#### What is Array

An array in C is a data structure that stores a fixed-size, sequential collection of elements of the same data type. These elements are stored in contiguous memory locations, allowing for efficient access using an index. The index of the first element in an array is 0, and the index of the last element is the array size minus 1.

Syntax:

data\_type array\_name[array\_size];

* **data\_type:** Specifies the type of elements the array will store (e.g., int, float, char).
* **array\_name:** The identifier used to refer to the array.
* **array\_size:** The number of elements the array can hold. This must be a constant value.

**Access the Elements of an Array**

* To access an array element, refer to its **index number**.
* Array indexes start with **0**: [0] is the first element. [1] is the second element, etc.

//To print any element of the array:

#include<stdio.h>

int main(){

int myNumber[] = {25, 50, 60, 40};

printf(“%d”, myNumber[0]);

Return 0;

}

//Output: 22

## **Change an Array Element**

Changing an **array element** means modifying the value stored at a specific index in the array. This is a basic operation in programming and is done using the array's name, the index of the element you want to change, and the new value.

//To change the 3rd element of the array:

#include<stdio.h>

int main(){

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

Numbers[2] = 99; // Change the 3rd element

for(int i = 0; i < 5; i++) { // Print the array

printf("%d ", numbers[i]);}

Return 0;}

1D and 2D Arrays

1D Array: A **1D array** is a linear collection of elements of the same data type stored in contiguous memory locations.

Syntax:

data\_type array\_name[array\_size];

Example – Declare, Access and Modify a 1D Array

#include <stdio.h>

int main() {

// Declare and initialize a 1D array

int numbers[5] = {10, 20, 30, 40, 50};

// Print original array

printf("Original array:\n");

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

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

}

// Change the 3rd element (index 2)

numbers[2] = 99;

// Print modified array

printf("\nModified array:\n");

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

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

}

return 0;

}

2D Array: A **2D array** is an array of arrays. It's used to represent a **table or matrix** structure with rows and columns.

data\_type array\_name[rows][columns];

Example – Declare, Access and Modify a 1D Array

#include <stdio.h>

int main() {

// Declare and initialize a 2D array (3 rows x 3 columns)

int matrix[3][3] = {

{1, 2, 3},

{4, 5, 6},

{7, 8, 9}

};

// Print original 2D array

printf("Original 2D array:\n");

for(int i = 0; i < 3; i++) {

for(int j = 0; j < 3; j++) {

printf("%d ", matrix[i][j]);

}

printf("\n");

}

// Change element at row 1, column 2 (matrix[1][2])

matrix[1][2] = 99;

// Print modified 2D array

printf("\nModified 2D array:\n");

for(int i = 0; i < 3; i++) {

for(int j = 0; j < 3; j++) {

printf("%d ", matrix[i][j]);

}

printf("\n");

}

return 0;

}

## Summary

| Concept | 1D Array | 2D Array |
| --- | --- | --- |
| **Structure** | Linear | Matrix (table of rows/cols) |
| **Syntax** | int a[5]; | int a[3][3]; |
| **Access** | a[i] | a[i][j] |
| **Use Cases** | List of values | Grids, matrices, tables |

**What is String**

In C, a string is represented as an array of characters, terminated by a null character \0. Since C doesn't have a built-in string type, character arrays are used to manipulate strings.

Declaration and Initialization:

A string can be declared and initialized in several ways: character array.

char greetings[] = "Hello World!";

String manipulation and functions

C language provides various built-in functions that can be used for various operations and manipulations on strings. These string functions make it easier to perform tasks such as string copy, concatenation, comparison, length, etc. The #include <string.h> directive in C is used to include the **Standard String Library**, which provides functions for manipulating **C-style strings** (i.e., arrays of characters ending with a null character '\0') and memory blocks.

🔹 Commonly Used Functions in <string.h>

| Function | Description | Syntax |
| --- | --- | --- |
| [strlen()](https://www.geeksforgeeks.org/strlen-function-in-c/) | Find the length of a string excluding ‘\0’ NULL character. | ****strlen****(str); |
| [strcpy()](https://www.geeksforgeeks.org/strcpy-in-c/) | Copies a string from the source to the destination. | ****strcpy****(dest, src); |
| [strncpy()](https://www.geeksforgeeks.org/strncpy-function-in-c/) | Copies n characters from source to the destination. | ****strncpy****( dest, src, n ); |
| [strcat()](https://www.geeksforgeeks.org/strcat-in-c/) | Concatenate one string to the end of another. | ****strcat****(dest, src); |
| [strncat()](https://www.geeksforgeeks.org/strncat-function-in-c-cpp/) | Concatenate n characters from the string pointed to by src to the end of the string pointed to by dest. | ****strncat****(dest, src, n); |
| [strcmp()](https://www.geeksforgeeks.org/strcmp-in-c/) | Compares these two strings lexicographically. | ****strcmp****(s1, s2); |
| strncmp() | Compares first n characters from the two strings lexicographically. | ****strncmp****(s1, s2, n); |
| [strchr()](https://www.geeksforgeeks.org/strchr-in-c/) | Find the first occurrence of a character in a string. | ****strchr****(s, c); |
| [strrchr()](https://www.geeksforgeeks.org/strrchr-in-c/) | Find the last occurrence of a character in a string. | ****strchr****(*s*, *ch*); |
| [strstr()](https://www.geeksforgeeks.org/strstr-in-ccpp/) | First occurrence of a substring in another string. | ****strstr****(s, subS); |
| [sprintf()](https://www.geeksforgeeks.org/sprintf-in-c/) | Format a string and store it in a string buffer. | ****sprintf****(s, format, …); |
| [strtok()](https://www.geeksforgeeks.org/strtok-strtok_r-functions-c-examples/) | Split a string into tokens based on specified delimiters. | ****strtok****(s, delim); |

Without #include <string.h>, functions like strlen, strcpy, or strcmp would **not be recognized by the compiler**, leading to warnings or errors.

Character arrays vs char\*

| Feature | char str[] = "Hello"; | char \*str = "Hello"; |
| --- | --- | --- |
| **Storage Type** | Character array (stack memory) | Pointer to string literal (read-only memory) |
| **Memory Location** | Stack | Read-only section (.rodata) |
| **Modifiable Content** | ✅ Yes | ❌ No (Undefined behavior if modified) |
| **Can Modify Pointer?** | ❌ No (array name is fixed) | ✅ Yes |
| **Null Terminator (**\0**) Added** | ✅ Yes | ✅ Yes |
| **Size Determined At Compile?** | ✅ Yes (fixed size) | ❌ No (only pointer, not actual data) |
| **Safe to Modify Characters?** | ✅ Yes | ❌ No (can crash or behave unexpectedly) |
| **Memory Deallocation Needed?** | ❌ No (stack memory auto-released) | ❌ No (literal, not malloc'ed) |
| **Best Use Case** | When string needs to be modified | When using constant/read-only strings |

* String library functions