# Programming in C

## ( String manipulation )

*Prepared By*

Alok Haldar

Assistant professor

Department of Computer Science & BCA

Kharagpur College

# C Programming Strings

In C programming, a string is a sequence of characters terminated with a null character `\0`. For example:

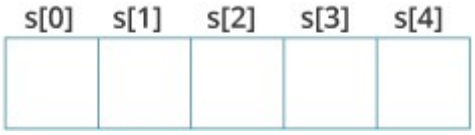
```
char c[] = "c string";
```

When the compiler encounters a sequence of characters enclosed in the double quotation marks, it appends a null character `\0` at the end by default.


## How to declare a string?

Here's how you can declare strings:

```
char s[5];
```



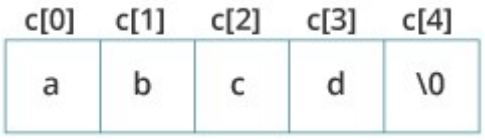

Here, we have declared a string of 5 characters.

```
char c[] = "abcd";

char c[50] = "abcd";

char c[] = {'a', 'b', 'c', 'd', '\0'};

char c[5] = {'a', 'b', 'c', 'd', '\0'};
```




Let's take another example:

```
char c[5] = "abcde";
```

Here, we are trying to assign 6 characters (the last character is `'\0'`) to a `char` array having 5 characters.

# Assigning Values to Strings

Arrays and strings are second-class citizens in C; they do not support the assignment operator once it is declared. For example,

```
char c[100];
c = "C programming";
```

# Read String from the user

You can use the `scanf()` function to read a string.

The `scanf()` function reads the sequence of characters until it encounters whitespace (space, newline, tab, etc.).

## Example 1: scanf() to read a string

```
#include <stdio.h>
int main()
{
    char name[20];
    printf("Enter name: ");
    scanf("%s", name);
    printf("Your name is %s.", name);
    return 0;
}
```

**Output:**

```
Enter name: Dennis Ritchie
Your name is Dennis.
```

## gets() and puts()

Functions gets() and puts() are two string functions to take string input from the user and display it respectively.

```
#include<stdio.h>

int main()
{
    char name[30];
    printf("Enter name: ");
    gets(name);     //Function to read string from user.
    printf("Name: ");
    puts(name);    //Function to display string.
    return 0;
}
```

## How to read a line of text?

You can use the `fgets()` function to read a line of string. And, you can use `puts()` to display the string.

### Example 2: fgets() and puts()

```
#include <stdio.h>
int main()
{
    char name[30];
    printf("Enter name: ");
    fgets(name, sizeof(name), stdin);  // read string
    printf("Name: ");
    puts(name);    // display string
    return 0;
}
```

**Output**

```
Enter name: Tom Hanks
Name: Tom Hanks
```

Here, we have used `fgets()` function to read a string from the user.

```
fgets(name, sizeof(name), stdlin); // read string
```

The `sizeof(name)` results to 30. Hence, we can take a maximum of 30 characters as input which is the size of the *name* string.

To print the string, we have used `puts(name);`.

**Note:** The `gets()` function can also be to take input from the user. However, it is removed from the C standard.

It's because `gets()` allows you to input any length of characters. Hence, there might be a buffer overflow.

## Passing Strings to Functions

Strings can be passed to a function in a similar way as arrays. Learn more about passing arrays to a function.

### Example 3: Passing string to a Function

```
#include <stdio.h>
void displayString(char str[]);

int main()
{
    char str[50];
    printf("Enter string: ");
    fgets(str, sizeof(str), stdin);
    displayString(str);     // Passing string to a function.
    return 0;
}
void displayString(char str[])
{
    printf("String Output: ");
    puts(str);
```

```
}
```

## String Manipulations In C Programming Using Library Functions

In this article, you'll learn to manipulate strings in C using library functions such as gets(), puts, strlen() and more. You'll learn to get string from the user and perform operations on the string.

You need to often manipulate strings according to the need of a problem. Most, if not all, of the time string manipulation can be done manually but, this makes programming complex and large.

To solve this, C supports a large number of string handling functions in the standard library `"string.h"`.

Few commonly used string handling functions are discussed below:

| Function | Work of Function |
|---|---|
| strlen() | computes string's length |
| strcpy() | copies a string to another |
| strcat() | concatenates(joins) two strings |
| strcmp() | compares two strings |
| strlwr() | converts string to lowercase |
| strupr() | converts string to uppercase |

Strings handling functions are defined under `"string.h"` header file.

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

**Note:** You have to include the code below to run string handling functions.

## C strcat()

In C programming, the strcat() function contcatenates (joins) two strings.

The function definition of `strcat()` is:

```
char *strcat(char *destination, const char *source)
```

It is defined in the `string.h` header file.

## strcat() arguments

As you can see, the `strcat()` function takes two arguments:

**destination** - destination string
**source** - source string

The `strcat()` function concatenates the `destination` string and the `source` string, and the result is stored in the `destination` string.

### Example: C strcat() function

```
#include <stdio.h>
#include <string.h>
int main() {
   char str1[100] = "This is ", str2[] = "programiz.com";
```

```
   // concatenates str1 and str2
   // the resultant string is stored in str1.
   strcat(str1, str2);

   puts(str1);
   puts(str2);

   return 0;
}
```

**Output :**

```
This is programiz.com
programiz.com
```

# C strcpy() :

In this tutorial, you will learn to use the strcpy() function in C programming to copy strings (with the help of an example).

# C strcpy() :

The function prototype of `strcpy()` is:

```
char* strcpy(char* destination, const char* source);
```

**The `strcpy()` function copies the string pointed by *source* (including the null character) to the destination.**

- The `strcpy()` function also returns the copied string.

**The `strcpy()` function is defined in the *string.h* header file.**

## Example: C strcpy()

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

int main() {
   char str1[20] = "C programming";
   char str2[20];

   // copying str1 to str2
   strcpy(str2, str1);

   puts(str2); // C programming

   return 0;
}
```

**Output :**

```
C programming
```

## C strlen()

The strlen() function calculates the length of a given string.

The `strlen()` function takes a string as an argument and returns its length. The returned value is of type `size_t` (the unsigned integer type).

It is defined in the <string.h> header file.

## Example: C strlen() function

```
#include <stdio.h>
#include <string.h>
int main()
{
    char a[20]="Program";
    char b[20]={'P','r','o','g','r','a','m','\0'};

    // using the %zu format specifier to print size_t
    printf("Length of string a = %zu \n",strlen(a));
    printf("Length of string b = %zu \n",strlen(b));

    return 0;
}
```

**Output :**

```
Length of string a = 7
Length of string b = 7
```

## C strcmp()

In this tutorial, you will learn to compare two strings using the strcmp() function.

**The `strcmp()` compares two strings character by character. If the strings are equal, the function returns 0.**

# C strcmp() Prototype

The function prototype of `strcmp()` is:

```
int strcmp (const char* str1, const char* str2);
```

# strcmp() Parameters

The function takes two parameters:

- **str1** - a string
- **str2** - a string

# Return Value from strcmp()

| Return Value | Remarks |
|---|---|
| 0 | if strings are equal |
| non-zero | if strings are not equal |

**The `strcmp()` function is defined in the `string.h` header file.**

## Example: C strcmp() function

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

int main() {
  char str1[] = "abcd", str2[] = "abCd", str3[] = "abcd";
  int result;

  // comparing strings str1 and str2
  result = strcmp(str1, str2);
  printf("strcmp(str1, str2) = %d\n", result);

  // comparing strings str1 and str3
  result = strcmp(str1, str3);
  printf("strcmp(str1, str3) = %d\n", result);

  return 0;
}
```

**Output :**

```
strcmp(str1, str2) = 1
strcmp(str1, str3) = 0
```

**In the program,**

- strings `str1` and `str2` are not equal. Hence, the result is a non-zero integer.
- strings `str1` and `str3` are equal. Hence, the result is 0.