* [Вывод в языке C. Функция printf.](https://foxford.ru/lessons/32262/conspects/1)
* [Ввод в языке C. Функция scanf.](https://foxford.ru/lessons/32262/conspects/2)
* [Вывод в языке C++ через cout](https://foxford.ru/lessons/32262/conspects/3)
* [Ввод в языке C++ через cin](https://foxford.ru/lessons/32262/conspects/4)
* [Простое использование файлового ввода-вывода при помощи функции freopen](https://foxford.ru/lessons/32262/conspects/5)
* [Ускорение ввода-вывода в C++](https://foxford.ru/lessons/32262/conspects/6)

**Вывод в языке C. Функция printf.**

**Функция printf**

Для вывода данных в языке C используется функция

**printf**

. Для использования этой функции необходимо подключить заголовочный файл

**stdio.h**

:

**#include<stdio.h>**

Если вы пишете на C++, то лучше подключать файл

**cstdio**

:

**#include<cstdio>**

Функция printf получает один или более аргументов. Первый аргумент - строка, которую нужно вывести. Например, если нужно просто вывести какой-то текст, то можно написать так:

**printf("Hello, world\n");**

Символ '\n' в конце вывода означает символ перехода на новую строку.

Если нужно вывести значения какой-то переменной, то в форматной строке вместо этой переменной указывается специальное выражение, начинающееся с символа "%". Например, для вывода целого числа типа int указывается строка "%d". Сами переменные указываются после форматной строки, следующими параметрами, передаваемыми функции

**printf**

.

Например, чтобы напечатать значения переменных a и b типа

**int**

 через пробел можно использовать следующую функцию:

**printf("%d %d", a, b);**

**Выводимые типы**

Перечислим разные типы данных, которые можно выводить при помощи функции

**printf**

 и соответствующие им форматные строки.

|  |  |
| --- | --- |
| **%d** или %**i** | Выводится целое число (значение типа int) в десятичной системе счисления. Для вывода значений типа short int используется %hd, long int - %ld, long long int - %lld. |
| **%u** | Выводится беззнаковое целое число (значение типа unsigned int) в десятичной системе счисления. Для вывода значений типа unsigned short используется %hu, unsigned long - %lu, unsigned long long  - %llu. |
| **%e** | Выводится значение типа double в экспоненциальной форме, например, число 123.456 будет выведено в виде 1.234560e+002. Для вывода значения типа double также используется %le, для вывода long double - %Le (в Windows может не работать). |
| **%f** | Выводится значение типа double с фиксированной точкой. Например, число 1e20 будет выведено, как 10000000000.000000. Для вывода значения типа double также используется %lf, для вывода long double - %Lf (в Windows может не работать). |
| **%g** | Значение типа double выводится либо с фиксированной точкой, либо с плавающей точкой, в зависимости от величины числа. Аналогично используется %lg для double и %Lg для long double. |
| **%c** | Выводится один символ (значение типа char). |
| **s** | выводится содержимое C-строки. В качестве параметра передается указатель char \* или массив символов char[]. |
| **%%** | Вывести один символ %, никакого значения переменной при этом не передается. |

 Значения целых чисел также можно выводить в восьмеричной или шестнадцатеричной системах счисления.

Обратите внимание, переменную типа float вывести при помощи функции printf нельзя (можно привести значение типа float к типу double и вывести его).

В операционной системе Windows не рекомендуется использовать тип long double.

**Модификаторы длины**

Эти модификаторы пишутся между знаком % и символом, обозначающим тип. Например, **%lld**.Эти модификаторы используются для того, чтобы вывести значения типов

**short int,**

**long int**

,

**long double**

 и т.д. и уже упоминались раньше.

|  |  |
| --- | --- |
| **hh** | Используется для значения типа char вместе с символами d, i, u для вывода в виде десятичного числа, соотвветствующего его ASCII-коду. |
| **h** | Используется для значения типа short int или unsigned short int вместе с модификаторами d, i, u. |
| **l** | Используется для значения типа long int или unsigned long int вместе с модификаторами d, i, u.  Используется для значения типа double вместе с модификаторами e, f, g. |
| **ll** | Используется для значения типа long long int или unsigned long long int вместе с модификаторами d, i, u. |
| **L** | Используется для значения типа long double вместе с модификаторами e, f, g. Не следует использовать в Windows. |

**Ширина поля вывода**

После знака % можно задать ширину поля для вывода в виде целого числа (не начинающегося с нуля). В этом случае число вывод этого числа будет занимать указанное число символов. Если в числе меньше цифр, чем заданная ширина поля вывода, то перед числом (слева) выводятся пробелы. Если же в числе больше цифр, чем ширина поля, то выводятся все цифры (то есть выводится больше цифр, чем вся ширина поля, поле расширяется, чтобы вывести все символы).

Если перед шириной поля поставить цифру 0, например, "%010d", то число будет дополняться слева не пробелами, а нулями. Это удобно, например, для вывода времени в формате hh:mm, что можно сделать, например, так:

**printf("%02d:%02d", h, m);**

Если перед шириной поля поставить знак "-", то число будет дополняться пробелами не слева, а справа (сначала выводится число, потом - дополнительные пробелы до заполнения всего поля).

Также вместо ширины поля вывода можно указать символ "\*" (звездочка), тогда значение ширины поля вывода нужно передать в качестве дополнительного параметра. Например, запись:

**printf("%\*d", n, a);**

означает вывести значение переменной a используя поле шириной n символов.

**Точность вывода действительных чисел**

По умолчанию действительные числа выводятся с 6 знаками после точки, при этом числа в экспоненциальной форме выводятся в виде одной цифры, точки и 6 цифр после точки (и показателя степени).

Для изменения точности вывода действительных чисел в форматной строке указывается символ точки ('.') и после нее число - количество выводимых знаков после точки. Например, так:

**printf("%.10lf", x);**

**Вывод целых чисел со знаком +/-**

Если при выводе целого числа в форматной строке задать символ "+", то положительные числа будут выводиться с обязательным знаком "+" перед ними.

Большинство перечисленных выше параметров комбинируются друг с другом. Например, такая строка:

**printf("%+020lld", n);**

означает вывод числа типа long long, со знаком "+" или "-" перед числом, с добавлением нулей таким образом, что общее число выводимых символов будет не меньше 20.

**Ввод в языке C. Функция scanf.**

**Функция scanf**

Для считывания данных в языке C используется функция scanf. Ее использование похоже на функцию prinf - сначала задается форматная строка, потом передаются переменные, в которые необходимо записать результат. Например, для считывания двух целых чисел функция вызывается так:

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

Основное отличие функции scanf в том, что при считывании чисел (или значений типа char) ей необходимо передавать **адреса** переменных (в языке C все параметры передаются по значению, поэтому чтобы функция scanf могла модифицировать переменную, необходимо передать в функцию адрес этой переменной). Поэтому перед названиями переменных мы пишем знак амперсанда ("&").

В функции scanf могут быть явно записаны какие-то символы, кроме форматных строк. Например, вызов

**scanf("%d:%d")**

можно использовать для считывания времени, заданного в виде hh:mm - функция считает число, затем символ двоеточия, затем опять число.

Если в форматной строке встречается пробел, то он означает, что будет считано любое число пробельных символов, возможно даже нулевое.

**Особенности считывания чисел**

Функция

**scanf**

 корректно считывает целые числа, если они  начинаются с символа 0, или со знака "+". То есть числа "+123" или "0123" будут корректно считаны по форматной строке "%d", никаких дополнительных параметров задавать не нужно.

Также при считывании чисел игнорируются пробелы перед числом. Это означает, что вызов

**scanf("%d:%d", &a, &b)**

сможет корректно считать время, заданное в формате hh:mm при наличии пробела после двоеточия (такая запись успешно считает строки "12:34", "01:02",  "01:␣23" или "␣01:␣23", поскольку дается указание считать число, затем сразу же двоеточие, затем - число, перед которым могут быть пробелы). Но такая запись не считает выражение, например, вида "01␣:␣23", поскольку после первого числа сразу должно идти двоеточие.

Чтобы считать записать вида "01␣:␣23" можно использовать форматную строку "%d :%d", причем пробел в форматной строке может означать и отсутствие пробелов.

**Возможные форматные символы**

Возможные форматные символы функции

**scanf**

 для разных типов данных в целом соответствуют форматным символам для функции

**printf**

, но имеют меньше различных модификаторов.

|  |  |
| --- | --- |
| Форматная строка | Соответствующий ей тип |
| **%hhd** | Считать число (десятичное) и записать его в переменную типа char (для unsigned char нужно использовать %hhu) |
| **%hd** | short int (для unsigned short int нужно использовать %hu) |
| **%d** | int (для unsigned int нужно использовать %u) |
| **%ld** | long int (для unsigned long int нужно использовать %lu) |
| **%lld** | long long int (для unsigned long long int нужно использовать %llu) |
| **%f** | float |
| **%lf** | double |
| **%Lf** | long double |
| **%с** | char. Считывается один символ, он может быть пробелом или символом конца строки. |
| **%s** | Считывается последовательность непробельных символов (строка), записывается в C-строку (типа char \* или char[]) |

**Особенность считывание символов**

Считывание одного символа "%c" считывает из потока ввода следующий символ, он может быть в том числе и пробельным символом, а также символом конца строки. Но если в форматной строке перед "%c" поставить пробел, то поскольку пробел в форматной строке обозначает последовательность пробельных символов любой длины, то в этом случае будет считан следующий непробельный символ.

**Особенность считывания строк**

При считывании строки результат записывается в С-строку, которая представляет собой массив символов (или указатель типа

**char \***

 с выделенной памятью). Поскольку строка в языке C является адресом (указателем) в памяти, где хранится начало строки символов, то передавать в функцию

**scanf**

 нужно имя переменной без указания амперсанда.

Например,

**char s[101];**

**scanf("%s", s);**

Функция

**scanf**

 читает строку до первого пробельного символа, также она пропускает предшествующие пробелы, то есть она записывает следующее слово целиком составленное из непробельных символов.

Иногда бывает полезно считать всю строку целиком вместе с пробелами до конца строки. Для этого используется функция

**gets**

. Например:

**gets(s);**

Функция

**gets**

 считается опасной и не рекомендуется для использования, так как она не контролирует количество считанных символов и не учитывает длину передаваемой строки, что может привести к записи данных за пределами строки.

Вместо нее рекомендуется использование функции

**fgets**

, у которой три параметра - строка для записи результата, размер строки и файловый поток, из которого читаются данные. Например:

**fgets(s, 101, stdin);**

В данном случае мы использовали

**stdin**

 для чтения со стандартного ввода.

Не следует забывать, что в языке C в конец строки добавляется нулевой символ для обозначения конца строки. То есть если необходимо считать строку, в которой может быть 4 символа, то для нее нужно создать массив

**char[5]**

, и функции

**fgets**

 нужно передавать число, не меньшее 5.

**Возвращаемое значение**

Функция

**scanf**

 возвращает значение, равное числу успешно считанных и записанных в переданные параметры значений, что можно использовать для анализа входных данных.

Например, пусть вызвали

**scanf("%d:%d", &a, &b)**

.

Тогда при вводе строки "12:34" функция

**scanf**

 считает два числа, запишет их в переменные a и b и вернет значение 2. А при вводе "12 34" будет считано только одно число, поскольку после него должно идти двоеточие, то второе число считано не будет и функция

**scanf**

 вернет значение 1.

**Вывод в языке C++ через cout**

**Использование объекта cout**

Для стандартного вывода (вывода на экран) в языке C++ используется объект

**cout**

 и операторы "<<". Например, для вывода значения переменной a можно написать:

**cout << a;**

Для вывода значений переменных a и b через пробел так:

**cout << a << " " << b;**

В этом примере мы даем команду сначала вывести значение переменной a, затем строку из одного пробела, затем вывести значение переменной b.

Для перехода на следующую строку нужно либо вывести специальный символ '\n', либо использовать объект

**endl**

. Приведем примеры использования обоих способов для вывода значений переменных в отдельных строках:

**cout << a << "\n" << b << endl;**

Через

**cout**

 можно выводить все стандартные типы данных - целые и действительные числа, символы,  строки (как C-строки, так и объекты

**string**

), заботиться о задании типов данных при этом не нужно, компилятор автоматически распознает необходимые типы.

Для использования стандартного ввода-вывода в языке C++ необходимо подключить заголовочный файл

**iostream**

:

**#include<iostream>**

**Задание ширины поля вывода**

Есть два способа управления выводом - через вызовы метода объекта cout (например,

**cout.width(20)**

) или при помощи "манипуляторов" (

**cout << setw(10)**

).

Второй способ удобней, поэтому рассмотрим его. Для использования манипуляторов необходимо подключить заголовочный файл

**iomanip**

:

**#include<iomanip>**

Для задания ширины поля вывода одного значения используется манипулятор setw с одним параметром - шириной поля вывода. Например:

**cout << setw(10) << a;**

выведет значение переменной a в поле шириной 10 символов. Аналогично правилам вывода в языке C, по умолчанию осуществляется выравнивание выводимого значения по правому краю (перед выводимым значением добавляются пробелы), если же число содержит больше символов, чем ширина поля, то будут выведены все символы.

Манипулятор

**setw**

 влияет только на значение ширины поля вывода для следующего выводимого значения, если нужно задать значение ширины поля вывода для двух чисел, то это нужно делать перед выводом каждого значения.

Манипулятор

**setfill**

 с одним параметром типа char позволяет задать символ-"заполнитель", который будет использоваться вместо пробела для заполнения поля. Например, вывести время в формате HH:MM (ровно по две цифры для вывода часов и минут) можно следующим образом:

**cout << setfill('0') << setw(2) << h << ":" << setw(2) << m << endl;**

Заметим, что манипулятор

**setfill**

 действует постоянно (до следующего вызова

**setfill**

), не нужно устанавливать его перед выводом каждого значения.

Также как и в языке C можно задавать тип выравнивания внутри поля. Выравнивание бывает левым, правым и внутренним. Внутреннее выравнивание означает, что сначала выводится знак числа (если число отрицательное), затем - пробелы, затем - цифры числа. Для задания выравнивания используются манипуляторы

**left**

,

**right**

,

**internal**

.

Примеры использования различных типов выравнивания:

|  |  |
| --- | --- |
| Пример кода | Вывод программы |
| **cout << left << setw(7) << -123 << "\*" << endl;** | **-123 \*** |
| **cout << right << setw(7) << -123 << "\*" << endl;** | **-123\*** |
| **cout << internal << setw(7) << -123 << "\*" << endl;** | **- 123\*** |

**Вывод действительных чисел**

Есть два способа вывода действительных чисел - с фиксированной точкой (соответствует форматной строке "%f" функции

**printf**

) и с плавающий точкой (соответствует "%e"). Для переключения режима вывода действительных чисел в формат с фиксированной точкой используется манипулятор fixed, для вывода с плавающей точкой - манипулятор scientific.

По умолчанию числа выводятся с точностью в 6 знаков после точки, это значение можно изменить при помощи манипулятора

**setprecision**

 с параметром - шириной поля вывода.

Например:

**cout << scientific << setprecision(15) << x << endl;**

Данные манипуляторы после включения действуют постоянно, до следующего задания значения, то есть их не нужно задавать перед выводом каждой переменной.

По умолчанию (если не задать явно манипулятор

**fixed**

 или

**scientific**

) число может выводиться или с плавающей точкой, или с фиксированной, в зависимости от величины числа. Если был установлен один из манипуляторов

**fixed**

 или

**scientific**

, то вернуть "автоматический" выбор формы представления числа можно при помощи манипулятора

**cout << resetiosflags(ios\_base::floatfield);**

**Ввод в языке C++ через cin**

**Использование объекта cin**

Для считывания данных со стандартного ввода используется объект

**cin**

, также определенных в заголовочном файле

**iostream**

. Его использование похоже на использование объекта

**cout**

 для вывода. Например, для считывания двух переменных a и b нужно дать команду:

**cin >> a >> b;**

Также как и в случае с

**cout**

, нет нужды беспокоиться о типах данных - можно считывать целые и действительные числа, символы и строки.

**Особенности считывания символов**

Пусть дано время в формате HH:MM. Считать его можно следующим образом:

**int h, m;**

**char c;**

**cin >> h >> c >> m;**

То есть сначала считывается число, затем - символ, затем - число.

При этом при считывании символа пропускаются пробельные символы (пробелы и концы строк) и считывается следующий непробельный символ. Например, если при запуске приведенного выше кода ввести "12 34", то переменная h станет равна 12, переменная с — '3', а переменная m — 4.

Если установить манипулятор

**noskipws**

:

**cin >> noskipws;**

то при считывании символов пробельные символы не будут пропускаться. Для восстановления режима пропуска пробельных символов используется манипулятор

**skipws**

:

**cin >> skipws;**

**Особенности считывания строк**

Считывание строк при помощи

**cin >> s**

, где s - объект класса string или C-строка приведет к считыванию строки из непробельных символов, при этом пробельные символы пропускаются.Использовать манипулятор

**noskipws**

 при чтении строк не следует.

Строку целиком до конца строки, вместе с пробельными символами можно считать следующим образом:

**getline(cin, s);**

Отметим следующую особенность такого считывания. Пусть входные данные заданы так - сначала число строк n, затем - n строк, возможно, содержащие пробелы. Требуется считать данные строки и сохранить их в массиве строк.

Правильное решение будет таким:

**int n;**

**string s, A[100];**

**cin >> n;**

**getline(cin, s);**

**for (int i = 0; i < n; ++i)**

**getline(cin, A[i]);**

Здесь нужно обратить внимание на "лишний" вызов

**getline**

 после считывания числа n. Дело в том, что

**cin >> n**

 не считывает конец строки из потока, поэтому последующий вызов

**getline**

 вернет пустую строку (но зато считает конец строки), поэтому нужно сделать один "холостой" вызов

**getline**

 перед считыванием всех остальных строк.

**Значение, возвращаемое при считывании**

Для проверки успешности считывания можно преобразовать значение, возвращаемое операцией считывания к типу

**bool**

. Например, следующим образом:

**bool is\_correct;**

**int a;**

**is\_correct = (cin >> a);**

Если значение переменной

**is\_correct**

 будет

**true**

, то считывание было успешно, иначе - нет.

Вместо записи результата в логическую переменную, можно непосредственно использовать его, например, в условной инструкции. Вот так, например, можно проверять, что на вход программе подано два числа через двоеточие:

**int h, m;**

**char c;**

**if (cin >> noskipws >> h >> c >> m)**

**{**

**if (c == ':') …**

**}**

Похожее использование - чтение текста "по словам", с обработкой каждого слова:

**string word;**

**while (cin >> word)**

**{**

**// Обработать слово в переменной word**

**}**

Или считывание текста по строкам:

**string line;**

**while (getline(cin, s))**

**{**

**// Обработать строку в переменной line**

**}**

#### Простое использование файлового ввода-вывода при помощи функции freopen

На олимпиадах часто для файлового ввода-вывода используют функцию freopen, определенную в языке C в заголовочном файле stdio.h. Эта функция позволяет "переоткрыть" файл, то есть связать уже открытый файловый дескриптор с существующим файлом на диске. На олимпиадах это устроено так - файловый дескриптор stdin, связанный со стандартным потоком ввода, переоткрывают на файл, который указан в условии задачи, как файл для хранения входных данных, а файловый дескриптор stdout переоткрывают на файл для выходных данных.

Пример использования:

**freopen("input.txt", "r", stdin);**

**freopen("output.txt", "w", stdout);**

После этого весь стандартный ввод-вывод будет перенаправляться в указанные файлы.

Функцию freopen можно использовать как вместе с выводом в языке C (функции printf и scanf), так и вместе с вводом-выводом в языке C++ (через потоки cin и cout).

#### Ускорение ввода-вывода в C++

Ввод-вывод в языке С++ работает, как правило, медленнее чем в языке C, однако многое может зависеть от компилятора, его версии, настроек, операционной системы.

Но есть приём, который в большинстве случаев позволяет ускорить ввод-вывод в языке C++ в критически важных случаях, например, при считывании более 100.000 чисел:

**sync\_with\_stdio(false);**

**cin.tie(0);**

Также лучше не использовать endl для вывода конца строки, вместо этого лучше выводить "\n", так как вывод endl приводит к сбросу буфера.

Однако в интерактивных задачах как раз требуется сбрасывать буфер после операций вывода, поэтому эти рекомендации в интерактивных задачах не подходят.