

Unit-5

Array and String

(Marks -11)

CTEVT Diploma in Computer Engineering

Subject : C Programming

Shree Sarwajanik Secondary Technical School

Prepared By: Er. Abinash Adhikari

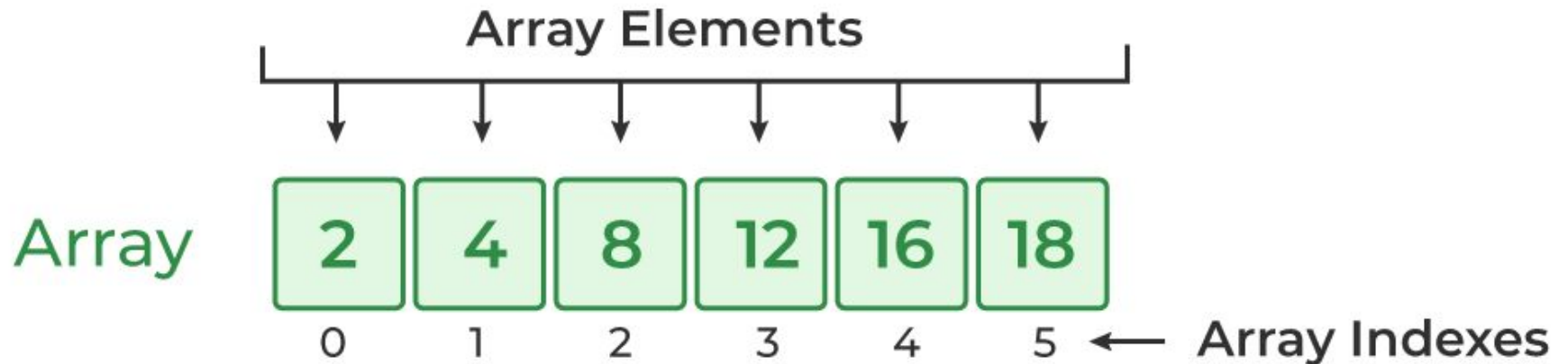
Introduction to Array, Declaration, Initialization

Array in C is one of the most used data structures in C programming. It is a simple and fast way of storing multiple values under a single name. In this article, we will study the different aspects of array in C language such as array declaration, definition, initialization, types of arrays, array syntax, advantages and disadvantages, and many more.

What is Array in C?

- ❖ An array in C 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 in C



C Array Declaration

In C, 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.

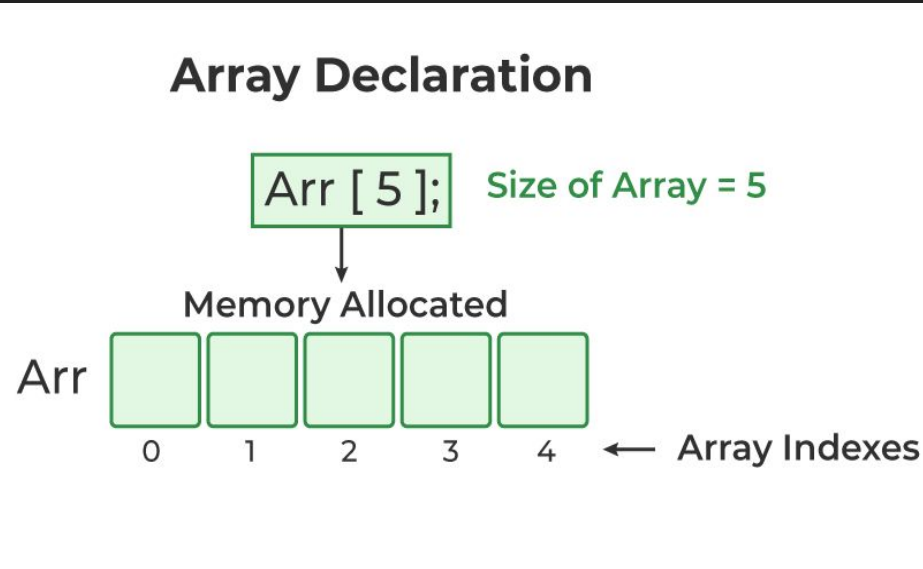
Syntax of Array Declaration

`data_type array_name [size];`

or

`data_type array_name [size1] [size2]...[sizeN];`

where N is the number of dimensions.



Example of Array Declaration



```
// C Program to illustrate the array declaration
#include <stdio.h>

int main()
{
    // declaring array of integers
    int arr_int[5];
    // declaring array of characters
    char arr_char[5];

    return 0;
}
```

C 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 by a comma.

```
data_type array_name [size] = {value1, value2, ... valueN};
```

Array Initialization

```
Arr [ 5 ] = { 2, 4, 8, 12, 16 };
```



Memory Allocated and Initialized

Arr



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;  
}
```




```
// C Program to demonstrate array initialization
#include <stdio.h>

int main()
{
    // array initialization using initializer list
    int arr[5] = { 10, 20, 30, 40, 50 };

    // array initialization using initializer list without
    // specifying size
    int arr1[] = { 1, 2, 3, 4, 5 };

    // array initialization using for loop
    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.

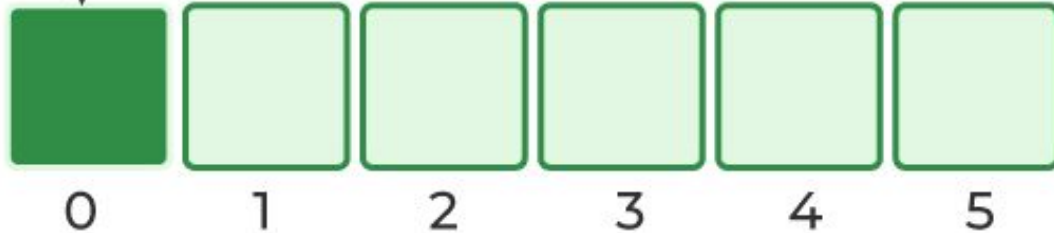
Access Array Element

Array
Variable

Index of the
Element to
be accessed

`Arr [0] ;`

Arr



Example of Accessing Array Elements using Array Subscript Operator



```
// C Program to illustrate element access using array
// subscript
#include <stdio.h>
int main()
{
    // array declaration and initialization
    int arr[5] = { 15, 25, 35, 45, 55 };
    // accessing element at index 2 i.e 3rd element
    printf("Element at arr[2]: %d\n", arr[2]);
    // accessing element at index 4 i.e last element
    printf("Element at arr[4]: %d\n", arr[4]);
    // accessing element at index 0 i.e first element
    printf("Element at arr[0]: %d", arr[0]);
    return 0;
}
```

Update Array Element

We can update the value of an element at the given index i in a similar way to accessing an element by using the array subscript operator `[]` and assignment operator `=`.

```
array_name[i] = new_value;
```

C Array Traversal

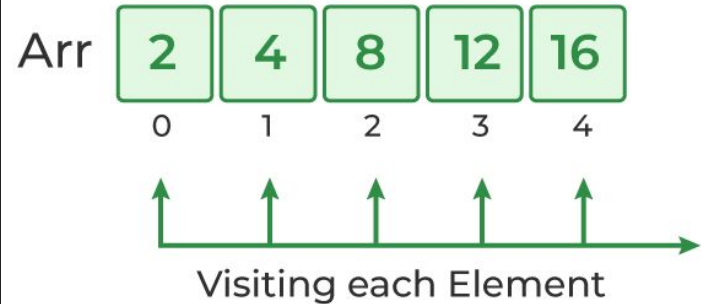
Traversal is the process in which we visit every element of the data structure. For C array traversal, we use loops to iterate through each element of the array.

Array Traversal using for Loop

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

Array Transversal

```
for ( int i = 0; i < Size; i++){  
    arr[i];  
}
```



Types of Array in C

There are two types of arrays based on the number of dimensions it has. They are as follows:

1. One Dimensional Arrays (1D Array)
2. Multidimensional Arrays

1. One Dimensional Array in C

The One-dimensional arrays, also known as 1-D arrays in C are those arrays that have only one dimension.

Syntax of 1D Array in C

```
array_name [size];
```

1D Array





```
// C Program to illustrate the use of 1D array
#include <stdio.h>
int main()
{
    // 1d array declaration
    int arr[5];
    // 1d array initialization using for loop
    for (int i = 0; i < 5; i++) {
        arr[i] = i * i - 2 * i + 1;
    }
    printf("Elements of Array: ");
    // printing 1d array by traversing using for loop
    for (int i = 0; i < 5; i++) {
        printf("%d ", arr[i]);
    }
    return 0;
}
```

2. Multidimensional Array in C

Multi-dimensional Arrays in C are those arrays that have more than one dimension. Some of the popular multidimensional arrays are 2D arrays and 3D arrays. We can declare arrays with more dimensions than 3d arrays but they are avoided as they get very complex and occupy a large amount of space.

A. Two-Dimensional Array in C

A Two-Dimensional array or 2D array in C is an array that has exactly two dimensions. They can be visualized in the form of rows and columns organized in a two-dimensional plane.

Syntax of 2D Array in C

```
array_name[size1] [size2];
```

size1: Size of the first dimension.

size2: Size of the second dimension.

2D Array

1	2	3	4
1	2	3	4
1	2	3	4
1	2	3	4



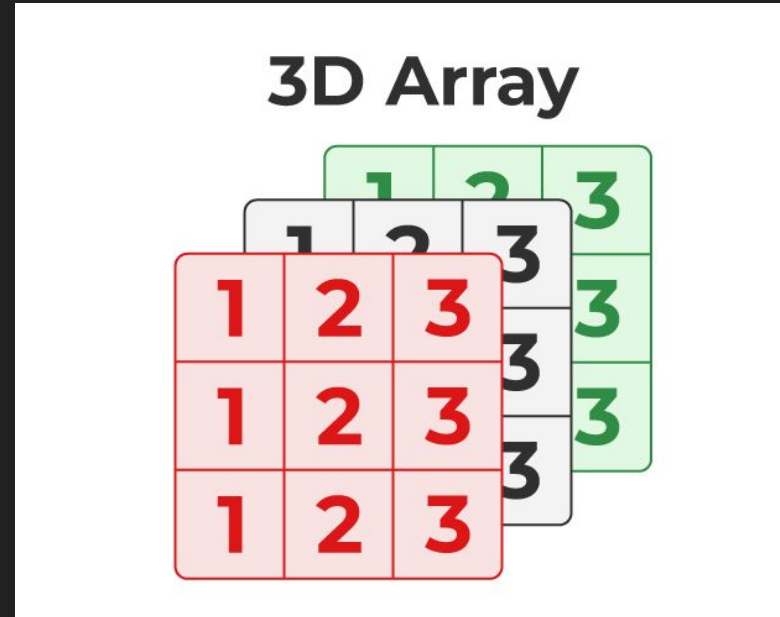
```
// C Program to illustrate 2d array
#include <stdio.h>
int main()
{
    // declaring and initializing 2d array
    int arr[2][3] = { 10, 20, 30, 40, 50, 60 };
    printf("2D Array:\n");
    // printing 2d array
    for (int i = 0; i < 2; i++) {
        for (int j = 0; j < 3; j++) {
            printf("%d ", arr[i][j]);
        }
        printf("\n");
    }
    return 0;
}
```

B. Three-Dimensional Array in C

Another popular form of a multi-dimensional array is Three Dimensional Array or 3D Array. A 3D array has exactly three dimensions. It can be visualized as a collection of 2D arrays stacked on top of each other to create the third dimension.

Syntax of 3D Array in C

```
array_name [size1] [size2] [size3];
```



```
#include <stdio.h>

int main() {
    // Declaration and Initialization of a 3D array
    int cube[2][3][4] = {
        { {1, 2, 3, 4}, {5, 6, 7, 8}, {9, 10, 11, 12} },
        { {13, 14, 15, 16}, {17, 18, 19, 20}, {21, 22, 23, 24} }
    };

    // Access and Print Elements of the 3D Array
    printf("Element at cube[0][0][0]: %d\n", cube[0][0][0]); // Output: 1
    printf("Element at cube[1][2][3]: %d\n", cube[1][2][3]); // Output: 24

    // Display All Elements in the 3D Array
    printf("All elements in the 3D array:\n");
    for (int i = 0; i < 2; i++) { // Loop for the first dimension
        for (int j = 0; j < 3; j++) { // Loop for the second dimension
            for (int k = 0; k < 4; k++) { // Loop for the third dimension
                printf("cube[%d][%d][%d] = %d\n", i, j, k, cube[i][j][k]);
            }
        }
    }
    return 0;
}
```

Properties of Arrays in C

1. Fixed Size

- The array in C is a fixed-size collection of elements. The size of the array must be known at the compile time and it cannot be changed once it is declared.

2. Homogeneous Collection

- We can only store one type of element in an array. There is no restriction on the number of elements but the type of all of these elements must be the same.

3. Indexing in Array

- The array index always starts with 0 in C language. It means that the index of the first element of the array will be 0 and the last element will be $N - 1$.

4. Dimensions of an Array

- A dimension of an array is the number of indexes required to refer to an element in the array. It is the number of directions in which you can grow the array size.

5. Contiguous Storage

- All the elements in the array are stored continuously one after another in the memory. It is one of the defining properties of the array in C which is also the reason why random access is possible in the array.

6. Random Access

- The array in C provides random access to its element i.e we can get to a random element at any index of the array in constant time complexity just by using its index number.

7. No Index Out of Bounds 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.

String, Array of String

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 in C

`char str[] = "Geeks"`

index → 0 1 2 3 4

str →

G	e	e	k	s	
---	---	---	---	---	--

Address →

--	--	--	--	--	--

C 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 "Engineering".

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[] = "Abinash";`

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] = "Abinash";`

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[8] = {'A','b','i','n','a','s','h','\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[] = {'A','b','i','n','a','s','h','\0'};`

Standard C Library – String.h Functions

string.h is a standard header file in the C language that contains functions for manipulating strings (arrays of characters). <string.h> header file contains some useful string functions that can be directly used in a program by invoking the #include preprocessor directive.

Syntax:

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

S.N.	Function	Description	Example
1	strlen()	Returns the length of a string, excluding the null character (\0).	<pre>char str[] = "Hello"; printf("%lu", strlen(str)); // Output: 5</pre>
2	strrev()	Reverses the given string. Note: Not part of the standard C library but available in some compilers.	<pre>char str[] = "World"; printf("%s", strrev(str)); // Output: dlroW</pre>
3	strupr()	Converts all characters of a string to uppercase. Note: Not part of the standard C library; platform-specific.	<pre>char str[] = "hello"; printf("%s", strupr(str)); // Output: HELLO</pre>
4	strlwr()	Converts all characters of a string to lowercase. Note: Platform-specific function.	<pre>char str[] = "HELLO"; printf("%s", strlwr(str)); // Output: hello</pre>
5	strcpy()	Copies the contents of one string into another.	<pre>char src[] = "C Programming"; char dest[20]; strcpy(, src); printf("%s", dest); // Output: C Programming</pre>
6	strcat()	Concatenates (appends) one string to the end of another.	<pre>char str1[20] = "Hello "; char str2[] = "World"; strcat(str1, str2); printf("%s", str1); // Output: Hello World</pre>
7	strcmp()	Compares two strings lexicographically and returns: 0 if equal, <0 if str1 < str2, >0 if str1 > str2.	<pre>char str1[] = "Apple"; char str2[] = "Banana"; printf("%d", strcmp(str1, str2)); // Output: -ve value</pre>

strlen() Function

- ❖ The strlen() function calculates the length of a given string. It doesn't count the null character '\0'.
- ❖ Syntax
 - int strlen(const char *str);
- ❖ Parameters
 - str: It represents the string variable whose length we have to find.
- ❖ Return Value
 - strlen() function in C returns the length of the string.

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

int main() {
    char str[] = "Hello, World!";
    printf("Length of the string: %lu\n", strlen(str)); // Output: 13
    return 0;
}
```

strrev() Function

❖ Definition:

- strrev() Function is a function which reverses the characters in a string in place.

❖ Parameters:

- char *str: A pointer to the null-terminated string to be reversed.

❖ Return Value

- char*: Returns a pointer to the same string (str) after reversing its contents.

❖ Syntax:

- char *strrev(char *str);



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

```
int main() {
    char str[] = "Hello";
    printf("Reversed string: %s\n", strrev(str)); // Output: olleH
    return 0;
}
```

strupr() Function

❖ Definition:

- strupr() Function is a function which converts all lowercase letters in a string to uppercase. Modifies the string in place.

❖ Parameters:


- `char *str`: A pointer to the null-terminated string to be converted to uppercase.

❖ Return Value

- `char*`: Returns a pointer to the same string (`str`) after converting all characters to uppercase.

❖ Syntax:

- `char *strupr(char *str);`



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

```
int main() {
    char str[] = "hello";
    printf("Uppercase string: %s\n", strupr(str)); // Output: HELLO
    return 0;
}
```

strlwr() Function

❖ Definition:

- strlwr() Function is a function which converts all uppercase letters in a string to lowercase. Modifies the string in place.

❖ Parameters:

- char *str: A pointer to the null-terminated string to be converted to lowercase.

❖ Return Value

- char*: Returns a pointer to the same string (str) after converting all characters to lowercase.

❖ Syntax:

- char *strlwr(char *str);



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

```
int main() {
    char str[] = "HELLO";
    printf("Lowercase string: %s\n", strlwr(str)); // Output: hello
    return 0;
}
```

strcpy() Function

❖ Definition:

- strcpy() Function is a function which copies a source string, including the null-terminator, into a destination string.

❖ Parameters:


- char *destination: A pointer to the destination string where the source will be copied.
- const char *source: A pointer to the string to copy.

❖ Return Value

- char*: Returns a pointer to the destination string after copying the contents of the source string.

❖ Syntax:

- `char *strcpy(char *destination, const char *source);`




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

```
int main() {
    char src[] = "C Programming";
    char dest[20];
    strcpy(dest, src);
    printf("Copied string: %s\n", dest); // Output: C Programming
    return 0;
}
```

strcat() Function

- ❖ Definition:
 - strcat() Function is a function which concatenates (appends) the source string to the end of the destination string.
- ❖ Parameters:
 - char *destination: A pointer to the destination string where the source will be appended.
 - const char *source: A pointer to the string to append.
- ❖ Return Value
 - char*: Returns a pointer to the destination string after appending the contents of the source string.
- ❖ Syntax:
 - char *strcat(char *destination, const char *source);



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

```
int main() {
    char str1[20] = "Hello ";
    char str2[] = "World";
    strcat(str1, str2);
    printf("Concatenated string: %s\n", str1); // Output: Hello World
    return 0;
}
```

strcmp() Function

❖ Definition:

- strcmp() Function is a function which compares two strings lexicographically (character by character).

❖ Parameters:


- const char *str1: A pointer to the first null-terminated string.
- const char *str2: A pointer to the second null-terminated string.

❖ Return Value

- int: Returns
 - 0 if the strings are equal.
 - A negative value if str1 is less than str2.
 - A positive value if str1 is greater than str2.

❖ Syntax:

- `int strcmp(const char *str1, const char *str2);`



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

```
int main() {
    char str1[] = "Apple";
    char str2[] = "Banana";

    int result = strcmp(str1, str2);

    if (result == 0) {
        printf("Strings are equal\n");
    } else if (result < 0) {
        printf("String 1 is less than String 2\n");
    } else {
        printf("String 1 is greater than String 2\n");
    }

    return 0;
}
```