Comprehensive Study of `string.h` in C

---

1. Introduction

The C programming language does not provide a built-in string data type. Instead, strings are represented as character arrays terminated by a null character (`\0`). To manipulate these strings efficiently, C provides a standard library header file called:

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

This header contains a collection of powerful functions used for string handling such as copying, comparing, concatenating, searching, and tokenizing strings. It is one of the most widely used libraries in system programming, embedded systems, and application development.

---

2. Importance of `string.h`

The `string.h` header plays a fundamental role in C programming:

- Provides built-in optimized string handling functions
- Allows low-level memory manipulation
- Used in operating systems and compilers
- High performance and portability
- Standardized by ANSI C

Applications of `string.h` include:

- File handling programs
- Text editors
- Operating system kernels
- Network applications
- Database systems

---

3. What is a String in C?

A string in C is a sequence of characters stored in a character array and terminated with a null character.

Example:

```
char name[] = "India";
```

Memory Representation:

```
I | n | d | i | a | \0
```

Each character occupies 1 byte in memory.

---

4. Difference Between String and Character Array

Feature	String	Character Array
Null Terminator	Mandatory (`\0`)	Not mandatory
Printing with `%s`	Safe	Unsafe
Library Support	Yes	No
Purpose	Text data	General storage

---

5. Overview of `string.h`

To use string functions in C:

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

`string.h` contains functions for:

- String length calculation
- Copying strings
- Concatenating strings
- Comparing strings
- Searching characters and substrings
- Tokenizing strings

- Memory operations

---

6. `strlen()` Function

Finds the length of a string.

Syntax:

```
size_t strlen(const char *str);
```

Example:

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

int main() {
    char name[] = "Computer";
    printf("Length = %lu", strlen(name));
    return 0;
}
```

Output:

```
Length = 8
```

---

7. `strcpy()` Function

Copies one string into another.

Syntax:

```
char* strcpy(char *dest, const char *src);
```

Example:

```
char src[] = "BTech";
char dest[20];
strcpy(dest, src);
```

⚠️ Risk: May cause buffer overflow if destination size is smaller.

---

8. `strncpy()` Function

Safer version of `strcpy()` with limit.

```
char* strncpy(char *dest, const char *src, size_t n);
```

---

9. `strcat()` Function

Concatenates two strings.

```
char a[30] = "Hello ";
char b[] = "World";
strcat(a, b);
```

Output:

```
Hello World
```

---

10. `strcmp()` Function

Compares two strings lexicographically.

```
int result = strcmp("Apple", "Banana");
```

Return Values:

- `0` → Equal
- Positive → First is greater
- Negative → Second is greater

---

11. `strchr()` Function

Finds the first occurrence of a character.

```
char *ptr = strchr("Programming", 'm');
```

---

12. `strrchr()` Function

Finds the last occurrence of a character.

---

13. `strstr()` Function

Finds the first occurrence of a substring.

```
strstr("Computer Science", "Science");
```

---

14. `strtok()` Function

Splits a string into tokens.

```
char str[] = "C,Java,Python";
char *token = strtok(str, ",");

while(token != NULL) {
    printf("%s\n", token);
    token = strtok(NULL, ",");
}
```

---

15. Memory Functions in `string.h`

- `memcpy()` – Copy memory
- `memset()` – Fill memory
- `memcmp()` – Compare memory

These functions do not stop at the null character.

16. Reverse a String (Without Library)

```
void reverse(char str[]) {
    int i, j;
    char temp;

    for(i = 0, j = strlen(str)-1; i < j; i++, j--) {
        temp = str[i];
        str[i] = str[j];
        str[j] = temp;
    }
}
```

17. Palindrome Program

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

int main() {
    char str[50], rev[50];

    printf("Enter a word: ");
    scanf("%s", str);

    strcpy(rev, str);
    strrev(rev);

    if(strcmp(str, rev) == 0)
        printf("Palindrome");
    else
        printf("Not a Palindrome");

    return 0;
}
```

18. Common Errors in `string.h`

- Buffer overflow
- Missing null terminator
- Using `gets()`
- Using `strcpy()` without size check
- Modifying string literals

---

19. Security Issues in `string.h`

- Stack smashing
- Memory corruption
- Data leakage
- Program crashes

✅ Secure practices:

- Use `strncpy()` and `strncat()`
- Validate buffer size
- Avoid unsafe input functions

---

20. `string.h` vs C++ `string`

C(`string.h`)	C++(`string`)
Manual memory management	Automatic
Unsafe	Safe
Faster execution	Slightly slower
Low-level	High-level

---

21. Interview Questions

1. Difference between `strlen()` and `sizeof()` ?
2. What is the use of `strtok()` ?
3. What is buffer overflow?

4. How is a string stored in memory?
5. Difference between `strcpy()` and `strncpy()` ?

---

22. Conclusion

The `string.h` header file is a core part of C programming. It provides essential tools for string manipulation and low-level memory operations. Proper understanding and careful use of these functions are necessary for secure and efficient programming.

---

23. References

1. Dennis Ritchie – *The C Programming Language*
2. ISO C Standard
3. GNU C Library
4. GeeksforGeeks – String Functions in C
5. TutorialsPoint – C Strings