[**Printf, Format Specifiers, Format Conversions and Formatted Output**](http://www.codingunit.com/printf-format-specifiers-format-conversions-and-formatted-output)

**printf Background**

The printf function is just a output function from the standard library of functions that are accessible by C programs. The behavior of printf is defined in the ANSI standard. If the compiler that you’re using conforms to this standard then all the features and properties should be available to you.

**Format Specifiers**

There are many format specifiers defined in C. Take a look at the following list:

|  |  |
| --- | --- |
| %i or %d | int |
| %c | char |
| %f | float (see also the note below) |
| %s | string |

Let us take a look at an example of printf formatted output (that why you here, isn’t it?):

#include<stdio.h>

main()

{

int a,b;

float c,d;

a = 15;

b = a / 2;

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

printf("%3d\n",b);

printf("%\*3d\n",b);

c = 15.3;

d = c / 3;

printf("%3.2f\n",d);

}

Output of the source above:

7

7

007

5.10

As you can see in the first printf statement we print a decimal. In the second printf statement we print the same decimal, but we use a width (%3d) to say that we want three digits (positions) reserved for the output.  
The result is that two “space characters” are placed before printing the character. In the third printf statement we say almost the same as the previous one. Print the output with a width of three digits, but fill the space with 0.

In the fourth printf statement we want to print a float. In this printf statement we want to print three position before the decimal point (called width) and two positions behind the decimal point (called precision).

The \n used in the printf statements is called an escape sequence. In this case it represents a newline character. After printing something to the screen you usually want to print something on the next line. If there is no \n then a next printf command will print the string on the same line. Commonly used escape sequences are:

* \n (newline)
* \t (tab)
* \v (vertical tab)
* \f (new page)
* \b (backspace)
* \r (carriage return)
* %d (print as a decimal integer)
* %6d (print as a decimal integer with a width of at least 6 wide)
* %f (print as a floating point)
* %4f (print as a floating point with a width of at least 4 wide)
* %.4f (print as a floating point with a precision of four characters after the decimal point)
* %3.2f (print as a floating point at least 3 wide and a precision of 2)

**Formatting other Types**

Until now we only used integers and floats, but there are more types you can use. Take a look at the following example:

#include<stdio.h>

main()

{

printf("The color: %s\n", "blue");

printf("First number: %d\n", 12345);

printf("Second number: %04d\n", 25);

printf("Third number: %i\n", 1234);

printf("Float number: %3.2f\n", 3.14159);

printf("Hexadecimal: %x\n", 255);

printf("Octal: %o\n", 255);

printf("Unsigned value: %u\n", 150);

printf("Just print the percentage sign %%\n", 10);

}

Output of the source example:

The color: blue

First number: 12345

Second number: 0025

Third number: 1234

Float number: 3.14

Hexadecimal: ff

Octal: 377

Unsigned value: 150

Just print the percentage sign %

**Note:** In the last printf statement only the percentage sign is printed.

The number 10 in this statement doesn’t matter; it’s not used in the output. So if you want to print a percentage number you would use something like this: printf(“%2d%%\n”, 10); (The output will be 10%)

**Formatting Strings**

By now you have seen most of the format conversion possible, but there is one type that is a little different  
and that are string format conversions. Take a look at the following example:

#include<stdio.h>

main()

{

printf(":%s:\n", "Hello, world!"); //:Hello, world!:

printf("\* %-15s \*\n", "Hello, world!"); //\*Hello, world! \*

printf(":%.10s:\n", "Hello, world!"); //:Hello, wor:

printf(":%-10s:\n", "Hello, world!");// :Hello, world!:

printf(":%-15s:\n", "Hello, world!");// :Hello, world! :

printf(":%.3s:\n", "Hello, world!"); :Hello, world!:

printf(":%15.10s:\n", "Hello, world!"); : Hello, wor:

printf(":%-15.10s:\n", "Hello, world!"); :Hello, wor :

}

The output of the example above:

As you can see, the string format conversion reacts very different from number format conversions.

* The printf(“:%s:\n”, “Hello, world!”); statement prints the string (nothing special happens.)
* The printf(“:%15s:\n”, “Hello, world!”); statement prints the string, but print 15 characters. If the string is smaller the “empty” positions will be filled with “whitespace.”
* The printf(“:%.10s:\n”, “Hello, world!”); statement prints the string, but print only 10 characters of the string.
* The printf(“:%-10s:\n”, “Hello, world!”); statement prints the string, but prints at least 10 characters. If the string is smaller “whitespace” is added at the end. (See next example.)
* The printf(“:%-15s:\n”, “Hello, world!”); statement prints the string, but prints at least 15 characters. The string in this case is shorter than the defined 15 character, thus “whitespace” is added at the end (defined by the minus sign.)
* The printf(“:%.15s:\n”, “Hello, world!”); statement prints the string, but print only 15 characters of the string. In this case the string is shorter than 15, thus the whole string is printed.
* The printf(“:%15.10s:\n”, “Hello, world!”); statement prints the string, but print 15 characters.  
  If the string is smaller the “empty” positions will be filled with “whitespace.” But it will only print a maximum of 10 characters, thus only part of new string (old string plus the whitespace positions) is printed.
* The printf(“:%-15.10s:\n”, “Hello, world!”); statement prints the string, but it does the exact same thing as the previous statement, accept the “whitespace” is added at the end.