B.Sc. (I.T.) / M.Sc. (I.T.) 1s Semester
Course: 104: Fundamentals of Programming Using C-1
Unit 5: Character Array & String • **5.1** Declaration & Initialization of String • **5.2** Input/Output functions for String • **5.3** Arithmetic operations on String • **5.4** In built Functions for handling String • **5.5** Array of String

In C, strings are handled differently compared to higher-level languages.
A **string is simply a character array ending with a null character '\0'**.
Example:
"HELLO" is stored as:
Index : 0 1 2 3 4 5
Value : H E L L O \0
The **'\0' (null terminator)** is important — it tells C where the string ends.

5.1 Declaration & Initialization of String

1. Declaration

```
char str[20];    // can store max 19 characters + '\0'
```

2. Initialization Methods

A. Character by Character

```
char name[5] = {'J', 'O', 'H', 'N', '\0'};
```

B. String Literal (Most Common)

```
char name[] = "JOHN";   // '\0' added automatically
```

C. Fixed Size Initialization

```
char city[10] = "Surat";
```

Unused positions are filled with garbage only if initialized partially:

```
char word[10] = {'A','B','C'};   // no '\0' added unless explicitly
```

⚠ When no '\0' exists, string functions misbehave
(because they search continuously in memory until they find '\0').

5.2 Input/Output Functions for String

C provides various ways to read and print strings.

A. Output Functions

1. printf()
printf("%s", str);
- Stops printing when it finds '\0'.

2. puts()
puts(str);
- Automatically adds a newline.
- Safer for strings.

B. Input Functions

1. scanf("%s", str)
Reads input until space, tab, or newline.

Example:
Input: Hello World
Stored only: "Hello"
Logic
Stops reading when whitespace is encountered.

2. gets(str) (Not Recommended)
Reads line with spaces, but unsafe → can cause memory overflow.

3. fgets(str, size, stdin) (Recommended)
fgets(str, 20, stdin);
- Reads spaces
- Maximum limit prevents overflow
- Safest method

Example Program

```
#include <stdio.h>
int main() {
   char name[20];

   printf("Enter name: ");
   fgets(name, 20, stdin);

   printf("Hello %s", name);
}
```

5.3 Arithmetic Operations on String (Conceptual in C)

In higher-level languages (Python, Java), strings support arithmetic like:

- concatenation using +
- repetition using *

But **C does not support direct string arithmetic operators**.
However, using **string functions**, we can perform string-like arithmetic operations:

5.3.1 Concatenation (Joining Strings)

Using strcat().

```
char s1[20] = "Hello ";
char s2[10] = "World";

strcat(s1, s2);
```

Result stored in s1: "Hello World"

Logic

- Finds end of s1 (till '\0')
- Starts copying s2 after it
- Adds new '\0'

5.3.2 Copying Strings

Using strcpy().

```
strcpy(dest, src);
```

Logic

Copies character by character until it reaches '\0'.

5.3.3 Comparing Strings

Using strcmp().

```
strcmp("Apple", "Mango");
```

Returns:

- 0 → if both strings are equal
- negative → if first string is smaller
- positive → if first string is greater

This helps in alphabetical sorting.

5.3.4 Length of a String

Using strlen().

```
strlen("hello") → 5
```

Logic

Counts characters until '\0' is found.

5.4 In-built Functions for Handling Strings

All available in **<string.h>**.
Below is a fully organized, exam-friendly list.

---

5. strlen(str)
Returns length of string (excluding '\0').

---

2. strcpy(dest, src)
Copies source to destination.

---

3. strncpy(dest, src, n)
Copies **first n characters**.

---

4. strcat(dest, src)
Appends src to end of dest.

---

5. strncat(dest, src, n)
Appends **n characters**.

---

6. strcmp(s1, s2)
Compares two strings lexicographically.

---

7. strncmp(s1, s2, n)
Compares **first n characters**.

---

8. strchr(str, ch)
Returns first occurrence of character.
Example:
strchr("Hello", 'l'); → points to "llo"

9. strstr(str, substr)
Finds first occurrence of substring.
Example:
strstr("Programming", "gram"); → points to "gramming"

---

10. strtok(str, delim)
Tokenizes (splits) string.
Example:
char s[] = "C,Java,Python";
strtok(s, ","); // returns "C"
strtok(NULL, ","); // "Java"

5.5 Array of Strings

Used when storing **multiple words**, like:

- list of names
- product items
- student names

Declaration

Method 1 — Array of Characters (2D array)

char names[5][20];

Meaning:

- 5 strings
- Each string max 19 chars

Example:

names[0] = "Ravi"

names[1] = "Neha"

...

Method 2 — Array of String Pointers

char *cities[] = {"Surat", "Vadodara", "Ahmedabad"};

Each pointer points to memory containing a string literal.

Example Program: Array of Strings (with Logic)

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

int main() {
   char names[3][20];
   int i;

   for(i=0;i<3;i++) {
      printf("Enter name %d: ", i+1);
      scanf("%s", names[i]);
   }

   printf("\nYou entered:\n");
   for(i=0;i<3;i++)
      printf("%s\n", names[i]);

   return 0;
}
```

Logic Explanation

1. names[3][20] stores **3 strings** of max length 19.
2. Loop reads each string into names[i].
3. Printing loop displays stored values.

Summary Table (Quick Revision – Useful for Exam)

Topic	Key Points
String	Character array ending with '\0'
Declaration	char str[20];
Input	scanf, fgets
Output	printf, puts
Concatenation	strcat()
Copy	strcpy()
Compare	strcmp()
Length	strlen()
Array of Strings	char names[5][20]