

Ravesli [Ravesli](#)


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

Урок №53. Пространства имен

 [Юрий](#) |

- [Уроки C++](#)

|

 Обновл. 24 Окт 2020 |

 40578

[1](#)  18

Этот урок является продолжением [урока №24](#).

Оглавление:

1. [Конфликт имен](#)
2. [Что такое пространство имен?](#)
3. [Доступ к пространству имен через оператор разрешения области видимости \(::\)](#)
4. [Пространства имен с одинаковыми названиями](#)
5. [Псевдонимы и вложенные пространства имен](#)

Конфликт имен

Конфликт имен возникает, когда два одинаковых идентификатора находятся в одной области видимости, и компилятор не может понять, какой из этих двух следует использовать в конкретной ситуации. Компилятор или линкер выдаст ошибку, так как у них недостаточно информации, чтобы решить эту неоднозначность. Как только программы увеличиваются в объемах, количество идентификаторов также увеличивается, следовательно, увеличивается и вероятность возникновения конфликтов имен.

Рассмотрим пример такого конфликта. `boo.h` и `doo.h` — это [заголовочные файлы](#) с функциями, которые выполняют разные вещи, но имеют одинаковые имена и [параметры](#).

`boo.h`:

```
1 // функция doOperation() выполняет операцию сложения своих параметров
2 int doOperation(int a, int b)
3 {
4     return a + b;
```

```
5 }
```

doo.h:

```
1 // функция doOperation() выполняет операцию вычитания своих параметров
2 int doOperation(int a, int b)
3 {
4     return a - b;
5 }
```

main.cpp:

```
1 #include <iostream>
2 #include "boo.h"
3 #include "doo.h"
4
5 int main()
6 {
7     std::cout << doOperation(5, 4); // какая версия doOperation() выполнится здесь?
8     return 0;
9 }
```

Если boo.h и doo.h скомпилировать отдельно, то всё пройдет без инцидентов. Однако, соединив их в одной программе, мы подключим две разные функции, но с одинаковыми именами и параметрами, в одну область видимости (глобальную), а это, в свою очередь, приведет к конфликту имен. В результате, компилятор выдаст ошибку. Для решения подобных проблем и добавили в язык C++ такую концепцию, как пространства имен.

Что такое пространство имен?

Пространство имен определяет область кода, в которой гарантируется уникальность всех идентификаторов. По умолчанию, глобальные переменные и обычные функции определены в **глобальном пространстве имен**. Например:

```
1 int g_z = 4;
2
3 int boo(int z)
4 {
5     return -z;
6 }
```

Глобальная переменная g_z и функция boo() определены в глобальном пространстве имен.

В примере, приведенном выше, при подключении файлов boo.h и doo.h обе версии doOperation() были включены в глобальное пространство имен, из-за чего, собственно, и произошел конфликт имен.

Чтобы избежать подобных ситуаций, когда два независимых объекта имеют идентификаторы, которые могут конфликтовать друг с другом при совместном использовании, язык C++ позволяет объявлять собственные пространства имен через **ключевое слово namespace**. Всё, что объявлено внутри пользовательского пространства имен, — принадлежит только этому пространству имен (а не глобальному).

Перепишем заголовочные файлы из вышеприведенного примера, но уже с использованием namespace:

boo.h:

```
1 namespace Boo
2 {
3     // Эта версия doOperation() принадлежит пространству имен Boo
4     int doOperation(int a, int b)
5     {
6         return a + b;
7     }
8 }
```

doo.h:

```
1 namespace Doo
2 {
3     // Эта версия doOperation() принадлежит пространству имен Doo
4     int doOperation(int a, int b)
5     {
6         return a - b;
7     }
8 }
```

Теперь doOperation() из файла boo.h находится в пространстве имен Boo, а doOperation() из файла doo.h — в пространстве имен Doo. Посмотрим, что произойдет при перекомпиляции main.cpp:

```
1 int main()
2 {
3     std::cout << doOperation(5, 4); // какая версия doOperation() здесь выполнится?
4     return 0;
5 }
```

Результатом будет еще одна ошибка:

```
C:\VCProjects\Test.cpp(15) : error C2065: 'doOperation' : undeclared
identifier
```

Случилось так, что когда мы попытались вызвать функцию doOperation(), компилятор заглянул в глобальное пространство имен в поисках определения doOperation(). Однако, поскольку ни одна из наших версий doOperation() не находится в глобальном пространстве имен, компилятор не смог найти определение doOperation() вообще!

Существует два разных способа сообщить компилятору, какую версию doOperation() следует использовать: через оператор разрешения области видимости или с помощью using-стейтментов (о них мы поговорим на следующем уроке).

Доступ к пространству имен через оператор разрешения области видимости (::)

Первый способ указать компилятору искать идентификатор в определенном пространстве имен — это использовать название необходимого пространства имен вместе с оператором разрешения области видимости (::) и требуемым идентификатором.

Например, сообщим компилятору использовать версию doOperation() из пространства имен Boo:

```
1 int main(void)
2 {
3     std::cout << Boo::doOperation(5, 4);
4     return 0;
5 }
```

Результат:

9

Если бы мы захотели использовать версию doOperation() из пространства имен Doo:

```
1 int main(void)
2 {
3     std::cout << Doo::doOperation(5, 4);
4     return 0;
5 }
```

Результат:

1

Оператор разрешения области видимости хорош, так как позволяет выбрать конкретное пространство имен. Мы даже можем сделать следующее:

```
1 int main(void)
2 {
3     std::cout << Boo::doOperation(5, 4) << '\n';
4     std::cout << Doo::doOperation(5, 4) << '\n';
5     return 0;
6 }
```

Результат:

9

1

Также этот оператор можно использовать без какого-либо префикса (например, ::doOperation). В таком случае мы ссылаемся на глобальное пространство имен.

Пространства имен с одинаковыми названиями

Допускается объявление пространств имен в нескольких местах (либо в нескольких файлах, либо в нескольких местах внутри одного файла). Всё, что находится внутри одного блока имен, считается частью только этого блока.

add.h:

```
1 namespace DoMath
2 {
```

```
3 // Функция add() является частью пространства имен DoMath
4 int add(int x, int y)
5 {
6     return x + y;
7 }
8 }
```

subtract.h:

```
1 namespace DoMath
2 {
3     // Функция subtract() является частью пространства имен DoMath
4     int subtract(int x, int y)
5     {
6         return x - y;
7     }
8 }
```

main.cpp:

```
1 #include "add.h" // импортируем DoMath::add()
2 #include "subtract.h" // импортируем DoMath::subtract()
3
4 int main(void)
5 {
6     std::cout << DoMath::add(5, 4) << '\n';
7     std::cout << DoMath::subtract(5, 4) << '\n';
8
9     return 0;
10 }
```

Всё работает, как нужно.

Стандартная библиотека C++ широко использует эту особенность, поскольку все заголовочные файлы, которые находятся в ней, реализуют свой функционал внутри пространства имен std.

Псевдонимы и вложенные пространства имен

Одни пространства имен могут быть вложены в другие пространства имен. Например:

```
1 #include <iostream>
2
3 namespace Boo
4 {
5     namespace Doo
6     {
7         const int g_x = 7;
8     }
9 }
10
11 int main()
12 {
```

```
13     std::cout << Boo::Doo::g_x;  
14     return 0;  
15 }
```

Обратите внимание, поскольку Doo находится внутри Boo, то доступ к g_x осуществляется через Boo::Doo::g_x.

Так как это не всегда удобно и эффективно, то C++ позволяет создавать псевдонимы для пространств имен:

```
1  #include <iostream>  
2  
3  namespace Boo  
4  {  
5      namespace Doo  
6      {  
7          const int g_x = 7;  
8      }  
9  }  
10  
11 namespace Foo = Boo::Doo; // Foo теперь считается как Boo::Doo  
12  
13 int main()  
14 {  
15     std::cout << Foo::g_x; // это, на самом деле, Boo::Doo::g_x  
16     return 0;  
17 }
```

Стоит отметить, что пространства имен в C++ не были разработаны, как способ реализации информационной иерархии — они были разработаны в качестве механизма предотвращения возникновения конфликтов имен. Как доказательство этому, вся Стандартная библиотека шаблонов находится в единственном пространстве имен std::.

Вложенность пространств имен не рекомендуется использовать, так как при неумелом использовании увеличивается вероятность возникновения ошибок и дополнительно усложняется логика программы.

Оценить статью:

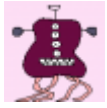
★★★★★ (329 оценок, среднее: 4,96 из 5)



[← Урок №52. Связи, область видимости и продолжительность жизни](#)

[Урок №54. using-стейтменты →](#)

Комментариев: 18



1. *cybersatori:*

[12 февраля 2020 в 21:42](#)

добавлю что если используете визуал студию, то необходимо тестировать код через консольное приложение иначе возникает ошибка линкёра, мало ли кто тоже столкнётся.

[Ответить](#)

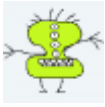


1. *stalker572:*

[5 марта 2020 в 16:23](#)

Эм... кажется он среди 33 моих ошибок увидел ещё и ошибку пространств имен...
Поэтому я здесь...

[Ответить](#)



2. *Борис:*

[29 апреля 2020 в 18:50](#)

Хорошо что не "ошибка линкора" 😊

[Ответить](#)



2. *Вася:*

[13 августа 2019 в 13:09](#)

Почему тут не используется Header guards или #pragma once? Это является необязательным?

[Ответить](#)



3. *Александр:*

[9 мая 2019 в 10:42](#)

Из урока 21 раздела Советы: "Не определяйте функции в заголовочных файлах".

Из текущего урока, в заголовочных файлах boo.h и doo.h ОПРЕДЕЛЕНА ФУНКЦИЯ.

Так где правда-то?

[Ответить](#)



1. *Денис:*

[30 января 2020 в 05:56](#)

Правда в том, что это просто пример. Не стал автор писать отдельно сrr файл для функции.

[Ответить](#)



4. Константин:

[31 декабря 2018 в 04:27](#)

Уважаемый Юра! Подскажи, пожалуйста, как выводить на экран название той переменной, которую инициализирует пользователь в процессе выполнения программы (run-time)? Пример: файлы GeoConClay.cpp и похожие содержат такой код:

```
1 #include <iostream>
2 #include "enNum.h" // enNum() - функция ввода числа типа double
3
4 namespace GeoConClay
5 {
6     extern const double w(enNum());
7     extern const double wl(enNum());
8     extern const double wp(enNum());
9     extern const double ip(enNum());
10    extern const double p(enNum());
11    extern const double ps(enNum());
12    extern const double e(enNum());
13 }
```

файлы GeoConClay.h и похожие содержат такой код:

```
1 #ifndef GEO_CON_CLAY_H
2 #define GEO_CON_CLAY_H
3
4 namespace GeoConClay
5 {
6     extern const double w;
7     extern const double wl;
8     extern const double wp;
9     extern const double ip;
10    extern const double p;
11    extern const double ps;
12    extern const double e;
13 }
14 #endif
```

файл zvit.cpp содержит фрагмент такого кода:

```
1 #include <iostream>
2 #include <Windows.h>
3
4 #include "enNum.h"
5
6 #include "GeoConSand.h"
7 #include "GeoConSand2.h"
8 #include "GeoConClay.h"
9 #include "GeoConClay2.h"
10
11 int main()
```



```

12 {
13     SetConsoleCP(1251);
14     SetConsoleOutputCP(1251);
15     using namespace std;
16
17     cout << " Природная влажность несвязного грунта
18     cout << " Плотность в природном залегании
19     cout << " Плотность частиц
20     cout << " Коэффициент пористости образца естественного сложения e = " << Ge
21     cout << endl;
22     cout << " Природная влажность связного грунта
23     cout << " Плотность в природном залегании
24     cout << " Плотность частиц
25     cout << " Коэффициент пористости образца естественного сложения e = " << Ge
26     cout << " Влажность грунта на границе текучести
27     cout << " Влажность грунта на границе раскатывания
28     cout << " Число пластичности
29     cout << endl;
30     cout << " Природная влажность несвязного грунта (2)
31     cout << " Плотность в природном залегании
32     cout << " Плотность частиц
33     cout << " Коэффициент пористости образца естественного сложения e = " << Ge
34     cout << endl;
35     cout << " Природная влажность связного грунта (2)
36     cout << " Плотность в природном залегании
37     cout << " Плотность частиц
38     cout << " Коэффициент пористости образца естественного сложения e = " << Ge
39     cout << " Влажность грунта на границе текучести
40     cout << " Влажность грунта на границе раскатывания
41     cout << " Число пластичности
42     return 0;
43 }

```

При запуске программы в консольном окне ввожу (с бумажки) данные, а при выводе они оказываются присвоенными не в те переменные...

[Ответить](#)



1. *Константин:*

[16 февраля 2019 в 15:11](#)

Ага! Путём эмпирического подбора разобрался в очередности присвоения значений пользователем в константные переменные!!!

[Ответить](#)



5. *Андрей:*

[14 декабря 2018 в 18:03](#)

Что делать если, в 2-х разных dll (от разных разработчиков) имена пространства имен одинаковые и нет исходного кода?

Компилятор пишет: "ошибка: C2757: GenApi: символ с этим именем уже существует, поэтому оно не может быть использовано в качестве имени пространства имен"

[Ответить](#)



6. *Константин:*

[22 августа 2018 в 13:10](#)

Юра, скажи, пожалуйста, освоив твои труды, можно ли написать некий код, который, подключив к AutoCad (или какой-то другой "рисовалке"), заставит его начертить некие простые геометрические фигуры или надо самому сидеть пялиться в монитор и орудовать мышкой? (я, собственно, и начал изучать C++ чтобы это сделать:-)

[Ответить](#)



1. *Юрий:*

[22 августа 2018 в 19:19](#)

Можно. Это ведь великий и могучий C++. Начертить фигуры можно и с помощью Visual Studio, не обязательно подключать AutoCad. Но сам принцип написания такой программы предполагает, что вы указываете координаты фигуры (т.е. как она рисуется), а затем уже можете добавить возможность изменения углов, длины стороны, радиуса, диаметра и т.д. и фигура будет нарисована автоматически с указанными параметрами.

[Ответить](#)



1. *Константин:*

[26 августа 2018 в 14:24](#)

И с помощью Code::Blocks (которым я пользуюсь) можно начертить?

[Ответить](#)



1. *Юрий:*

[27 августа 2018 в 22:09](#)

Code::Blocks не использовал, поэтому не могу сказать точно.



2. *Артемий:*

[21 февраля 2020 в 10:04](#)

CodeBlocks местами гибче... Мне даже больше нравится чем VS



7. *Константин:*

[21 августа 2018 в 13:09](#)

Юрий, Ваши труды спасают меня от депрессии:-) В этом уроке мне не понятно появление void в скобках после main. Растолкуйте, пожалуйста.

[Ответить](#)

1.  *Юрий:*
[21 августа 2018 в 21:19](#)

Спасибо, что читаете 😊 void указано здесь для того, чтобы сообщить, что функция main не принимает никаких параметров.

[Ответить](#)

1.  *Константин:*
[22 августа 2018 в 12:59](#)

То есть именно в данном фрагменте void употреблён как пережиток языка C. а суть дела не изменилась — не впускаю никаких параметров и баста! Я правильно понял?

[Ответить](#)

1.  *Юрий:*
[22 августа 2018 в 19:14](#)

Да, в принципе в C++ void не обязательно указывать, можно просто пустые скобки, но в первых версиях языка эта практика применялась. И да, это всё пошло от языка C.

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