

Ravesli [Ravesli](#)


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


Урок №15. Локальная область видимости

 [Юрий](#) |

- [Уроки C++](#)

|

 Обновл. 19 Сен 2020 |

 44320

[1](#)  [15](#)

Как мы уже знаем из предыдущих уроков, при выполнении процессором **стейтмента** `int x;` создается переменная. Возникает вопрос: «Когда эта переменная уничтожается?».

Область видимости переменной определяет, кто может видеть и использовать переменную во время её существования. И параметры функции, и переменные, которые объявлены внутри функции, имеют **локальную область видимости**. Другими словами, эти параметры и переменные используются только внутри функции, в которой они объявлены. Локальные переменные создаются в точке объявления и уничтожаются, когда выходят из области видимости.

Рассмотрим следующую программу:

```
1  #include <iostream>
2
3  int add(int a, int b) // здесь создаются переменные a и b
4  {
5      // a и b можно видеть/использовать только внутри этой функции
6      return a + b;
7  } // здесь a и b выходят из области видимости и уничтожаются
8
9  int main()
10 {
11     int x = 7; // здесь создается и инициализируется переменная x
12     int y = 8; // здесь создается и инициализируется переменная y
13     // x и y можно использовать только внутри функции main()
14     std::cout << add(x, y) << std::endl; // вызов функции add() с a = x и b = y
15     return 0;
16 } // здесь x и y выходят из области видимости и уничтожаются
```

Параметры `a` и `b` функции `add()` создаются при вызове этой функции, используются только внутри нее и уничтожаются по завершении выполнения этой функции.

Переменные `x` и `y` функции `main()` можно использовать только внутри `main()` и они также уничтожаются по завершении выполнения функции `main()`.

Для лучшего понимания давайте детально разберем ход выполнения этой программы:

- ➔ выполнение начинается с функции `main()`;
- ➔ создается переменная `x` в функции `main()` и ей присваивается значение 7;
- ➔ создается переменная `y` в функции `main()` и ей присваивается значение 8;
- ➔ вызывается функция `add()` с параметрами 7 и 8;
- ➔ создается переменная `a` в функции `add()` и ей присваивается значение 7;
- ➔ создается переменная `b` в функции `add()` и ей присваивается значение 8;
- ➔ выполняется операция сложения чисел 7 и 8, результатом является значение 15;
- ➔ функция `add()` возвращает значение 15 обратно в caller (в функцию `main()`);
- ➔ переменные `a` и `b` функции `add()` уничтожаются;
- ➔ функция `main()` выводит значение 15 на экран;
- ➔ функция `main()` возвращает 0 в операционную систему;
- ➔ переменные `x` и `y` функции `main()` уничтожаются.

Всё!

Обратите внимание, если бы функция `add()` вызывалась дважды, параметры `a` и `b` создавались и уничтожались бы также дважды. В программе с большим количеством функций, переменные создаются и уничтожаются часто.

Локальная область видимости предотвращает возникновение конфликтов имен

Из примера, приведенного выше, понятно, что переменные `x` и `y` отличаются от переменных `a` и `b`.

Теперь давайте рассмотрим следующую программу:

```
1  #include <iostream>
2
3  int add(int a, int b) // здесь создаются переменные a и b функции add()
4  {
5      return a + b;
6  } // здесь a и b функции add() выходят из области видимости и уничтожаются
7
8  int main()
9  {
10
```

```
11 | int a = 7; // здесь создается переменная a функции main()
12 | int b = 8; // здесь создается переменная b функции main()
13 | std::cout << add(a, b) << std::endl; // значения переменных a и b функции main()
14 | return 0;
    | } // здесь a и b функции main() выходят из области видимости и уничтожаются
```

Здесь мы изменили имена переменных `x` и `y` функции `main()` на `a` и `b`. Программа по-прежнему работает корректно, несмотря на то, что функция `add()` также имеет переменные `a` и `b`. Почему это не вызывает конфликта имен? Дело в том, что `a` и `b`, принадлежащие функции `main()`, являются локальными переменными, функция `add()` не может их видеть, точно так же, как функция `main()` не может видеть переменные `a` и `b`, принадлежащие функции `add()`. Ни `add()`, ни `main()` не знают, что они имеют переменные с одинаковыми именами!

Это значительно снижает возможность возникновения конфликта имен. Любая функция не должна знать или заботиться о том, какие переменные находятся в другой функции. Это также предотвращает возникновение ситуаций, когда одни функции могут непреднамеренно (или намеренно) изменять значения переменных других функций.

Правило: Имена, которые используются внутри функции (включая параметры), доступны/видны только внутри этой же функции.

Тест

Каким будет результат выполнения следующей программы?

```
1 | #include <iostream>
2 |
3 | void doMath(int a)
4 | {
5 |     int b = 5;
6 |     std::cout << "doMath: a = " << a << " and b = " << b << std::endl;
7 |     a = 4;
8 |     std::cout << "doMath: a = " << a << " and b = " << b << std::endl;
9 | }
10 |
11 | int main()
12 | {
13 |     int a = 6;
14 |     int b = 7;
15 |     std::cout << "main: a = " << a << " and b = " << b << std::endl;
16 |     doMath(a);
17 |     std::cout << "main: a = " << a << " and b = " << b << std::endl;
18 |     return 0;
19 | }
```

Ответ

Результат выполнения программы:

```
main: a = 6 and b = 7
doMath: a = 6 and b = 5
doMath: a = 4 and b = 5
main: a = 6 and b = 7
```

Вот ход выполнения этой программы:

- выполнение начинается с функции `main()`;
- создается переменная `a` в функции `main()`, ей присваивается значение 6;
- создается переменная `b` в функции `main()`, ей присваивается значение 7;
- `cout` выводит `main: a = 6 and b = 7`;
- вызывается `doMath()` с аргументом 6;
- создается переменная `a` в функции `doMath()`, ей присваивается значение 6;
- выполняется инициализация переменной `b` функции `doMath()` значением 5;
- `cout` выводит `doMath: a = 6 and b = 5`;
- переменной `a` функции `doMath()` присваивается значение 4;
- `cout` выводит `doMath: a = 4 and b = 5`;
- переменные `a` и `b` функции `doMath()` уничтожаются;
- `cout` выводит `main: a = 6 and b = 7`;
- функция `main()` возвращает 0 в операционную систему;
- переменные `a` и `b` функции `main()` уничтожаются.

Обратите внимание, даже когда мы присвоили значения переменным `a` и `b` внутри функции `doMath()`, на переменные внутри функции `main()` это никак не повлияло.

Оценить статью:



(591 оценок, среднее: 4,91 из 5)



[← Урок №14. Почему функции — полезны, и как их эффективно использовать?](#)

[Урок №16. Ключевые слова и идентификаторы](#)



Комментариев: 15



1. Андрей:

[6 июля 2020 в 22:47](#)

Здравствуйте.

Материал отличны, легко и интересно читать.

Спасибо

Относительно последнего теста в уроке №15

Исходя из правила: "Имена, которые используются внутри функции (включая параметры), доступны/видны только внутри этой же функции."

Исходя из правила немного не понятно как и откуда переменная "a" получает значение "6". По моему пониманию значение все таки взято из `main` так называемого `caller` и передано в функцию `doMath` и присвоено опять же переменной "a". ведь в функции `doMath` до исполнения строки кода `std::cout << "doMath: a = " << a << " and b = " << b << std::endl;` значение переменной "a" определено не было.

Но вот что меня совсем запутало так это то что все происходит и самое главное компилируется без ошибок — так это то что функция `doMath` с типом возврата `void`.

Поясните пожалуйста для тех кто в танке.

Заранее благодарен.

[Ответить](#)



1. aaa:

[11 августа 2020 в 19:30](#)

компилятор все выполняет построчно. научись читать его построчно будет легче учить.
так вот:

16: компилятор обращается к строке 3 с параметром `a=6`. так как в `main` видит только свою переменную `a`. в функции `doMath` создается своя локальная переменная `a` значение которой взято из `main`.

7: переменной `a` присваивается значение из 6 на 4. переменной `a` из `main` в этом блоке не существует.

может тебя запутало что они одинаково называются.

могло быть вот так

```
1 void doMath(int x)
2 {
3     int y = 5;
4     std::cout << "doMath: x = " << x << " and y = " << y << std::endl;
5     x = 4;
6     std::cout << "doMath: x = " << x << " and y = " << y << std::endl;
7 }
8
9 int main()
10 {
11     int a = 6;
```

```
12 |     int b = 7;  
13 |     std::cout << "main: a = " << a << " and b = " << b << std::endl;  
14 |     doMath(a);  
15 |     std::cout << "main: a = " << a << " and b = " << b << std::endl;  
16 |     return 0;  
17 | }
```

функция doMath ничего не возвращает. она просто выводит на экран и сразу закрывается(ничего не нужно хранить в памяти чисто инструкции). а переменной a и b в функции doMath выделяется место в памяти и потом сразу удаляется

[Ответить](#)



1. *Андрей:*

[16 августа 2020 в 08:02](#)

Спасибо за развернутый ответ. Как только отправил вопрос сразу понял что "функция doMath ничего не возвращает. она просто выводит на экран и сразу закрывается"

[Ответить](#)



2. *Михаил:*

[2 июня 2020 в 16:28](#)

не понял как формируются две срединных строки. Почему сначала аргументу "a" сначала присваивается значение из main-на, а потом из doMath ? А "b" всегда остается из doMath ?

[Ответить](#)



1. *Сергей:*

[14 июня 2020 в 09:20](#)

В ответе написан ход выполнения программы. Перечитайте его

[Ответить](#)



3. *Андрей:*

[5 мая 2020 в 19:49](#)

Можете объяснить, что означает выход из области видимости(какойнибудь жизненный пример привести), и как переменные после этого уничтожаются.

Я хоть и внимательно прочитал и законспектировал, но эти 2 момента не понял

[Ответить](#)



1. *Константин:*

[7 мая 2020 в 21:23](#)

Андрей, ну вот представь подсобные хозяйства на селе: фигурные скобки в функции это ограничители вольера (область видимости), животные это переменные, килограммы массы — их значения. Какие имена хрюшек? Правильно! Внутри одной области видимости уникальные, а вне — одинаковые. Пусть дело происходит в мультфильме. Тогда мы можем взять килограммы одной хрюшки и не отобрать их у неё, а именно скопировать в другую хрюшку при этом вытеснив уже имевшиеся кг в последней! Андэсденд ми?

[Ответить](#)



1. Андрей:

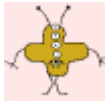
[9 мая 2020 в 17:19](#)

Спасибо за ответ.

Про область видимости я давно понял, а про уничтожение переменных не до конца понимал.

Я тут у прогеров поспрашивал и они сказали, что уничтожение — вынос мусора из памяти. И тогда да меня дошло

[Ответить](#)



4. данила:

[5 декабря 2018 в 12:00](#)

судя из теста понятно так же что уничтожаются аргументы ,сами то параметры остаются. Но аргумент так же уничтожается после объявления нового аргумента к этому же параметру . Как пример ,в функции майн() переопределил значение переменной В (В=8),получается что предыдущее значение В (7) уничтожено. Так ли я понял урок?

```

1 void doMath(int a)
2 {
3     int b = 5;
4     std::cout << "doMath: a = " << a << " and b = " << b << std::endl;
5     a = 4;
6
7     std::cout << "doMath: a = " << a << " and b = " << b << std::endl;
8 }
9
10 int main()
11 {
12     int a = 6;
13     int b = 7;
14     std::cout << "main: a = " << a << " and b = " << b << std::endl;
15     doMath(a);
16     b = 8;
17     std::cout << "main: a = " << a << " and b = " << b << std::endl;
18     return 0;
19 }
```

[Ответить](#)



5. Александра:

[30 августа 2018 в 11:59](#)

Отличный урок! Да и все остальные тоже =) Понятным и доступным языком все объясняется. Но вот проблема с этим уроком: не посмотреть ответ на тест =(

[Ответить](#)



1. *Юрий:*

[30 августа 2018 в 23:18](#)

Починил.

[Ответить](#)



6. *Степан:*

[13 апреля 2018 в 16:37](#)

Привет! Будешь дальше переводить оригинальный сайт?
Интересно почитать про жесты из STL.

Отличный сайт, спасибо!

[Ответить](#)



1. *Юрий:*

[14 апреля 2018 в 19:14](#)

Привет. Да, малость уроков осталась. Самому тоже интересно дойти до STL.

Пожалуйста 😊

[Ответить](#)



7. *rainkiller:*

[31 марта 2018 в 17:34](#)

Очень полезный урок. Спасибо!

[Ответить](#)



1. *Юрий:*

[31 марта 2018 в 21:24](#)

Пожалуйста 😊

[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