

Ravesli [Ravesli](#)


- [Уроки по C++](#)
- [OpenGL](#)
- [SFML](#)
- [Qt5](#)
- [RegExр](#)
- [Ассемблер](#)
- [Купить .PDF](#)

## Урок №18. Базовое форматирование кода

 [Юрий](#) |

- [Уроки C++](#)

|

 Обновл. 2 Сен 2020 |

 43162

[↓](#)  25

**Пробелы** относятся к символам, которые используются в форматировании кода, вместе с символами табуляции и, иногда, разрывом строки. Компилятор, как правило, игнорирует пробелы, но все же есть небольшие исключения.

В следующем примере все строки кода выполняют одно и то же:

```
1 std::cout << "Hello, world!";
2
3 std::cout          <<          "Hello, world!";
4
5 std::cout << "Hello, world!";
6
7 std::cout
8     << "Hello, world!";
```

Даже последний стейтмент с разрывом строки успешно скомпилируется.

Аналогично:

```
1 int add(int x, int y) { return x + y; }
2
3 int add(int x, int y) {
4     return x + y; }
5
6 int add(int x, int y)
7 {     return x + y; }
8
9 int add(int x, int y)
```

```
10 | {  
11 |     return x + y;  
12 | }
```

Исключением, где компилятор учитывает пробелы, является цитируемый текст, например: "Hello, world!".

"Hello, world!"

отличается от

"Hello, world!"

**Разрыв/перевод строки** не допускается в цитируемом тексте:

```
1 | std::cout << "Hello,  
2 |     world!" << std::endl; // Не допускается!
```

Еще одним исключением, где компилятор обращает внимание на пробелы, являются [однострочные комментарии](#): они занимают только одну строку. Перенос однострочного комментария на вторую строку вызовет ошибку компиляции, например:

```
1 | std::cout << "Hello, world!" << std::endl; // это однострочный комментарий  
2 | А это уже не комментарий
```

## Основные рекомендации

В отличие от других языков программирования, C++ не имеет каких-либо ограничений в форматировании кода со стороны программистов. Основное правило заключается в том, чтобы использовать только те способы, которые максимально улучшают читабельность и логичность кода.

**Вот 6 основных рекомендаций:**

**Рекомендация №1:** Вместо символа табуляции (клавиша «Tab») используйте 4 пробела. В некоторых [IDE](#) по умолчанию используются три пробела в качестве одного символа табуляции — это тоже нормально (количество пробелов можно легко настроить в соответствующих пунктах меню вашей IDE).

Причиной использования пробелов вместо символов табуляции является то, что если вы откроете свой код в другом редакторе, то он сохранит правильные отступы, в отличие от использования символов табуляции.

**Рекомендация №2:** Открытие и закрытие фигурных скобок функции должно находиться на одном уровне на отдельных строках:

```
1 | int main()  
2 | {  
3 | }
```

Хотя есть еще и следующий вариант (вы также можете его использовать):

```
1 | int main() {  
2 |     //...  
3 | }
```

Первый вариант хорош тем, что в случае возникновения ошибки несоответствия скобок, найти те самые проблемные скобки визуально будет проще.

**Рекомендация №3: Каждый стейтмент функции должен быть с соответствующим отступом** (клавиша Tab или 4 пробела):

```
1 int main()
2 {
3     std::cout << "Hello world!" << std::endl; // один Tab (4 пробела)
4     std::cout << "Nice to meet you." << std::endl; // один Tab(4 пробела)
5 }
```

**Рекомендация №4: Строки не должны быть слишком длинными.** 72, 78 или 80 символов — это оптимальный максимум строки. Если она будет длиннее, то её следует разбить на несколько отдельных строк:

```
1 int main()
2 {
3     std::cout << "This is a really, really, really, really, really, really, really, really,
4         "really long line" << std::endl; // один дополнительный отступ для строки-продолжения
5
6     std::cout << "This is another really, really, really, really, really, really, really, really,
7         "really long line" << std::endl; // отступ + выравнивание с учетом главы
8
9     std::cout << "This one is short" << std::endl;
10 }
```

**Рекомендация №5: Если длинная строка разбита на части с помощью определенного оператора** (например, << или +), то этот оператор должен находиться в конце этой же строки, а не в начале следующей (так читабельнее).

Правильно:

```
1 std::cout << "This is a really, really, really, really, really, really, really, really, " <<
2     "really long line" << std::endl;
```

Неправильно:

```
1 std::cout << "This is a really, really, really, really, really, really, really, really, "
2     << "really long line" << std::endl;
```

**Рекомендация №6: Используйте пробелы и пропуски строк между стейтментами для улучшения читабельности вашего кода.**

Менее читабельно:

```
1 nCost = 57;
2 nPricePerItem = 24;
3 nValue = 5;
4 nNumberOfItems = 17;
```

Более читабельно:

```
1 |
```

```
1 nCost          = 57;  
2 nPricePerItem  = 24;  
3 nValue         = 5;  
4 nNumberOfItems = 17;
```

Менее читабельно:

```
1 std::cout << "Hello world!" << std::endl; // cout и endl находятся в библиотеке iostream  
2 std::cout << "It is very nice to meet you!" << std::endl; // эти комментарии ухудшают  
3 std::cout << "Yeah!" << std::endl; // особенно, когда строки разной длины
```

Более читабельно:

```
1 std::cout << "Hello world!" << std::endl; // cout и endl находятся в библиотеке iostream  
2 std::cout << "It is very nice to meet you!" << std::endl; // эти комментарии более  
3 std::cout << "Yeah!" << std::endl; // не так ли?
```

Менее читабельно:

```
1 // cout и endl находятся в библиотеке iostream  
2 std::cout << "Hello world!" << std::endl;  
3 // эти комментарии ухудшают читабельность кода  
4 std::cout << "It is very nice to meet you!" << std::endl;  
5 // особенно, когда они в одной куче  
6 std::cout << "Yeah!" << std::endl;
```

Более читабельно:

```
1 // cout и endl находятся в библиотеке iostream  
2 std::cout << "Hello world!" << std::endl;  
3  
4 // эти комментарии читать легче  
5 std::cout << "It is very nice to meet you!" << std::endl;  
6  
7 // ведь они разделены дополнительными строками  
8 std::cout << "Yeah!" << std::endl;
```

Язык C++ позволяет выбрать вам тот стиль форматирования вашего кода, в котором вам будет наиболее комфортно работать.

Оценить статью:

★★★★★ (727 оценок, среднее: 4,94 из 5)



← [Урок №17. Операторы](#)



## Комментариев: 25



1. *Сергей:*

[4 июня 2020 в 05:28](#)

Я, конечно, не далеко не профи в программировании, мягко говоря, но логично предположить, что спор о скобочках решается просто — у нормальной конторы наверняка существует определенный стандарт по оформлению кода, проекта, да даже рабочего стола. Думаю скобочки (и прочее, ну вы поняли) лучше писать там, где их пишут все в вашем микроклимате. "Часть корабля, часть команды", так сказать )))

[Ответить](#)



2. *ЛевБонифаций:*

[28 декабря 2019 в 12:50](#)

Учить использовать пробелы вместо табуляции это признак дурного тона и извращение. Мифические плюсы основанные на том что не все редакторы корректно обрабатывают табуляции — это прошлый век и проблемы сортирных редакторов. В цивилизованном мире белых людей в 99,9% случаев все в полном порядке с этим во всех нормальных редакторах и IDE

Итого: пробелы = плохо, табуляция = хорошо.

[Ответить](#)



1. *Алексей:*

[4 апреля 2020 в 18:46](#)

Очень часто разработчики пользуются текстовыми редакторами, например Sublime (не реклама), для того чтобы быстро открыть один или несколько файлов и посмотреть интересующий фрагмент кода, вместо того, чтобы ждать запуска тяжеловесной IDE с подгрузкой кода проекта на несколько гигабайт. Кроме того, фрагменты кода иногда копируются и вставляются, например, в презентацию. И когда в коде используются одновременно и табы и пробелы, то выглядит это настолько страшно, что быстро понять какие строки находятся внутри блока, а какие — уже за его пределами, просто невозможно. Хотя бы уже по этим причинам "Учить использовать пробелы вместо табуляции" — это НЕ признак дурного тона и извращение, а правильное понимание сути проблем.

[Ответить](#)



3. *Максим:*

[15 июля 2019 в 13:42](#)

Использование пробелов вместо табуляции, на мой взгляд, является извратом. Мало когда написанный в команде или же для себя код будет открываться в другом редакторе.

[Ответить](#)



4. *Александр:*

[11 апреля 2019 в 04:07](#)

Спасибо Юрий! Всё понятно и просто. За ваши труды вам воздастся!

[Ответить](#)



1. *Юрий:*

[11 апреля 2019 в 09:27](#)

Пожалуйста 😊

[Ответить](#)



2. *ManiaC:*

[19 декабря 2019 в 05:43](#)

"Мало когда будет открываться"=="Будет открываться". Поэтому лучше использовать только пробелы. Но это, конечно, пожалуй, наименее значимый фактор качества форматирования. Грамотное построение кода, грамотное комментирование, расстановка отступов и скобок дают наибольший вклад в читабельность.

Главное тут то, что подход с позиции "больше никто это читать не будет" изначально дефектен: наступит день и будут. А ещё более вероятное событие — что однажды понадобится самому пересматривать и вспоминать когда-то написанный и давно забытый код. И с привычкой писать как-попало и то и другое будет выглядеть печально.

А реально неудобства с табуляцией бывают в тех случаях, когда символы табуляции смешиваются с пробелами. Так или иначе, но на практике такое встречается и вот это как раз уже больше тянет на изврат.

PS: Поясню, пожауй, некоторые моменты по теме, которые тут изложены достаточно декларативно:

Форматирование кода — суть есть вёрстка (в типографском/издательском смысле этого слова) и потому всё в этом процессе поистекает из методов компоновки текста, применяемых в издательском деле. А именно: отступы, абзацы, колонки. И применять их следует сообразно их смыслу.

В этом ключе поясню всё по пунктам статьи:

2. Не смотря на частое использование второго варианта, где открывающая скобка ставится в той-же строке, что и выражение (предпочту использовать именно перевод слова "стейтмент"), к которому относится этот составной оператор (то, что в скобках), всё-таки такой вариант имеет явный недостаток: если с функциями всё более однозначно, то после условных операторов и в циклах используются не только составные операторы, но и одиночные, которые не требуется заключать в скобки. И тут читабельность кода падает значительно. И именно поэтому лучше открывающую скобку всё-таки переносить на новую строку и ставить на одном уровне с закрывающей.

3. Каждый следующий уровень вложенности операторов/выражений должен идти с

[равным] отступом относительно предыдущего уровня. Во всех конструкциях с вложенностью, не только в функциях (если вдруг это не очевидно).

4. Строка — это предложение. Короткое предложение лучше слишком длинного, но если это оправдано — то можно. Если длинная строка — это просто значение строки, которое мало значимо для понимания программной логики, то можно и длинную написать.

6. Первое — это колонки. Колонки, конечно читабельней, но если между ними слишком большие промежутки, то лучше поблочно эти промежутки сократить.

Второе — это параграфы и абзацы. В начале параграфа (крупного смыслового блока) ставится заголовок, ёмко отражающий его содержание. В коде — это комментарий, однострочный как правило. Меньшие смысловые блоки — абзацы — , которые обычно не требуют отдельного заглавия, отделяются просто вертикальным отступом, т.е. пустой строкой. Тут ещё небольшая практическая ремарка: не надо понимать эти "параграфы" и "абзацы" формально и жёстко: если короткий комментарий относится к целому небольшому блоку кода, то иногда читабельней поместить его перед этим блоком а не колонкой слева.

PPS:

И самое главное: называйте идентификаторы по их смыслу, отвечая на вопрос: "что делает эта функция" и "что содержит эта переменная/константа". Это не только просто "улучшает читаемость", но и помогает в отладке. Если форматирование кода — это скорее к поиску опечаток и забытых скобок, т.е. тому, о чём подскажет добрый компилятор (но надо помнить, что не во всех языках он столь любезен), то осмысленное именование — это уже к отладке логики. Гораздо проще понять, верно-ли написана функция, выражение, и т.д., когда ясно, что она/оно должно делать или содержать. Потому ошибка — это и есть ситуация, когда действительное не совпадает с должным.

[Ответить](#)



1. *Пётр:*

[27 февраля 2020 в 14:33](#)

По вашему комментарию к пункту 2 есть также рекомендация, которую советую придерживаться новичкам особенно. ВСЕГДА ставить фигурные скобки, даже если оператор один. Во-первых читаемость лучше. Во-вторых, в рабочих проектах даже встречал, когда был один оператор, а потом потребовалось что-то добавить в обработку условия, про скобки благополучно не вспомнили, компилятор ошибки никакой не выдаёт. А в работу программу запустили через месяц и очень долго потом искали, а почему же работает не так, как надо. На эту же тему, кстати, почему обязательно надо использовать систему управления версиями, когда пишешь код.

[Ответить](#)



1. *ManiaC:*

[4 марта 2020 в 12:40](#)

Фигурные скобки — это т.н. "составной оператор". Соответственно если они ставятся, значит предполагается, что в них будет несколько операторов. И это надо понимать, а не просто бездумно писать какие-то скобочки и другие значки. Это в смысле новичков. И новичкам я бы посоветовал всегда думать над тем "какой смысл и назначение в языке у тех букв, знаков, операторов, конструкций и т.д., которые ты пишешь?", "соответствует-ли то, что ты используешь тому, что ты хочешь выразить?".

А по принципу "а если потом потребуется" можно наворотить такого, что при самом прекрасном форматировании кода никто просто не поймёт, за чем там всё это понаписано. Если человек не осознаёт, что и для чего он пишет, то ни какие ухищрения не помогут избежать кривого кода. Скорее наоборот, имхо. Ну и от того, чтобы банально где-то ступить, что-то забыть — от этого, имхо, тоже никто не застрахован. Но если код написан осмысленно (ключевое слово — осмысленно) и осмысленно прокомментирован, то не будет проблем с поиском явных ошибок.

ps: Комментарии в коде — это как избыточное кодирование в передаче данных. Этакая контрольная сумма. Если смысл, заявленный в комментарии не совпадает со смыслом, написанным в коде — значит где-то ошибка. То же самое и со смыслами в идентификаторах переменных, функций и т.д.

Поэтому рекомендация, которой я советую придерживаться особенно всем, — всегда и везде ставьте смысл.



5. *Максим:*

[19 февраля 2019 в 22:21](#)

Автор в этой статье начал молча использовать `using namespace std`, не предупредив, не объяснив что это и для чего, и в дальнейшем без объяснения причин и как это работает продолжает использовать.

Не красиво, совсем.

[Ответить](#)



1. *Юрий:*

[21 февраля 2019 в 01:22](#)

Исправил.

[Ответить](#)



1. *Максим:*

[21 февраля 2019 в 17:52](#)

Смысла в этом исправлении особого не вижу. Пролистав два урока дальше — я вижу тоже самое, молчаливый переход на `using namespace std` без объяснения причин, даже не обращая внимание на это.

Я так бросил уроков 10, потому что неожиданно в тексте появляется функции, которые раньше не использовались, и что это, и для чего они — не объясняется.

[Ответить](#)



1. *Юрий:*

[21 февраля 2019 в 18:43](#)

Читайте уроки №24 и №54. В них всё детально рассказывается.





2. *Максим:*

[21 февраля 2019 в 18:52](#)

То есть, я сейчас должен бросать урок 20 и перепрыгивать на 54, чтобы понять урок 20?

Кажется, это звучит глупо.



3. *Юрий:*

[21 февраля 2019 в 19:14](#)

Вы название урока прочитали? Повторю, если не удосужились — "Урок 18. Whitespace и базовое форматирование". Вам это о чём-либо говорит? В этом уроке рассказывается о whitespaces и базовом форматировании. Здесь не рассказывается о пространствах имён или о стейтментах using.

Для объяснения материала о WHITESPACES И БАЗОВОМ ФОРМАТИРОВАНИИ прилагаются примеры в коде для лучшего усваивания материала. В примерах кода используются параметры, пространство имён std, операторы << и return. О них в этом уроке тоже ничего не сообщается. Как думаете, почему? Правильно, потому что это урок о whitespaces и базовом форматировании. На каждую тему — отдельный урок. Если тема какого-либо объекта сейчас не раскрывается, значит на данном этапе это не нужно. Дойдёте до соответствующего урока — прочитаете и узнаете.

**О том, что вы должны. Вы ничего мне не должны.** Вы можете закрыть этот сайт и забыть его, как самый страшный сон. Если вас не удовлетворяет структура или логика этих уроков — пожалуйста, нажимаете на крестик возле вкладки вверху браузера. Теперь, надеюсь, понятно объяснил?



4. *Михаил:*

[3 мая 2019 в 20:23](#)

Для вас столько бесплатно сделано, а Вы к мелочам цепляетесь. Непонятная строка в коде — это трудность для человека, решившего стать программистом? Сказали бы спасибо Юрию за проделанный труд.



6. *Илья:*

[31 марта 2018 в 17:15](#)

простите, всё что вы объясняете, относится только к консольным приложениям?

[Ответить](#)

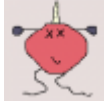


1. *Юрий:*

[31 марта 2018 в 21:23](#)

В этих уроках объясняются фундаментальные основы и понятия в программировании не только на языке C++, но вообще в программировании (основы одинаковы в большинстве всех языков программирования). В качестве примеров да, используются только консольные приложения.

[Ответить](#)



1. *Илья:*

[2 апреля 2018 в 16:07](#)

спасибо,я не знал!

[Ответить](#)



7. *Дима:*

[9 августа 2017 в 19:40](#)

Терпень не могу такой стиль:

```
1 int main()
2 {
3 }
```

Он мне кажется каким-то неаккуратным.  
Мне куда больше нравится такой способ:

```
1 int main() {
2
3 }
```

Для меня он даже выглядит солиднее.

Может быть у меня такое мнение, только после другого языка, в котором хорошим тоном считается именно такой способ (хотя с переносом скобки тоже допускается, хоть и редко используется).

Указанная здесь причина — легче искать несоответствие скобок — их в обоих стилях легко искать. Хотя в любом случае некрасиво писать код в котором десяток вложенных скобок.

Помимо того в способе с переносом скобок получается значительно больше кода, особенно когда есть много мелких функций.

А работать с большим файлом сложнее. Конечно, код все равно разбивается на много файлов, но все же...

Хотя это все конечно дело вкуса)

[Ответить](#)



1. *Юрий:*

[10 августа 2017 в 00:24](#)

В статье описываются рекомендации к базовому форматированию. А каждый уже решает сам, чему ему придерживаться и как писать код. Как уже было сказано, о вкусах не спорят



[Ответить](#)2. *Мгер:*[3 января 2018 в 11:34](#)<https://github.com/Microsoft/WinObjC/wiki/Coding-Standards>

Майкрасофт с Димой согласен

[Ответить](#)1. *Серж:*[8 февраля 2018 в 11:50](#)

По вашей ссылке, если внимательно почитать, есть еще одна ссылка на [https://en.wikipedia.org/wiki/Indentation\\_style#Variant:\\_Stroustrup](https://en.wikipedia.org/wiki/Indentation_style#Variant:_Stroustrup). И вот там очень подробно описываются различные стили. Многие из них (Allman, GNU, Whitesmiths, Horstmann) придерживаются именно такого стиля как в этой статье. Мне это тоже кажется удобнее. Да и при программировании микроконтроллеров используется этот же стиль. Для примера <http://narodstream.ru/avr-urok-3-pishem-kod-na-si-zazhigaem-svetodi/>

[Ответить](#)1. *Мгер:*[31 марта 2018 в 22:33](#)

только у МК обычно язык C, а не C++.

3. *ManiaC:*[4 марта 2020 в 13:32](#)

Дело вкуса — это подход ненаучный. Вкусы у всех разные, а нам нужен общий метод, дающий лучшие результаты. И в данном случае лучшим является тот вариант, который позволяет легче визуально отслеживать начала и концы составных операторов.

И мои аргументы за написание

```
if (условие)
{
    оператор1;
    оператор2;
    ...
}
```

и

```
if (условие)
    только_один_оператор;
и всех остальных аналогичных конструкций такой:
```

— Просматривать только начало строк проще, чем искать наличие нужного символа в конце строки, и тем более — то в конце, то в начале. Кроме того, условие может быть достаточно сложным, и иногда его приходится даже разбивать на несколько строк (да, не

лучший вариант, но иногда оправдано, например чтобы аккуратно откомментировать по частям).

— Проще сверять соответствие пар скобок, когда они на одном уровне по вертикали (естественно весь код внутри должен быть с отступом и не попадать на линию между ними).

— Наличие "фигурных скобок", т.е. обозначения составного оператора дополнительно указывает на то, что там должно быть именно несколько операторов, а не один. (Лишний признак для проверки правильности кода.)

НО:

1)

```
if (условие) {  
    операторы;  
    ...  
}
```

тоже годный вариант. Главное, чтобы остальные правила выполнялись, иначе что так, что этак — всё будет плохо. (Это второй по важности фактор.)

2) Если человек писал код, не осознавая доподлинно, что делают все эти буквы и значки — разобраться в нём не поможет ни какое форматирование. (И это самый главный фактор, который определяет 70-80% читаемости кода, а все эти отступы, где скобочку поставить и прочее, как говорится, — догадайтесь сами 😊 )

\* Всё это выведено из опыта разбирания больших объёмов чужого кода, в некоторых случаях написанного явно в состоянии полной невменяемости.

[Ответить](#)

## Добавить комментарий

Ваш E-mail не будет опубликован. Обязательные поля помечены \*

Имя \*

Email \*

Комментарий

☐ Сохранить моё Имя и E-mail. Видеть комментарии, отправленные на модерацию






☐ Получать уведомления о новых комментариях по электронной почте. Вы можете [подписаться](#) без комментирования.

[TELEGRAM](#)  [КАНАЛ](#)



[ПАБЛИК](#) 

## ТОП СТАТЬИ

-  [Словарь программиста. Сленг, который должен знать каждый кодер](#)
-  [Урок №1. Введение в программирование](#)
-  [70+ бесплатных ресурсов для изучения программирования](#)
-  [Урок №1: Введение в создание игры «Same Game»](#)
-  [Урок №4. Установка IDE \(Интегрированной Среды Разработки\)](#)

- [Ravesli](#)
- - [О проекте](#) -
- - [Пользовательское Соглашение](#) -
- - [Все статьи](#) -
- Copyright © 2015 - 2020