printf, fprintf, sprintf, snprintf, printf_s, fprintf_s, sprintf_s, snprintf_s

Defined in header <stdio.h>

`int printf( const char *format, ... );`	(1)	(until C99)
`int printf( const char *restrict format, ... );`		(начиная с C99)
`int fprintf( FILE *stream, const char *format, ... );`	(2)	(до C99)
`int fprintf( FILE *restrict stream, const char *restrict format, ... );`		(начиная с C99)
`int sprintf( char *buffer, const char *format, ... );`	(3)	(до C99)
`int sprintf( char *restrict buffer, const char *restrict format, ... );`		(начиная с C99)
`int snprintf( char *restrict buffer, size_t bufsz, const char *restrict format, ... );`	(4)	(начиная с C99)
`int printf_s( const char *restrict format, ... );`	(5)	(начиная с C11)
`int fprintf_s( FILE *restrict stream, const char *restrict format, ... );`	(6)	(начиная с C11)
`int sprintf_s( char *restrict buffer, rsize_t bufsz, const char *restrict format, ... );`	(7)	(начиная с C11)
`int snprintf_s( char *restrict buffer, rsize_t bufsz, const char *restrict format, ...);`	(8)	(начиная с C11)

Загружает данные из заданных местоположений, преобразует их в эквиваленты символьных строк и записывает результаты в различные приемники / потоки:

1) Записывает результаты в выходной поток `stdout`.
2) Записывает результаты в выходной поток`stream`.
3) Записывает результаты в символьную строку`buffer`. Поведение не определено, если записываемая строка (плюс завершающий нулевой символ) превышает размер массива , на который указывает `buffer`.
4) Записывает результаты в символьную строку`buffer`. Написано не более `bufsz`- 1 символа. Результирующая символьная строка будет заканчиваться нулевым символом, если `bufsz`только он не равен нулю. Если `bufsz`равно нулю, ничего не записывается и `buffer`может быть нулевым указателем, однако возвращаемое значение (количество байтов, которые будут записаны, не включая нулевой терминатор) по-прежнему вычисляется и возвращается.

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

- спецификатор преобразования `%n`присутствует в `format`
- любой из аргументов, соответствующих `%s`нулевому указателю
- `stream` или `format`или `buffer`является нулевым указателем
- `bufsz` равно нулю или больше `RSIZE_MAX`
- ошибки кодирования возникают в любом из спецификаторов преобразования строк и символов
- (`sprintf_s`только для) строка, в которой будет храниться `buffer`(включая конечный `null`), будет превышать `bufsz`

Как и во всех функциях с проверкой границ, `printf_s`, `fprintf_s`, `sprintf_s`, и `snprintf_s`гарантированно доступны только в том случае, если `__STDC_LIB_EXT1__` определяется реализацией и если пользователь определяет `__STDC_WANT_LIB_EXT1__` для целочисленной константы 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 многобайтовую строку с нулевым окончанием, указывающую, как интерпретировать данные
...	-	аргументы, указывающие данные для печати. Если какой-либо аргумент после продвижения аргумента по умолчанию не соответствует типу, ожидаемому соответствующим спецификатором преобразования, или если аргументов меньше, чем требуется`format`, поведение не определено. Если аргументов больше, чем требуется `format`, то посторонние аргументы оцениваются и игнорируются.

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 задана 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 том случае, когда *используется, ширина задается дополнительным аргументом типа `int`, который появляется перед преобразуемым аргументом и аргументом, обеспечивающим точность, если он задан. Если значение аргумента отрицательное, то в результате -будет указан флаг и положительная ширина поля. (Примечание: это минимальная ширина: значение никогда не усекается.)

- (необязательно) . за которым следует целое число или*, или ни то, ни другое, определяющее *точность* преобразования. В случае, когда *используется, *точность* задается дополнительным аргументом типа `int`, который появляется перед аргументом, подлежащим преобразованию, но после аргумента, указывающего минимальную ширину поля, если он указан. Если значение этого аргумента отрицательно, оно игнорируется. Если ни число ни *не используется, то точность принимается равной нулю. Точные эффекты точности приведены в таблице ниже.

- (необязательно) *модификатор длины*, задающий размер аргумента (в сочетании со спецификатором формата преобразования он 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, записывается каждый байт до первого нулевого терминатора, но не включая его. Если используется спецификатор l, то аргумент должен быть указателем на начальный элемент массива `wchar_t`, который преобразуется в массив char как бы вызовом wcrtomb с нулевой инициализацией состояния преобразования.	N/A	N/A	`char*`	`wchar_t*`	N/A	N/A	N/A	N/A	N/A
d i	преобразует целое **число со знаком** в десятичное представление *[-]dddd*. *Точность* определяет минимальное количество цифр, которые должны появиться. Точность по умолчанию `равна 1`. Если и преобразованное значение, и точность равны `0`, то преобразование не приводит к появлению символов.	`подпись char`	`коротко`	`инт`	`долго`	`долго долго`	`intmax_t`	подписанный `size_t`	`ptrdiff_t`	N/A
o	преобразует целое **число без знака** в восьмеричное представление *oooo*. *Precision* задает минимальное количество отображаемых цифр. Точность по умолчанию `равна 1`. Если преобразованное значение и точность равны `0`, преобразование не приводит к символам. В *альтернативной реализации* точность увеличивается, если необходимо записать один ведущий ноль. В этом случае, если и преобразованное значение, и точность равны `0`, записывается одиночный `0`.	`символ без знака`	`без знака короткий`	`unsigned int`	`длинный без знака`	`без знака долго долго`	`uintmax_t`	`size_t`	неподписанная версия `ptrdiff_t`	N/A
x X	преобразует целое число без знака в шестнадцатеричное представление *hhhh*. Для **x**преобразования abcdefиспользуются буквы. Для **X**преобразования ABCDEFиспользуются буквы. *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. В *альтернативной реализации* **0x** или **0X**имеет префикс results , если преобразованное значение не равно нулю.									N/A
u	преобразует целое **число без знака** в десятичное представление *dddd*. *Точность* определяет минимальное количество цифр, которые должны появиться. Точность по умолчанию `равна 1`. Если преобразованное значение и точность равны `0`, преобразование не приводит к символам.									N/A
f F	преобразует **число с плавающей запятой** в десятичную систему счисления в стиле *[-]ddd.ddd*. *Точность* указывает точное количество цифр, которые будут отображаться после символа десятичной точки. Точность по умолчанию `- 6`. В *альтернативной реализации* символ десятичной запятой записывается даже в том случае, если за ним не следует никаких цифр. Стиль преобразования бесконечности и не-числа см. В примечаниях.	N/A	N/A	`двойной`	`двойной` (C99)	N/A	N/A	N/A	N/A	`длинный двойной`
e E	преобразует **число** с плавающей запятой в десятичную экспоненту. Для **e**преобразования используется стиль *[-]d.ddd***e**±*dd*. Для **E**преобразования используется стиль *[-]d.ddd***E**±*dd*. Показатель степени содержит не менее двух цифр, больше цифр используется только в случае необходимости. Если значение равно `0`, показатель степени также `равен 0`. *Точность* указывает точное количество цифр, которые будут отображаться после символа десятичной	N/A	N/A			N/A	N/A	N/A	N/A	

	точки. Точность по умолчанию `равна 6`. В *альтернативной реализации* символ десятичной точки записывается, даже если за ним не следуют цифры. Стиль преобразования бесконечности и не-числа см. В примечаниях.									
a **A** (C99)	преобразует **число** с плавающей запятой в шестнадцатеричную экспоненту. Для **a**преобразования используется стиль *[-]***0x***h.hhh***p±d***.* Для **A**преобразования используется стиль *[-]***0X***h.hhh***P±d***.* Первая шестнадцатеричная цифра не 0является, если аргумент является нормализованным значением с плавающей запятой. Если значение равно `0`, то показатель степени также `равен 0`. *Точность* определяет точное количество цифр, которые должны появляться после шестнадцатеричного символа точки. Точность по умолчанию достаточна для точного представления значения. В *альтернативной реализации* символ десятичной точки записывается, даже если за ним не следуют цифры. Стиль преобразования бесконечности и не-числа см. В примечаниях.	N/A	N/A			N/A	N/A	N/A	N/A	
g **G**	преобразует **число с плавающей запятой** в десятичную или десятичную экспоненту в зависимости от значения и *точности*. Для **g**стиля преобразования **ef**будет выполнено преобразование со стилем или. Для **G**стиля преобразования **EF**будет выполнено преобразование со стилем или. Пусть Рравна точности, если она ненулевая, `6`, если точность не указана, или `1`, если точность равна `0`. Тогда, если преобразование со стилем Ебудет иметь показательX: ▪ если *P> X ≥ -4*, преобразование выполняется со стилем **f**или **F**и точностью *P − 1 − X*. ▪ в противном случае преобразование выполняется со стилем **e**или **E**и точностью *P − 1*. Если *не запрашивается альтернативное представление*, конечные нули удаляются, а также символ десятичной точки удаляется, если не осталось дробной части. Стиль преобразования бесконечности и не-числа см. В примечаниях.	N/A	N/A			N/A	N/A	N/A	N/A	
n	возвращает **количество символов, записанных** до сих пор этим вызовом функции. Результат *записывается* в значение, на которое указывает аргумент. Спецификация может не содержать *флага*, *ширины поля*или *точности*.	`подписанный символ*`	`короткое*`	`int*`	`долго*`	`долго долго*`	`intmax_t*`	подписанный `size_t*`	`ptrdiff_t*`	N/A
p	записывает определенную реализацией последовательность символов, определяющую **указатель**.	`подписанный символ*`	N/A	`пустота*`	N/A	N/A	N/A	N/A	N/A	N/A

Функции преобразования с плавающей запятой преобразуют бесконечность в `inf`или `infinity`. Какой из них используется, определяется реализацией.

Not-a-number преобразуется в `nan`или `nan(`*`char_sequence`*`)`. Какой из них используется, определяется реализацией.

Вместо этого преобразования **F**, **E**, **G**, **A**выводят `INF`, `INFINITY`, `NAN`.

Несмотря `%c`на то, что ожидает `int`аргумент, безопасно передавать а `char`из-за целочисленного продвижения, которое происходит при вызове переменной функции.

Правильные спецификации преобразования для типов символов фиксированной ширины (`int8_t`и т. Д.) Определены в заголовке `<inttypes.h>`(хотя `PRIdMAX`, `PRIuMAX` ит. Д. Являются синонимами `%jd`, `%ju`и т. Д.).

Спецификатор преобразования для записи в память `%n` является общей целью эксплойтов безопасности, где строки формата зависят от пользовательского ввода и не поддерживаются семейством функций с проверкой границ`printf_s`.

После действия каждого спецификатора преобразования существует точка последовательности; это позволяет хранить несколько результатов `%n` в одной и той же переменной или, в крайнем случае, печатать строку, измененную более ранним `%n` в том же вызове.

Если спецификация преобразования недопустима, то поведение не определено.

Возвращаемое значение

1,2) количество символов, передаваемых в выходной поток, или отрицательное значение, если произошла ошибка вывода или ошибка кодирования (для спецификаторов преобразования строк и символов)

3) количество записанных символов `buffer`(не считая завершающего нулевого символа) или отрицательное значение, если произошла ошибка кодирования (для спецификаторов преобразования строк и символов).

4) количество символов (не включая завершающий нулевой символ), которые были бы записаны`buffer`, если `bufsz`бы были проигнорированы, или отрицательное значение, если бы произошла ошибка кодирования (для спецификаторов

преобразования строк и символов)

5,6) количество символов, передаваемых в выходной поток, или отрицательное значение, если произошла ошибка вывода, ошибка нарушения ограничений времени выполнения или ошибка кодирования.

7) количество символов, записанных `buffer`, не считая нулевого символа (который всегда записывается до тех пор, пока `buffer`не является нулевым указателем и `bufsz`не равен нулю и не большеRSIZE_MAX), или ноль при нарушениях ограничений времени выполнения и отрицательное значение при ошибках кодирования

8) количество символов, не включающих завершающий нулевой символ (который всегда записывается до тех пор, пока `buffer`не является нулевым указателем и `bufsz`не равен нулю и не большеRSIZE_MAX), который был бы записан`buffer`, если `bufsz`бы был проигнорирован, или отрицательное значение, если бы произошло нарушение ограничений времени выполнения или ошибка кодирования

Стандарт C и POSIX (http://pubs.opengroup.org/onlinepubs/9699919799/functions/fprintf.html) указывают, что поведение `sprintf`и его варианты не определены, когда аргумент перекрывается с целевым буфером. Пример:

```
sprintf(dst, "%s и %s", dst, t); // <- broken: неопределенное поведение
```

POSIX указывает (http://pubs.opengroup.org/onlinepubs/9699919799/functions/fprintf.html) , что `errno` устанавливается при ошибке. Он также определяет дополнительные спецификации преобразования, в частности поддержку переупорядочения аргументов (`n $` сразу после `%` указывает n'th аргумент).

Вызов `snprintf`с нулевым `bufsz`и нулевым указателями for `buffer`полезен для определения необходимого размера буфера для хранения выходных данных:

```
const char *fmt = "sqrt(2) = %f";
int sz = snprintf(NULL, 0, fmt, sqrt(2));
char buf[sz + 1]; // примечание +1 для завершения нулевого байта
snprintf(buf, sizeof buf, fmt, sqrt(2));
```

`snprintf_s`, точно так же , как `snprintf`, но в отличие `sprintf_s`от , будет усекать вывод , чтобы вписаться `bufsz-1`.

Запустите этот код

```
#include <stdio.h>

int main(void)
{
    const char* s = "Hello";
 printf("Strings - padding:\n");
 printf("\t.%10s.\n\t.%-10s.\n\t.%*s.\n", s, s, 10, s);
 printf("Strings - truncating:\n");
 printf("\t%.4s\n\t%.*s\n", s, 3, s);

 printf("Символы:\t%c%%%\n", 65);

 printf("Целые**числа**\n");
 printf("Десятичная дробь:\t%i %d%.6i %i %.0i %+i %i\n", 1, 2, 3, 0, 0, 4, -4);
 printf("Шестнадцатеричный:\t%x %x %X %#x\n", 5, 10, 10, 6);
 printf("Восьмеричный:\t\t%o%#o%#o\n", 10, 10, 4);

 printf("Плавающая точка\n");
 printf("Округление:\t%f%.0f%.32f\n", 1.5, 1.5, 1.3);
 printf("Заполнение:\t%05.2f %.2f %5.2f\n", 1.5, 1.5, 1.5);
 printf("Scientific:\t%E%e\n", 1.5, 1.5);
 printf("Hexadecimal:\t%a%A\n", 1.5, 1.5);
}
```

Вывод:

```
Струны - подкладка:
 . Привет.
 Привет .
 . Привет.
Строки - усечение:
 Ад
 Hel
Символы: A %
```

```
Целые числа
Десятичное: 1 2 000003 0 +4 -4
Шестнадцатеричное: 5 a A 0x6
Восьмеричное: 12 012 04
Плавающая точка
Округление: 1.500000 2 1.3000000000000000044408920985006261
Заполнение: 01.50 1.50 1.50
Научный: 1.500000E +00 1.500000e+00
Шестнадцатеричный: 0x1.8p +0 0X1.8P +0
```

Ссылки

- Стандарт C17 (ISO/IEC 9899:2018):
  - 7.21.6.1 Функция fprintf (p: 225-230)
  - 7.21.6.3 Функция printf (p: 236)
  - 7.21.6.5 Функция snprintf (p: 237)
  - 7.21.6.6 Функция sprintf (p: 237)
  - K.3.5.3.1 Функция fprintf_s (p: 430)
  - K.3.5.3.3 Функция printf_s (p: 432)
  - K.3.5.3.5 Функция snprintf_s (p: 432-433)
  - K.3.5.3.6 Функция sprintf_s (p: 433)
- Стандарт C11 (ISO/IEC 9899:2011):
  - 7.21.6.1 Функция fprintf (p: 309-316)
  - 7.21.6.3 Функция printf (p: 324)
  - 7.21.6.5 Функция snprintf (p: 325)
  - 7.21.6.6 Функция sprintf (p: 325-326)
  - K.3.5.3.1 Функция fprintf_s (p: 591)
  - K.3.5.3.3 Функция printf_s (p: 593-594)
  - K.3.5.3.5 Функция snprintf_s (p: 594-595)
  - K.3.5.3.6 Функция sprintf_s (p: 595-596)
- Стандарт C99 (ISO/IEC 9899:1999):
  - 7.19.6.1 Функция fprintf (p: 274-282)
  - 7.19.6.3 Функция printf (p: 290)
  - 7.19.6.5 Функция snprintf (p: 290-291)
  - 7.19.6.6 Функция sprintf (p: 291)
- Стандарт C89/C90 (ISO/IEC 9899:1990):
  - 4.9.6.1 Функция fprintf
  - 4.9.6.3 Функция printf
  - 4.9.6.5 Функция sprintf

См. Также

wprintf (C95) **fwprintf** (C95) **swprintf** (C95) **wprintf_s** (C11) (C11) **fwprintf_s** (C11) **swprintf_s** (C11) **snwprintf_s** (C11)	печатает отформатированный широкий символьный вывод в `stdout`, поток файла или буфер (функция)
	печатает отформатированный вывод в `stdout`, поток файла или буфер с помощью списка аргументов переменной (функции)

vprintf **vfprintf** **vsprintf** **vsnprintf** (C99) **vprintf_s** (C11) **vfprintf_s** (C11) **vsprintf_s** (C11) **vsnprintf_s** (C11)	
fputs	запись символьной строки в файловый поток (функция)
scanf **fscanf** **sscanf** **scanf_s** (C11) **fscanf_s** (C11) **sscanf_s** (C11)	считывает форматированные входные данные из `stdin`, файлового потока или буфера (функции)

C++ документация для **printf**, **fprintf**, **sprintf**, **snprintf**