strcpy

strcpy() is a standard library function in C/C++ and is used to copy one string to another. In C it is present in **string.h** header file and in C++ it is present in **cstring** header file.

Syntax:

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

Paramters: This method accepts following paramters:

- **dest**: Pointer to the destination array where the content is to be copied.
- **src:** string which will be copied.

Return Value: After copying the source string to the destination string, the strcpy() function returns a pointer to the destination string.

Below program explains different usages of this library function:

```
// C program to illustrate
// strcpy() function ic C/C++
#include<stdio.h>
#include<string.h>

int main ()
{
   char str1[]="Hello Geeks!";
   char str2[] = "GeeksforGeeks";
   char str3[40];
   char str4[40];
   char str5[] = "GfG";

   strcpy(str2, str1);
   strcpy(str3, "Copy successful");
   strcpy(str4, str5);
   printf ("str1: %s\nstr2: %s\nstr3: %s\nstr4: %s\n", str1, str2, str3, str4);
   return 0;
}
```

Output:

```
str1: Hello Geeks!
str2: Hello Geeks!
str3: copy successful
str4: GfG
```

strncpy() function

The strncpy() function is similar to strcpy() function, except that at most n bytes of src are copied. If there is no NULL character among the first n character of src, the string placed in dest will not be NULL-terminated. If the length of src is less than n, strncpy() writes additional NULL character to dest to ensure that a total of n character are written.

Syntax:

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

Parameters: This function accepts two parameters as mentioned above and described below:

- **src:** The string which will be copied.
- **dest:** Pointer to the destination array where the content is to be copied.
- **n:** The first n character copied from src to dest.

Return Value: It returns a pointer to the destination string.

```
#include <stdio.h>
#include <string.h>
int main()
{
   char src[] = "geeksforgeeks";
   // The destination string size is 14.
   char dest[14];
   // copying n bytes of src into dest.
   strncpy(dest, src, 14);
   printf("Copied string: %s\n", dest);
   return 0;
}
```

Output:

Copied string: geeksforgeeks

strcat() function will append a copy of the source string to the end of destination string. The strcat() function takes two arguments:

1) dest

2) src

It will append copy of the source string in the destination string. **The terminating character at the end of dest is replaced by the first character of src .**

Return value: The strcat() function returns dest, the pointer to the destination string.

```
#include <cstring>
#include <iostream>
int main()
{
   char dest[50] = "This is an";
   char src[50] = " example";
   strcat(dest, src);
   printf("%s",dest);
   return 0;
}
```

Output:

This is an example

strncat() is a predefined function used for string handling. **string.h** is the header file required for string functions.

This function appends not more than **n** characters from the string pointed to by src to the end of the string pointed to by dest plus a terminating Null-character. The initial character of string(src) overwrites the Null-character present at the end of string(dest). Thus, length of the string(dest) becomes strlen(dest)+n. But, if the length of the string(src) is less than **n**, only the content up to the terminating null-character is copied and length of the string(dest) becomes strlen(src) + strlen(dest).

The behavior is undefined if

- the strings overlap.
- the dest array is not large enough to append the contents of src.

Syntax:

char *strncat(char *dest, const char *src, size_t n)

Parameters: This method accepts following paramters:

- **dest**: the string where we want to append.
- **src**: the string from which 'n' characters are going to append.
- **n**: represents maximum number of character to be appended. size_t is an unsigned integral type.

Return Value: The strncat() function shall return the pointer to the string(dest).

Program:

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

```
int main()
{

    // Take any two strings
    char src[50] = "efghijkl";
    char dest[50] = "abcd";

    // Appends 5 character from src to dest
    strncat(dest, src, 5);

    // Prints the string
    printf("Source string : %s\n", src);
    printf("Destination string : %s", dest);

    return 0;
}
```

Output:

```
Source string : efghijkl
Destination string : abcdefghi
```

strchr()

The function strchr() searches the occurrence of a specified character in the given string and returns the pointer to it.

Syntax:

```
char *strchr(const char *str, int ch)
```

str – The string in which the character is searched.
ch – The character that is searched in the string str.

```
#include <stdio.h>
#include <string.h>
int main () {
   const char str[] = "This is just a String";
   const char ch = 'u';
   char *p;
   p = strchr(str, ch);
   printf("String starting from %c is: %s", ch, p);
   return 0;
}
```

Output:

```
String starting from u is: ust a String
```

strrchr() function in C/C++ locates the last occurrence of a character in a string. It returns a pointer to the last occurrence in the string. The terminating null character is considered part of the C string. Therefore, it can also be located to retrieve a pointer to the end of a string. It is defined in **cstring** header file.

Syntax :

```
const char* strrchr( const char* str, int ch )
          or
char* strrchr( char* str, int ch )
```

Parameter : The function takes two mandatory parameters which are described below:

- **str :** specifies the pointer to the null terminated string to be searched for.
- **ch:** specifies the character to be search for.

Return Value: The function returns a pointer to the last location of **ch** in string, if the **ch** is found. If not found, it returns a null pointer.

Below programs illustrate the above function:

Program 1:

```
#include <stdio.h>
#include <string.h>
int main()
{

    // initializing variables
    char st[] = "GeeksforGeeks";
    char ch = 'e';
    char* val;

    val = strrchr(st, ch);
    printf("String after last %c is : %s \n",  ch, val);

    char ch2 = 'm';
    val = strrchr(st, ch2);
    printf("String after last %c is : %s ", ch2, val);
    return (0);
}
```

Output:

```
String after last e is : eks
String after last m is : (null)
```

strcmp()

strcmp() is a built-in library function and is declared in **<string.h>** header file. This function takes two strings as arguments and compare these two strings lexicographically.

Syntax::

int strcmp(const char *leftStr, const char *rightStr);

In the above prototype, function srtcmp takes two strings as parameters and returns an integer value based on the comparison of strings.

- strcmp() compares the **two strings lexicographically** means it starts comparison character by character starting from the first character until the characters in both strings are equal or a NULL character is encountered.
- If first character in both strings are equal, then this function will check the second character, if this is also equal then it will check the third and so on
- This process will be continued until a character in either string is NULL or the characters are unequal.

Zero (0): A value equal to zero when both strings are found to be identical. That is, That is, All of the characters in both strings are same.

- All characters of strings are same

```
#include<stdio.h>
#include<string.h>
int main()
{
   char leftStr[] = "g f g";
   char rightStr[] = "g f g";
   // Using strcmp()
   int res = strcmp(leftStr, rightStr);
   if (res==0)
      printf("Strings are equal");
   else
      printf("Strings are unequal");

   printf("\nValue returned by strcmp() is:  %d" , res);
   return 0;
}
```

Output:

Strings are equal
Value returned by strcmp() is: 0

Greater than zero (>0): A value greater than zero is returned when the first not matching character in leftStr have the greater ASCII value than the corresponding character in rightStr or we can also say If character in leftStr is lexicographically **after** the character of rightStr

```
#include<stdio.h>
#include<string.h>
int main()
{
   // z has greater ASCII value than g
   char leftStr[] = "zfz";
   char rightStr[] = "gfg";

   int res = strcmp(leftStr, rightStr);

   if (res==0)
      printf("Strings are equal");
   else
      printf("Strings are unequal");

   printf("\nValue of result: %d" , res);

   return 0;
}
```

Output:

Strings are unequal
Value returned by strcmp() is: 19

Less than Zero (<0): A value less than zero is returned when the first not matching character in leftStr have lesser ASCII value than the corresponding character in rightStr. If character in leftStr is lexicographically **before** the character of rightStr

```
#include<stdio.h>
#include<string.h>
int main()
{
   // b has less ASCII value than g
   char leftStr[] = "bfb";
   char rightStr[] = "gfg";

   int res = strcmp(leftStr, rightStr);
```

```
    if (res==0)
        printf("Strings are equal");
    else
        printf("Strings are unequal");

    printf("\nValue returned by strcmp() is:  %d" , res);


    return 0;
}
```

Output:

```
Strings are unequal
Value returned by strcmp() is:  -5
```

strncmp() function lexicographically compares not more than count characters from the two null-terminated strings and returns an integer based on the outcome.

- This function takes two strings and a number **num** as arguments and compare at most first **num** bytes of both the strings.
- **num** should be at most equal to the length of the longest string. If **num** is defined greater than the string length than comparison is done till the null-character('\0') of either string.
- This function compares the two strings lexicographically. It starts comparison from the first character of each string. If they are equal to each other, it continues and compare the next character of each string and so on.
- This process of comparison stops until a terminating null-character of either string is reached or **num** characters of both the strings matches.

Syntax :

int strncmp(const char *str1, const char *str2, size_t count);

Parameters:
str1 and str2: C string to be compared.
count: Maximum number of characters to compare.
size_t is an unsigned integral type.

Return Value:

Value	**Meaning**
Less than zero	str1 is less than str2.
Zero	str1 is equal to str2.
Greater than zero	str1 is greater than str2.

1. **Greater than zero (>0):** A positive value is returned, if a character of **str1** and **str2** doesn't match before the **num** characters and the ASCII value of *str1* character is **greater** than ASCII value of *str2* character. As a result, we can say that *str1* is lexicographically greater than *str2*.

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

int main()
{
   // Take any two strings
   char str1[10] = "aksh";
   char str2[10] = "akash";

   // Compare strings using strncmp()
   int result = strncmp(str1, str2, 4);

   if (result == 0) {
      // num is the 3rd parameter of strncmp() function
      printf("str1 is equal to str2 upto num characters\n");
   }
   else if (result > 0)
      printf("str1 is greater than str2\n");
   else
      printf("str2 is greater than str1\n");

   printf("Value returned by strncmp() is: %d", result);

   return 0;
}
```

Output:

```
str1 is greater than str2
Value returned by strncmp() is: 18
```

2. Less than zero (<0): A negative value is returned, if a character of **str1** and **str2** doesn't match before the **num** characters and the ASCII value of *str1* character is **lesser** than ASCII value of *str2* character. As a result, we can say that *str2* is lexicographically greater than *str1*

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

```
int main()
{
    // Take any two strings
    char str1[10] = "akash";
    char str2[10] = "aksh";

    // Compare strings using strncmp()
    int result = strncmp(str1, str2, 4);

    if (result == 0) {
        // num is the 3rd parameter of strncmp() function
        printf("str1 is equal to str2 upto num characters\n");
    }
    else if (result > 0)
        printf("str1 is greater than str2\n");
    else
        printf("str2 is greater than str1\n");

    printf("Value returned by strncmp() is: %d", result);

    return 0;
}
```

Output:

```
str2 is greater than str1
Value returned by strncmp() is: -18
```

strspn()

The **strspn()** function returns the length of the initial substring of the string pointed to by *str1* that is made up of only those character contained in the string pointed to by *str2*.

Syntax :

size_t strspn(const char *str1, const char *str2)
str1 : string to be scanned.
str2 : string containing the
characters to match.
Return Value : This function
returns the number of characters
in the initial segment of str1
which consist only of characters
from str2.

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

int main () {
   int len = strspn("geeks for geeks","geek");
   printf("Length of initial segment matching : %d\n", len );
   return(0);
}
```

Output:

```
Length of initial segment matching 4
```

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

int main () {
   int len = strspn("i am","xyz");
   printf("Length of initial segment matching : %d\n", len );
   return(0);
}
```

Output:

```
Length of initial segment matching 0
```

strcspn()

The C library function **strcspn()** calculates the length of the number of characters before the 1st occurrence of character present in both the string.
Syntax :

strcspn(const char *str1, const char *str2)

Parameters:
str1 : The Target string in which search has to be made.
str2 : Argument string containing characters
to match in target string.

Return Value:
This function returns the number of characters before the 1st occurrence of character present in both the string.

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

```
int main()
{

    int size;

    // initializing strings
    char str1[] = "geeksforgeeks";
    char str2[] = "kfc";

    // using strcspn() to calculate initial chars
    // before 1st matching chars.
    // returns 3
    size = strcspn(str1, str2);

    printf("The unmatched characters before first matched character :  %d\n", size);
}
```

Output:

The unmatched characters before first matched character : 3

strlen( )

function in C gives the length of the given string. Syntax for strlen( ) function is given below.

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

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

int main( )
{
    int len;
    char array[20]="fresh2refresh.com" ;

    len = strlen(array) ;

    printf ( "\string length  = %d \n" , len ) ;
    return 0;
}
```

Output:

string length = 17

strpbrk() in C

This function **finds the first character in the string s1 that matches any character specified in s2** (It excludes terminating null-characters).

Syntax :
char *strpbrk(const char *s1, const char *s2)

Parameters :
s1 : string to be scanned.
s2 : string containing the characters to match.

Return Value :
It returns a pointer to the character in s1 that
matches one of the characters in s2, else returns NULL.

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

int main()
{
    // Declaring three strings
    char s1[] = "geeksforgeeks";
    char s2[] = "app";
    char s3[] = "kite";
    char* r, *t;
    // Checks for matching character
    // no match found
    r = strpbrk(s1, s2);
    if (r != 0)
        printf("First matching character: %c\n", *r);
    else
        printf("Character not found");
    // Checks for matching character
    // first match found at "e"
    t = strpbrk(s1, s3);
    if (t != 0)
        printf("\nFirst matching character: %c\n", *t);
    else
        printf("Character not found");
    return (0);
}
```

Output:

Character not found
First matching character: e

strstr()

strstr() is a predefined function used for string handling. **string.h** is the header file required for string functions.

This function takes two strings **s1** and **s2** as an argument and finds the first occurrence of the sub-string **s2** in the string **s1**

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

int main()
{
    // Take any two strings
    char s1[] = "GeeksforGeeks";
    char s2[] = "for";
    char* p;

    // Find first occurence of s2 in s1
    p = strstr(s1, s2);

    // Prints the result
    if (p) {
        printf("String found\n");
        printf("First occurrence of string '%s' in '%s' is '%s'", s2, s1, p);
    } else
        printf("String not found\n");
    return 0;
}
```

Output:

String found
First occurrence of string 'for' in 'GeeksforGeeks' is 'forGeeks'

========== 2

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

int main()
```

```
{
    // Take any two strings
    char s1[] = "Fun with STL";
    char s2[] = "STL";
    char* p;

    // Find first occurence of s2 in s1
    p = strstr(s1, s2);

    // Prints the result
    if (p) {
        strcpy(p, "Strings");
        printf("%s", s1);
    } else
        printf("String not found\n");
    return 0;
}
```

Output: Fun with Strings

strtok()

C provides two functions strtok() and strtok_r() for splitting a string by some delimiter. Splitting a string is a very common task. For example, we have a comma separated list of items from a file and we want individual items in an array.

Syntax : char * strtok(char str[], const char *delims);

```
#include <stdio.h>
#include <string.h>
int main()
{
    char str[] = "Geeks-for-Geeks";
    // Returns first token
    char* token = strtok(str, "-");
    // Keep printing tokens while one of the
    // delimiters present in str[].
    while (token != NULL) {
        printf("%s\n", token);
        token = strtok(NULL, "-");
    }

    return 0;
}
```

Output:

Geeks
for
Geeks