

Ravesli [Ravesli](#)

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





Урок №24. Конфликт имен и std namespace

 [Юрий](#) |

- [Уроки C++](#)

|

 Обновл. 2 Сен 2020 |

 41140

[1](#)  [12](#)

На этом уроке мы рассмотрим, что такое конфликт имен в языке C++ и как его решить с помощью пространств имен и оператора разрешения области видимости.

Оглавление:

1. [Конфликт имен в C++](#)
2. [Пространство имен](#)
3. [Оператор разрешения области видимости ::](#)

Конфликт имен в C++

Допустим, что вам нужно съездить к дальним родственникам в другой город. У вас есть только их адрес: *г. Ржев, ул. Вербовая, 13*. Попав в город Ржев, вы открываете Google Карты/Яндекс.Карты и видите, что есть две улицы с названием *Вербовая*, еще и в противоположных концах города! Какая из них нужна вам? Если у вас нет никакой дополнительной информации (например, вы знаете, что их дом находится возле аптеки или школы), вам придется позвонить им и спросить. Чтобы подобной путаницы не возникало, все названия улиц в городе должны быть уникальными.

Аналогично и в языке C++ все идентификаторы (имена переменных/функций/классов и т.д.) должны быть уникальными. Если в вашей программе находятся два одинаковых идентификатора, то будьте уверены, что ваша программа не скомпилируется: вы получите ошибку **конфликта имен**.

Пример конфликта имен:

a.cpp:

```
1 #include <iostream>
2
3 void doSomething(int x)
4 {
5     std::cout << x;
6 }
```

b.cpp:

```
1 #include <iostream>
2
3 void doSomething(int x)
4 {
5     std::cout << x * 2;
6 }
```

main.cpp:

```
1 void doSomething(int x); // предварительное объявление функции doSomething()
2
3 int main()
4 {
5     doSomething(5);
6
7     return 0;
8 }
```

По отдельности файлы a.cpp, b.cpp и main.cpp скомпилируются. Однако, если a.cpp и b.cpp разместить в одном проекте — произойдет конфликт имен, так как определение функции doSomething() находится сразу в обоих файлах.

Большинство конфликтов имен происходят в двух случаях:

- ➔ Файлы, добавленные в один проект, имеют функцию (или глобальную переменную) с одинаковыми именами (ошибка на этапе [линкинга](#)).
- ➔ Файл .cpp подключает [заголовочный файл](#), в котором идентификатор конфликтует с идентификатором из файла .cpp (ошибка на этапе компиляции).

Как только программы становятся больше, то и идентификаторов используется больше. Следовательно, вероятность возникновения конфликта имен значительно возрастает. Хорошая новость заключается в том, что язык C++ предоставляет достаточно механизмов для предотвращения возникновения конфликтов имен (например, [локальная область видимости](#) или пространства имен).

Пространство имен

В первых версиях языка C++ все идентификаторы из Стандартной библиотеки C++ (такие как [cin/cout](#) и т.д.) можно было использовать напрямую. Тем не менее, это означало, что любой идентификатор из Стандартной библиотеки C++ потенциально мог конфликтовать с именем, которое вы выбрали для ваших собственных идентификаторов. Код, который работал, мог внезапно получить конфликт имен

при подключении нового заголовочного файла из Стандартной библиотеки C++. Или, что еще хуже, код, написанный по стандартам одной версии языка C++, мог уже не работать в новой версии языка C++. Чтобы устранить данную проблему, весь функционал Стандартной библиотеки C++ перенесли в специальную область — **пространство имен** (англ. «*namespace*»).

Аналогично тому, как город гарантирует, что все улицы в его пределах имеют уникальные названия, так и пространство имен гарантирует, что все его идентификаторы — уникальны.

Таким образом, `std::cout` состоит из двух частей: идентификатор `cout` и пространство имен `std`. Весь функционал Стандартной библиотеки C++ определен внутри пространства имен `std` (сокр. от «*standard*»).

Мы еще поговорим о пространствах имен на следующих уроках, а также рассмотрим создание своего собственного пространства имен. Сейчас, главное, что вам нужно запомнить, — это то, что всякий раз, когда вы используете идентификаторы из Стандартной библиотеки C++ (например, `cout`), вы должны сообщать компилятору, что этот идентификатор находится внутри пространства имен `std`.

Правило: При использовании идентификаторов из пространства имен — указывайте используемое пространство имен.

Оператор разрешения области видимости ::

Самый простой способ сообщить компилятору, что определенный идентификатор находится в определенном пространстве имен — использовать **оператор разрешения области видимости ::**. Например:

```
1 | std::cout << "Hello, world!";
```

Здесь мы сообщаем компилятору, что хотим использовать объект `cout` из пространства имен `std`.

Оценить статью:



(464 оценок, среднее: 4,92 из 5)



[← Урок №23. Header guards и #pragma once](#)



[Урок №25. Разработка ваших первых программ →](#)

Комментариев: 12



1. *Сергей:*

[29 августа 2020 в 14:21](#)

Спасибо! Все доступно и понятно.

[Ответить](#)2. *Stern:*[8 мая 2020 в 22:40](#)

Вот скажите, зачем вскользь наводить тень на плетень уроком №24, когда суть namespace у вас рассматривается аж в уроке №53 (да и то, с каким-то подвывихом, как будто вы с иврита переводите)? Вас же и начинающие смотрят, а потом в головах у них сплошной сумбур и отвращение к "плюсам" на уровне рефлексов. Неужели нельзя излагать тему без перлов, подобных "Или, что ещё хуже, код, написанный по стандартам одной версии C++, мог уже не работать в новой версии C++."? Остальных ваших уроков тоже касается, всё подано кусочно-прерывистым методом, напрочь отсутствует целостность. Хотите нормальных отзывов о своих уроках — "причешите" их в нормальное, структурированное и последовательное состояние. Ничего личного.

[Ответить](#)3. *Александр:*[23 апреля 2019 в 04:02](#)

Спасибо Юрий! Двигаюсь по вашим урокам с трудом. Понимаю не сразу и не так просто. Пытаюсь читать и видеть больше, чем просто новый язык.

[Ответить](#)4. *Денис:*[27 января 2019 в 18:07](#)

Я попробовал скомпилировать ваш пример, где якобы, как написано в статье, всё скомпилируется, так как по отдельности эти функции в порядке. Но в итоге я получил ошибку именно из-за повтора. И что это значит?

[Ответить](#)1. *Юрий:*[27 января 2019 в 18:26](#)

Ошибку вы получите, если файлы будут в одном проекте. В вас, с вероятностью 99%, эти файлы находятся в одном проекте, либо вообще в одном файле. Перечитайте внимательнее урок и попробуйте создать 2 ОТДЕЛЬНЫХ проекта и в них разместить код.

В статье сообщается: "если a.cpp и b.cpp разместить в одном проекте – произойдет конфликт". Здесь имеется в виду, что в одном проекте у вас должны быть a.cpp и main.cpp, либо b.cpp и main.cpp, но никак не a.cpp, b.cpp, main.cpp.

[Ответить](#)5. *Михаил:*[22 сентября 2018 в 21:53](#)

После php изучать такие языки очень интересно, ибо не слишком похоже на то, что делаю на работе. Юзаю CLion на домашнем Linux-е, ничем не хуже Visual Studio 2017, если кто-то тоже на Linux-е, рекомендую. А Вам, Юрий, большое спасибо за труд, очень легко двигаться по Вашим урокам.

[Ответить](#)

1. *Юрий:*

[23 сентября 2018 в 12:02](#)

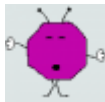
Да, плюсы после PHP это что-то новенькое. Для программистов обязательно если не работать, то хотя бы неплохо разбираться в ОС Linux, поэтому спасибо за совет в выборе IDE 😊

[Ответить](#)

6. *Александр:*

[21 сентября 2018 в 15:27](#)

Я конечно понимаю что использовать std::cout безопаснее, но разве не проще использовать using namespace std;? Или в других пространствах имён есть cout или cin?

[Ответить](#)

1. *Александр:*

[29 января 2019 в 15:21](#)

```
1 | using namespace std;
```

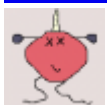
может приводить к очень неожиданным ошибкам, как только Ваш код начинает усложняться. Подключение пространства имен ограничивает те имена, которые Вы выбираете для своих переменных. Причем иногда код может даже компилироваться и работать... но совсем не так, как Вы можете ожидать

[Ответить](#)

7. *Андрей:*

[28 августа 2018 в 23:56](#)

Доброе утро. Не понял всё-таки, как надо использовать это пространство имён, в случае с приведённым выше примером, да и вообще...

[Ответить](#)

8. *Илья:*

[16 июня 2018 в 12:13](#)

извините что вопрос не по теме. существует такая программа visual studio code вроде тоже для программирования, я её установил(ради интереса) она похожа на visual studio community 2017 (которой пользуюсь я) но я не понял для чего же она?

[Ответить](#)**R**

1. Юрий:

[19 июня 2018 в 21:23](#)

Visual Studio Code — это кроссплатформенный редактор с подсветкой синтаксиса и отладчиком. Намного упрощенная версия полноценной IDE, которой является Visual Studio.

[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