printf

```
<cstdio>
int printf ( const char * format, ... );
```

Print formatted data to stdout

Writes to the standard output (stdout) a sequence of data formatted as the *format* argument specifies. After the *format* parameter, the function expects at least as many additional arguments as specified in *format*.

Parameters

format

String that contains the text to be written to `stdout`.
It can optionally contain embedded format tags that are substituted by the values specified in subsequent argument(s) and formatted as requested.
The number of arguments following the *format* parameters should at least be as much as the number of format tags.
The format tags follow this prototype:

```
%[flags][width][.precision][length]specifier
```

Where *specifier* is the most significant one and defines the type and the interpretation of the value of the coresponding argument:

specifier	**Output**	**Example**
`c`	Character	`a`
`d` or `i`	Signed decimal integer	`392`
`e`	Scientific notation (mantise/exponent) using `e` character	`3.9265e+2`
`E`	Scientific notation (mantise/exponent) using `E` character	`3.9265E+2`
`f`	Decimal floating point	`392.65`
`g`	Use the shorter of `%e` or `%f`	`392.65`
`G`	Use the shorter of `%E` or `%f`	`392.65`
`o`	Signed octal	`610`
`s`	String of characters	`sample`
`u`	Unsigned decimal integer	`7235`
`x`	Unsigned hexadecimal integer	`7fa`
`X`	Unsigned hexadecimal integer (capital letters)	`7FA`
`p`	Pointer address	`B800:0000`
`n`	Nothing printed. The argument must be a pointer to a signed `int`, where the number of characters written so far is stored.	
`%`	A `%` followed by another `%` character will write `%` to `stdout`.	

The tag can also contain *flags*, *width*, *.precision* and *modifiers* sub-specifiers, which are optional and follow these specifications:

flags	**description**
`-`	Left-justify within the given field width; Right justification is the default (see *width* 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 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** the 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

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