1. Introduction to Strings

In C language, a **string** is a **sequence of characters** stored in an array of characters and **terminated by a null character ('\0')**.

Unlike other languages, **C does not have a built-in string data type**. Strings are handled using **character arrays** and **string library functions**.

Example

```
char name[] = "C Language";
```

---

2. Importance of Null Character ('\0')

The **null character** marks the **end of a string**.

Example

```
char str[] = {'H','E','L','L','O','\0'};
```

Without '\0', C cannot identify the end of the string, which may cause errors.

---

3. Declaration of Strings

3.1 Using Character Array

```
char str[20];
```

3.2 Initialization of Strings

Compile-Time Initialization

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

Character-wise Initialization

```
char str[] = {'H','i','\0'};
```

3.3 Run-Time Input

```
scanf("%s", str);
```

□□ scanf() stops input at whitespace.

---

4. String Input and Output Functions

4.1 gets() and puts()

```
gets(str);   // Not recommended (unsafe)
puts(str);
```

4.2 fgets() and puts()

```
fgets(str, 20, stdin);
puts(str);
```

☑fgets() is safer and recommended.

---

5. String Handling Functions

C provides many built-in string functions in <string.h>.

5.1 strlen()

Returns the length of string (excluding '\0').

```
int len = strlen(str);
```

5.2 strcpy()

Copies one string into another.

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

5.3 strcat()

Concatenates two strings.

```
strcat(str1, str2);
```

5.4 strcmp()

Compares two strings.

```
strcmp(str1, str2);
```

Returns:

- 0 → equal
- <0 → string1 < string2
- 0 → string1 > string2

5.5 strlwr() and strupr()

Convert string to lowercase or uppercase.

```
strlwr(str);
strupr(str);
```

---

6. String and Pointers

Strings can be accessed using **character pointers**.

Example

```
char *str = "Hello";
printf("%s", str);
```

Pointer points to the **base address of the string**.

---

7. Array of Strings

An array of strings is a **2D character array**.

Example

```
char names[3][20] = {"Ram", "Shyam", "Mohan"};
```

Access:

```
printf("%s", names[1]);
```

---

8. Passing Strings to Functions

Strings are passed as **character arrays or pointers**.

Example

```
void display(char str[])
{
   printf("%s", str);
}
```

---

9. String Operations Using Loops

9.1 String Length Without strlen()

```
int i = 0;
while(str[i] != '\0')
   i++;
```

9.2 String Copy Without strcpy()

```
int i = 0;
while(str2[i] = str1[i])
   i++;
```

10. Searching in Strings

Example: Count Vowels

```
int i, count = 0;
for(i = 0; str[i] != '\0'; i++)
{
   if(str[i]=='a'||str[i]=='e'||str[i]=='i'||str[i]=='o'||str[i]=='u')
      count++;
}
```

11. String Comparison Using Loop

```
int i = 0;
while(str1[i] == str2[i])
{
   if(str1[i] == '\0')
      break;
   i++;
}
```

12. Common String Programs

12.1 Reverse a String

```
int i, len;
len = strlen(str);
for(i = len - 1; i >= 0; i--)
   printf("%c", str[i]);
```

12.2 Palindrome Check

// Compare original and reversed string

13. Difference Between Character Array and String

Feature	Character Array	String
Data Type	char array	char array + '\0'
Termination	No	Yes
Functions	Not applicable	Available

---

14. Advantages of Strings

- Easy text manipulation
- Used in input/output
- Essential for file handling
- Widely used in applications

---

15. Limitations of Strings in C

- Fixed size
- No built-in string type
- Unsafe functions like gets()
- Manual memory handling

---

16. Common Errors in Strings

1. Missing null character
2. Buffer overflow
3. Using gets()
4. Incorrect size declaration
5. Uninitialized strings

---

17. Best Practices

- Always use fgets() instead of gets()
- Allocate sufficient memory

- Use string functions carefully
- Validate input length

---

18. Applications of Strings

- Text editors
- File handling
- Command-line tools
- Web development
- Data processing

---

19. Interview / Exam Important Points

- Strings are character arrays
- Null character is mandatory
- <string.h> is required
- Strings are passed by reference

---

20. Conclusion

Strings are a fundamental part of C programming used for handling text data. Understanding string declaration, manipulation, and functions is essential for writing effective and reliable C programs.