**UNIT 4**

Strings are actually one-dimensional array of characters terminated by a **null** character '\0'. Thus a null-terminated string contains the characters that comprise the string followed by a **null**.

The following declaration and initialization create a string consisting of the word "Hello". To hold the null character at the end of the array, the size of the character array containing the string is one more than the number of characters in the word "Hello."

char greeting[6] = {'H', 'e', 'l', 'l', 'o', '\0'};

If you follow the rule of array initialization then you can write the above statement as follows −

char greeting[] = "Hello";

Following is the memory presentation of the above defined string in C/C++ −



Actually, you do not place the *null* character at the end of a string constant. The C compiler automatically places the '\0' at the end of the string when it initializes the array. Let us try to print the above mentioned string −

[Live Demo](http://tpcg.io/P0muN5)

#include <stdio.h>

int main () {

 char greeting[6] = {'H', 'e', 'l', 'l', 'o', '\0'};

 printf("Greeting message: %s\n", greeting );

 return 0;

}

When the above code is compiled and executed, it produces the following result −

Greeting message: Hello

C supports a wide range of functions that manipulate null-terminated strings −

|  |  |
| --- | --- |
| **Sr.No.** | **Function & Purpose** |
| 1 | **strcpy(s1, s2);**Copies string s2 into string s1. |
| 2 | **strcat(s1, s2);**Concatenates string s2 onto the end of string s1. |
| 3 | **strlen(s1);**Returns the length of string s1. |
| 4 | **strcmp(s1, s2);**Returns 0 if s1 and s2 are the same; less than 0 if s1<s2; greater than 0 if s1>s2. |
| 5 | **strchr(s1, ch);**Returns a pointer to the first occurrence of character ch in string s1. |
| 6 | **strstr(s1, s2);**Returns a pointer to the first occurrence of string s2 in string s1. |

#include <stdio.h>

#include <string.h>

int main () {

 char str1[12] = "Hello";

 char str2[12] = "World";

 char str3[12];

 int len ;

 /\* copy str1 into str3 \*/

 strcpy(str3, str1);

 printf("strcpy( str3, str1) : %s\n", str3 );

 /\* concatenates str1 and str2 \*/

 strcat( str1, str2);

 printf("strcat( str1, str2): %s\n", str1 );

 /\* total lenghth of str1 after concatenation \*/

 len = strlen(str1);

 printf("strlen(str1) : %d\n", len );

 return 0;

}

When the above code is compiled and executed, it produces the following result −

**Output**

strcpy( str3, str1) : Hello

strcat( str1, str2): HelloWorld

strlen(str1) : 10

## Declaration

Refer to the declaration given below −

char stringname [size];

For example - char a[50]; a string of length 50 characters.

## Initialization

The initialization is as follows −

* Using **single character** constant −

char string[20] = { ‘H’, ‘i’, ‘l’, ‘l’, ‘s’ ,‘\0’}



* Using string constants −

char string[20] = "Hello":;



* ‘\0’ is called a null character. It marks the end of the string.
* ‘\0’ is automatically placed by the compiler, if a string is given as input. The user has to take care of placing ‘\0’ at the end if a single character is given.

**Accessing** − There is a control string "%s" used for accessing the string, till it encounters ‘\0’.

## Example

Following is the C program for a string −

#include<stdio.h>

main ( ){

   char a[10] = "Hello";

   clrscr ( );

   printf ( " given string is %s",a)

   getch ( );

}

## Output

When the above program is executed, it produces the following result −

Given string is Hello

Reading Strings From Terminal

• Using scanf function

An example for reading a string is given below:

char name[10];

scanf("%s", name);

• scanf terminates the input on the first space it finds. i.e if we enter <NEW

YORK= as input, only <NEW= will be read into the array name.

• scanf function automatically terminates the string that is read, with a null

character and therefore the character array should be large enough to hold

the input string plus the null character.

• No <&= sign is required before the variable name while reading a string

using scanf function

• In the above example if we give <New= as input string the name array is

created in the memory as shown below:

N

e

w

\0

?

?

?

?

?

?

Unused locations will be filled with garbage.

• Using gets Function

A string containing white spaces can be read from the keyboard using gets

function. For example:

char line[50];

gets(line);

This will read a line of text from the keyboard.

Reading Strings From Terminal

• Using scanf function

An example for reading a string is given below:

char name[10];

scanf("%s", name);

• scanf terminates the input on the first space it finds. i.e if we enter <NEW

YORK= as input, only <NEW= will be read into the array name.

• scanf function automatically terminates the string that is read, with a null

character and therefore the character array should be large enough to hold

the input string plus the null character.

• No <&= sign is required before the variable name while reading a string

using scanf function

• In the above example if we give <New= as input string the name array is

created in the memory as shown below:

N

e

w

\0

?

?

?

?

?

?

Unused locations will be filled with garbage.

• Using gets Function

A string containing white spaces can be read from the keyboard using gets

function. For example:

char line[50];

gets(line);

This will read a line of text from the keyboard.

Reading Strings From Terminal

• Using scanf function

An example for reading a string is given below:

char name[10];

scanf("%s", name);

• scanf terminates the input on the first space it finds. i.e if we enter <NEW

YORK= as input, only <NEW= will be read into the array name.

• scanf function automatically terminates the string that is read, with a null

character and therefore the character array should be large enough to hold

the input string plus the null character.

• No <&= sign is required before the variable name while reading a string

using scanf function

• In the above example if we give <New= as input string the name array is

created in the memory as shown below:

N

e

w

\0

?

?

?

?

?

?

Unused locations will be filled with garbage.

• Using gets Function

A string containing white spaces can be read from the keyboard using gets

function. For example:

char line[50];

gets(line);

This will read a line of text from the keyboard.

Reading Strings From Terminal

• Using scanf function

An example for reading a string is given below:

char name[10];

scanf("%s", name);

• scanf terminates the input on the first space it finds. i.e if we enter <NEW

YORK= as input, only <NEW= will be read into the array name.

• scanf function automatically terminates the string that is read, with a null

character and therefore the character array should be large enough to hold

the input string plus the null character.

• No <&= sign is required before the variable name while reading a string

using scanf function

• In the above example if we give <New= as input string the name array is

created in the memory as shown below:

N

e

w

\0

?

?

?

?

?

?

Unused locations will be filled with garbage.

## 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](https://stackoverflow.com/questions/30033582/what-is-the-symbol-for-whitespace-in-c) (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.**

###  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: f gets() 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

**Arithmetic operators**

**Arithmetic operators** are used to perform arithmetic/mathematical operations on operands. For instance, the symbols “+” and “-” are used for addition and subtraction, respectively, while “\*” is used for multiplication on numeric values

Based on the number of operands, C Arithmetic operators can be of two types:

1. **Binary Arithmetic Operators**
2. **Unary Arithmetic Operators**

**1. Binary Arithmetic Operators**

The binary arithmetic operators operate or work on two operands. C provides**5** Binary Arithmetic Operators for performing arithmetic functions which are as follows:

| **Operator** | **Name of the Operator** | **Arithmetic Operation** | **Syntax** |
| --- | --- | --- | --- |
| **+** | Addition | Add two operands. | x **+**y |
| **–** | Subtraction | Subtract the second operand from the first operand. | x **–** y |
| **\*** | Multiplication | Multiply two operands. | x **\*** y |
| **/** | Division | Divide the first operand by the second operand. | x **/**y |
| **%** | [Modulus](https://www.geeksforgeeks.org/modulo-operator-in-c-cpp-with-examples/) | Calculate the remainder when the first operand is divided by the second operand. | x **%** y |

* C

|  |
| --- |
| // C program to demonstrate // working of binary arithmetic// operators#include <stdio.h>  int main(){    int a = 10, b = 4, res;      // printing a and b    printf("a is %d and b is %d\n", a, b);      res = a + b; // addition    printf("a + b is %d\n", res);      res = a - b; // subtraction    printf("a - b is %d\n", res);      res = a \* b; // multiplication    printf("a \* b is %d\n", res);      res = a / b; // division    printf("a / b is %d\n", res);      res = a % b; // modulus    printf("a %% b is %d\n", res);      return 0;} |

**Output**

a is 10 and b is 4

a + b is 14

a - b is 6

a \* b is 40

a / b is 2

a % b is 2

## **2. Unary Arithmetic Operators**

The unary arithmetic operators operate or work with a single operand. In C, we have two unary arithmetic operators which are as follows:

| **Operator** | **Symbol** | **Operation** | **Implementation** |
| --- | --- | --- | --- |
| Decrement Operator | — | Decreases the integer value of the variable by one. | –h or h– |
| Increment Operator | ++ | Increases the integer value of the variable by one. | ++h or h++ |

**Increment:** The **‘++’** operator is used to increment the value of an integer. When placed before the variable name (also called the [pre-increment](https://www.geeksforgeeks.org/pre-increment-and-post-increment-in-c/) operator), its value is incremented instantly. For example, **++x**.

And when it is placed after the variable name (also called [post-increment](https://www.geeksforgeeks.org/pre-increment-and-post-increment-in-c/) operator), its value is preserved temporarily until the execution of this statement and it gets updated before the execution of the next statement. For example, **x++**.

**Decrement:** The **‘ – – ‘** operator is used to decrement the value of an integer. When placed before the variable name (also called the **pre-decrement** operator), its value is decremented instantly. For example, **– – x**.

And when it is placed after the variable name (also called **post-decrement** operator), its value is preserved temporarily until the execution of this statement and it gets updated before the execution of the next statement. For example, **x – –**.

**Example:**

* C

|  |
| --- |
| // C program to demonstrate working// of Unary arithmetic// operators#include <stdio.h>  int main(){    int a = 10, b = 4, res;      printf("Post Increment and Decrement\n");    // post-increment example:    // res is assigned 10 only, a is not updated yet    res = a++;    printf("a is %d and res is %d\n", a,           res); // a becomes 11 now      // post-decrement example:    // res is assigned 11 only, a is not updated yet    res = a--;    printf("a is %d and res is %d\n", a,           res); // a becomes 10 now      printf("\nPre Increment and Decrement\n");    // pre-increment example:    // res is assigned 11 now since    // a is updated here itself    res = ++a;      // a and res have same values = 11    printf("a is %d and res is %d\n", a, res);      // pre-decrement example:    // res is assigned 10 only since a is updated here    // itself    res = --a;      // a and res have same values = 10    printf("a is %d and res is %d\n", a, res);      return 0;} |

**Output**

Post Increment and Decrement

a is 11 and res is 10

a is 10 and res is 11

Pre Increment and Decrement

a is 11 and res is 11

a is 10 and res is 10

**Character arithmetic**

Character arithmetic is used to implement arithmetic operations like addition, subtraction ,multiplication ,division on characters in C and C++ language.
In character arithmetic character converts into integer value to perform task. For this ASCII value is used.

It is used to perform action the strings.

#include <stdio.h>

int main()

{

 char ch1 = 125, ch2 = 10;

 ch1 = ch1 + ch2;

 printf("%d\n", ch1);

 printf("%c\n", ch1 - ch2 - 4);

 return 0;

}

**Output**

-121

y

# Compare the two strings

Strings can be compared either by using the string function or without using string function. First, we will look at how we can compare the strings with the help of string function, i.e., **strcmp(),** which is defined in a **string.h** header file.

### String comparison by using string function

The string function which is pre-defined in a **string.h** header file is a **strcmp()** function. The strcmp() function consider two strings as a parameter, and this function returns an integer value where the integer value can be **zero**, **positive** or **negative**.

### The syntax of the strcmp() function is given below:

1. **int** strcmp (**const** **char**\* str1, **const** **char**\* str2);

In the above syntax, two parameters are passed as strings, i.e., **str1** and **str2**, and the return type is **int** means that the strcmp() returns an integer value.

The strcmp() function compares the character of both the strings. If the first character of both the strings are same, then this process of comparison will continue until all the characters are compared or the pointer points to the null character '\0'.

**String-handling Functions**

The string handling functions are defined in a header file called **string.h**. Whenever we want to use any string handling function we must include the header file called **string.h**.

The following table provides most commonly used string handling function and their use...

|  |  |  |
| --- | --- | --- |
| **Function** | **Syntax (or) Example** | **Description** |
| **strcpy()** | strcpy(string1, string2) | Copies string2 value into string1 |
| **strncpy()** | strncpy(string1, string2, 5) | Copies first 5 characters string2 into string1 |
| **strlen()** | strlen(string1) | returns total number of characters in string1 |
| **strcat()** | strcat(string1,string2) | Appends string2 to string1 |
| **strncat()** | strncpy(string1, string2, 4) | Appends first 4 characters of string2 to string1 |
| **strcmp()** | strcmp(string1, string2) | Returns 0 if string1 and string2 are the same;less than 0 if string1<string2; greater than 0 if string1>string2 |
| **strncmp()** | strncmp(string1, string2, 4) | Compares first 4 characters of both string1 and string2 |
| **strcmpi()** | strcmpi(string1,string2) | Compares two strings, string1 and string2 by ignoring case (upper or lower) |
| **stricmp()** | stricmp(string1, string2) | Compares two strings, string1 and string2 by ignoring case (similar to strcmpi()) |

1. **Write a Program in C to handle any 3 string functions**

#include<stdio.h>

#include<string.h>

main()

{

char a[10],b[10];

int ch,len;

printf("enter str1 ");

scanf("%s",a);

printf("enter str2 ");

scanf("%s",b);

while(1)

{

printf("\n choose ur option");

printf("\n 1.length\n 2.compare\n 3.copy\n 4.concat\n");

printf("enter ur choice: ");

scanf("%d",&ch);

switch(ch)

{

case 1: len=strlen(a);

printf("length is %d\n",len);

break;

case 2:if(strcmp(a,b)==0)

{

printf("both strings are equal\n");

}

else

if(strcmp(a,b)>0)

printf("%s is greater than %s\n",a,b);

else

printf("%s is greater than %s\n",b,a);

break;

case 3: printf(" str1 %s\n",a);

printf("str2 %s\n",b);

strcpy(a,b);

printf("after copy strings are\n");

printf(" str1 %s\n",a);

printf("str2 %s\n",b);

break;

case 4:printf(" str1 %s\n",a);

printf("str2 %s\n",b);

strcat(a,b);

printf(" str1 %s\n",a);

break;

}

}

}

**2. Program to find the length of the string without using strcat().**

#include <string.h>

#include <stdio.h>

int main()

{

 int i,len=0,j;

 char str[50],str1[50],ch,ch1;

 printf("\nEnter the First String : " );

 gets(str);

 printf("\nEnter the Second String : " );

 gets(str1);

 for(i=0; str[i]!='\0'; i++)

 {

 len++;

 }

 for(i=0; str1[i]!='\0'; i++)

 {

 str[len++]=str1[i];

 }

 str[len]='\0';

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

 return 0;

}

OUTPUT

Enter the First String : Programming9

Enter the Second String : .com

Concatenated String is: Programming.com