**Input and Output**

**Input** means to provide the program with some data to be used in the program and **Output** means to display data on screen or write the data to a printer or a file.

C programming language provides many built-in functions to read any given input and to display data on screen when there is a need to output the result.

In this tutorial, we will learn about such functions, which can be used in our program to take input from user and to output the result on screen.

All these built-in functions are present in C header files, we will also specify the name of header files in which a particular function is defined while discussing about it.

## scanf() and printf() functions

The standard input-output header file, named stdio.h contains the definition of the functions printf() and scanf(), which are used to display output on screen and to take input from user respectively.

#include<stdio.h>

void main()

{

// defining a variable

int i;

/\*

displaying message on the screen

asking the user to input a value

\*/

printf("Please enter a value...");

/\*

reading the value entered by the user

\*/

scanf("%d", &i);

/\*

displaying the number as output

\*/

printf( "\nYou entered: %d", i);

}

When you will compile the above code, it will ask you to enter a value. When you will enter the value, it will display the value you have entered on screen.

You must be wondering what is the purpose of %d inside the scanf() or printf() functions. It is known as **format string** and this informs the scanf() function, what type of input to expect and in printf() it is used to give a heads up to the compiler, what type of output to expect.

|  |  |
| --- | --- |
| **Format String** | **Meaning** |
| %d | Scan or print an integer as signed decimal number |
| %f | Scan or print a floating point number |
| %c | To scan or print a character |
| %s | To scan or print a character string. The scanning ends at whitespace. |

We can also **limit the number of digits or characters** that can be input or output, by adding a number with the format string specifier, like "%1d" or "%3s", the first one means a single numeric digit and the second one means 3 characters, hence if you try to input 42, while scanf() has "%1d", it will take only 4 as input. Same is the case for output.

In C Language, computer monitor, printer etc output devices are treated as files and the same process is followed to write output to these devices as would have been followed to write the output to a file.

**NOTE :** printf() function returns the number of characters printed by it, and scanf() returns the number of characters read by it.

int i = printf("studytonight");

In this program printf("studytonight"); will return 12 as result, which will be stored in the variable i, because studytonight has 12 characters.

## getchar() & putchar() functions

The getchar() function reads a character from the terminal and returns it as an integer. This function reads only single character at a time. You can use this method in a [loop](https://www.studytonight.com/c/loops-in-c.php) in case you want to read more than one character. The putchar() function displays the character passed to it on the screen and returns the same character. This function too displays only a single character at a time. In case you want to display more than one characters, use putchar() method in a loop.

#include <stdio.h>

void main( )

{

int c;

printf("Enter a character");

/\*

Take a character as input and

store it in variable c

\*/

c = getchar();

/\*

display the character stored

in variable c

\*/

putchar(c);

}

When you will compile the above code, it will ask you to enter a value. When you will enter the value, it will display the value you have entered.

## gets() & puts() functions

The gets() function reads a line from **stdin**(standard input) into the buffer pointed to by str [pointer](https://www.studytonight.com/pointers-in-c.php), until either a terminating newline or EOF (end of file) occurs. The puts() function writes the string str and a trailing newline to **stdout**.

str → This is the pointer to an array of chars where the C string is stored. (Ignore if you are not able to understand this now.)

#include<stdio.h>

void main()

{

/\* character array of length 100 \*/

char str[100];

printf("Enter a string");

gets( str );

puts( str );

getch();

}

When you will compile the above code, it will ask you to enter a string. When you will enter the string, it will display the value you have entered.

### Difference between scanf() and gets()

The main difference between these two functions is that scanf() stops reading characters when it encounters a space, but gets() reads space as character too.

If you enter name as **Study Tonight** using scanf() it will only read and store **Study** and will leave the part after space. But gets() function will read it completely.

# C Input Output (I/O)

#### In this tutorial, you will learn to use scanf() function to take input from the user, and printf() function to display output to the user.

## C Output

In C programming, printf() is one of the main output function. The function sends formatted output to the screen. For example,

### Example 1: C Output

#include <stdio.h>

int main()

{

// Displays the string inside quotations

printf("C Programming");

return 0;

}

**Output**

C Programming

How does this program work?

* All valid C programs must contain the main() function. The code execution begins from the start of the main() function.
* The printf() is a library function to send formatted output to the screen. The function prints the string inside quotations.
* To use printf() in our program, we need to include stdio.h header file using the #include <stdio.h> statement.
* The return 0; statement inside the main() function is the "Exit status" of the program. It's optional.

### Example 2: Integer Output

#include <stdio.h>

int main()

{

int testInteger = 5;

printf("Number = %d", testInteger);

return 0;

}

**Output**

Number = 5

We use %d format specifier to print int types. Here, the %d inside the quotations will be replaced by the value of testInteger.

### Example 3: float and double Output

#include <stdio.h>

int main()

{

float number1 = 13.5;

double number2 = 12.4;

printf("number1 = %f\n", number1);

printf("number2 = %lf", number2);

return 0;

}

**Output**

number1 = 13.500000

number2 = 12.400000

To print float, we use %f format specifier. Similarly, we use %lf to print double values.

### Example 4: Print Characters

#include <stdio.h>

int main()

{

char chr = 'a';

printf("character = %c.", chr);

return 0;

}

**Output**

character = a

To print char, we use %c format specifier.

## C Input

In C programming, scanf() is one of the commonly used function to take input from the user. The scanf() function reads formatted input from the standard input such as keyboards.

### Example 5: Integer Input/Output

#include <stdio.h>

int main()

{

int testInteger;

printf("Enter an integer: ");

scanf("%d", &testInteger);

printf("Number = %d",testInteger);

return 0;

}

**Output**

Enter an integer: 4

Number = 4

Here, we have used %d format specifier inside the scanf() function to take int input from the user. When the user enters an integer, it is stored in the testInteger variable.

Notice, that we have used &testInteger inside scanf(). It is because &testInteger gets the address of testInteger, and the value entered by the user is stored in that address.

### Example 6: Float and Double Input/Output

#include <stdio.h>

int main()

{

float num1;

double num2;

printf("Enter a number: ");

scanf("%f", &num1);

printf("Enter another number: ");

scanf("%lf", &num2);

printf("num1 = %f\n", num1);

printf("num2 = %lf", num2);

return 0;

}

**Output**

Enter a number: 12.523

Enter another number: 10.2

num1 = 12.523000

num2 = 10.200000

We use %f and %lf format specifier for float and double respectively.

### Example 7: C Character I/O

#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: g

You entered g.

When a character is entered by the user in the above program, the character itself is not stored. Instead, an integer value (ASCII value) is stored.

And when we display that value using %c text format, the entered character is displayed. If we use %d to display the character, it's ASCII value is printed.

### Example 8: ASCII Value

#include <stdio.h>

int main()

{

char chr;

printf("Enter a character: ");

scanf("%c", &chr);

// When %c is used, a character is displayed

printf("You entered %c.\n",chr);

// When %d is used, ASCII value is displayed

printf("ASCII value is % d.", chr);

return 0;

}

Output

Enter a character: g

You entered g.

ASCII value is 103.

## I/O Multiple Values

Here's how you can take multiple inputs from the user and display them.

#include <stdio.h>

int main()

{

int a;

float b;

printf("Enter integer and then a float: ");

// Taking multiple inputs

scanf("%d%f", &a, &b);

printf("You entered %d and %f", a, b);

return 0;

}

**Output**

Enter integer and then a float: -3

3.4

You entered -3 and 3.400000

## Format Specifiers for I/O

As you can see from the above examples, we use

* %d for int
* %f for float
* %lf for double
* %c for char

Here's a list of commonly used C data types and their format specifiers.

|  |  |
| --- | --- |
| **Data Type** | **Format Specifier** |
| int | %d |
| char | %c |
| float | %f |
| double | %lf |
| short int | %hd |
| unsigned int | %u |
| long int | %li |
| long long int | %lli |
| unsigned long int | %lu |
| unsigned long long int | %llu |
| signed char | %c |
| unsigned char | %c |
| long double | %Lf |