**🧠 Unit: Arrays, Strings, and Functions in C**

**🔹 1. Introduction to Array**

**Definition**

An **array** is a **collection of elements of the same data type** stored in **contiguous memory locations**.  
It helps to store multiple values in a single variable instead of declaring many separate variables.

**Syntax**

data\_type array\_name[size];

**Example**

int marks[5];

float salary[10];

char name[20];

**Example Program: Store and Display Array Elements**

#include <stdio.h>

int main() {

int marks[5] = {80, 85, 90, 75, 95};

printf("Student Marks:\n");

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

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

}

return 0;

}

**Output**

Student Marks:

80 85 90 75 95

**🔹 2. One-Dimensional Array**

**Definition**

A **one-dimensional array** stores a list of values in a single row.

**Syntax**

data\_type array\_name[size];

**Example Program: Sum of Array Elements**

#include <stdio.h>

int main() {

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

int sum = 0;

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

sum = sum + num[i];

}

printf("Sum of array elements = %d", sum);

return 0;

}

**Output**

Sum of array elements = 150

**🔹 3. Multi-Dimensional Array**

**Definition**

A **multi-dimensional array** is an array of arrays.  
The most commonly used is the **two-dimensional array** (like a matrix).

**Syntax**

data\_type array\_name[row][column];

**Example Program: Display 2D Array (Matrix)**

#include <stdio.h>

int main() {

int a[2][3] = { {1, 2, 3}, {4, 5, 6} };

printf("Matrix Elements:\n");

for (int i = 0; i < 2; i++) {

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

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

}

printf("\n");

}

return 0;

}

**Output**

Matrix Elements:

1 2 3

4 5 6

**🔹 4. Introduction to String**

**Definition**

A **string** is a sequence of **characters** terminated by a **null character (\0)**.

**Syntax**

char string\_name[size];

**Example**

char name[10] = "Ravi";

**Program: Input and Output of String**

#include <stdio.h>

int main() {

char name[20];

printf("Enter your name: ");

scanf("%s", name);

printf("Hello %s", name);

return 0;

}

**Output**

Enter your name: Meera

Hello Meera

**🔹 5. Character and String Functions**

C provides built-in functions in **<string.h>** for string operations.

| **Function** | **Description** | **Example** |
| --- | --- | --- |
| strlen(str) | Returns string length | strlen("Hello") → 5 |
| strcpy(dest, src) | Copies one string to another |  |
| strcat(str1, str2) | Concatenates two strings |  |
| strcmp(str1, str2) | Compares two strings |  |
| strupr(str) | Converts string to uppercase |  |
| strlwr(str) | Converts string to lowercase |  |

**Example Program: String Functions**

#include <stdio.h>

#include <string.h>

int main() {

char str1[20] = "Hello";

char str2[20] = "World";

char str3[20];

strcpy(str3, str1);

strcat(str3, str2);

printf("Copied String: %s\n", str3);

printf("Length: %lu\n", strlen(str3));

printf("Comparison: %d\n", strcmp(str1, str2));

return 0;

}

**Output**

Copied String: HelloWorld

Length: 10

Comparison: -15

**🔹 6. Array of Strings**

**Definition**

An **array of strings** is a 2D array of characters, where each row represents one string.

**Example**

#include <stdio.h>

int main() {

char names[3][10] = {"Ravi", "Meera", "Amit"};

printf("List of Students:\n");

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

printf("%s\n", names[i]);

}

return 0;

}

**Output**

List of Students:

Ravi

Meera

Amit

**🧩 Functions in C**

**🔹 1. Introduction to User-Defined Functions**

**Definition**

A **function** is a block of code that performs a specific task and can be reused multiple times.  
C allows **user-defined functions** (created by programmers).

**Syntax**

return\_type function\_name(parameter\_list) {

// body of function

}

**Types of Functions**

1. Library Functions (e.g., printf(), scanf())
2. User-Defined Functions (created by the programmer)

**🔹 2. Function Declaration / Prototype**

Tells the compiler about function name, return type, and parameters.

int add(int, int);

**🔹 3. Function Definition**

Contains actual code (logic) of the function.

int add(int a, int b) {

return a + b;

}

**🔹 4. Function Call**

Function is executed when called by name.

**Example Program: Function Example**

#include <stdio.h>

// Function Declaration

int add(int, int);

int main() {

int a = 10, b = 20, sum;

sum = add(a, b);

printf("Sum = %d", sum);

return 0;

}

// Function Definition

int add(int x, int y) {

return x + y;

}

**Output**

Sum = 30

**🔹 5. Return Statement**

Used to return a value from a function.

return expression;

**🔹 6. Passing Parameters to Function**

**Call by Value Example**

#include <stdio.h>

void display(int n) {

printf("Number = %d", n);

}

int main() {

int x = 5;

display(x);

return 0;

}

**Output**

Number = 5

**🔹 7. Scope of Variables**

| **Scope Type** | **Description** |
| --- | --- |
| **Local Variable** | Declared inside function; used only within it. |
| **Global Variable** | Declared outside all functions; accessible in all functions. |

**Example**

#include <stdio.h>

int g = 10; // global variable

int main() {

int l = 5; // local variable

printf("Global = %d\n", g);

printf("Local = %d", l);

return 0;

}

**Output**

Global = 10

Local = 5

**🔹 8. Storage Classes**

| **Storage Class** | **Scope** | **Lifetime** | **Default Value** | **Keyword** |
| --- | --- | --- | --- | --- |
| Automatic | Local | Until function ends | Garbage | auto |
| Register | Local | Until function ends | Garbage | register |
| Static | Local/Global | Throughout program | 0 | static |
| Extern | Global | Throughout program | 0 | extern |

**Example**

#include <stdio.h>

void demo() {

static int count = 0;

count++;

printf("Count = %d\n", count);

}

int main() {

demo();

demo();

demo();

return 0;

}

**Output**

Count = 1

Count = 2

Count = 3

🧩 **Explanation:**  
The variable count retains its value between function calls due to the **static** keyword.

**🔹 9. Recursive Function**

**Definition**

A **recursive function** is a function that calls **itself** to solve a problem.

**Example: Factorial using Recursion**

#include <stdio.h>

int factorial(int n) {

if (n == 0)

return 1;

else

return n \* factorial(n - 1);

}

int main() {

int num = 5;

printf("Factorial of %d = %d", num, factorial(num));

return 0;

}

**Output**

Factorial of 5 = 120

🧩 **Explanation:**  
factorial(5) calls itself repeatedly until n == 0.

**✅ Summary**

| **Concept** | **Meaning** | **Example** |
| --- | --- | --- |
| Array | Collection of same type elements | int a[5] |
| String | Sequence of characters | "Hello" |
| Function | Reusable block of code | int add(int, int) |
| Recursion | Function calling itself | factorial() |