



# Программирование на C и C++

Онлайн справочник программиста на C и C++

[Главная](#)[Язык C](#)[Язык C++](#)[Функции](#)[Термины](#)[Блоги](#)[Главная](#)

## Случайные статьи

Двоичный ввод/вывод

enable, \_enable

floor, floorl

registerbgidriver, registerbgifont

Корректное использование аргументов по умолчанию

Инициализация переменных

Вариации цикла for

\_heapmin

strdup, \_fstdup

Шаблоны, исключения и RTTI

Глобальные переменные

absread, abswrite

getimage

perror

atexit

ftell

unlock

Использование defined

Файлы

Передача ссылок на объекты

Битовые операторы

vprintf, vfprintf, vsprintf

Использование множества сканирования

unlink

Обзор языка C

Использование feof()

Оператор ?

clreol, clrscr

rewind

DO/WHILE



## scanf



### int scanf(const char \*format, arg-list)

Прототип:

stdio.h

Описание:

Функция `scanf()` является процедурой ввода общего назначения, считывающей данные из потока `stdin`. Она может считывать данные всех базовых типов и автоматически конвертировать их в нужный внутренний формат. Если бы `printf()` выполняла ввод, а не вывод, ее можно было бы назвать аналогом `scanf()`.

Управляющая строка, на которую указывает `format`, состоит из символов трех типов:

- Спецификаторы формата
- Специальные символы
- Прочие символы (не специальные)

Спецификаторы формата следуют за символом процент и сообщают `scanf()`, данные какого типа будут считаны следующими. Коды спецификаторов приведены в таблице.

Таблица: Коды форматов для `scanf()`

Код	Значение
%c	Считать один символ
%d	Считать десятичное число целого типа
%i	Считать десятичное число целого типа
%e	Считать число с плавающей запятой
%f	Считать число с плавающей запятой
%g	Считать число с плавающей запятой
%o	Считать восьмеричное число
%s	Считать строку
%x	Считать шестнадцатеричное число
%p	Считать указатель
%n	Принимает целое значение, равное количеству считанных до текущего момента символов
%u	Считывает беззнаковое целое
%[]	Просматривает набор символов
%%	Считывает символ %

Например, `%s` считывает строку, а `%d` считывает переменную целого типа.

Строка формата считывается слева направо, при этом устанавливается соответствие между кодами формата и аргументами из списка аргументов.

Специальные символы в управляющей строке заставляют `scanf()` пропускать один или больше специальных символов во входном потоке. Специальные символы – это пробел, табуляция или новая строка. Один специальный символ в управляющей строке заставляет `scanf()` считывать, не запоминая, любое количество (включая нуль) идущих подряд специальных символов из входного потока, пока не встретится символ, не являющийся специальным символом.

Наличие обычного символа заставляет `scanf()` считать и отбросить соответствующий символ. Например, `"%d,%d"` заставляет `scanf()` считать целое число, считать и отбросить запятую и затем считать еще одно целое число. Если указанный символ не обнаружен во входном потоке, `scanf()` останавливается.

Все переменные, используемые для приема значений с помощью функции `scanf()`, должны отыскиваться по их адресам. Это значит, что все аргументы функции должны быть указателями на переменные. Таким образом, C создает возможность передачи по ссылке, и это позволяет функции изменять содержимое аргумента.

Например, чтобы считать целое число и присвоить его значение переменной `count`, необходимо воспользоваться следующим обращением к `scanf()`:

```
scanf("%d", &count);
```

Строки считываются в массивы символов, и имя массива, без всякого указателя, является адресом первого элемента массива. Поэтому, чтобы считать строку в массив символов `address`, можно использовать команду

```
scanf("%s", address);
```

В этом случае имя `address` уже является указателем и не нуждается в префиксе `&`.

Элементы вводимых данных должны разделяться пробелами, знаками табуляции или новой строки. Знаки пунктуации, такие как запятая, точка с запятой и т.п., не считаются разделителями. Это значит, что для оператора

```
scanf("%d%d", &r, &c);
```

последовательность `10 20` будет воспринята, а последовательность `10,20` – нет. Спецификаторы формата `scanf()` расположены в том же порядке, что и переменные в списке аргументов, которым присваиваются значения принимаемых переменных.

Знак `*`, помещенный после `%` и перед спецификатором формата, считывает данные указанного типа, но подавляет их присваивание. Таким образом, код

```
scanf ("%d*c%d", &x, &y);
```

при вводе последовательности `10/20` присваивает значение `10` переменной `x`, отбрасывает символ `/` и присваивает значение `20` переменной `y`.

Командами форматирования может задаваться модификатор максимальной ширины поля. Он представляет собой целое число, которое помещается между знаком `%` и спецификатором формата. Он ограничивает количество считываемых символов для любого поля. Например, если необходимо считать не больше, чем `20` символов в массив `address`, следует написать

```
scanf ("%20s", address);
```

Если входной поток содержал больше `20` символов, то при последующем вызове функция ввода начнет ввод с того места, где был остановлен ввод при текущем обращении. Ввод поля может быть прерван и до достижения максимальной длины поля, если встретится пробел. В этом случае `scanf()` переходит к следующему полю.

Хотя пробелы, символы табуляции и новых строк используются как разделители полей, они считаются как любой другой символ при вводе одиночного символа. Например, при входном потоке `x y` функция

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

поместит символ `x` в переменную `a`, пробел – в переменную `b` и `y` – в переменную `c`.

Надо быть внимательным: любые другие символы в управляющей строке – включая пробелы, символы табуляции и новых строк – используются для указания и отбрасывания символов из входного потока.

Например, при входном потоке `10t20` функция

```
scanf ("%st%s", &x, &y);
```

поместит `10` в `x`, а `20` в `y`. Символ `t` будет отброшен, поскольку в управляющей строке имеется `t`.

Еще одна возможность функции `scanf()` называется множеством сканирования. С помощью множества сканирования определяются символы, которые будут считываться функцией `scanf()` и присваиваться элементам соответствующего массива символов. Чтобы задать множество сканирования, надо символы, ввод которых допустим, поместить в квадратные скобки. Перед первой квадратной скобкой ставится знак процента. Например, следующий перечень множества сканирования задает считывание функцией `scanf()` только символов `A`, `B` и `C`:

```
%[ABC]
```

Соответствующий перечню множества сканирования аргумент должен быть указателем на массив символов. При использовании множества сканирования функция `scanf()` считывает символы и помещает их в указанный массив до тех пор, пока не встретится символ, не входящий в множество сканирования (то есть считываются только символы, входящие в множество сканирования).

Массив, возвращенный `scanf()`, будет содержать строку с нулевым символом в конце. Перечень считываемых символов можно задать и в инверсной форме. Для этого в качестве первого символа надо поместить `^`. Тогда `scanf()` будет принимать любой символ, не входящий в множество сканирования.

С помощью кавычек можно задать диапазон воспринимаемых символов. Например, следующее выражение дает указание `scanf()` принимать буквы от «`A`» до «`Z`»:

```
%[A-Z]
```

Множество сканирования различает прописные и строчные буквы. Если необходимо, чтобы `scanf()` принимала те и другие, необходимо перечислить их в множестве сканирования отдельно.

Функция `scanf()` возвращает число, равное количеству полей, значения которых были действительно присвоены переменным. В это количество не входят поля, которые были считаны, но их значения не были ничему присвоены вследствие использования модификатора `*` для подавления присваивания. Если до присвоения значения первого поля произошла ошибка, возвращается `EOF`.

При использовании Borland C++ в 16-разрядной среде можно изменить модель памяти, используемую по умолчанию для компилирования программы, путем явного указания размера каждого указателя, используемого при вызове `scanf()`. Ближний указатель задается модификатором `N`, а дальний – модификатором `F`. (Нельзя использовать модификатор `N`, если программа скомпилирована для модели памяти `huge`.)

#### Пример:

Действия функции `scanf()` в следующих примерах поясняются комментариями:

```
char str[80];
int i;
/* чтение строки и целого */
scanf ("%s%d", str, &i);
/* чтение до 79 символов в str */
```

```
scanf ("%79s", str);
/* пропуск целого между двумя строками */
scanf (" %s%d%s", str, &i, str);
```

Назначение платежа

**На хостинг**

Сумма

100 ₽

**Пожертвовать**

## Функции ввода/вывода

vprintf, vfprintf, vsprintf	access	chmod
chsize	clearerr	close, _rtl_close
creat, _rtl_creat, creatnew, creattemp	dup, dup2	eof
fclose, fcloseall	fdopen	feof
ferror	fflush	fgetc, fgetchar
fgetpos	fgets	filelength
fileno	flushall	fopen
fprintf	fputc	fputchar
fputs	fread	freopen

1 2 3

следующая › последняя »


АМА

СОВКОМБАНК

О «СОВКОМБАНК»  
ЛИЦЕНЗИЯ ЦБ РФ №963

Выдаем кредиты  
каждый день!

Денежный кредит под 19,9%



ЗАПОЛНИТЬ ЗАЯВКУ

[Главная](#)[Язык С](#)[Язык С++](#)[Функции](#)[Термины](#)[Блоги](#)

4 258

2 546

889



Рамблер

ТОП100

