vprintf, vfprintf, vsprintf, vsnprintf, vprintf_s, vfprintf_s, vsprintf_s, vsnprintf_s

Определено в заголовке <stdio.h>

`int vprintf( const char *format, va_list vlist );`	(1)	(до C99)
`int vprintf( const char *restrict format, va_list vlist );`		(с C99)
`int vfprintf( FILE *stream, const char *format, va_list vlist);`	(2)	(до C99)
`int vfprintf( FILE *restrict stream, const char *restrict format, va_list vlist );`		(с C99)
`int vsprintf( char *buffer, const char *format, va_list vlist);`	(3)	(до C99)
`int vsprintf( char *restrict buffer, const char *restrict format, va_list vlist );`		(с C99)
`int vsnprintf( char *restrict buffer, size_t bufsz, const char *restrict format, va_list vlist );`	(4)	(с C99)
`int vprintf_s( const char *restrict format, va_list arg);`	(5)	(начиная с C11)
`int vfprintf_s( FILE *restrict stream, const char *restrict format, va_list arg);`	(6)	(начиная с C11)
`int vsprintf_s( char *restrict buffer, rsize_t bufsz, const char *restrict format, va_list arg);`	(7)	(начиная с C11)
`int vsnprintf_s(char *restrict buffer, rsize_t bufsz, const char *restrict format, va_list arg);`	(8)	(начиная с C11)

Загружает данные из местоположений, определенных vlist, преобразует их в эквиваленты символьных строк и записывает результаты в различные приемники.

1) Записывает результаты в stdout.
2) Записывает результаты в поток файловstream.
3) Записывает результаты в символьную строкуbuffer.
4) Записывает результаты в символьную строкуbuffer. Записывается не более bufsz - 1символов. Результирующая символьная строка будет заканчиваться нулевым символом, если bufsztолько он не равен нулю. Если bufszравно нулю, то ничего не записывается и bufferможет быть нулевым указателем, однако возвращаемое значение (количество байтов, которые будут записаны без учета нулевого терминатора) все равно вычисляется и возвращается.

5-8) То же самое, что и (1-4), за исключением того, что следующие ошибки обнаруживаются во время выполнения и вызывают установленную в данный момент функцию обработчика ограничений:

- спецификатор преобразования %nприсутствует в format
- любой из аргументов, соответствующих %snулевому указателю
- format или bufferэто нулевой указатель
- bufsz равен нулю или больше RSIZE_MAX
- ошибки кодирования возникают в любом из спецификаторов преобразования строк и символов
- (vsprintf_sтолько для), строка для хранения buffer(включая конечный null)) будет превышать bufsz

Как и во всех функциях с проверкой границ, vprintf_s, vfprintf_s, vsprintf_s, и vsnprintf_sгарантированно доступны только в том случае, если `__STDC_LIB_EXT1__` определяется реализацией и если пользователь определяет `__STDC_WANT_LIB_EXT1__` целочисленной константой 1 перед включением stdio.h.

Параметры

поток - поток выходного файла для записи
буфер - указатель на символьную строку для записи
bufsz - может быть записано до bufsz - 1 символов плюс нулевой терминатор
формат - указатель на символьную строку с нулевым окончанием, указывающую, как интерпретировать данные
vlist - список аргументов переменной, содержащий данные для печати.

Строка **формата** состоит из обычных многобайтовых символов (кроме %), которые копируются без изменений в выходной поток, и спецификаций преобразования. Каждая спецификация преобразования имеет следующий формат:

- вводный %символ
- (необязательно) один или несколько флагов, изменяющих поведение преобразования:

- **-:** результат преобразования выравнивается по левому краю поля (по умолчанию выравнивается по правому краю)
- **+:** знак подписанных преобразований всегда предшествует результату преобразования (по умолчанию результату предшествует минус только тогда, когда он отрицательный)
- *пробел*: если результат преобразования со знаком не начинается со знака или пуст, к результату добавляется пробел. Он игнорируется, если +флаг присутствует.
- **# :** *выполняется альтернативная форма* преобразования. Точные эффекты см. В таблице ниже, в противном случае поведение не определено.
- **0 :** для преобразования целых чисел и чисел с плавающей запятой начальные нули используются для заполнения поля вместо пробелов. Для целых чисел он игнорируется, если точность указана явно. Для других преобразований использование этого флага приводит к неопределенному поведению. Он игнорируется, если -флаг присутствует.

- (необязательно) целочисленное значение или *задающее минимальную ширину поля. Результат дополняется пробелами (по умолчанию), если требуется, слева при выравнивании по правому краю или справа при выравнивании по левому краю. В случае, когда *используется, ширина задается дополнительным аргументом типа `int`, который появляется перед преобразуемым аргументом и аргументом, обеспечивающим точность, если он указан. Если значение аргумента отрицательное, то результат с указанным -флагом и положительной шириной поля. (Примечание: это минимальная ширина: значение никогда не усекается.)

- (необязательно) . за которым следует целое число или *, или ни то, ни другое, указывающее *точность* преобразования. В том случае, когда *используется when, *точность* задается дополнительным аргументом типа `int`, который появляется перед преобразуемым аргументом, но после аргумента, задающего минимальную ширину поля, если таковой задан. Если значение этого аргумента отрицательно, оно игнорируется. Если ни число ни *не используется, то точность принимается равной нулю. См. Таблицу ниже для точных эффектов *точности*.

- (необязательно) *модификатор длины*, задающий размер аргумента (в сочетании со спецификатором формата преобразования задает тип соответствующего аргумента)

- спецификатор формата преобразования

Доступны следующие спецификаторы формата:

Спецификатор преобразования	Объяснение	Ожидаемый тип аргумента								
	Модификатор длины →	hh (C99)	h	(нет)	l	ll (C99)	j (C99)	z (C99)	t (C99)	L
%	пишет литерал %. Полная спецификация преобразования должна быть%%.	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A
c	записывает **один символ.** Аргумент сначала преобразуется в `unsigned char`. Если используется модификатор l, аргумент сначала преобразуется в символьную строку, как если **бы %ls** с `аргументом wchar_t[2]` .	N/A	N/A	int	wint_t	N/A	N/A	N/A	N/A	N/A
s	записывает **символьную строку** Аргумент должен быть указателем на начальный элемент массива символов. *Точность* определяет максимальное количество записываемых байтов. Если *точность* не указана, записывает каждый байт до первого нулевого терминатора, не включая его. Если используется спецификатор l, аргумент должен быть указателем на начальный элемент массива `wchar_t`, который преобразуется в массив char как бы вызовом wcrtomb с нулевым инициализированным состоянием преобразования.	N/A	N/A	голец*	wchar_t*	N/A	N/A	N/A	N/A	N/A
d i	преобразует целое **число со знаком** в десятичное представление *[-]dddd*. *Точность* указывает минимальное количество цифр, которые должны отображаться. Точность по умолчанию `равна 1`. Если преобразованное значение и точность равны `0`, преобразование не приводит к символам.	подписанный символ	короткое	int	длинный	долго долго	intmax_t	подписанный size_t	ptrdiff_t	N/A
o	converts an **unsigned integer** into octal representation *oooo*. *Precision* specifies the minimum number of digits to appear. The default precision is `1`. If both the converted value and the precision are `0` the conversion results in no characters. In the *alternative implementation* precision is increased if necessary, to write one leading zero. In that case if both the converted value and the precision are `0`, single `0` is written.	unsigned char	unsigned short	unsigned int	unsigned long	unsigned long long	uintmax_t	size_t	unsigned version of ptrdiff_t	N/A
x X	converts an **unsigned integer** into hexadecimal representation *hhhh*. For the **x** conversion letters abcdef are used. For the **X** conversion letters ABCDEF are used. *Precision* specifies the minimum number of digits to appear. The default precision is `1`. If both the converted value and the precision are `0` the conversion results in no characters. In the *alternative implementation* **0x** or **0X** is prefixed to results if the converted value is nonzero.									N/A
u	converts an **unsigned integer** into decimal representation *dddd*. *Precision* specifies the minimum number of digits to appear. The default precision is `1`. If both the converted value and the precision are `0` the conversion results in no characters.									N/A
f F	converts **floating-point number** to the decimal notation in the style *[-]ddd.ddd*. *Precision* specifies the exact number of digits to appear after the decimal point character. The default precision is `6`. In the *alternative implementation* decimal point character is written even if no digits follow it. For infinity and not-a-number conversion style see notes.	N/A	N/A	double	double (C99)	N/A	N/A	N/A	N/A	long double
e E	converts **floating-point number** to the decimal exponent notation. For the **e** conversion style *[-]d.ddd***e***±dd* is used. For the **E** conversion style *[-]d.ddd***E***±dd* is used. The exponent contains at least two digits, more digits are used only if necessary. If the value is `0`, the exponent is also `0`. *Precision* specifies the exact number of digits to appear after the decimal point character. The default precision is `6`. In the *alternative implementation* decimal point character is written even if no digits follow it. For infinity and not-a-number conversion style see notes.	N/A	N/A			N/A	N/A	N/A	N/A	
a	converts **floating-point number** to the hexadecimal exponent	N/A	N/A			N/A	N/A	N/A	N/A	

A (C99)	notation. For the **a** conversion style *[-]***0x***h.hhh***p±***d* is used. For the **A** conversion style *[-]***0X***h.hhh***P±***d* is used. The first hexadecimal digit is not 0 if the argument is a normalized floating point value. If the value is `0`, the exponent is also `0`. *Precision* specifies the exact number of digits to appear after the hexadecimal point character. The default precision is sufficient for exact representation of the value. In the *alternative implementation* decimal point character is written even if no digits follow it. For infinity and not-a-number conversion style see notes.									
g **G**	converts **floating-point number** to decimal or decimal exponent notation depending on the value and the *precision*. For the **g** conversion style conversion with style **e** or **f** will be performed. For the **G** conversion style conversion with style **E** or **F** will be performed. Let P equal the precision if nonzero, `6` if the precision is not specified, or `1` if the precision is `0`. Then, if a conversion with style E would have an exponent of X: ▪ if *P > X ≥ −4*, the conversion is with style **f** or **F** and precision *P − 1 − X*. ▪ otherwise, the conversion is with style **e** or **E** and precision *P − 1*. Unless *alternative representation* is requested the trailing zeros are removed, also the decimal point character is removed if no fractional part is left. For infinity and not-a-number conversion style see notes.	N/A	N/A			N/A	N/A	N/A	N/A	
n	returns the **number of characters written** so far by this call to the function. The result is *written* to the value pointed to by the argument. The specification may not contain any *flag*, *field width*, or *precision*.	`signed char*`	`short*`	`int*`	`long*`	`long long*`	`intmax_t*`	`signed size_t*`	`ptrdiff_t*`	N/A
p	writes an implementation defined character sequence defining a **pointer**.	N/A	N/A	`void*`	N/A	N/A	N/A	N/A	N/A	N/A

The floating point conversion functions convert infinity to `inf` or `infinity`. Which one is used is implementation defined.

Not-a-number is converted to `nan` or `nan(`*`char_sequence`*`)`. Which one is used is implementation defined.

The conversions **F**, **E**, **G**, **A** output `INF`, `INFINITY`, `NAN` instead.

Even though `%c` expects `int` argument, it is safe to pass a `char` because of the integer promotion that takes place when a variadic function is called.

The correct conversion specifications for the fixed-width character types (`int8_t`, etc) are defined in the header `<inttypes.h>` (although `PRIdMAX`, `PRIuMAX`, etc is synonymous with `%jd`, `%ju`, etc).

The memory-writing conversion specifier `%n` is a common target of security exploits where format strings depend on user input and is not supported by the bounds-checked `printf_s` family of functions.

There is a sequence point after the action of each conversion specifier; this permits storing multiple `%n` results in the same variable or, as an edge case, printing a string modified by an earlier `%n` within the same call.

If a conversion specification is invalid, the behavior is undefined.

Return value

1-3) The number of characters written if successful or negative value if an error occurred.

4) The number of characters written if successful or negative value if an error occurred. If the resulting string gets truncated due to `buf_size` limit, function returns the total number of characters (not including the terminating null-byte) which would have been written, if the limit was not imposed.

5,6) number of characters transmitted to the output stream or negative value if an output error, a runtime constrants violation error, or an encoding error occurred.

7) number of characters written to `buffer`, not counting the null character (which is always written as long as `buffer` is not a null pointer and `bufsz` is not zero and not greater than `RSIZE_MAX`), or zero on runtime constraint violations, and negative value on encoding errors

8) number of characters not including the terminating null character (which is always written as long as `buffer` is not a null pointer and `bufsz` is not zero and not greater than `RSIZE_MAX`), which would have been written to `buffer` if `bufsz` was ignored, or a negative value if a runtime constraints violation or an encoding error occurred

Notes

All these functions invoke `va_arg` at least once, the value of `arg` is indeterminate after the return. These functions do not invoke `va_end`, and it must be done by the caller.

`vsnprintf_s`, unlike `vsprintf_s`, will truncate the result to fit within the array pointed to by `buffer`.

Example

Run this code

```
#include <stdio.h>
#include <stdarg.h>
#include <time.h>

void debug_log(const char *fmt, ...)
{
    struct timespec ts;
    timespec_get(&ts, TIME_UTC);
    char time_buf[100];
    size_t rc = strftime(time_buf, sizeof time_buf, "%D %T", gmtime(&ts.tv_sec));
    snprintf(time_buf + rc, sizeof time_buf - rc, ".%06ld UTC", ts.tv_nsec / 1000);

    va_list args1;
    va_start(args1, fmt);
    va_list args2;
    va_copy(args2, args1);
    char buf[1+vsnprintf(NULL, 0, fmt, args1)];
    va_end(args1);
    vsnprintf(buf, sizeof buf, fmt, args2);
    va_end(args2);

    printf("%s [debug]: %s\n", time_buf, buf);
}

int main(void)
{
    debug_log("Logging, %d, %d, %d", 1, 2, 3);
}
```

Possible output:

```
02/20/15 21:58:09.072683 UTC [debug]: Logging, 1, 2, 3
```

References

- C11 standard (ISO/IEC 9899:2011):
    - 7.21.6.8 The vfprintf function (p: 326-327)
    - 7.21.6.10 The vprintf function (p: 328)
    - 7.21.6.12 The vsnprintf function (p: 329)
    - 7.21.6.13 The vsprintf function (p: 329)
    - K.3.5.3.8 The vfprintf_s function (p: 597)
    - K.3.5.3.10 The vprintf_s function (p: 598-599)
    - K.3.5.3.12 The vsnprintf_s function (p: 600)
    - K.3.5.3.13 The vsprintf_s function (p: 601)
- C99 standard (ISO/IEC 9899:1999):
    - 7.19.6.8 The vfprintf function (p: 292)
    - 7.19.6.10 The vprintf function (p: 293)
    - 7.19.6.12 The vsnprintf function (p: 294)
    - 7.19.6.13 The vsprintf function (p: 295)

- C89/C90 standard (ISO/IEC 9899:1990):
  - 4.9.6.7 The vfprintf function
  - 4.9.6.8 The vprintf function
  - 4.9.6.9 The vsprintf function

See also

vwprintf (C95) **vfwprintf** (C95) **vswprintf** (C95) **vwprintf_s** (C11) **vfwprintf_s** (C11) **vswprintf_s** (C11) **vsnwprintf_s** (C11)	prints formatted wide character output to stdout, a file stream or a buffer using variable argument list (function)
printf **fprintf** **sprintf** **snprintf** (C99) **printf_s** (C11) **fprintf_s** (C11) **sprintf_s** (C11) **snprintf_s** (C11)	prints formatted output to stdout, a file stream or a buffer (function)
vscanf (C99) **vfscanf** (C99) **vsscanf** (C99) **vscanf_s** (C11) **vfscanf_s** (C11) **vsscanf_s** (C11)	reads formatted input from stdin, a file stream or a buffer using variable argument list (function)
C++ documentation for **vprintf**, **vfprintf**, **vsprintf**, **vsnprintf**	