

Ravesli [Ravesli](#)


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

Урок №54. using-стейтменты

 [Юрий](#) |

- [Уроки C++](#)

|

 Обновл. 22 Сен 2020 |

 30677

[1](#)  8

Если вы часто используете Стандартную библиотеку C++, то постоянное добавление `std::` к используемым объектам может быть несколько утомительным, не правда ли? Язык C++ предоставляет альтернативы в виде using-стейтментов.

Оглавление:

1. [Использование «using-объявления»](#)
2. [Использование «using-директивы»](#)
3. [Пример конфликта с «using-директивой»](#)
4. [Область видимости «using-объявления» и «using-директивы»](#)
5. [Отмена/замена using-стейтментов](#)

Использование «using-объявления»

Одной из альтернатив является использование «**using-объявления**». Вот программа «Hello, world!» с «using-объявлением» в строке №5:

```
1 #include <iostream>
2
3 int main()
4 {
5     using std::cout; // "using-объявление" сообщает компилятору, что cout следует обра
6     cout << "Hello, world!"; // и никакого префикса std:: уже здесь не нужно!
7     return 0;
8 }
```

Строка `using std::cout;` сообщает компилятору, что мы будем использовать объект [cout](#) из [пространства имен](#) `std`. И каждый раз, когда компилятор будет сталкиваться с `cout`, он будет

понимать, что это `std::cout`.

Конечно, в этом случае, мы не сэкономили много усилий, но в программе, где объекты из пространства имен `std` используются сотни, если не тысячи раз, «`using`-объявление» неплохо так экономит время, усилия + улучшает читабельность кода. Также для каждого объекта нужно использовать отдельное «`using`-объявление» (например, отдельное для `std::cout`, отдельное для `std::cin` и отдельное для `std::endl`).

Хотя этот способ является менее предпочтительным, чем использование префикса `std::`, он все же является абсолютно безопасным и приемлемым.

Использование «`using`-директивы»

Второй альтернативой является использование «`using`-директивы». Вот программа «Hello, world!» с «`using`-директивой» в строке №5:

```
1 #include <iostream>
2
3 int main()
4 {
5     using namespace std; // "using-директива" сообщает компилятору, что мы используем
6     cout << "Hello, world!"; // так что никакого префикса std:: здесь уже не нужно!
7     return 0;
8 }
```

Много разработчиков спорят насчет использования «`using`-директивы». Так как с её помощью мы подключаем ВСЕ имена из пространства имен `std`, то вероятность возникновения конфликтов имен значительно возрастает (но все же эта вероятность в глобальном масштабе остается незначительной). `using namespace std;` сообщает компилятору, что мы хотим использовать всё, что находится в пространстве имен `std`, так что, если компилятор найдет имя, которое не сможет распознать, он будет проверять его наличие в пространстве имен `std`.

Совет: Старайтесь избегать использования «`using`-директивы» (насколько это возможно).

Пример конфликта с «`using`-директивой»

Рассмотрим пример, где использование «`using`-директивы» создает неопределенность:

```
1 #include <iostream>
2
3 int cout() // объявляем нашу собственную функцию "cout"
4 {
5     return 4;
6 }
7
8 int main()
9 {
10    using namespace std; // делаем std::cout доступным по "cout"
11    cout << "Hello, world!"; // какой cout компилятор здесь должен использовать? Тот
12 }
```

```
13 |     return 0;  
14 | }
```

Здесь компилятор не сможет понять, использовать ли ему `std::cout` или функцию `cout()`, которую мы определили сами. В результате, получим ошибку неоднозначности. Хотя это и банальный пример, но если бы мы добавили префикс `std::` к `cout`:

```
1 |     std::cout << "Hello, world!"; // сообщаем компилятору, что хотим использовать std
```

Или использовали бы «using-объявление» вместо «using-директивы»:

```
1 |     using std::cout; // сообщаем компилятору, что cout означает std::cout  
2 |     cout << "Hello, world!"; // так что здесь следует использовать std::cout
```

Тогда наша программа была бы без ошибок.

Большинство программистов избегают использования «using-директивы» именно по этой причине. Другие считают это приемлемым до тех пор, пока «using-директива» используется только в пределах отдельных функций (что значительно сокращает масштабы возникновения конфликтов имен).

Области видимости «using-объявления» и «using-директивы»

Если «using-объявление» или «using-директива» используются в блоке, то они применяются только внутри этого блока (по обычным правилам локальной области видимости). Это хорошо, поскольку уменьшает масштабы возникновения конфликтов имен до отдельных блоков. Однако многие начинающие программисты пишут «using-директиву» в глобальной области видимости (вне функции `main()` или вообще вне любых функций). Этим они *вытаскивают* все имена из пространства имен `std` напрямую в глобальную область видимости, значительно увеличивая вероятность возникновения конфликтов имен. А это уже не хорошо.

Правило: Никогда не используйте using-стейтменты вне тела функций.

Отмена/замена using-стейтментов

Как только один using-стейтмент был объявлен, его невозможно отменить или заменить другим using-стейтментом в пределах области видимости, в которой он был объявлен. Например:

```
1 | int main()  
2 | {  
3 |     using namespace Boo;  
4 |  
5 |     // Отменить «использование пространства имен Boo» здесь невозможно!  
6 |     // Также нет никакого способа заменить «using namespace Boo» на другой using-сте  
7 |  
8 |     return 0;  
9 | } // действие using namespace Boo заканчивается здесь
```

Лучшее, что вы можете сделать — это намеренно ограничить область применения using-стейтментов с самого начала, используя правила [локальной области видимости](https://ravesli.com/urok-54-using-statements/):

```
1 int main()
2 {
3     {
4         using namespace Boo;
5         // Здесь всё относится к пространству имен Boo::
6     } // действие using namespace Boo заканчивается здесь
7
8     {
9         using namespace Foo;
10        // Здесь всё относится к пространству имен Foo::
11    } // действие using namespace Foo заканчивается здесь
12
13    return 0;
14 }
```

Конечно, всей этой головной боли можно было бы избежать, просто используя оператор разрешения области видимости (::).

Оценить статью:

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

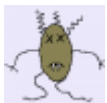


[← Урок №53. Пространства имен](#)



[Урок №55. Неявное преобразование типов данных →](#)

Комментариев: 8



1. Александр:
[17 июня 2020 в 07:55](#)

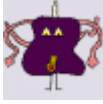
Здравствуйте. Можете подсказать список всех объектов std?
Это было бы полезно держать "под боком" во время написания программы, да и поможет запомнить наиболее часто используемые объекты.
Заранее спасибо.

[Ответить](#)



2. Глеб:
[6 апреля 2020 в 07:51](#)

Не будет ли лучшим решением написать свою функцию <<cout>> для вывода текста на дисплей экрана?

[Ответить](#)1. *Sarad:*[23 апреля 2020 в 15:53](#)

И как вы себе это представляете?

[Ответить](#)3. *Артеми́й:*[21 февраля 2020 в 11:04](#)

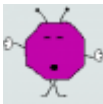
Вот с первого урока в комментариях пишут, что надо использовать стандартное пространство имен и из-за этого комментария повылазило множество ошибок, т.к. глобальное определение вытаскивает все имена о которых учащийся даже не знает. Предлагаю кратко в начале ООП прописать или паттерны что ли, как надо делать, писать структуру кода, с чего можно начинать, а с чего нельзя начинать.

[Ответить](#)4. *Константин:*[12 августа 2018 в 17:22](#)

Артем, поддерживаю — дело говоришь!

[Ответить](#)5. *Артем:*[13 июля 2018 в 19:13](#)

Не понимаю, почему такой большой проблемой является использование `using`. Программисты должны знать, по идее, с какими словами будут возникать ошибки (мне, например, и в голову не придет назвать функцию или переменную `cout` или `cin` + когда мы пишем исходный код, если вдруг кому и придет идея назвать переменную `cout`, то программа покажет, что такое слово есть в библиотеке `std`). Только даун, уж извините, ошибется в таком моменте. При этом использование `using` экономит много времени. Не нужно постоянно прописывать имя библиотеки и `::`.

[Ответить](#)1. *Александр:*[2 февраля 2019 в 11:27](#)

Я искренне за Вас рад, если Вы помните ВСЕ имена из `std`...

иногда возникают очень неожиданные ошибки при переходе на другой компилятор: у Вас все может работать превосходно, а на другой системе не компилироваться вообще

в примерах речь об пространстве `std`, но на практике может идти речь о подключении пространств имен других программистов... Вы и эти все имена запоминать планируете?

Или будете вводить различные правила по использованию различных пространств имен?

[Ответить](#)



2. *Dmitry:*

[10 мая 2020 в 09:18](#)

Проблема в том, мой друг Горацио, что принимаются новые стандарты языка. Может быть на сегодняшний день ты выучил все имена из стандартной библиотеки и написал суперский код, настолько популярный, что его подрубают как библиотеку во все серверные приложения. Но вот принимается новый стандарт и в этот злосчастный std могут записывают новое имя, которое вызывает конфликт с твоим. Угадай, к кому будут вопросы после того, как либа перестанет работать?

[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 «SameGame» на C++/MFC](#)
-  [Урок №4. Установка IDE \(Интегрированной Среды Разработки\)](#)

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