Know Program

Different Methods to Read and Write String in C

C TUTORIAL

C PROGRAMS

String in C

➤ Read Display String in C

➤ 2D Array of Strings in C

➤ Find Length of String

➤ Copy Two Strings in C

➤ Concatenate Two Strings

➤ Compare two strings

➤ Reverse a String in C

➤ Palindrome string in C
➤ Uppercase to Lowercase
➤ Lowercase to Uppercase
➤ Remove all characters in String except alphabet
➤ Find Frequency of Characters
➤ Count Number of Words
➤ Count lines, words, & characters
➤ Count Vowel, Consonant, Digit, Space, Special Character
➤ String Pattern in C
➤ Copy Input to Output

There are several methods to read the strings in the C language. To display the string also there are many methods. In this tutorial, we will learn how to read and display string in the C language. What is the most preferred function to read a string in C? We will discuss them with the programs.

A string is an array of characters. It is terminated by the null character ('\0'). The collection of characters enclosed in double quotation marks is called a string constant.

Table Of Contents

. . .

Read the string in C

First, we will learn how to read the strings in C language. There are some predefined functions in C language to read the strings.

Read string in C using gets() and fgets() functions

Read the string using gets() and fgets() functions is a very popular way to read the array of characters i.e. string.

Read the strings in C using gets()

This is the most popular approach to read string is using gets() library function. The gets() function is defined inside “stdio.h” library. The gets() function takes the start address of an area of memory suitable to hold the input as a single parameter. The gets() function read a line (terminated by a newline character \n) from the input stream and make a null-terminates string out of it. It reads data until it finds a newline or end-of-file. the gets() function declaration is,

```
char* gets(char* strptr);
```

Program to see the demonstration of gets() function

```
#include<stdio.h>
int main()
{
  char str[100];
  printf("Enter a string: ");
  gets(str);
  printf("The Entered string : %s\n",str);
  return 0;
}
```

Output:-

```
Enter a string: Know Program C language
The Entered string: Know Program C language
```

. . .

Read string in C using fgets()

The fgets() function is also defined inside “stdio.h” library. It also takes a line (terminated by a newline) from the input stream and makes a null-terminates string out of it. The fgets() function declaration is,

```
char* fgets(char* strptr, int length, FILE * stream);
```

The fgets() function can get input from a file or standard input. The fgets() function requires two additional parameters: one specifying the array size that is available to receive the data, and the other a stream. It can be used with the keyword by specifying the ***stdin***.

```c
#include<stdio.h>
int main()
{
   char str[100];
   printf("Enter a string: ");
   fgets(str, sizeof(str), stdin);
   printf("The entered string: %s",str);
   return 0;
}
```

Output:-

```
Enter a string: Reading input from keyword
The entered string: Reading input from keyword
```

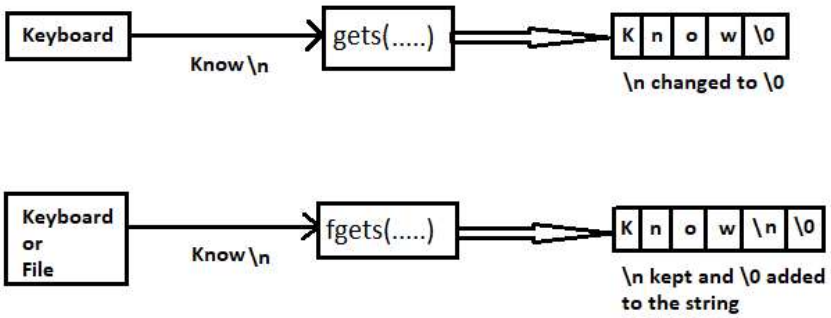
The most preferred function to read a string in C is fgets() function.

Comparison between gets() and fgets()

Both gets() and fgets() functions take a line (terminated by a newline) from the input stream and make a null-terminates string out of it. They are therefore sometimes called line-to-string input functions. Both accept a string pointer and return the same pointer if the input is successful. If any input problems occur, such as detecting end-of-file before reading any data, they return NULL.

The source of data for the gets() is standard input but the source of data for fgets() can be a file or standard input. To receive input from keyword using fgets() function we should use ***stdin***.

The gets() and fgets() do not work the same. The gets() function converts newline character to the end-of-string character (‘\0’) but fgets() don’t convert newline character, it puts a newline character in the string and appends an end-of-string character (‘\0’).




```c
#include<stdio.h>
int main()
{
   char str1[100], str2[100];
   printf("Enter two string: ");
   gets(str1);
   fgets(str2, sizeof(str2), stdin);

   // notice displaying order

   printf("String 1: %s ",str1);
   printf("String 2: %s",str2);
   return 0;
}
```

Output:-

```
Enter two string: Know Program
C language
String 1: Know Program String 2: C language
```

Now, Change the order of display. First, display str2 and later display str1. Observe the difference in the output.

```c
printf("String 2: %s",str2);
printf("String 1: %s",str1);
```

Output:-

```
Enter two string: Know Program
C language
```

```
String 2: C language
String 1: Know Program
```

Notice that we didn't use \n in the program but there is a newline after displaying String 2 because fgets() put a newline character in the string but gets() converted it to '\0'.

The C11 standard ISO/IEC 9899:2011 eliminated gets() as a standard function. But it will remain in libraries for many years (meaning 'decades') for reasons of backward compatibility. ***It is recommended to use fgets() instead of gets()*** function to read the string in C language. For more information read:- Why gets function is dangerous and should not be used

Read string in C using scanf()

In C language, the scanf() function is used to read primitive type data. The string is also the group of characters, so it also can be used to read the string. There are several ways and limitations for using the scanf() function to read the string.

Read string in C using scanf() with %s

The string may be read by using the %s conversion with the function scanf(), but there are some restrictions are there.

The String may be read by using the %s conversion with the function scanf(), but there are some restrictions are there.

1) An address operator is not required to read string since it is already a pointer. It is optional to use the address operator (&) to read string. If we use it then no error occurs.

```c
#include<stdio.h>
int main()
{
   char str[100];
   printf("Enter a string:");
   scanf("%s",&str);
   printf("String = %s",str);
   return 0;
}
```

Output:-

```
Enter String: Language
String = Language
```

2) The scanf() function read-only first word, when white space came then it stops reading. If we try to read "`Know Program`" then it will read-only "`Know`".

```c
#include<stdio.h>
int main()
{
   char str[100];
   printf("Enter String: ");
   scanf("%s",str);
   printf("String = %s", str);
   return 0;
}
```

Output:-

```
Enter String: Language
String = Language
```

```
Enter String: Know Program
String = Know
```

3) There should be enough space to read the string. If there is no enough space then the program will be terminated.

```c
#include<stdio.h>
int main()
{
   char str[5];
   printf("Enter String: ");
   scanf("%s",str);
   printf("String = %s", str);
   return 0;
}
```

```
Output:-

 Enter String: Language
 *** stack smashing detected ***:  terminated
 Aborted (core dumped)
```

In string, size should be large enough to store all data. If it is not large enough then it will destroy whatever follows the array in memory. Therefore, we must make sure we don't exceed the length of the data.

4) We can protect against the user entering too much data by using width in the field specification. We already discussed in detail about width:- width modifier using printf. The width specifies the maximum number of characters to be read.

```
#include<stdio.h>
int main()
{
   char str[5];
   printf("Enter a string: ");
   scanf("%4s",str);
   printf("String = %s",str);
   return 0;
}
```

Output:-

```
Enter a string: Programming
String = Prog
```

The string can read up to 4 characters, last for '\0'. It will not read more than 4 characters. The advantage is that if the user gives more characters than the size of string then the program will not terminate, it stops reading after reaching the size of the string.

. . .

Read string in C using scanf() with %c

An alternative method for the input of strings is to use scanf() with the %c conversion which may have a count associated with it. This conversion does not recognize the new-line character as special. The count specifies the number of characters to be read in. Unlike %s and %[ ] conversions, the %c conversion does not automatically generate the string terminating NULL and strange effects will be noted if the wrong number of characters is supplied.

```
#include<stdio.h>
int main()
{
   char str[10];
   printf("Enter a string of 9 characters: ");
   scanf("%10c",str);
   str[9]='\0';
   printf("String = %s\n",str);
   return 0;
}
```

It is not a good idea to use this method. The above program read exactly 9 characters if we try to give less than 9 characters than it will not give output until at least 9 character size is reached.

. . .

Read string in C using scanset conversion code ([…])

The scanset conversion facility provided by scanf() is a useful string input method. This conversion facility allows the programmer to specify the set of characters that are acceptable as part of the string. A scanset conversion consists of a list of acceptable characters enclosed within square brackets.

```c
#include<stdio.h>
int main()
{
   char str[100];
   printf("Enter a string in lower case: ");
   scanf("%[a-z]",str);
   printf("String = %s\n",str);
   return 0;
}
```

Output:-

```
Enter a string in lower case: knowProgram
String = know
```

```
Enter a string in lower case: know program
String = know
```

```
Enter a string in lower case: Know Program
String =
```

```
Enter a string in lower case: know9program
String = know
```

The above program only reads lowercase characters whenever it encounters another character except for lowercase charcater then it stops reading. In “knowProgram” it reads only “know” because the next character is in capital letter. When “know program” input is given then also it reads only “know” because space is a special character, not a lowercase character. In the third input-output “Know Program”, the first character is an uppercase character, so it will not read the string. If any digits came then also it will not read from the digit.

```
// program2:-
scanf("%[A-Z]",str);
```

Output:-

```
Enter a string: KnowProgram
String = K
```

It will read only uppercase characters, whenever other characters came then it will stop reading the characters.

```
//program3:-
scanf("%[a-z,A-Z]",str);
```

Output:-

```
Enter a string: Know9Program C Language
String = Know
```

```
Enter a string: KnowProgram C Language
String = KnowProgram
```

The above line will read all uppercase and lowercase characters. Except for these any other character encounters then scanf() will stop reading.

```
//program4:-
scanf("%[a-z,A-Z,0-9]",str);
```

Output:-

```
Enter a string: Know999Program C
String = Know999Program
```

The above program will read uppercase characters, lowercase characters, and digits. Except these whenever any other character is encountered then scanf() will stop reading.

Program-5

```c
#include<stdio.h>
int main()
{
   char str[100];
   printf("Enter a string: ");
   scanf("%[a-z,A-Z,0-9,' ']",str);
   printf("String = %s\n",str);
   return 0;
}
```

Output:-

```
Enter a string: Know Program9 C String
String = Know Program9 C String
```

In scanf() function the main program was to read space. By this program, we can read space also. Above program will read all lowercase, uppercase, digits and one white space (only one space) characters.

We see that by default scanf() function is not able to read a string with spaces. To avoid this problem scanset conversion is used.

. . .

Read string in C using scanset with [^\n] (single line)

The circumflex (^) plays an important role while taking input. [^\n] will read input until the user press the enter key, otherwise, it will keep reading input from the screen.

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

```
{
   char str[100];
   printf("Enter a string: ");
   scanf("%[^\n]",str);
   printf("String = %s\n",str);
   return 0;
}
```

Output:-

```
Enter a string: It will read until the user presses the enter key. By this way, we can read one line.
String = It will read until the user presses the enter key. By this way, we can read one line.
```

. . .

Multiline input using scanset

One can use a bracketed string read, %[…] where the square brackets are used to enclose all characters which are permissible in the input. If any character other than those listed within the brackets occurs in the input string, further reading is terminated.

```
#include<stdio.h>
int main()
{
   char str[100];
   printf("Enter a string, terminated with a star(*): ");
   scanf("%[^*]",str);
   printf("String = %s\n",str);
   return 0;
}
```

Output:-

```
Enter a string, terminated with a star(*): Welcome to the programming world*
String = Welcome to the programming world
```

```
Enter a string, terminated with a star(*): Hello * World
String = Hello
```

The above program will read input until the user enters the star * symbol. After * symbol, it will stop reading. In this way, we can read multiple lines of characters. It will read white space also, Only stop reading when a * symbol came. Otherwise, it will keep reading the input from the user.

```
[^*] :- It will stop reading only when a * is entered.
[^~] :- It will stop reading only when a tilde(~) is entered.
[^k] :- It will stop reading only when a
        lowecase character 'k' is entered.

[^9] :- It will stop reading only when the digit 9 is entered.
[^(0-9)]:-It will stop reading only when any digit is entered.
[^*~?] :- It will stop reading only when * or ~ or ? is entered.
```

Display string in C

There are various methods to display string in C language. We can use puts() or fputs() or printf() with %s format specifier.

Display string in C using puts() or fputs()

The library function puts()/fputs() writes a line of output to the standard output or a file. It terminates the line with a new line, '\n'.

The puts() function change '\0' to a new line but fputs() function doesn't change it. So, using puts() function control came to next line as printf("\n") do, but using fputs() control remains in same line. The declarations for these functions are:-

```
int puts(const char* strptr);
int fputs(const char* strptr, FILE* stream);
```

Demonstrating puts() function

```
#include<stdio.h>
int main()
{
   char str[100];
   printf("Enter a string: ");
   fgets(str, sizeof(str), stdin);
   puts(str);
   return 0;
}
```

Output:-

```
Enter a string: Know Program@999
Know Program@999
```

Demonstrating fputs() function

```
#include<stdio.h>
int main()
{
   char str[100];
   printf("Enter a string: ");
   fgets(str, sizeof(str), stdin);
   fputs(str, stdout);
   return 0;
}
```

Output:-

```
Enter a string: KnowProgram 2355>{}
KnowProgram 2355>{}
```

The puts() and fputs() return an EOF if an error occur. It will return a positive number upon success. Thus, they are sometimes called string-to-line output functions.

Returning values from puts() and fputs() function

```c
#include<stdio.h>
int main()
{
   char str[100];
   printf("Enter a string: ");
   fgets(str, sizeof(str), stdin);
   int c = puts(str);
   printf("%d\n",c);
   return 0;
}
```

Output:-

```
Enter a string: Know Program String
Know Program String

21
```

On successful reading, the puts() function return the number of characters read from the standard input, including '\n' (newline) and '\0'. On successful reading the fputs() function returns 1.

```c
#include<stdio.h>
int main()
{
   char str[100];
   printf("Enter a string: ");
   fgets(str, sizeof(str), stdin);
   int c = fputs(str, stdout);
   printf("%d\n",c);
   return 0;
}
```

Output:-

```
Enter a string: Know Program
Know Program
1



Enter a string: C Language String fputs() function
C Language String fputs() function
1
```

Note:- puts() and gets() function can be nested.

Display string in C using printf() with %s format code

The corresponding output for scanf() is printf(). By using this function a string can be display using with %s format.

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

```
{
   char str[100];
   printf("Enter a string: ");
   fgets(str, sizeof(str), stdin);
   printf("%s",str);
   return 0;
}
```

Output:-

```
Enter a string: Know Program
Know Program
```

We can use width modifiers as we discussed in the previous tutorial, printf() in C

```
#include<stdio.h>
int main()
{
   printf("%s\n","program");
   printf("%3s\n","program");
   printf("%12s\n","program");
   printf("%-12s\n","program");
   printf("%.3s\n","program");
   printf("%12.3s\n","program");
   printf("%-12.3s\n","program");
   return 0;
}
```

String Input-Output using fscanf() and fprintf() functions

Each C program has three input-output streams:- stdin, stdout, and stderr. The input stream is called standard-input (stdin), the output stream is called standard-output (stdout), and the side stream of output characters from errors is called the standard error (stderr). Internally they occupy file descriptors 0, 1, and 2 respectively. The fprintf() sends formatted output to a stream and fscanf() scans and formats input from a stream.

Demonstrating fscanf() and fprintf() function

```
#include<stdio.h>
int main()
{
   char str[10];
   printf("Enter a string: ");
   fscanf(stdin, "%s", str);
   fprintf(stdout, "String = %s", str);
   return 0;
}
```

Output:-

```
Enter a string: Know Program C language
String = Know
```

Similarly scanf(), by default fscanf() also reads only one word (If the size is enough). We can also use scanset and many others as we used with scanf().

```
fscanf(stdin, "%9s", str);
fscanf(stdin, "%[a-z]", str);
fscanf(stdin, "%[a-z,A-Z]", str);
fscanf(stdin, "%[^\n]", str);
fscanf(stdin, "%[^*]", str);
```

If you enjoyed this post, share it with your friends. Do you want to share more information about the topic discussed above or you find anything incorrect? Let us know in the comments. Thank you!

‹ Scope of Variables 2D Array of Strings ›

C TUTORIAL

C PROGRAMS

String in C

Basic C Programs
Flow Control Programs
C Function Programs
C Array Programs
C String Programs
C Pointer Programs
Others

Leave a Comment

Your email address will not be published. Required fields are marked *

Type here..

Name*

Email*

Website

Post Comment »