# Strings

A C string is a set of characters,terminated by **null** character '\0'. Each Character Occupies 1 byte of Memory and are stored in consecutive memory location.

## syntax

**char stringname[size].**

**eg. char str[20];**

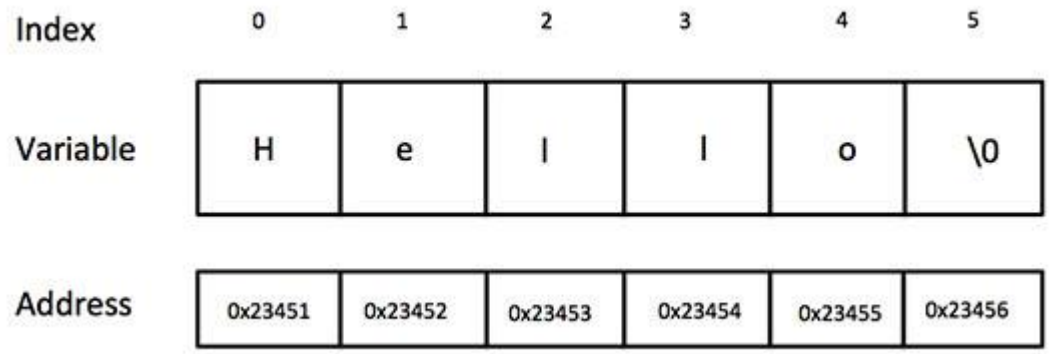
## String Initialization:

**Method 1:**

char address[]={'H', 'e', 'l', 'l','o','\0'};

**Method 2: The above string can also be defined as**

char address[]="Hello";




In the above declaration NULL character (\0) will automatically be inserted at the end of the string.

**What is NULL Char "\0"?**
'\0' represents the end of the string. It is also referred as String terminator or Null Character.The ASCII value of NULL is zero

## Character processing

**1. Read and Write characters Using Formatted I/O**

**Printf and Scanf –**

**Program:**

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

```
    char chr;
    printf("Enter a character: ");
    scanf("%c",&chr);
    printf("You entered %c.",chr);
    return 0;
}
```

**Output:**

**Enter a character: m**

**You entered m.**

## String Processing :

**Input and output operations:**

**1. Reading & Writing Strings Using Formatted I/O**

**Printf and Scanf –**

**Program1:**

```
#include <stdio.h>
#include <string.h>
int main()
{
    /* String Declaration*/
    char name[20];

    printf("Enter your name:");

    /* reading the input string and storing it in name*/
    scanf("%s", name);

    /*Displaying String*/
    printf("%s",name);

    return 0;
}
```

**Output**

**Enter your name: PES**

**PES**

**Note1: Observe the above output .Input PES is given after prefixing some spaces .scanf skips all leading spaces and accepts only string**

**Output**

**Enter your name:PES UNIVERSITY**

**PES**

**Observe the above output, scanf statement stops reading the input when it encounters whitespace**

**Note2: %s is the format specifier to read the successive characters till it reaches whitespace .**

**Note3: Base address is sufficient for string %s,it increments internally and reads successive characters. No need to specify address operator '&' in scanf statement.**

## Variations of scanf statements.

scanf("%*s%*s%*s%s",s);

**input: a b c d**

**output:d**

scanf("%[a-z]",s);

**input: abc123**

**output:abc**

scanf("%[a-z0-9]",s);

**input:abc123**

**output:abc123**

scanf("%[a-z0-9 ]",s);

**input:abc 123**

**output:abc 123**

scanf("%[^a-z]",s);

**NOTE: ^ - perplex character**

**input: 45dsf@]e**

**output:45**

## 2. Reading & Writing Strings Using Unformatted I/O

## gets() and puts()

**char *gets(char *str)**

**R**eads a line from stdin and stores it into the string pointed to by str. It stops when either the newline character is read or when the end-of-file is reached, whichever comes first.

**int puts(const char *str)**

Writes a string to stdout up to but not including the null character. A newline character is appended to the output.

### program:

```
#include <stdio.h>
#include <string.h>
int main()
{
   /* String Declaration*/
   char name[20];

   /* Console display using puts */
   puts("Enter your name:");

   /*Input using gets*/
   gets(name);

   puts(name);

   return 0;
}
```

**Output**

**Enter your name:PES UNIVERSITY**

**PES UNIVERSITY**

**Output**

**Enter your name:**

**PES**

**PES**

**Note1: Observe the above output gets() accepts whitespace as a character and reads into the string variable. Leading spaces are preserved.**

**Note2: gets() is dangerous to use in the program .you can see the warning when you use gets() in your program. because it maintains buffer of limited capacity, if given input string exceeds the maximum size of the buffer we may loose the input data.**

This function is avoided to use in the program which accepts huge data.for classroom execution and smaller projects still gets() can be used.

## getchar() & putchar()

**int getchar(void);**

Reads a single character of input and returns that character as the value of the function. If there is an error reading the character, or if the end of input is reached, getchar() returns a special value, represented by "EOF".

**int putchar(int c);**

writes a string to stdout up to but not including the null character. A newline character is appended to the output.

## Program1:

```
#include <stdio.h>

int main ()
{
  char c;

  printf("Enter character: ");
  c = getchar();

  printf("Character entered: ");
  putchar(c);

  return(0);
}
```

**output:**

**Enter character: b**

**Character entered: b**

## Consider the following program

## program2:

```
#include <stdio.h>
int main()
{
   char name[30], ch;
   int i = 0;
   printf("Enter name: ");
   for(i=0;i<3;i++)
   {
   name[i]=getchar();
   printf(“HELLO”);
   }
   name[i] = '\0';       // safe operation inserting null character at end
   printf("Name: %s\n",name);
   return 0;
}
```

The above program is suppose to allow the user to enter one character at a time at each iteration.

**Output:**

**Enter name: abc**

**HELLO**

**HELLO**

**HELLO**

**Name: abc**

**Explaination:Output looks strange!!!!**

So, if you compile/run this, and type in 'abc', it will just take each one character, and send it through the loop. The requirement is only to take the very *first* character that someone types in, no matter how many they do type in**.**

Problem here is because of keyboard buffer,Input entered by the user is stored in keyboard

buffer.When the loop executes,it looks for the availablity of data in then buffer,if present ,input is fetched directly from the keyboard buffer instead of prompting the user for furthur inputs.

When, i =0 a is read from keyboard buffer

i=1 b

i=2 c

**Hence the output is abc.**

**The solution to the above problem is to call __fpurge(stdin) to clear the contents of the keyboard buffer.**

**Ubuntu/Linux Operating system supports __fpurge(STDIN) function defined in include header file stdio_ext.h**

**Windows Operating system supports fflush(STDIN).**

## program3:

```
#include <stdio.h>
#include<stdio_ext.h>
int main()
{
   char name[30], ch;
   int i = 0;
   printf("Enter name: ");
   for(i=0;i<3;i++)
   {
    __fpurge(stdin);
   name[i]=getchar();
   }
   name[i] = '\0';       // safe operation inserting null character at end
   printf("Name: %s\n", name);
   return 0;
}
```

**Output:**

**Enter name:**

**abc**

**egf**

**dasd**

**Name: aed**

## String Handling Functions:

**1.strlen(size_t )**

where size_t represents unsigned short
It returns the length of the string without including end character **(terminating char '\0')**.

**Example:**

```
#include <stdio.h>
#include <string.h>
int main()
{
    char str1[20] = "PESUNIVERSITY";
    printf("Length of string str1: %d", strlen(str1));
    return 0;
}
```

**Output:**

**Length of string str1: 13**

### strlen vs sizeof

strlen returns you the length of the string stored in array, however sizeof returns the total allocated size assigned to the array. So if I consider the above example again then the following statements would return the below values.

strlen(str1) returned value 13.
sizeof(str1) would return value 20 as the array size is 20 (see the first statement in main function).

**2.strnlen function:** size_t strnlen(const char *str, size_t maxlen)

size_t represents unsigned short
It returns length of the string if it is less than the value specified for maxlen (maximum length) otherwise it returns maxlen value.

**Example:**

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

```
    char str1[20] = "PESUNIVERSITY";
    printf("Length of string str1 when maxlen is 30: %d", strnlen(str1, 30));
    printf("Length of string str1 when maxlen is 10: %d", strnlen(str1, 10));
    return 0;
}
```

**Output:**

Length of string str1 when maxlen is 30: 13

Length of string str1 when maxlen is 10: 10

Have you noticed the output of second printf statement, even though the string length was 13 it returned only 10 because the maxlen was 10.

## 3.strcmp function:

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

It compares the two strings and returns an integer value. If both the strings are same (equal) then this function would return 0 otherwise it may return a negative or positive value based on the comparison.

Function takes two Strings as parameter. It returns integer.

Return Values

| Return values | Condition |
|---|---|
| -ve Value | String1 < String2 |
| +ve Value | String1 > String2 |
| 0 Value | String1 = String2 |

**Example:**

```
#include <stdio.h>
#include <string.h>
int main()
{
    char s1[20] = "PESUNIVERSITY";
    char s2[20] = "PESUNIVERSITY.COM";
    if (strcmp(s1, s2) ==0)
    {
      printf("string 1 and string 2 are equal");
    }else
     {
       printf("string 1 and 2 are different");
     }
    return 0;
}
```

**Output:**

string 1 and 2 are different

## 4.strncmp function:

**int strncmp(const char *str1, const char *str2, size_t n)**

size_t is for unassigned short
It compares both the string till n characters or in other words it compares first n characters of both the strings.

**Example:**

```
#include <stdio.h>
#include <string.h>
int main()
{
    char s1[20] = "PESUNIVERSITY";
    char s2[20] = "PESUNIVERSITY.COM";
    /* below it is comparing first 8 characters of s1 and s2*/
    if (strncmp(s1, s2, 8) ==0)
    {
       printf("string 1 and string 2 are equal");
    }else
    {
       printf("string 1 and 2 are different");
    }
    return 0;
}
```

**Output:**

string1 and string 2 are equal

## 5.strcat function:

char *strcat(char *str1, char *str2)

It concatenates two strings and returns the combined one string.

**Example:**

```
#include <stdio.h>
#include <string.h>
int main()
{
    char s1[10] = "Hello";
    char s2[10] = "World";
    strcat(s1,s2);
    printf("Output string after concatenation: %s", s1);
    return 0;
```

```
}
```

**Output:**

Output string after concatenation: HelloWorld

## 6.strncat function:

char *strncat(char *str1, char *str2, int n)

It concatenates n characters of str2 to string str1. A terminator char (‘\0’) will always be appended at the end of the concatenated string.

**Example:**

```
#include <stdio.h>
#include <string.h>
int main()
{
    char s1[10] = "Hello";
    char s2[10] = "World";
    strncat(s1,s2, 3);
    printf("Concatenation using strncat: %s", s1);
    return 0;
}
```

**Output:**

Concatenation using strncat: HelloWor

## 7.strcpy function:

**char *strcpy( char *str1, char *str2)**

It copies the string str2 into string str1, including the end character (terminator char ‘\0’).

**Example:**

```
#include <stdio.h>
#include <string.h>
int main()
{
    char s1[30] = "string 1";
    char s2[30] = "string 2 : I’m gonna copied into s1";
    /* this function has copied s2 into s1*/
    strcpy(s1,s2);
    printf("String s1 is: %s", s1);
    return 0;
}
```

**Output:**

String s1 is: string 2: I’m gonna copied into s1

## 8.Strncpy function:

**char *strncpy( char *str1, char *str2, size_t n)**

size_t is unassigned short and n is a number.

**Case1:** If length of str2 > n then it just copies first n characters of str2 into str1.

**Case2:** If length of str2 < n then it copies all the characters of str2 into str1 and appends several terminator chars('\0') to accumulate the length of str1 to make it n.

**Example:**

```
#include <stdio.h>
#include <string.h>
int main()
{
    char first[30] = "string 1";
    char second[30] = "string 2: I'm using strncpy now";
    /* this function has copied first 10 chars of s2 into s1*/
    strncpy(s1,s2, 12);
    printf("String s1 is: %s", s1);
    return 0;
}
```

**Output:**

String s1 is: string 2: I'm

## 9.strchr function:

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

It searches string str for character ch data type of ch as int, The thing is when we give any character while using strchr then it internally gets converted into integer for better searching.

**Example:**

```
#include <stdio.h>
#include <string.h>
int main()
{
    char mystr[30] = "I'm an example of function strchr";
    printf ("%s", strchr(mystr, 'f'));
    return 0;
}
```

**Output2:**

f function strchr

**Note:If character not present in the given string it returns NIL**

## 10.Strrchr function:

**char *strrchr(char *str, int ch)**

It is similar to the function strchr, the only difference is that it searches the string in reverse order,

Now let's take the same above example:

```
#include <stdio.h>
#include <string.h>
int main()
{
    char mystr[30] = "I'm an example of function strchr";
    printf ("%s", strrchr(mystr, 'f'));
    return 0;
}
```

**Output:**

**function strchr**

**Note:If character not present in the given string it returns NIL**

**Why output is different than strchr?** It is because it started searching from the end of the string and found the first 'f' in function instead of 'of'.

## 11.strstr function:

**char *strstr(char *str, char *srch_term)**

It is similar to strchr, except that it searches for string srch_term instead of a single char.

**Example:**

```
#include <stdio.h>
#include <string.h>
int main()
{
    char inputstr[70] = "String Function in C at PESUNIVERSITY.COM";
    printf ("Output string is: %s", strstr(inputstr, 'PES'));
    return 0;
}
```

**Output:**

**Output string is: PESUNIVERSITY.COM**

## 12. Strtok function:

**char *strtok(char *str, const char *delim);**

The strtok() function breaks a string into a sequence of zero or more nonempty tokens. On the first call to strtok() the string to be parsed should be specified in str. In each subsequent call that should parse the same string, str must be NULL.

The delim argument specifies a set of bytes that delimit the tokens in the parsed string. The caller may specify different strings in delim in successive calls that parse the same string.

```
#include <string.h>

'#include <stdio.h>

#include<stdio.h>

int main(){

  char str[80] = "c-python-physics-chemistry-maths-mechanical";

  const char s[2] = "-";

  char *token;

  /* get the first token */

   printf("input string is %s\n",str);

   printf("first token %s\n", strtok(str, s));// first token "c"

  /*get subsequent token*/

   token = strtok(NULL, s);

   printf( "second token %s\n", token );//second token "python"

   token = strtok(NULL, s);

   printf( "third token %s\n", token );//third token "java"

  // inspite of having repeated c statements to

  // get subsequent tokens use while loop

  // until pointer reaches end of string i.e.,'\0'

  /* walk through other tokens */

   while( token != NULL )

    {

     printf( " %s\n", token );
```

```
    token = strtok(NULL, s);
   }
return(0);
}
```

**Output:**

**input string is c-python-physics-chemistry-maths-mechanical**

**first token c**

**second token python**

**third token physics**

**physics**

**chemistry**

**maths**

**mechanical**