

Search:

Go

Not logged in

Reference <stdio> printf

register log in

C++

Information

Tutorials

Reference

Articles

Forum

Reference

C library:

<assert> (assert.h)

<cctype> (ctype.h)

<errno> (errno.h)

<cfenv> (fenv.h)

<cmath> (math.h)

<climits> (limits.h)

<locale> (locale.h)

<cmath> (math.h)

<setjmp> (setjmp.h)

<signal> (signal.h)

<stdarg> (stdarg.h)

<stdbool> (stdbool.h)

<stddef> (stddef.h)

<stdint> (stdint.h)

<stdio> (stdio.h)

<stdlib> (stdlib.h)

<string> (string.h)

<tgmath> (tgmath.h)

<time> (time.h)

<uchar> (uchar.h)

<wchar> (wchar.h)

<wctype> (wctype.h)

Containers:

Input/Output:

Multi-threading:

Other:

<stdio> (stdio.h)

functions:

clearerr

fclose

feof

ferror

fflush

fgetc

fgetpos

fgets

fopen

fprintf

fputc

fputs

fread

freopen

fscanf

fseek

fsetpos

ftell

fwrite

getc

getchar

gets

perror

printf

putc

putchar

puts

remove

rename

rewind

scanf

setbuf

setvbuf

snprintf

sprintf

sscanf

tmpfile

tmpnam

ungetc

vfprintf

vfprintf

vscanf

vsnprintf

vsprintf

vsscanf

objects:

stderr

stdin

stdout

function

printf

<stdio>

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

Print formatted data to stdout

Writes the C string pointed by *format* to the standard output (*stdout*). If *format* includes *format specifiers* (subsequences beginning with %), the additional arguments following *format* are formatted and inserted in the resulting string replacing their respective specifiers.

Parameters

format

C string that contains the text to be written to *stdout*.
It can optionally contain embedded *format specifiers* that are replaced by the values specified in subsequent additional arguments and formatted as requested.

A *format specifier* follows this prototype: [see compatibility note below]
%[flags][width][.precision][length]specifier

Where the *specifier character* at the end is the most significant component, since it defines the type and the interpretation of its corresponding argument:

specifier	Output	Example
d or i	Signed decimal integer	392
u	Unsigned decimal integer	7235
o	Unsigned octal	610
x	Unsigned hexadecimal integer	7fa
X	Unsigned hexadecimal integer (uppercase)	7FA
f	Decimal floating point, lowercase	392.65
F	Decimal floating point, uppercase	392.65
e	Scientific notation (mantissa/exponent), lowercase	3.9265e+2
E	Scientific notation (mantissa/exponent), uppercase	3.9265E+2
g	Use the shortest representation: %e or %f	392.65
G	Use the shortest representation: %E or %F	392.65
a	Hexadecimal floating point, lowercase	-0xc.90fep-2
A	Hexadecimal floating point, uppercase	-0XC.90FEP-2
c	Character	a
s	String of characters	sample
p	Pointer address	b8000000
n	Nothing printed. The corresponding argument must be a pointer to a signed int. The number of characters written so far is stored in the pointed location.	
%	A % followed by another % character will write a single % to the stream.	%

The *format specifier* can also contain sub-specifiers: *flags*, *width*, *.precision* and *modifiers* (in that order), which are optional and follow these specifications:

flags	description
-	Left-justify within the given field width; Right justification is the default (see <i>width</i> sub-specifier).
+	Forces to precede 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 preceded with 0, 0x or 0X respectively for values different than zero. Used with a, A, e, E, f, F, g or G it forces the written output to contain a decimal point even if no more digits follow. By default, if no digits follow, no decimal point is written.
0	Left-pads the number with zeroes (0) instead of spaces when padding is specified (see <i>width</i> 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 <i>width</i> is not specified in the <i>format</i> 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): <i>precision</i> 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 <i>precision</i> of 0 means that no character is written for the value 0. For a, A, e, E, f, F, g and G specifiers: this is the number of digits to be printed after the decimal point (by default, this is 6). 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. If the period is specified without an explicit value for <i>precision</i> , 0 is assumed.
.*	The <i>precision</i> is not specified in the <i>format</i> string, but as an additional integer value argument preceding the argument that has to be formatted.

The *length* sub-specifier modifies the length of the data type. This is a chart showing the types used to interpret the corresponding arguments with and without *length* specifier (if a different type is used, the proper type promotion or conversion is performed, if allowed):

	specifiers						
length	d i	u o x X	f F e E g G a A	c	s	p	n
(none)	int	unsigned int	double	int	char*	void*	int*
hh	signed char	unsigned char					signed char*
h	short int	unsigned short int					short int*

types:

FILE

fpos_t

size_t

macro constants:

BUFSIZ

EOF

FILENAME_MAX

FOPEN_MAX

L_tmpnam

NULL

TMP_MAX

l	long int	unsigned long int		wint_t	wchar_t*	long int*
ll	long long int	unsigned long long int				long long int*
j	intmax_t	uintmax_t				intmax_t*
z	size_t	size_t				size_t*
t	ptrdiff_t	ptrdiff_t				ptrdiff_t*
L			long double			

Note regarding the c specifier: it takes an int (or `wint_t`) as argument, but performs the proper conversion to a char value (or a `wchar_t`) before formatting it for output.

Note: Yellow rows indicate specifiers and sub-specifiers introduced by C99. See <inttypes> for the specifiers for extended types.

... (additional arguments)
Depending on the *format* string, the function may expect a sequence of additional arguments, each containing a value to be used to replace a *format specifier* in the *format* string (or a pointer to a storage location, for n). There should be at least as many of these arguments as the number of values specified in the *format specifiers*. Additional arguments are ignored by the function.

Return Value

On success, the total number of characters written is returned.

If a writing error occurs, the *error indicator* (`ferror`) is set and a negative number is returned.

If a multibyte character encoding error occurs while writing wide characters, `errno` is set to `EILSEQ` and a negative number is returned.

Example

```
1 /* printf example */
2 #include <stdio.h>
3
4 int main()
5 {
6     printf ("Characters: %c %c \n", 'a', 65);
7     printf ("Decimals: %d %ld\n", 1977, 650000L);
8     printf ("Preceding with blanks: %10d \n", 1977);
9     printf ("Preceding with zeros: %010d \n", 1977);
10    printf ("Some different radices: %d %x %o %#x %u \n", 100, 100, 100, 100, 100);
11    printf ("floats: %4.2f %+.0e %E \n", 3.1416, 3.1416, 3.1416);
12    printf ("Width trick: %*d \n", 5, 10);
13    printf ("%s \n", "A string");
14    return 0;
15 }
```

Output:

```
Characters: a A
Decimals: 1977 650000
Preceding with blanks:      1977
Preceding with zeros: 0000001977
Some different radices: 100 64 144 0x64 0144
floats: 3.14 +3e+000 3.141600E+000
Width trick:    10
A string
```

Compatibility

Particular library implementations may support additional *specifiers* and *sub-specifiers*. Those listed here are supported by the latest C and C++ standards (both published in 2011), but those in yellow were introduced in C99 (only required for C++ implementations since C++11), and may not be supported by libraries that comply with older standards.

See also

puts	Write string to stdout (function)
scanf	Read formatted data from stdin (function)
fprintf	Write formatted data to stream (function)
fwrite	Write block of data to stream (function)