Synopsis

```
#include <stdio.h>

int fprintf(
  FILE * restrict stream, const char * restrict format, ... );
int printf(
  const char * restrict format, ...);
int sprintf(
  char * restrict s,
  const char * restrict format, ...);
int snprintf(
  char * restrict s, size_t n,
  const char * restrict format, ...);     // C99
```

```
#include <stdio.h>
#include <wchar.h>

int fwprintf(
  FILE * restrict stream,
  const wchar_t * restrict format, ... );
int wprintf(
  const wchar_t * restrict format, ...);
int swprintf(
  wchar_t *s, size_t n, const wchar_t *format, ...);
```

Функция **fprintf** выполняет форматирование вывода, отправляя вывод в поток, указанный в качестве первого аргумента. Второй аргумент - это строка управления format (форматом). В зависимости от содержимого управляющей строки могут потребоваться дополнительные аргументы. Ряд выходных символов генерируется по указанию управляющей строки; эти символы отправляются в указанный поток.

Функция **printf** связана с **fprintf**, но отправляет символы в стандартный выходной поток **stdout**.

Функция **sprintf** позволяет сохранять выходные символы в строковом буфере **s**. Последний нулевой символ выводится в s после того, как все символы, указанные в строке управления, были выведены. Программист несет ответственность за то, чтобы область строки назначения **sprintf** была достаточно большой, чтобы содержать выходные данные, сгенерированные операцией форматирования. Однако функция **swprintf**, в отличие от sprintf, включает в себя счетчик максимального количества широких символов (включая завершающий нулевой символ), которые должны быть записаны в выходную строку **s**. В C99 был добавлен **sprintf** для подсчета неширокой функции.

Значением, возвращаемым этими функциями, является **EOF**, если во время операции вывода произошла ошибка; в противном случае результатом является некоторое значение, отличное от **EOF**. В стандарте C и в большинстве современных реализаций функции возвращают количество символов, отправленных в выходной поток, если ошибки не возникает. В случае **sprintf** в число не входит завершающий нулевой символ. (Стандарт C позволяет этим функциям возвращать любое отрицательное значение в случае ошибки.)

C89 (Поправка 1) определяет три широкоформатных версии этих функций: **fwprintf**, **wprintf** и **swprintf**. Результатом этих функций является концептуально широкая строка, и они преобразуют свои дополнительные аргументы в широкие строки под управлением операторов преобразования. Мы обозначаем эти функции как семейство функций **wprintf** или просто функции **wprintf**, чтобы отличать их от оригинальных байтово-ориентированных функций printf. Согласно поправке 1 спецификатор размера l может применяться к операторам преобразования **c** и **s** в функциях printf и wprintf.

C99 вводит операторы преобразования a и A для шестнадцатеричных преобразований с плавающей точкой и модификаторы длины **hh, ll(el-el), j, z** и **t**.

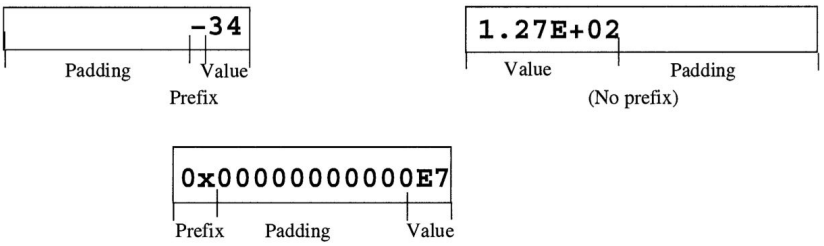
Ссылки **EOF** 15.1; шестнадцатеричный формат с плавающей запятой 2.7.2; **scanf** 15.8; **stdout** 15.4; широкие символы 2.1.4

15.11.1 Формат Вывода

Строка управления - это просто текст, который нужно скопировать дословно, за исключением того, что строка может содержать спецификации преобразования. В стандарте C управляющая строка представляет собой (не интерпретируемую) многобайтовую последовательность символов, которая начинается и заканчивается в своем начальном состоянии сдвига. В функциях **wprintf** это строка широких символов.

Спецификация преобразования может требовать обработки некоторого числа дополнительных аргументов, что приводит к операции форматированного преобразования, которая генерирует выходные символы, явно не содержащиеся в управляющей строке. Должно быть точно правильное количество аргументов, каждый из которых точно подходящего типа, чтобы соответствовать спецификациям преобразования в управляющей строке. Дополнительные аргументы игнорируются, но результат от слишком малого количества аргументов непредсказуем. Если какая-либо спецификация преобразования искажена, то последствия непредсказуемы. Спецификации преобразования для вывода аналогичны тем, которые используются для ввода в **fscanf** и связанных с ним функциях; различия обсуждаются в разделе 15.8.2. Сразу после действий, вызываемых каждой спецификацией преобразования, есть точка последовательности.

Последовательность символов или широких символов, выводимых для спецификации преобразования, может быть концептуально разделена на три элемента: собственно преобразованное значение, которое отражает значение преобразованного аргумента; префикс, который, если присутствует, обычно **+, -** или **пробел**; и заполнение, которое представляет собой последовательность пробелов или нулевых цифр, добавляемых при необходимости, чтобы увеличить ширину выходной последовательности до указанного минимума. Префикс всегда предшествует преобразованному значению. В зависимости от спецификации преобразования, заполнение может предшествовать префиксу, отделять префикс от преобразованного значения или следовать за преобразованным значением. Примеры показаны на следующем рисунке; заключенные в них поля показывают степень вывода, определяемую спецификацией преобразования.заключенные в них поля показывают степень вывода, определяемую спецификацией преобразования.



15.11.2 Характеристики преобразования

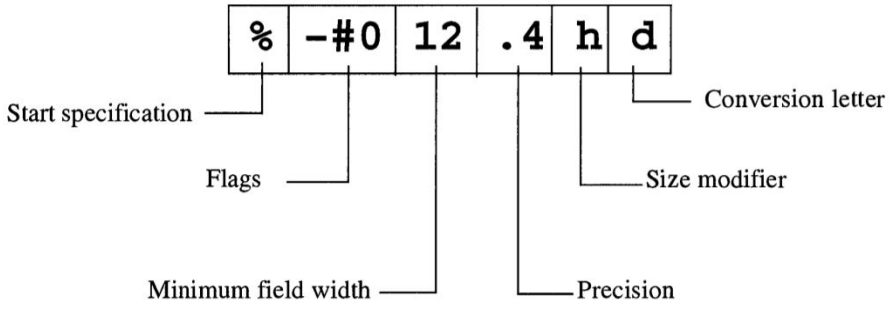
В дальнейшем термины символы, буквы и т. д. Следует понимать как обычные символы или буквы (байты) в случае функций **printf** и широкие символы или буквы в случае функций **wprintf**. Например, в **wprintf** спецификации преобразования начинаются со знака процента широкого символа, **%.**

Спецификация преобразования начинается со знака процента,**%,** и имеет следующие элементы в порядке:

1. Ноль или более символов флага (**- ,+, 0, # или пробел**), которые изменяют значение операции преобразования.
2. Необязательная минимальная ширина поля, выраженная в виде десятичной целочисленной константы.
3. Необязательная спецификация точности, выраженная в виде точки, за которой необязательно следует десятичное целое число.
4. Необязательная спецификация размера, выраженная в виде одной из букв **ll, l, L, h, hh, j, z** или **t.**
5. Операция преобразования, один символ из набора **a, A, c, d, e, E, f, g, G,i, n, 0, p, s, u, x, X и %.**

Буквы спецификации размера **L** и **h**, а также операции преобразования **i, p** и **n** были введены в C89. Буквы спецификации размера **ll, hh, j, z** и **t**, а также операции преобразования **a и A** были введены в C99.

Конверсионное письмо завершает спецификацию. Спецификация преобразования **% - #012.4 hd** показана далее разбитой на составные элементы:



15.11.3 Флаги преобразования

Необязательные символы флага изменяют значение основной операции преобразования:

-	Выровнять по левому краю значение в пределах ширины поля.
0	Используйте 0 для символа пэда, а не пробела.
+	Всегда производите знак + или -.
пробел	Всегда производите либо знак - либо пробел.
#	Использовать вариант основной операции преобразования.

Эффекты флаговых символов теперь описаны более подробно.

Флаг - Если присутствует флаг минус, то преобразованное значение будет выровнено по левому краю поля, то есть любой отступ будет помещен справа от преобразованного значения. Если знак минус отсутствует, преобразованное значение будет выровнено по полю справа. Этот флаг имеет значение только в том случае, если указана явная минимальная ширина поля и преобразованное значение меньше этой минимальной ширины; в противном случае значение заполнит поле без заполнения.

Флаг 0 Если присутствует флаг **0** (ноль), то **0** будет использоваться в качестве символа заполнения, если заполнение должно быть размещено слева от преобразованного значения. Флаг **0** имеет значение только тогда, когда указана явная минимальная ширина поля и преобразованное значение меньше этой минимальной ширины. В целочисленных преобразованиях этот флаг заменяется спецификацией точности.

Если флаг с нулевой цифрой отсутствует, то в качестве символа pad будет использоваться пробел. Пробел всегда используется как символ заполнения, если заполнение должно располагаться справа от преобразованного значения, даже если присутствует символ **- флаг**.

Флаг + Если присутствует флаг +, то результат преобразования со знаком всегда будет начинаться со знака, то есть явный + будет предшествовать преобразованному положительному значению. (Отрицательным значениям всегда предшествует - независимо от того, указан ли флаг плюса.) Этот флаг имеет значение только для операций преобразования a, **A, d, e, E, f, g, G и i.**

Флаг пробел Если присутствует флаг пробела и первый символ в преобразованном значении, полученном в результате преобразования со знаком, не является знаком (+ или -), то пробел будет добавлен перед преобразованным значением. Добавление этого пробела слева не зависит от любого отступа, который может быть размещен слева или справа под контролем символа - flag. Если в одной спецификации преобразования указаны флаги **пробела** и **+,** флаг пробела игнорируется, поскольку флаг + гарантирует, что преобразованное значение всегда будет начинаться со знака. Этот флаг относится только к операциям преобразования a, **A, d, e, E, f, g, G и i.**

Флаг # Если присутствует флаг **#,** то используется альтернативная форма основной операции преобразования. Этот флаг относится только к операциям

преобразования **a, A, e, E, f, g, G, i, o, x и X.** Модификации, подразумеваемые флагом **#**, описаны в связи с соответствующими операциями преобразования.

15.11.4 Минимальная ширина поля.

Можно указать необязательную минимальную ширину поля, выраженную в виде десятичной целочисленной константы. Константа должна быть непустой последовательностью десятичных цифр, которая не начинается с нулевой цифры (которая будет принята за флаг **0**). Если преобразованное значение (включая префикс) приводит к меньшему количеству символов, чем указанная ширина поля, то символы заполнения используются для дополнения значения до указанной ширины. Если преобразованное значение приводит к большему количеству символов, чем указанная ширина поля, то поле расширяется, чтобы вместить его без заполнения.

Ширина поля также может быть указана звездочкой *, и в этом случае используется аргумент типа **int**, который определяет минимальную ширину поля. Результат задания отрицательной ширины непредсказуем.

Пример

Следующие два вызова **printf** приводят к одному и тому же выводу:

```
int width=5, value;
…
printf("%5d", value);
printf("%*d", width, value);
```

15.11.5 Точность

Необязательная спецификация точности может быть указана и выражена в виде периода, за которым следует необязательное десятичное целое число. Спецификация точности используется для контроля:

1. минимального количества цифр, которые будут напечатаны для преобразования **d, i, o, u, x** и **X**
2. количества цифр справа от десятичной точки в преобразованиях **e, E** и **f**
3. количества значащих цифр в **g** и **G** преобразованиях
4. максимального количества символов, которое будет записано из строки в преобразовании **s**

Если период появляется, но целое число отсутствует, то предполагается, что целое число равно нулю, что обычно имеет иной эффект, чем пропуск всей спецификации точности.

Точность также может быть указана звездочкой после точки, в этом случае используется аргумент типа **int**, который задает точность. Если ширина и точность поля указаны звездочками, то аргумент ширины поля предшествует аргумент точности.

15.11.6 Спецификация размера

Необязательный модификатор размера, одна из последовательностей букв **ll (ell-ell), l (ell), L, h, hh, j, z,** или **t,** может предшествовать некоторым операциям преобразования.

Буква **l** в сочетании с операциями преобразования **d, i, o, u, x и X** указывает, что аргумент преобразования имеет тип **long** или **unsigned long.** В сочетании с преобразованием n он указывает, что аргумент имеет тип **long ***. В **C89** модификатор **l** также может использоваться с **c**, в этом случае аргумент имеет тип **wint_t** или с **s**, и в этом случае он указывает, что аргумент имеет тип **wchar_t ***. Модификатор **l** не действует при использовании с **a, A, e, E, f, F, g и G**; сравните это с модификатором **L** и будьте осторожны, который вы используете.

Модификатор **ll** в сочетании с операциями преобразования **d, i, o, u, x и X** указывает, что аргумент преобразования имеет тип **long long int** или **unsigned long long int.** В сочетании с преобразованием n модификатор **ll** указывает, что аргумент имеет тип **long long int ***. Модификатор размера **ll** был представлен в **C99**.

Буква **h** в сочетании с операциями преобразования **d, i, o, u, x и X** указывает, что аргумент преобразования имеет тип short или unsigned short. То есть, хотя аргумент был бы преобразован в **int** или без знака при продвижении аргумента, он должен быть преобразован в короткий или без знака незадолго до преобразования. В сочетании с преобразованием n модификатор **h** указывает, что аргумент имеет тип **short ***. Модификатор размера **h** был представлен в **C89**.

Модификатор **hh** в сочетании с операциями преобразования **d, i, o, u, x и X** указывает, что аргумент преобразования имеет тип char или unsigned char. То есть, хотя аргумент был бы преобразован в **int** или **unsigned** при продвижении аргумента, он должен быть преобразован в **char** или **unsigned char** перед преобразованием. В сочетании с преобразованием n модификатор hh указывает, что аргумент имеет тип **char *** со знаком. Модификатор размера **hh** доступен в **C99**.

Буква **L** в сочетании с операциями преобразования **a, A, e, E, f, F, g и G** указывает, что аргумент имеет тип **long double**. Модификатор размера **L** был представлен в **C89.** Будьте осторожны, используйте **L** а не **l** для **long double**, так как **l** не влияет на эти операции.

Модификатор **j** в сочетании с операциями преобразования **d, i, o, u, x и X** указывает, что аргумент преобразования имеет тип **intmax_t** или **uintmax_t**. В сочетании с преобразованием n модификатор j указывает, что аргумент имеет тип **intmax_t ***. Модификатор размера **j** был представлен в **C99**.

Модификатор **z** в сочетании с операциями преобразования **d, i, o, u, x и X** указывает, что аргумент преобразования имеет тип **size_t**. В сочетании с

преобразованием **n** модификатор **z** указывает, что аргумент имеет тип **siize_t** *. Модификатор размера **z** был представлен в **C99**.

Модификатор **t** в сочетании с операциями преобразования **d, i, o, u, x** и **X** указывает, что аргумент преобразования имеет тип **ptrdiff_t.** В сочетании с преобразованием **n** модификатор **t** указывает, что аргумент имеет тип **ptrdiff_t** *. Модификатор размера был представлен в **C99**.

15.11.7 Операции преобразования

Операция преобразования выражается одним символом: **a, A, c, d, e, E, f, g, G, i, n, o, p, s, u, x, X** или **%**. Указанное преобразование определяет допустимые символы флага и размера, ожидаемый тип аргумента и внешний вид выходных данных. Таблица **15-6** суммирует операции преобразования. Каждая операция затем обсуждается индивидуально.

d и i конверсии. Десятичное преобразование со знаком выполняется. Аргумент должен иметь тип **int**, если модификатор размера не используется, тип **short**, если используется **h,** или тип **Long**, если используется **l**. Оператор **i** присутствует в стандарте С для совместимости с **fscanf**; он распознается на выходе для единообразия, где он идентичен оператору **d**.

Преобразованное значение состоит из последовательности десятичных цифр, которая представляет абсолютное значение аргумента. Эта последовательность максимально короткая, но не короче указанной точности. Преобразованное значение будет иметь ведущие нули, если это необходимо для обеспечения точности спецификации; эти ведущие нули не зависят от каких-либо дополнений, которые могут также вводить ведущие нули. Если точность равна **1** (по умолчанию), то преобразованное значение не будет иметь начального **0,** если аргумент не равен **0**, и в этом случае выводится один **0**. Если точность равна **0**, а аргумент равен **0,** тогда преобразованное значение будет пустым (пустая строка).

Префикс вычисляется следующим образом. Если аргумент отрицательный, префикс является знаком минус. Если аргумент неотрицательный и указан флаг **+,** то префикс является знаком плюс. Если аргумент неотрицательный, указан флаг пробела, а флаг **+** не указан, то префикс - это пробел. В противном случае префикс пуст. Флаг **#** не имеет отношения к преобразованиям **d и i.** В таблице 15-7 приведены примеры преобразования d.

Table 15–6 Output conversion specifications

Conversion	Defined flags - + # 0 *space*	Size modifier	Argument type	Default precision[a]	Output
d, i[b]	- + 0 *space*	*none*	**int**	1	dd...d
		h	**short**		-dd...d
		l	**long**		+dd...d
u	- + 0 *space*	*none*	**unsigned int**	1	dd...d
		h	**unsigned short**		
		l	**unsigned long**		
o	- + # 0 *space*	*none*	**unsigned int**	1	oo...o
		h	**unsigned short**		**0**oo...o
		l	**unsigned long**		
x, X	- + # 0 *space*	*none*	**unsigned int**	1	hh...h
		h	**unsigned short**		**0x**hh...h
		l	**unsigned long**		**0X**hh...h
f	- + # 0 *space*	*none*	**double**	6	d...d.d...d
		l	**double**		-d...d.d...d
		L	**long double**		+d...d.d...d
e, E	- + # 0 *space*	*none*	**double**	6	d.d...d**e**+dd
		l	**double**		-d.d...d**E**-dd
		L	**long double**		
g, G	- + # 0 *space*	*none*	**double**	6	*like* **e**, **E**,
		l	**double**		*or* **f**
		L	**long double**		
a, A[c]	- + # 0 *space*	*none*	**double**	6	**0x**h.h...h**p+**dd
		l	**double**		**-0X**h.h...h**P-**
		L	**long double**		dd
c	-	*none*	**int**	1	c
		l[d]	**wint_t**		
s	-	*none*	**char ***	×	cc...c
		l[c]	**wchar_t ***		
p[b]	*impl. defined*	*none*	**void ***	1	*impl. defined*
n[b]		*none*	**int ***	n/a	*none*
		h	**short ***		
		l	**long ***		
%		*none*	*none*	n/a	**%**

a - Точность по умолчанию, если ничего не указано.
б - Введено в C89. Преобразования i и d эквивалентны на выходе.
c - введено в C99
d - Введено в C99 (Поправка 1).

U преобразование. Выполняется десятичное преобразование без знака. Аргумент должен иметь тип **unsigned**, если не используется модификатор размера, тип **unsigned short**, если используется **h**, или тип **unsigned long**, если используется **l**.

Преобразованное значение состоит из последовательности десятичных цифр, представляющих значение аргумента. Эта последовательность максимально короткая, но не короче указанной точности.

Table 15–7 Examples of the **d** conversion

Sample format	Sample output Value = 45	Sample output Value = –45
%12d	45	-45
%012d	000000000045	-00000000045
% 012d	00000000045	-00000000045
%+12d	+45	-45
%+012d	+00000000045	-00000000045
%-12d	45	-45
%- 12d	45	-45
%-+12d	+45	-45
%12.4d	0045	-0045
%-12.4d	0045	-0045

Преобразованное значение будет иметь ведущие нули, если это необходимо для обеспечения точности спецификации; эти ведущие нули не зависят от каких-либо дополнений, которые могут также вводить ведущие нули. Если точность равна 1 (по умолчанию), то преобразованное значение не будет иметь начального 0, если аргумент не равен **0**, и в этом случае выводится один **0**. Если точность и аргумент равны **0**, то преобразованное значение является пустым (нулевая строка). Префикс всегда пуст. Флаги **+,** пробел и **#** не имеют отношения к операции преобразования **u**. Таблица 15-8 показывает примеры преобразования.

Table 15–8 Examples of the **u** conversion

Sample format	Sample output Value = 45	Sample output Value = –45
%14u	45	4294967251
%014u	00000000000045	00004294967251
%#14u	45	4294967251
%#014u	00000000000045	00004294967251
%-14u	45	4294967251
%-#14u	45	4294967251
%14.4u	0045	4294967251
%-14.4u	0045	4294967251

O преобразование. Восьмеричное преобразование без знака выполняется. Аргумент должен иметь тип **unsigned**, если не используется модификатор размера, тип **unsigned short**, если используется **h**, или тип **unsigned long**, если используется **l.** Преобразованное значение состоит из последовательности восьмеричных цифр, представляющих значение аргумента. Эта последовательность максимально короткая, но не короче указанной точности. Преобразованное значение будет иметь ведущие нули, если это необходимо для обеспечения точности спецификации; эти ведущие нули не зависят от каких-либо дополнений, которые могут также вводить ведущие

нули. Если точность равна **1** (по умолчанию), то преобразованное значение не будет иметь начального **0**, если аргумент не равен **0**, и в этом случае выводится один **0**. Если точность равна **0**, а аргумент равен **0**, тогда преобразованное значение будет пустым (пустая строка).
Если присутствует флаг **#,** то префикс равен **0**. Если флаг **#** отсутствует, то префикс пуст. Флаги **+** и **пробел** не имеют отношения к операции преобразования **o**. В таблице **15-9** приведены примеры преобразования **o.**

Table 15–9 Examples of the o conversion

Sample format	Sample output Value = 45	Sample output Value = –45
%14o	55	37777777723
%014o	00000000000055	00037777777723
%#14o	055	037777777723
%#014o	00000000000055	00037777777723
%-14o	55	377777777723
%-#14o	055	037777777723
%14.4o	0055	37777777723
%-#14.4o	00055	037777777723

X и X преобразований. Выполняется шестнадцатеричное преобразование без знака. Аргумент должен иметь тип **unsigned**, если не используется модификатор размера, тип **unsigned short**, если используется **h**, или тип **unsigned long**, если используется **l**.

Преобразованное значение состоит из последовательности шестнадцатеричных цифр, представляющих значение аргумента. Эта последовательность максимально короткая, но не короче указанной точности. Операция x использует **0123456789abcdef** в качестве цифр, тогда как операция **X** использует **0123456789ABCDEF.** Преобразованное значение будет иметь ведущие нули, если это необходимо для обеспечения точности спецификации; эти ведущие нули не зависят от каких-либо дополнений, которые могут также вводить ведущие нули. Если точность равна **1**, то преобразованный

Значение не будет иметь начального **0**, если аргумент не равен **0**, и в этом случае выводится один **0**.

Если точность равна **0**, а аргумент равен **0**, тогда преобразованное значение будет пустым (пустая строка). Если точность не указана, то предполагается, что точность равна **1**. Если присутствует флаг **#,** то префикс равен **0x** (для операции x) или **0X** (для операции X). Если флаг **#** отсутствует, то префикс пуст. **+** и **пробелы** не имеют значения. В таблице **15-10** приведены примеры X и X преобразований.

Table 15–10 Examples of the **x** and **X** conversions

Sample format	Sample output Value = 45	Sample output Value = –45
`%12x`	`2d`	`fffffd3`
`%012x`	`00000000002d`	`0000fffffd3`
`%#12X`	`0X2D`	`0XFFFFFFD3`
`%#012X`	`0X000000002D`	`0X00FFFFFFD3`
`%-12x`	`2d`	`fffffd3`
`%-#12x`	`0x2d`	`0xfffffd3`
`%12.4x`	`002d`	`fffffd3`
`%-#12.4x`	`0x002d`	`fffffd3`

Преобразование c. Аргумент печатается в виде символа или расширенного символа. Один аргумент потребляется. Флаги **+, пробел** и **#**, а также спецификация точности не имеют отношения к операции преобразования **c**. Преобразования, применяемые к символу аргумента, зависят от того, присутствует ли спецификатор размера **l** и используется ли **printf** или **wprintf**. Эти возможности перечислены в таблице **15-13**. В таблице **15-12** приведены примеры преобразования c.

Table 15–11 Conversions of the **c** specifier

Function	Size specifier	Argument type	Conversion
`printf`	none	`int`	argument is converted to **`unsigned char`** and copied to the output
	`l`	`wint_t`	argument is converted to **`wchar_t`**, converted to a multibyte characters as if by **`wcrtomb`**[a], and output
`wprintf`	none	`int`	argument is converted to a wide character as if by **`btowc`** and copied to the output
	`l`	`wint_t`	argument is converted to **`wchar_t`** and copied to the output

[a] The conversion state for the **`wcrtomb`** function is set to zero before the character is converted.

Table 15–12 Examples of the **c** conversion

Sample format	Sample output Value = `'*'`
`%12c`	`*`
`%012c`	`00000000000*`
`%-12c`	`*`

s преобразование. Аргумент печатается в виде строки. Один аргумент потребляется. Если спецификатор размера **l** отсутствует, аргумент должен быть указателем на массив любого символьного типа. Если присутствует **l**, аргумент должен иметь тип **wchar_t** * и обозначать последовательность широких символов. Префикс всегда пуст. Флаги **+, пробел** и **#** не имеют отношения к **s**-преобразованию.

Если спецификация точности не указана, то преобразованное значение представляет собой последовательность символов в строковом аргументе, но не включая завершающий нулевой символ или нулевой широкий символ. Если задана точная спецификация **p**, то преобразованное значение - это первые **p** символов выходной строки или до, но не включая завершающий нулевой символ, в зависимости от того, что меньше. Когда задана точность, строка аргумента не обязательно должна заканчиваться нулевым символом, если она содержит достаточно символов для получения максимального количества выходных символов. При записи многобайтовых символов (**printf**, с **l**) ни в коем случае не будет записываться частичный многобайтовый символ, поэтому фактическое количество записанных байтов может быть меньше **p**.

Преобразования, которые происходят в строке аргумента, зависят от того, присутствует ли спецификатор размера **l** и используются ли функции **printf** или **wprintf**. Возможности перечислены в таблице **15-13**. Таблица **15-14** показывает примеры преобразования.

Table 15–13 Conversions of the **s** specifier

Function	Size specifier	Argument type	Conversion
`printf`	none	`char *`	characters from the argument string are copied to the output
	`l`	`wchar_t *`	wide characters from the argument string are converted to multibyte characters as if by `wcrtomb`[a]
`wprintf`	none	`char *`	multibyte characters from the argument string are converted to wide characters as if by `mbrtowc`[a]
	`l`	`wchar_t *`	wide characters from the argument string are copied to the output

a - The conversion state for the wertomb or mbrtowc function is set to zero before the first character is converted. Subsequent conversions use the state as modified by the preceding characters.

Table 15–14 Examples of the **s** conversion

Sample format	Sample output Value = `"zap"`	Sample output Value = `"longish"`
`%12s`	`zap`	`longish`
`%12.5s`	`zap`	`longi`
`%012s`	`000000000zap`	`00000longish`
`%-12s`	`zap`	`longish`

p преобразование. Аргумент должен иметь тип **void ***, и он печатается в формате, определяемом реализацией. Для большинства компьютеров это, вероятно, будет таким же, как формат, создаваемый преобразованиями **o, x** или **X**. Этот оператор преобразования встречается в стандарте **C**, но в остальном он необычен.

n преобразование. Аргумент должен иметь тип i**nt ***, если модификатор размера не используется, введите **long ***, если используется спецификатор **l**, или введите **short ***, если используется спецификатор **h**. Вместо вывода символов этот оператор преобразования вызывает вывод числа символов, поэтому далеко записать в обозначенное целое число. Этот оператор преобразования встречается в стандарте **C**, но в остальном он необычен.

f и F преобразования. Выполняется десятичное преобразование со знаком и плавающей точкой. Используется один аргумент, который должен иметь тип **double**, если модификатор размера не используется, или тип **long double**, если используется **L**. Если предоставляется аргумент типа **float**, он преобразуется в тип **double** обычными аргументами продвижения, поэтому он работает с использованием **%f** для вывода числа типа **float**.

Преобразованное значение состоит из последовательности десятичных цифр, возможно, со встроенной десятичной точкой, которая представляет приблизительное абсолютное значение аргумента.

По крайней мере одна цифра появляется перед десятичной точкой. Точность определяет количество цифр после десятичной точки. Если точность равна **0**, то после десятичной точки цифры не отображаются. Кроме того, десятичная точка также не появляется, если присутствует флаг **#.** Если точность не указана, то предполагается точность **6**.

Если значение с плавающей запятой не может быть представлено точно в количестве произведенных цифр, тогда преобразованное значение должно быть результатом округления точного значения с плавающей запятой до числа произведенных десятичных разрядов. (Некоторые реализации C не выполняют правильное округление во всех случаях.)

В **C99**, если значение с плавающей запятой представляет бесконечность, тогда преобразованное значение с использованием оператора f является одним из значений **inf, -inf, infinity** или **-infinity**. (То, что выбрано, определяется реализацией.) Если значение с плавающей запятой представляет **NaN**, то преобразованное значение с использованием оператора **f** является одним из **nan, -nan, nan (...)** или **-nan (...),** где «...» - это определенная реализацией последовательность букв, цифр или подчеркиваний. Оператор **F** преобразует бесконечность и **NaN**, используя заглавные буквы. Флаги **#** и **0** не влияют на преобразование бесконечности или **NaN**.

Префикс вычисляется следующим образом. Если аргумент отрицательный, префикс является знаком минус. Если аргумент неотрицательный и указан флаг **+**, то префикс является знаком плюс. Если аргумент неотрицательный, указан флаг пробела, а флаг **+** не указан, то префикс - это **пробел**. В противном случае префикс пуст. В таблице **15-15** приведены примеры **f-**преобразования.

Table 15–15 Examples of the **f** conversion

Sample format	Sample output Value = 12.678	Sample output Value = –12.678
%10.2f	12.68	-12.68
%010.2f	000000012.68	-00000012.68
% 010.2f	00000012.68	-00000012.68
%+10.2f	+12.68	-12.68
%+010.2f	+00000012.68	-00000012.68
%-10.2f	12.68	-12.68
%- 10.2f	12.68	-12.68
%-+10.4f	+12.6780	-12.6780

e и E преобразования. Преобразование с плавающей запятой со знаком выполняется. Используется один аргумент, который должен иметь тип **double**, если не используется спецификатор размера, или тип **long double**, если используется **L.** Аргумент типа **float** разрешен, как для преобразования **f**. Электронное преобразование описано; **E** - преобразование отличается только тем, что буква **E** появляется всякий раз, когда **e** появляется в **e-преобразовании**.

Преобразованное значение состоит из десятичной цифры, затем, возможно, десятичной точки и нескольких десятичных цифр, затем буквы **e**, затем знака **плюс** или **минус**, и, наконец, как **минимум** еще **двух десятичных цифр**. Если значение не равно нулю, часть перед буквой e представляет значение в диапазоне от **1.0** до **9.99** .... Часть после буквы e представляет значение экспоненты в виде десятичного целого числа со знаком. Значение первой части, умноженное на **10** и увеличенное до значения **второй части**, приблизительно равно абсолютному значению аргумента. Количество цифр экспоненты одинаково для всех значений и является максимальным числом, необходимым для представления диапазона типов с плавающей точкой реализации. Таблица **15-16** показывает примеры **e и E преобразований**.

Table 15–16 Examples of **e** and **E** conversions

Sample format	Sample output Value = 12.678	Sample output Value = –12.678
%10.2e	1.27e+01	-1.27e+01
%010.2e	00001.27e+01	-0001.27e+01
% 010.2e	0001.27e+01	-0001.27e+01
%+10.2E	+1.27E+01	-1.27E+01
%+010.2E	+0001.27E+01	-0001.27E+01
%-10.2e	1.27e+01	-1.27e+01
%- 10.2e	1.27e+01	-1.27e+01
%-+10.2e	+1.27e+01	-1.27e+01

Точность определяет количество цифр после десятичной точки; если не указан, то предполагается **6**. Если точность равна **0**, то после десятичной точки цифры не отображаются. Кроме того, десятичная точка также не появляется, если присутствует флаг **#.** Если значение с плавающей запятой не может быть представлено точно в количестве произведенных цифр, тогда преобразованное значение получается округлением точного значения с плавающей запятой. Префикс рассчитывается так же, как и для конвертации в фунтах стерлингов. Значения бесконечности или **NaN** конвертируются, как указано для преобразования **f** и **F**.

g и G преобразования. Преобразование с плавающей запятой со знаком выполняется. Используется один аргумент, который должен иметь тип **double**, если не используется спецификатор размера, или тип long double, если используется **L**. Аргумент типа float разрешен, как для преобразования **f.** Только данный оператор преобразования обсуждается позже; операция **G** идентична, за исключением того, что где **g** использует **e** преобразование, **G** использует **E** преобразование. Если указанная точность меньше **1**, то используется точность **1**. Если точность не указана, то предполагается точность **6**.

Данное преобразование начинается так же, как преобразование **f** или **e**; какой из них будет выбран, зависит от значения, которое нужно преобразовать. В спецификации стандарта **C** говорится, что **e**-преобразование используется только в том случае, если показатель, полученный в результате e-преобразования, меньше - **4** или больше или равен указанной точности. Некоторые другие реализации используют **e**-преобразование, если показатель степени меньше - **3** или строго больше, чем указанная точность.

Преобразованное значение (с помощью **f** или **e**) затем дополнительно модифицируется путем удаления конечных нулей справа от десятичной точки. Если результат не имеет цифр после десятичной точки, то десятичная точка также удаляется. Если присутствует флаг **#**, это удаление нулей и десятичной точки не происходит.

Префикс вычисляется как для **f** и **e** преобразований. Значения бесконечности или NaN конвертируются, как указано для преобразования **f** и **F**.

a и A преобразования. Эти преобразования являются новыми в **C99**. Выполняется шестнадцатеричное преобразование со знаком с плавающей запятой. Используется один аргумент, который должен иметь тип double, если не используется спецификатор размера, или тип **long double**, если используется **L**.

Разрешается аргумент типа float, как и для других преобразований с плавающей точкой. Конверсия описана; преобразование A отличается использованием заглавных букв для **шестнадцатеричных цифр**, **префикса** (0X) и **буквы степени** (P).

Преобразованное значение состоит из **шестнадцатеричной цифры**, затем, возможно, **десятичной точки** и нескольких **шестнадцатеричных цифр**, затем буквы **p**, затем знака **плюс** или **минус**, а затем, наконец, одной или нескольких **десятичных цифр**. Если значение не равно нулю или не денормализовано, начальная шестнадцатеричная цифра не равна нулю. Часть после буквы **p** представляет значение двоичного показателя в виде десятичного целого числа со знаком.

Точность определяет количество **шестнадцатеричных цифр**, которые должны появляться после **десятичной точки**; если не указан, то появляется достаточно цифр, чтобы различать значения типа **double**. (Если **FLT RADIX** равен **2**, то точности по умолчанию достаточно для точного представления значений.) Если точность равна **0**, то после **десятичной точки цифры** не отображаются; кроме того, **десятичная точка** также не появляется, если не присутствует флаг **#.** Если значение с плавающей запятой не может быть представлено точно в количестве произведенных **шестнадцатеричных цифр**, тогда преобразованное значение получается округлением точного значения с плавающей запятой. Префикс вычисляется как для **f-преобразования**.

Значения бесконечности или **NaN** конвертируются, как указано для преобразования **f** и **F**.

% Преобразования. Один знак процента напечатан. Поскольку знак процента используется для обозначения начала спецификации преобразования, необходимо написать два из них, чтобы один из них был напечатан. Аргументы не используются, а префикс пуст.

Стандарт C не допускает присутствия каких-либо флаговых символов, модификаторов минимальной ширины, точности или размера; полная спецификация преобразования должна быть **%%**. Однако другие реализации C выполняют заполнение так же, как и для любой другой операции преобразования; например, спецификация преобразования**% 05%** печатает **0000%** в этих реализациях. Флаги **+**, **пробел** и **#**, спецификация точности и спецификации размера никогда не имеют отношения к операции **% преобразования**.

Пример

Следующая двухстрочная программа известна как **quine** - самовоспроизводящаяся программа. При выполнении он напечатает свою копию на стандартном выходе. (Первая строка программы слишком длинна, чтобы поместиться на напечатанной строке в этой книге, поэтому мы разбили ее после **% cmain ()**, вставив обратную косую черту и разрыв строки.)

```
char*f="char*f=%c%s%c,q='%c',n='%cn',b='%c%c';%cmain()\
{printf(f,q,f,q,q,b,b,b,n,n);}%c",q='"',n='\n',b='\\';
main(){printf(f,q,f,q,q,b,b,b,n,n);}
```

Следующая однострочная программа - это почти **quine**. (Мы разбили его после **"; main ()**, вставив обратную **косую черту** и разрыв **строки**, поскольку он не помещается на печатной строке.) Мы оставляем читателю выяснить, почему это не совсем **quine (**самовоспроизводящаяся программа**)**.

```
char*f="char*f=%c%s%c;main(){printf(f,34,f,34);}";main()\
{printf(f,34,f,34);}
```