1. strcat()

Used to **concatenate (join)** two strings.
Syntax:strcat(destination, source);

- destination → must have enough space to hold both strings.
- source → string to be appended at the end of destination.

Example:

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

int main() {
   char str1[20] = "Hello";
   char str2[] = "World";
   strcat(str1, str2);   // str1 = "HelloWorld"
   printf("Result: %s\n", str1);
   return 0;
}
```

Tracing:

- Step 1: str1 = "Hello\0"
- Step 2: str2 = "World\0"
- Step 3: strcat(str1, str2); → copies "World" after "Hello".
- Final: str1 = "HelloWorld\0"

2. strncat()

Same as strcat() but allows you to **limit the number of characters appended**.
Syntax: strncat(destination, source, n);

- Appends at most **n characters** from source.
- Still adds a \0 at the end.

Example:

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

```
int main() {
    char str1[20] = "Hello";
    char str2[] = "World";
    strncat(str1, str2, 3);   // Append first 3 chars of str2
    printf("Result: %s\n", str1);
    return 0;
}
```

Tracing:

- Step 1: str1 = "Hello\0"
- Step 2: str2 = "World\0"
- Step 3: strncat(str1, str2, 3); → copies "Wor" after "Hello".
- Final: str1 = "HelloWor\0"

Key Difference:

- strcat() → copies entire source string.
- strncat() → copies only n characters from source.

2) **strcmp()**

Compares **two strings** character by character.
Syntax: strcmp(str1, str2)

- Returns **0** → if both strings are equal.
- Returns **>0** → if first non-matching char in str1 is greater than str2.
- Returns **<0** → if first non-matching char in str1 is smaller than str2.

Example:

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

int main() {
   char s1[] = "Hello";
   char s2[] = "Hello";
   char s3[] = "World";

   printf("strcmp(s1, s2) = %d\n", strcmp(s1, s2)); // equal → 0
   printf("strcmp(s1, s3) = %d\n", strcmp(s1, s3)); // "H" < "W" → negative
   printf("strcmp(s3, s1) = %d\n", strcmp(s3, s1)); // "W" > "H" → positive

   return 0;
}
```

Tracing:

- Compare s1 = "Hello" with s2 = "Hello"
  → All characters same → result = **0**.
- Compare s1 = "Hello" with s3 = "World"
  → Compare 'H' (72) vs 'W' (87) → 'H' < 'W' → result = **negative**.
- Compare s3 = "World" with s1 = "Hello"
  → 'W' (87) vs 'H' (72) → 'W' > 'H' → result = **positive**.

2. strncmp()

Same as strcmp(), but compares **only first n characters**.
Syntax: strncmp(str1, str2, n);

- Returns result based on first **n characters**.

Example:

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

int main() {
   char s1[] = "Hello";
   char s2[] = "Helium";

   printf("strncmp(s1, s2, 3) = %d\n", strncmp(s1, s2, 3)); // "Hel" vs "Hel" → equal → 0
   printf("strncmp(s1, s2, 5) = %d\n", strncmp(s1, s2, 5)); // "Hello" vs "Heliu" → 'l' vs 'i' →
positive

   return 0;
}
```

Tracing:

- strncmp(s1, s2, 3) → "Hel" vs "Hel" → first 3 chars same → result = **0**.
- strncmp(s1, s2, 5) → Compare char by char:
    - H == H
    - e == e
    - l == l
    - Next: l (108) vs i (105) → 'l' > 'i' → result = **positive**.

Key Difference:

- strcmp() → compares **entire strings** until \0.
- strncmp() → compares **only n characters**, then stops.

1. strcpy()

Used to **copy one string into another**.
Syntax: strcpy(destination, source);

- Copies all characters from source into destination, including the null character \0.
- Destination must have **enough memory** to hold the source string.

Example:

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

int main() {
   char src[] = "Mahesh";
   char dest[20];   // empty buffer

   strcpy(dest, src);

   printf("Source: %s\n", src);
   printf("Destination: %s\n", dest);

   return 0;
}
```

Tracing:

- src = "Mahesh\0"
- dest = ????? (garbage initially)

strcpy(dest, src) → copies each character:

'M' → dest[0]
'a' → dest[1]
'h' → dest[2]
'e' → dest[3]
's' → dest[4]
'h' → dest[5]
'\0' → dest[6]

- Final: dest = "Mahesh\0"

2. strncpy()

Same as strcpy() but allows you to **copy only first n characters**.
Syntax:strncpy(destination, source, n);

- Copies at most **n characters**.
- If source is shorter than n, remaining space is filled with \0.
- If source is longer than n, \0 may not be added automatically → must add manually.

Example 1 (when n > length of source):

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

int main() {
   char src[] = "Hello";
   char dest[10];

   strncpy(dest, src, 10);  // copy 10 chars (Hello + pad with \0)

   printf("Destination: %s\n", dest);
   return 0;
}
```

Tracing:

- src = "Hello\0" (6 chars including \0)
- Copy "Hello\0" into dest → remaining positions filled with \0.
- Final: dest = "Hello\0\0\0\0\0"

Example 2 (when n < length of source):

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

int main() {
   char src[] = "Programming";
   char dest[20];
```

```
    strncpy(dest, src, 5);  // copy only "Progr"
    dest[5] = '\0';         // manually add null terminator

    printf("Destination: %s\n", dest);
    return 0;
}
```

Tracing:

- src = "Programming\0"
- Copy only first 5 chars → "Progr"
- Add \0 manually.
- Final: dest = "Progr\0"

Key Difference:

- strcpy() → copies full string (always null-terminated).
- strncpy() → copies only n characters, **may not null-terminate**, so we must handle \0 carefully.

🔹 **1. strlen()**

👉 Used to **find the length of a string** (number of characters **before \0**, not including \0).
👉 Syntax:

size_t strlen(const char *str);

- Counts characters until it hits the null terminator \0.
- Return type = size_t (unsigned integer).

Example:

```
#include <stdio.h>

#include <string.h>

int main() {

   char str[] = "Mahesh";

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

   printf("Length = %lu\n", strlen(str));  // 6 (without '\0')

   return 0;

}
```

Tracing:

- str = "Mahesh\0"
- Counts: M=1, a=2, h=3, e=4, s=5, h=6 → stops at \0.
- Final Result = **6**.

2. strnlen()

Safer version of strlen().

Syntax: size_t strnlen(const char *str, size_t maxlen);

- Returns the length of string, **but at most maxlen characters**.
- Prevents scanning memory endlessly if \0 is missing.

Example:

```
#include <stdio.h>

#include <string.h>


int main() {

   char str[] = "Programming";


   printf("Using strlen: %lu\n", strlen(str));          // 11

   printf("Using strnlen (max=5): %lu\n", strnlen(str, 5));   // 5

   printf("Using strnlen (max=20): %lu\n", strnlen(str, 20)); // 11


   return 0;

}
```

Tracing:

- str = "Programming\0" → actual length = 11.
- strlen(str) → scans until \0 → 11.

- strnlen(str, 5) → scans only 5 chars max → result = 5.
- strnlen(str, 20) → scans up to 20, but finds \0 at 11 → result = 11.

Key Difference:

- strlen() → gives exact length until \0.
- strnlen() → gives min(actual length, maxlen). Safer when string may not be null-terminated.

Summary:

Function	Purpose	Key Behavior	Risk / Notes
strlen(str)	Get length of string	Counts characters until \0 (not included).	If \0 missing → undefined behavior.
strnlen(str, n)	Get length safely	Returns min(actual length, n).	Safe – avoids scanning memory beyond n.
strcpy(dest, src)	Copy string	Copies entire src (including \0) into dest.	dest must be large enough → risk of overflow.
strncpy(dest, src, n)	Copy first n chars	Copies at most n chars; if src shorter → fills with \0. If longer → may miss \0.	Must add \0 manually if src ≥ n.
strcat(dest, src)	Concatenate strings	Appends full src to dest.	dest must have enough extra space.
strncat(dest, src, n)	Concatenate safely	Appends at most n chars from src, always adds \0.	Still check buffer size.
strcmp(s1, s2)	Compare strings	Returns 0 if equal; <0 if s1<s2; >0 if s1>s2.	Compares until \0.

strncmp(s1, s2, n)	Compare first n chars	Same as strcmp, but checks only n chars.	Safer for partial comparison.