Description

The C library function **int printf(const char *format, ...)** sends formatted output to stdout.

Declaration

Following is the declaration for printf() function.

```
int printf(const char *format, ...)
```

Parameters

format -- This is the string that contains the text to be written to stdout.It can optionally contain embedded format tags that are replaced by the values specified in subsequent additional arguments and formatted as requested. Format tags prototype is **%[flags][width][.precision][length]specifier**, which is explained below:

specifier	Output
c	Character.
d or i	Signed decimal integer
e	Scientific notation (mantissa/exponent) using e character
E	Scientific notation (mantissa/exponent) using E character
f	Decimal floating point
g	Use the shorter of %e or %f.
G	Use the shorter of %E or %f
o	Signed octal
s	String of characters
u	Unsigned decimal integer
x	Unsigned hexadecimal integer
X	Unsigned hexadecimal integer (capital letters)
p	Pointer address
n	Nothing printed.

%	Character.

flags	Description
-	Left-justify within the given field width; Right justification is the default (see width sub-specifier).
+	Forces to preceed the result with a plus or minus sign (+ or -) even for positive numbers. By default, only negative numbers are preceded with a - sign..
(space)	If no sign is going to be written, a blank space is inserted before the value.
#	Used with o, x or X specifiers the value is preceeded with 0, 0x or 0X respectively for values different than zero. Used with e, E and f, it forces the written output to contain a decimal point even if no digits would follow. By default, if no digits follow, no decimal point is written. Used with g or G the result is the same as with e or E but trailing zeros are not removed.
0	Left-pads the number with zeroes (0) instead of spaces, where padding is specified (see width sub-specifier).

width	Description
(number)	Minimum number of characters to be printed. If the value to be printed is shorter than this number, the result is padded with blank spaces. The value is not truncated even if the result is larger.
*	The width is not specified in the format string, but as an additional integer value argument preceding the argument that has to be formatted.

.precision	Description

.number	For integer specifiers (d, i, o, u, x, X): precision specifies the minimum number of digits to be written. If the value to be written is shorter than this number, the result is padded with leading zeros. The value is not truncated even if the result is longer. A precision of 0 means that no character is written for the value 0. For e, E and f specifiers: this is the number of digits to be printed after de decimal point. For g and G specifiers: This is the maximum number of significant digits to be printed. For s: this is the maximum number of characters to be printed. By default all characters are printed until the ending null character is encountered. For c type: it has no effect. When no precision is specified, the default is 1. If the period is specified without an explicit value for precision, 0 is assumed.
.*	The precision is not specified in the format string, but as an additional integer value argument preceding the argument that has to be formatted.

length	Description
h	The argument is interpreted as a short int or unsigned short int (only applies to integer specifiers: i, d, o, u, x and X).
l	The argument is interpreted as a long int or unsigned long int for integer specifiers (i, d, o, u, x and X), and as a wide character or wide character string for specifiers c and s.
L	The argument is interpreted as a long double (only applies to floating point specifiers: e, E, f, g and G).

additional arguments -- Depending on the format string, the function may expect a sequence of additional arguments, each containing one value to be inserted instead of each %-tag specified in the format parameter, if any. There should be the same number of these arguments as the number of %-tags that expect a value.

Return Value

If successful, the total number of characters written is returned.On failure, a negative number is returned.

Example

The following example shows the usage of printf() function.

```
#include <stdio.h>

int main ()
{
```

```
   int ch;

   for( ch = 75 ; ch <= 100; ch++ ) {
      printf("ASCII value = %d, Character = %c\n", ch , ch );
   }

   return(0);
}
```

Let us compile and run the above program, this will produce the following result:

```
ASCII value = 75, Character = K
ASCII value = 76, Character = L
ASCII value = 77, Character = M
ASCII value = 78, Character = N
ASCII value = 79, Character = O
ASCII value = 80, Character = P
ASCII value = 81, Character = Q
ASCII value = 82, Character = R
ASCII value = 83, Character = S
ASCII value = 84, Character = T
ASCII value = 85, Character = U
ASCII value = 86, Character = V
ASCII value = 87, Character = W
ASCII value = 88, Character = X
ASCII value = 89, Character = Y
ASCII value = 90, Character = Z
ASCII value = 91, Character = [
ASCII value = 92, Character = \
ASCII value = 93, Character = ]
ASCII value = 94, Character = ^
ASCII value = 95, Character = _
ASCII value = 96, Character = `
ASCII value = 97, Character = a
ASCII value = 98, Character = b
ASCII value = 99, Character = c
ASCII value = 100, Character = d
```