
```

Example

Following is the C program on **runtime initialization** −

```
#include<stdio.h>
main (){
  int a[5],i;
  printf ("enter 5 elements");
  for( i=0; i<5; i++)
    scanf("%d",&a[i]);
  printf("elements of the array are");
  for(i=0; i<5; i++)
    printf("%d", a[i]);
}
```

Output

The output is as follows −

```
enter 5 elements 10 20 30 40 50

elements of the array are : 10 20 30 40 50
```

Two Dimensional Array in C

The two-dimensional array can be defined as an array of arrays. The 2D array is organized as matrices which can be represented as the collection of rows and columns. However, 2D arrays are created to implement a relational database lookalike data structure. It provides ease of holding the bulk of data at once which can be passed to any number of functions wherever required.

Declaration of two dimensional Array

The syntax to declare the 2D array is given below.

1. data_type array_name[rows][columns];

Consider the following example.

1. **int** twodimen[4][3];

Initialization of 2D Array

In the 1D array, we don't need to specify the size of the array if the declaration and initialization are being done simultaneously. However, this will not work with 2D arrays. We will have to define at least the second dimension of the array. The two-dimensional array can be declared and defined in the following way.

1. **int** arr[4][3]={{1,2,3},{2,3,4},{3,4,5},{4,5,6}};

Two-dimensional array example

```
#include<stdio.h>
int main(){
int i=0,j=0;
int arr[4][3]={{1,2,3},{2,3,4},{3,4,5},{4,5,6}};
//traversing 2D array
for(i=0;i<4;i++){
 for(j=0;j<3;j++){
   printf("arr[%d] [%d] = %d \n",i,j,arr[i][j]);
 }//end of j
}//end of i
return 0;
}
```

Output

```
arr[0][0] = 1
arr[0][1] = 2
arr[0][2] = 3
arr[1][0] = 2
arr[1][1] = 3
arr[1][2] = 4
arr[2][0] = 3
arr[2][1] = 4
arr[2][2] = 5
arr[3][0] = 4
arr[3][1] = 5
arr[3][2] = 6
```

Difference Between One-Dimensional and Two-Dimensional Array

Parameters	**One-Dimensional Array**	**Two-Dimensional Array**
Basics	A one-dimensional array stores a single list of various elements having a similar data type.	A two-dimensional array stores an *array of various arrays*, or a *list of various lists*, or an *array of various one-dimensional arrays*.
Representation	It represents multiple data items in the form of a list.	It represents multiple data items in the form of a table that contains columns and rows.
Dimensions	It has only one dimension.	It has a total of two dimensions.

Parameters of Receiving	One can easily receive it in a pointer, an unsized array, or a sized array.	The parameters that receive it must define an array's rightmost dimension.
Total Size (in terms of Bytes)	Total number of Bytes = The size of array x the size of array variable or datatype.	Total number of Bytes = The size of array visible or datatype x the size of second index x the size of the first index.

Introduction to strings:

A String in C programming is a sequence of characters terminated with a null character '\0'. The C String is stored as an array of characters. The difference between a character array and a C string is that the string in C is terminated with a unique character '\0'.

String Declaration Syntax

Declaring a string in C is as simple as declaring a one-dimensional array. Below is the basic syntax for declaring a string.

char *string_name*[*size*];
In the above syntax **string_name** is any name given to the string variable and size is used to define the length of the string, i.e the number of characters strings will store.
There is an extra terminating character which is the ***Null character ('\0') used to indicate the termination of a string that differs strings from normal character arrays***.

C String Initialization

A string in C can be initialized in different ways. We will explain this with the help of an example. Below are the examples to declare a string with the name str and initialize it with "GeeksforGeeks".

We can initialize a C string in 4 different ways which are as follows:

1. Assigning a String Literal without Size

String literals can be assigned without size. Here, the name of the string str acts as a pointer because it is an array.

```
char str[] = "GeeksforGeeks";
```

2. Assigning a String Literal with a Predefined Size

String literals can be assigned with a predefined size. But we should always account for one extra space which will be assigned to the null character. If we want to store a string of size n then we should always declare a string with a size equal to or greater than n+1.

```
char str[50] = "GeeksforGeeks";
```

3. Assigning Character by Character with Size

We can also assign a string character by character. But we should remember to set the end character as '\0' which is a null character.

```
char str[14] = { 'G','e','e','k','s','f','o','r','G','e','e','k','s','\0'};
```

4. Assigning Character by Character without Size

We can assign character by character without size with the NULL character at the end. The size of the string is determined by the compiler automatically.

```
char str[] = { 'G','e','e','k','s','f','o','r','G','e','e','k','s','\0'};
```

Printing Strings

- The conversion type 's' may be used for output of strings using printf().
- Width and precision specifications may be used with the %s conversion specifier.
- The width specifies the minimum output field width; if the string is shorter, then space padding is generated.
- The precision specifies the maximum number of characters to display.
- If the string is too long, it is truncated. A negative width implies left justification of short strings rather than the default right justification.

For example,   printf("%7.3s",name)

- This specifies that only the first three characters have to be printed in a total field width of seven characters and right justified in the allocated width by default.
- We can include a minus sign to make it left justified (%-7.3). The following points should be noted.
- When the field width is less than the length of the string, the entire string is printed.
- The integer value on the right side of the decimal point specifies the number of characters to be printed.
- When the number of characters to be printed is specified as zero, nothing is printed.
- The minus sign in the specification causes the string to be printed as left justified.

PROGRAM

```
 #include <stdio.h>
int main()
{
         char s[]="Hello, World";
        printf(">>%s<<\n",s);
        printf(">>%20s<<\n",s);
        printf(">>%-20s<<\n",s);
        printf(">>%.4s<<\n",s);
        printf(">>%-20.4s<<\n",s);
        printf(">>%20.4s<<\n",s);
        return 0;
```

}

String Input:

Using %s control string with scanf():

 Strings may be read by using the %s conversion with the function scanf() but there are some irksome restrictions.

 The first is that scanf() only recognizes a sequence of characters delimited by white space characters as an external string.

 The second is that it is the programmer's responsibility to ensure that there is enough space to receive and store the incoming string along with the terminating null which is automatically generated and stored by scanf() as part of the %s conversion.

 The associated parameter in the value list must be the address of the first location in an area of memory set aside to store the incoming string.

```
int main()
{
         char str[50];
        printf("Enter a string");
        scanf("%s",str);
        printf("The string was :%s\n",str);
        return 0;
}
```

OUTPUT:

```
Enter a string manas
The string was :manas
```

Using gets():

 The gets() function enables the user to enter some characters followed by the enter key.

 All the characters entered by the user get stored in a character array.

 The null character is added to the array to make it a string.

 The gets() allows the user to enter the space-separated strings.

  It returns the string entered by the user.

PROGRAM:

```
#include<stdio.h>
void main ()
{
char s[30];
printf("Enter the string? ");

gets(s);

printf("You entered %s",s);

}
```

OUTPUT:

```
Enter the string?Hello World

You entered Hello World
```

Example:

```
#include <stdio.h>
#include <string.h>

int main()
{
   // declare and initialize string
   char s1[] = “NECN",rev[],s2[]=”Hai”;

   // print string
   printf("%s\n", s1);

   int length = 0;
   length = strlen(s1);
   rev[]=strrev(str);
   s3=strcat(s2,s1);

   printf("Length of string str is %d", length);
   printf("Reverse of string str is %s", rev);
   printf("Concatenation of string s2 and s1 is is %s", s3);



   return0;
}
```

Output

```
NECN

Length of string str is 4

Reverse of string s1 is  NCEN

Concatenation of string s2 and s1 is  HaiNECN
```