scanf, fscanf, sscanf, scanf_s, fscanf_s, sscanf_s

Определено в заголовке <stdio.h>

`int scanf( const char *format, ... );`	(1)	(до C99)
`int scanf( const char *restrict format, ... );`		(начиная с C99)
`int fscanf( FILE *stream, const char *format, ...);`	(2)	(до C99)
`int fscanf( FILE *restrict stream, const char *restrict format, ...);`		(начиная с C99)
`int sscanf( const char *buffer, const char *format, ...);`	(3)	(до C99)
`int sscanf( const char *ограничить буфер, const char *ограничить формат, ...);`		(начиная с C99)
`int scanf_s(const char *restrict format, ...);`	(4)	(начиная с C11)
`int fscanf_s(FILE *restrict stream, const char *restrict format, ...);`	(5)	(начиная с C11)
`int sscanf_s(const char *ограничить буфер, const char *ограничить формат, ...);`	(6)	(начиная с C11)

Считывает данные из различных источников, интерпретирует их в соответствии `format`и сохраняет результаты в заданных местах.

1) считывает данные из `stdin`
2) считывает данные из потока файлов `stream`
3) считывает данные из символьной строки с нулевым окончанием`buffer`. Достижение конца строки эквивалентно достижению условия конца файла для `fscanf`

4-6) То же, что и (1-3), за исключением того, что `%c` , `%s` и `%[` спецификаторы преобразования ожидают два аргумента (обычный указатель и значение типа `rsize_t`, указывающее размер принимающего массива, который может быть равен 1 при чтении с `%c` в один символ) и за исключениемчто следующие ошибки обнаруживаются во время выполнения и вызывают установленную в данный момент функцию обработчика ограничений:

- любой из аргументов типа указателя является нулевым указателем
- `format`, `stream`, или `buffer`является нулевым указателем
- количество символов, которые будут записаны с помощью `%c`, `%s` или `%[` , плюс завершающий нулевой символ, превысит второй аргумент (`rsize_t`), предоставленный для каждого из этих спецификаторов преобразования
- опционально любая другая обнаруживаемая ошибка, например неизвестный спецификатор преобразования

Как и во всех функциях с проверкой границ, `scanf_s`, `fscanf_s`, и `sscanf_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 входного файла для чтения
буфер	–	указатель на символьную строку с нулевым окончанием для чтения
формат	–	указатель на символьную строку с нулевым окончанием, указывающую, как читать входные данные
`...`	–	получение аргументов.

Строка **формата** состоит из

- небельные многобайтовые символы, за исключением`%`: каждый такой символ в строке формата потребляет ровно один идентичный символ из входного потока или вызывает сбой функции, если следующий символ в потоке не сравнивается равным.
- символы пробела: любой отдельный символ пробела в строке формата потребляет все доступные последовательные символы пробела из входных данных (определяется как вызов `isspace` в цикле). Обратите внимание, что нет никакой разницы между `"\ n"` , `" "` , `"\ t\ t"` или другими пробелами в строке формата.
- спецификации преобразования. Каждая спецификация преобразования имеет следующий формат:
  - вводный `%`символ
  - (необязательно) присваивание -подавляющий символ `*`. Если этот параметр присутствует, функция не назначает результат преобразования ни одному получающему аргументу.
  - (необязательно) целое число (больше нуля), задающее *максимальную ширину поля*, то есть максимальное количество символов, которые функция может использовать при выполнении преобразования, указанного в

текущей спецификации преобразования. Обратите внимание, что %s и %[могут привести к переполнению буфера, если ширина не указана.

- (необязательно) *модификатор длины*, определяющий размер принимающего аргумента, то есть фактический тип назначения. Это влияет на точность преобразования и правила переполнения. Тип назначения по умолчанию отличается для каждого типа преобразования (см. Таблицу ниже).
- спецификатор формата преобразования

Доступны следующие спецификаторы формата:

Спецификатор преобразования	Объяснение	Тип аргумента								
	Модификатор длины →	hh (C99)	h	(нет)	l	ll (C99)	j (C99)	z (C99)	t (C99)	L
%	соответствует буквальному %	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A
c	соответствует **символу** или последовательности **символов** Если используется спецификатор ширины, он точно соответствует символам ширины (аргумент должен быть указателем на массив с достаточным пространством). В отличие от %s и %[, не добавляет нулевой символ в массив.	N/A	N/A	char*	wchar_t*	N/A	N/A	N/A	N/A	N/A
s	соответствует последовательности символов без пробелов (**строка**) Если используется спецификатор ширины, соответствует *ширине* или до первого пробельного символа, в зависимости от того, что появится первым. Всегда хранит нулевой символ в дополнение к совпадающим символам (поэтому массив аргументов должен иметь место как минимум для *символов width + 1*)									
[*установить*]	соответствует непустой последовательности символов из *набора* символов. Если первый символ набора равен^, то все символы, не входящие в набор, совпадают. Если набор начинается с]или^], то]символ также включается в набор. Определяется реализацией, может ли символ -в начальной позиции в наборе развертки указывать диапазон, как в [0-9]. Если используется спецификатор ширины, соответствует только *ширине*. Всегда хранит нулевой символ в дополнение к совпадающим символам (поэтому массив аргументов должен иметь место как минимум для *символов width + 1*)									
d	соответствует **десятичному целому**числу. Формат числа такой же, как и ожидалось strtol() со значением 10 для baseаргумента	подписанный символ* или неподписанный символ*	короткое со знаком* или короткое без знака*	signed int* или unsigned int*	signed long* или unsigned long*	signed long long* или unsigned long long*	intmax_t* или uintmax_t*	size_t*	ptrdiff_t*	N/A
i	соответствует целому **числу**. Формат числа такой же, как и ожидалось strtol() со значением 0 для baseаргумента (база определяется первыми проанализированными символами)									
u	соответствует **десятичному целому**числу без знака. Формат числа такой же, как и ожидалось strtoul() со значением 10 для baseаргумента.									
o	соответствует **восьмеричному целому числу без знака.** Формат числа такой же, как и ожидалось strtoul() со значением 8 для baseаргумента									
x, X	соответствует **шестнадцатеричному целому**числу без знака. Формат числа такой же, как и ожидалось strtoul() со значением 16 для baseаргумента									
n	возвращает **количество прочитанных символов.** Входные данные не используются. Не увеличивает количество назначений. Если в спецификаторе определен оператор подавления присваивания, поведение не определено									
a, A(C99) e, E f,F g, G	соответствует **числу с плавающей запятой.** Формат числа такой же, как и ожидалось strtof()	N/A	N/A	поплавок*	двойной*	N/A	N/A	N/A	N/A	длинный двойной*
p	соответствует определенной реализацией последовательности символов, определяющей **указатель.** printf семейство функций должно выдавать одну и ту же последовательность с использованием **%p**спецификатора формата	N/A	N/A	пустота**	N/A	N/A	N/A	N/A	N/A	N/A

Для каждого спецификатора преобразования, отличного от n, самая длинная последовательность входных символов, которая не превышает заданной ширины поля и которая либо является именно тем, что ожидает спецификатор преобразования, либо является префиксом последовательности, которую он ожидает, – это то, что потребляется из потока. Первый символ, если таковой имеется, после этой используемой последовательности остается непрочитанным. Если

потребляемая последовательность имеет нулевую длину или если потребляемая последовательность не может быть преобразована, как указано выше, сбой сопоставления происходит, если только конец файла, ошибка кодирования или ошибка чтения не предотвратили ввод из потока, и в этом случае это сбой ввода.

Все спецификаторы преобразования, отличные от [, c, и nпотребляют и отбрасывают все ведущие символы пробела (определяемые как бы вызовом isspace) перед попыткой анализа входных данных. Эти потребляемые символы не учитываются при указанной максимальной ширине поля.

Спецификаторы преобразования lc, ls, и l[выполняют многобайтовое преобразование символов, как если бы вызывали mbrtowc() с объектом mbstate_t, инициализированным до нуля перед преобразованием первого символа.

Спецификаторы преобразования s и [всегда сохраняют нулевой терминатор в дополнение к соответствующим символам. Размер целевого массива должен быть как минимум на единицу больше указанной ширины поля. Использование `%s` или `%[` без указания размера целевого массива так же небезопасно, как и gets

Правильные спецификации преобразования для целочисленных типов с фиксированной шириной (int8_t, etc) Определены в заголовке <inttypes.h>(хотя SCNdMAX, SCNuMAX ит. Д. Являются синонимами %jd, %ju, etc) .

После действия каждого спецификатора преобразования существует точка последовательности; это позволяет хранить несколько полей в одной переменной "приемник".

При анализе неполного значения с плавающей запятой, которое заканчивается показателем без цифр, например при анализе `"100er"` со спецификатором преобразования `%f`, последовательность `"100e"` (самый длинный префикс возможно допустимого числа с плавающей запятой) потребляется, что приводит к ошибке сопоставления (потребляемая последовательностьне может быть преобразовано в число с плавающей запятой), при `этом остается "r"`. Некоторые существующие реализации не следуют этому правилу и откатываются, чтобы потреблять только `"100"`, оставляя `"er"`, например ошибка glibc 1765 (https://sourceware.org/bugzilla/show_bug.cgi?id=1765)

Если спецификация преобразования недопустима, поведение не определено.

1-3) Количество успешно назначенных аргументов приема (которое может быть равно нулю, если сбой согласования произошел до назначения первого аргумента приема), или EOF, если сбой ввода произошел до назначения первого аргумента приема.

4-6) То же, что и (1-3), за исключением того, что EOF также возвращается, если есть нарушение ограничений времени выполнения.

Не гарантируется. Примечательно, что некоторые реализации sscanfявляются *O(N)*, где `N = std::strlen(buffer)` [1] (https://sourceware.org/bugzilla/show_bug.cgi?id=17577) . Для выполнения синтаксического анализа строк см.std::from_chars

Поскольку большинство спецификаторов преобразования сначала потребляют все последовательные пробелы, код, такой как

```
scanf("%d", &a);
scanf("%d", &b);
```

считывает два целых числа, введенные в разных строках (второй %d будет использовать новую строку, оставшуюся после первой) или в одной строке, разделенные пробелами или табуляциями (второй %d будет использовать пробелы или табуляции).

Спецификаторы преобразования, которые не используют начальные пробелы, такие как %c, могут быть сделаны с помощью пробельного символа в строке формата:

```
scanf("%d", &a);
scanf("%c", &c); // потребляйте все последовательные пробелы после %d, затем считывайте симв
```

Запустите этот код

```
#define __STDC_WANT_LIB_EXT1__ 1
#include <stdio.h>
#include <stddef.h>
#include <locale.h>

int main(void)
```

```
{
    int i, j;
    float x, y;
    char str1[10], str2[4];
    wchar_t warr[2];
    setlocale(LC_ALL, "en_US.utf8");

    char input[] = "25 54.32E-1 Thompson 56789 0123 56ß水";
    /* разбор следующим образом:
 %d: целое число
 %f: значение с плавающей запятой
 %9s: строка не более 9 символов без пробелов
 %2d: двузначное целое число (цифры 5 и 6)
 %f: значение с плавающей запятой (цифры 7, 8, 9)
 %*d: целое число, которое нигде не хранится
 ' ': все последовательные пробелы
 %3[0-9]: строка не более 3 десятичных цифр (цифры 5 и 6)
 %2lc: два широких символа, используя многобайтовое преобразование */
    int ret = sscanf(вход, "%d%f%9s%2d%f%*d%3[0-9]%2lc",
&i, &x, str1, &j, &y, str2,warr);

    printf("Преобразованные поля %d:\ni = %d\nx = %f\nstr1 = %s\n"
           "j = %d\ny = %f\nstr2 = %s\n"
           "warr[0] = U+%x warr[1] = U+%x\n",
ret, i, x, str1, j, y, str2, warr[0], warr[1]);

#ifdef __STDC_LIB_EXT1__
    int n = sscanf_s(input, "%d%f%s", &i, &x, str1, (rsize_t)sizeof str1);
    // записывает 25 в i, 5.432 в x, 9 байт "thompson\0" в str1 и 3 в n.
#endif
}
```

Вывод:

```
Преобразовано 7 полей:
i = 25
x = 5.432000
str1 = Томпсон
j = 56
y = 789.000000
str2 = 56
warr[0] = U+df warr[1] = U+6c34
```

- Стандарт C11 (ISO/IEC 9899:2011):
  - 7.21.6.2 Функция fscanf (p: 317-324)
  - 7.21.6.4 Функция scanf (p: 325)
  - 7.21.6.7 Функция sscanf (p: 326)
  - K.3.5.3.2 Функция fscanf_s (p: 592-593)
  - K.3.5.3.4 Функция scanf_s (p: 594)
  - K.3.5.3.7 Функция sscanf_s (p: 596)
- Стандарт C99 (ISO/IEC 9899:1999):
  - 7.19.6.2 Функция fscanf (p: 282-289)
  - 7.19.6.4 Функция scanf (p: 290)
  - 7.19.6.7 Функция sscanf (p: 291)
- Стандарт C89/C90 (ISO/IEC 9899:1990):
  - 4.9.6.2 Функция fscanf
  - 4.9.6.4 Функция scanf
  - 4.9.6.6 Функция sscanf

См. Также

vscanf (C99) **vfscanf** (C99) **vsscanf** (C99) **vscanf_s** (C11) **vfscanf_s** (C11) **vsscanf_s** (C11)	считывает форматированные входные данные из stdin, потока файлов или буфера с помощью списка переменных аргументов (функции)
fgets	получает символьную строку из потока файлов (функция)
printf **fprintf** **sprintf** **snprintf** (C99) **printf_s** (C11) **fprintf_s** (C11) **sprintf_s** (C11) **snprintf_s** (C11)	печатает форматированный вывод в stdout, поток файла или буфер (функция)
C++ документация для **scanf**, **fscanf**, **sscanf**	