

Ravesli [Ravesli](#)


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

## Урок №27. Отладка программ: стек вызовов и отслеживание переменных

 [Юрий](#) |

- [Уроки C++](#)

|

 Обновл. 2 Сен 2020 | 35973[1](#)  [12](#)

На предыдущем уроке о [степпинге и точках останова](#), мы узнали, как, с их использованием, контролировать выполнение программы шаг за шагом. Тем не менее, на этом не заканчиваются все полезные возможности отладчика. Он также позволяет отслеживать значения переменных.

**Примечание:** Перед тем как продолжить, убедитесь, что вы находитесь в [режиме конфигурации «Debug»](#).

Оглавление:

1. [Отслеживание переменных](#)
2. [Окно просмотра](#)
3. [Стек вызовов](#)
4. [Заключение](#)

### Отслеживание переменных

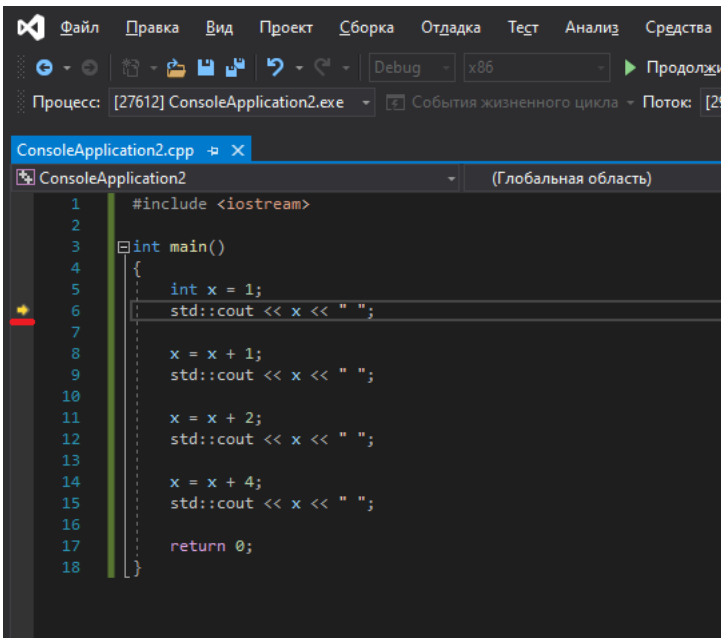
**Отслеживание переменных** — это процесс проверки значений переменных во время отладки. Например:

```
1 #include <iostream>
2
3 int main()
4 {
5     int x = 1;
6     std::cout << x << " ";
7
8     x = x + 1;
9     std::cout << x << " ";
10
11    x = x + 2;
12    std::cout << x << " ";
13
14    x = x + 4;
15    std::cout << x << " ";
16
17    return 0;
18 }
```

Результат выполнения программы:

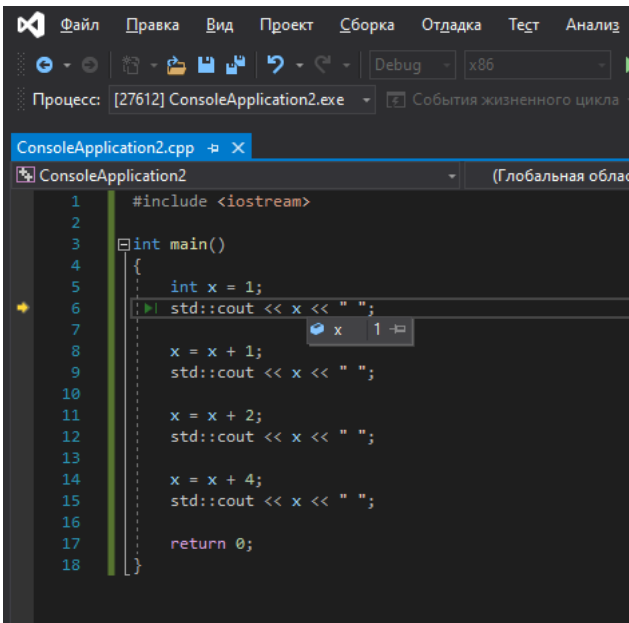
1 2 4 8

Используя команду «Выполнить до текущей позиции» переместитесь к строке `std::cout << x << " ";`:

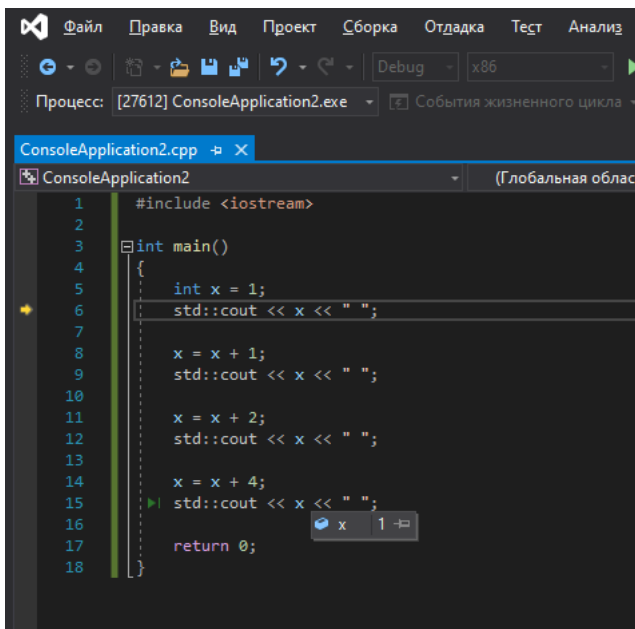


К этому моменту переменная `x` уже создана и инициализирована, поэтому, при проверке этой переменной, вы должны увидеть число `1`.

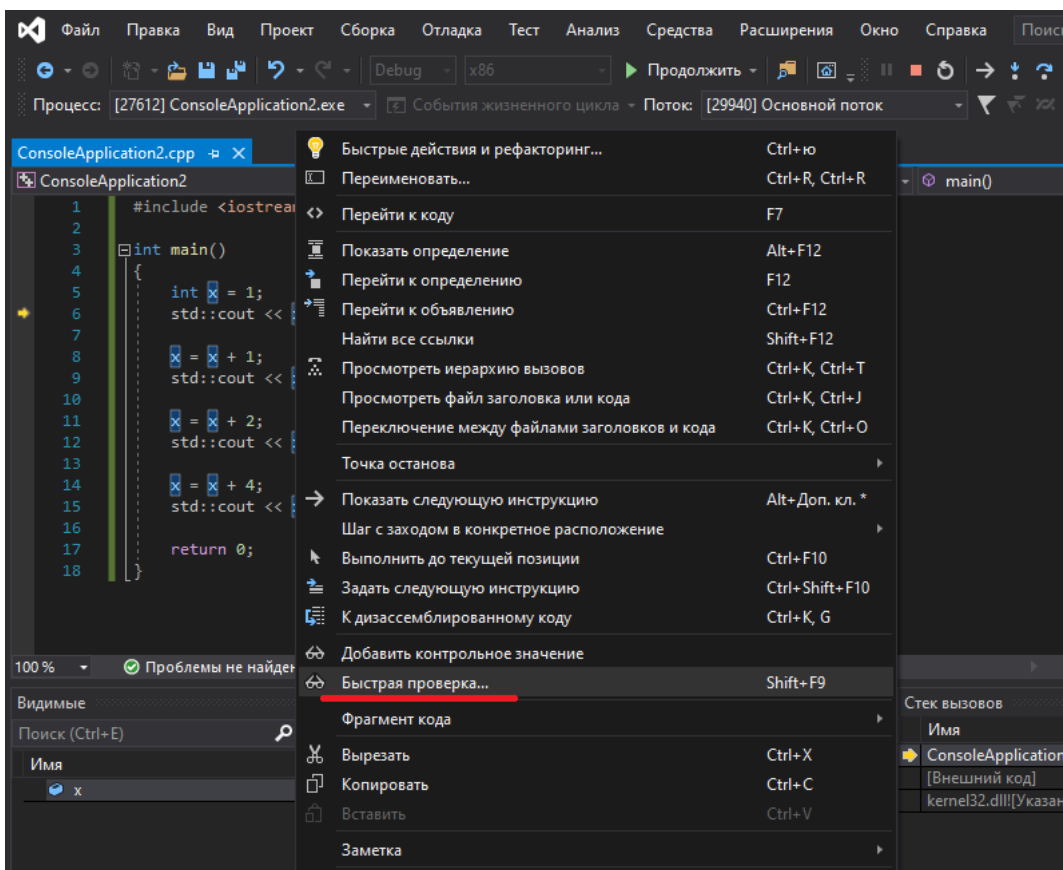
Самый простой способ отслеживания простых переменных (как `x`) — это наведение курсора мыши на элемент. Большинство современных отладчиков поддерживают эту возможность:



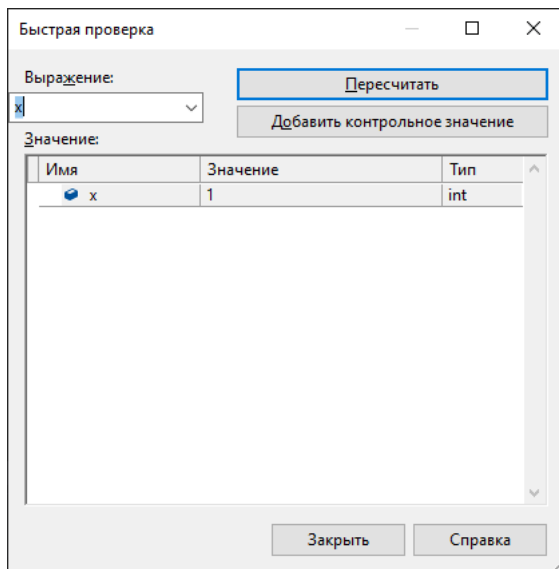
Обратите внимание, вы можете навести курсор мыши на любую другую переменную (и на любой строке):



В Visual Studio есть еще возможность использовать «Быструю проверку». Выделите переменную x с помощью мыши > ПКМ (правая кнопка мыши) > "Быстрая проверка...":

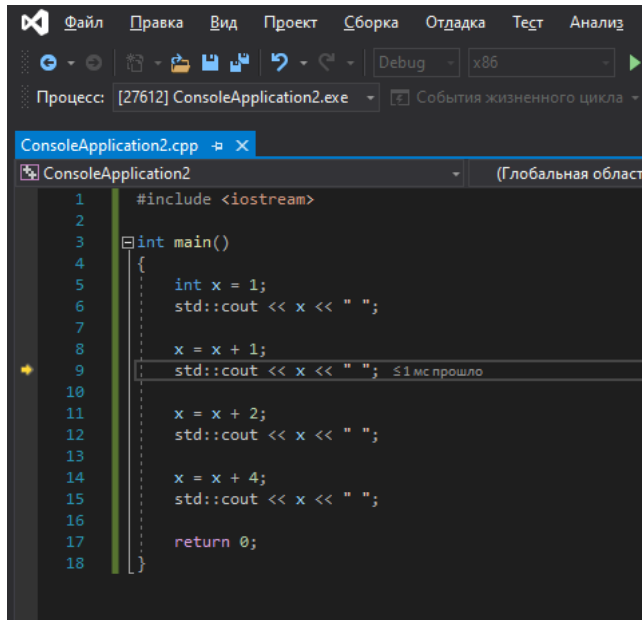


Появится специальное окно с текущим значением переменной:



Хорошо, теперь закройте это окно.

Значения переменных можно отслеживать и во время выполнения отладки. Переместитесь с помощью [команды «Шаг с обходом»](#) к строке `std::cout << x << " ";`:



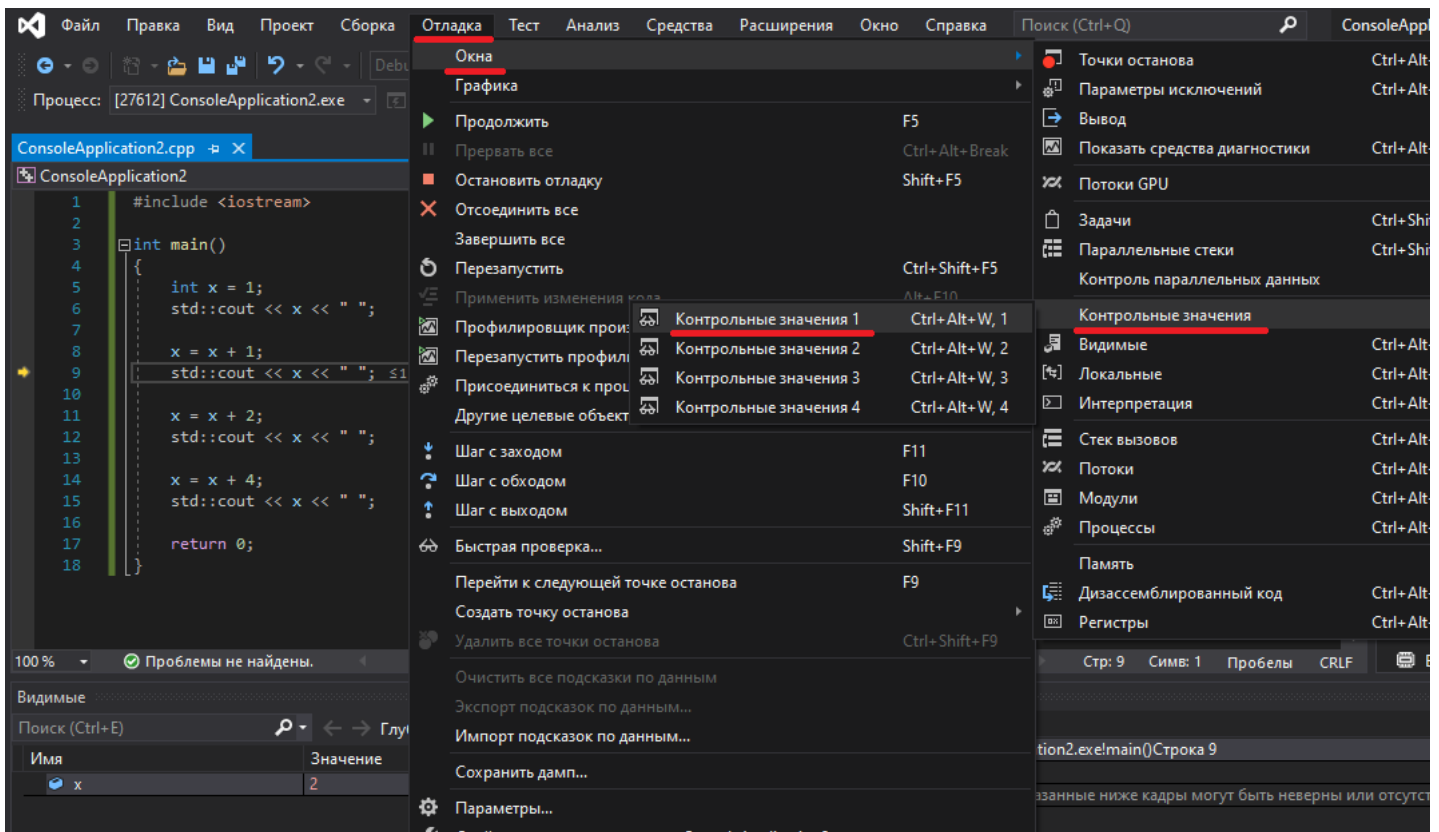
Значение переменной `x` должно поменяться на 2. Проверьте!

## Окно просмотра

Команда «Быстрая проверка» или наведение курсора на элемент подходят для статического просмотра переменных, но не очень подходят для отслеживания изменений переменной во время выполнения программы, так как с каждой новой выполненной строкой придется заново наводить курсор на элемент или использовать «Быструю проверку».

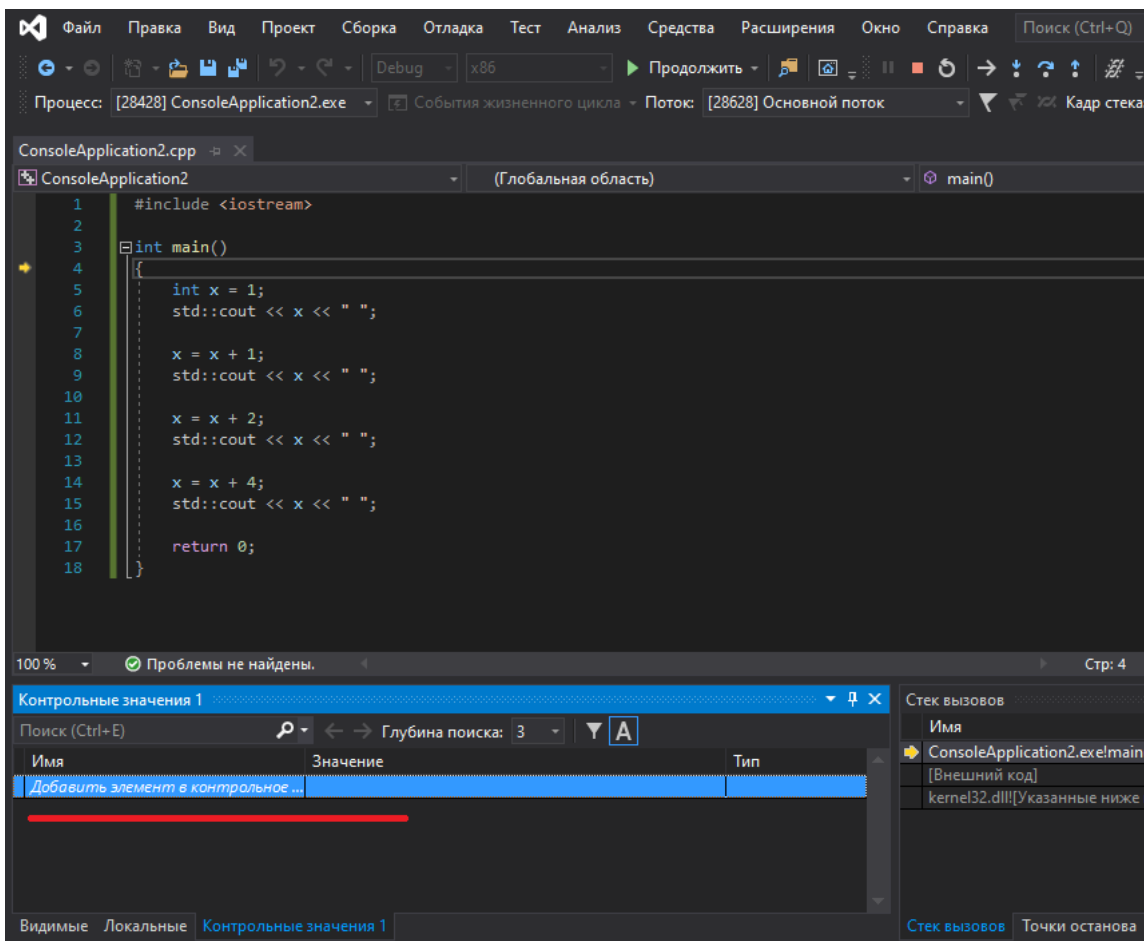
Для решения этой проблемы, все современные отладчики предлагают еще один инструмент — окно просмотра. **Окно просмотра** — это окно, в которое можно добавлять переменные для постоянного отслеживания. Данные переменные будут автоматически обновляться при последовательном выполнении программы. Окно просмотра уже может быть подключено и отображаться в вашей рабочей области, но если это не так, то вы можете это исправить, перейдя в настройки вашей [IDE](#).

В Visual Studio для отображения окна просмотра вам нужно перейти в "Отладка" > "Окна" > "Контрольные значения" > "Контрольные значения 1":



**Примечание:** Вы должны находиться в режиме отладки — используйте для этого [команду «Шаг с заходом»](#).

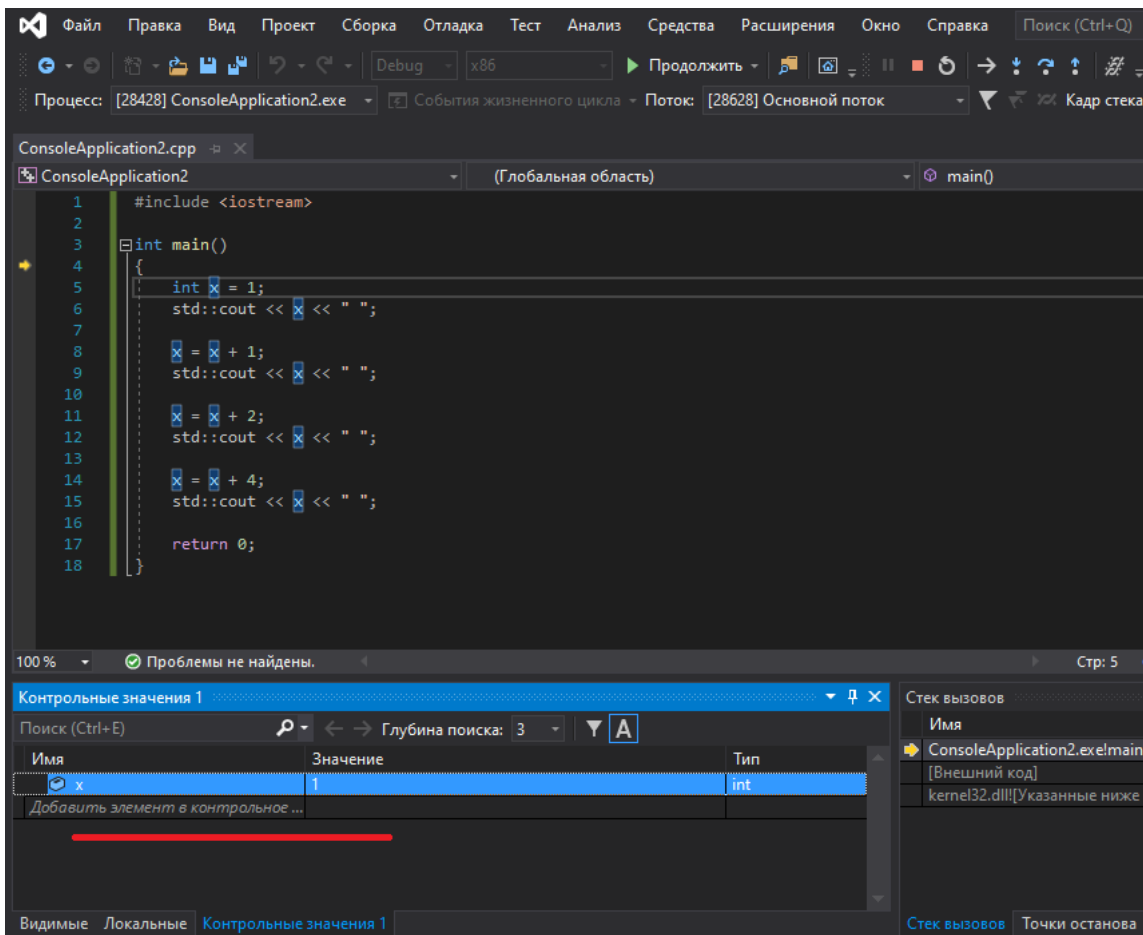
Вы должны увидеть следующее:



Пока что в этом окне ничего нет, так как вы еще ничего в него не добавили. Есть 2 пути:

- Ввести имя переменной, которую нужно отслеживать, в колонку "Имя" в окне просмотра.
- Выделить переменную, которую нужно отслеживать > ПКМ > "Добавить контрольное значение".

Попробуйте добавить переменную x в окно просмотра:



Теперь выберите команду «Шаг с обходом» несколько раз и следите за изменениями значения вашей переменной!

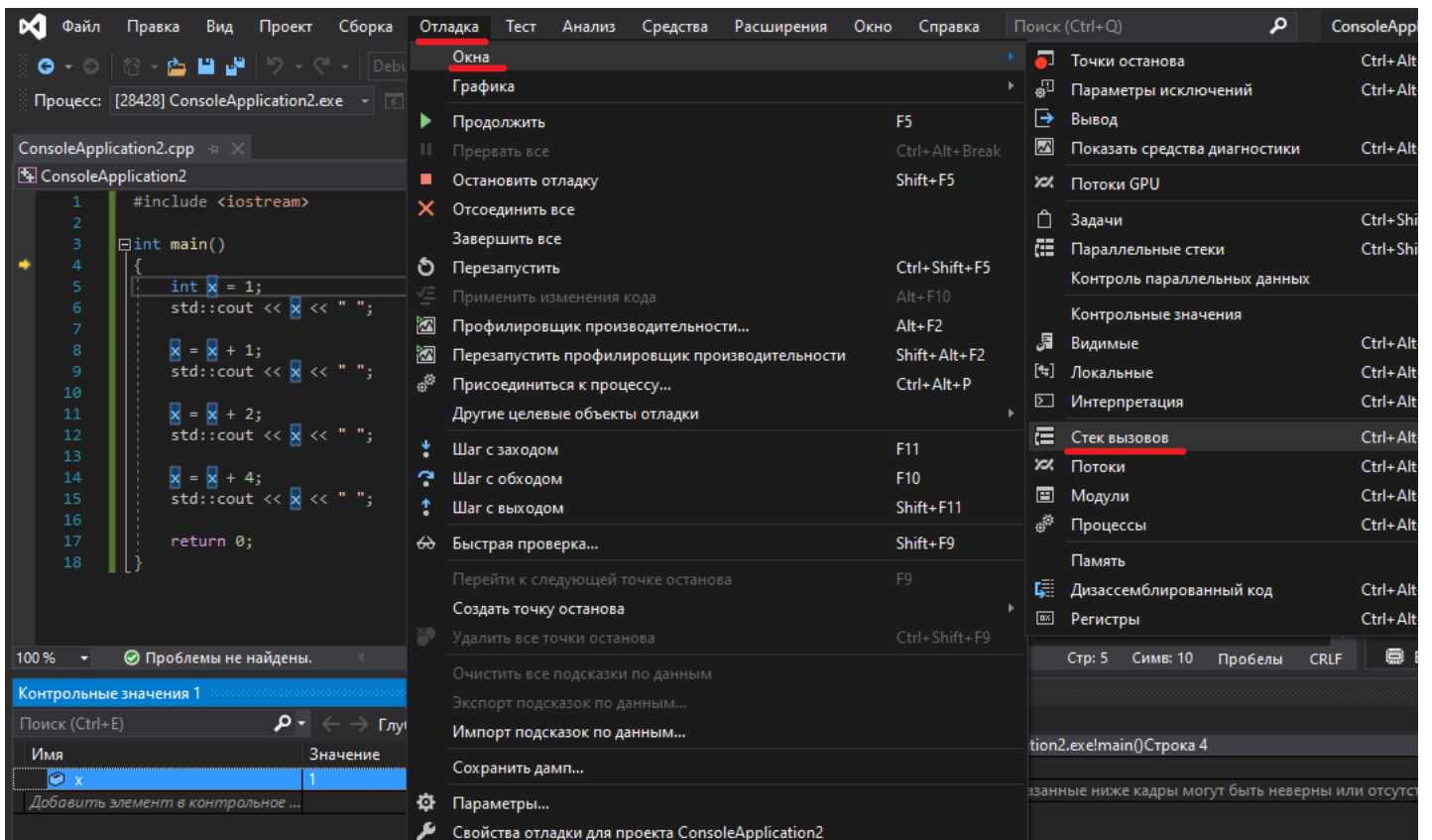
## Стек вызовов

Современные отладчики имеют еще одно информационное окно, которое может быть очень полезным при отладке программ — «Стек вызовов».

Как вы уже знаете, при вызове функции программа оставляет закладку в текущем местоположении, выполняет функцию, а затем возвращается в место закладки. Программа отслеживает все вызовы функций в стеке вызовов.

**Стек вызовов** — это список всех активных функций, которые вызывались до текущего местоположения. В стек вызовов записывается вызываемая функция и выполняемая строка. Всякий раз, когда происходит вызов новой функции, эта новая функция добавляется в верх стека. Когда выполнение текущей функции прекращается, она удаляется из верхней части стека и управление переходит к функции ниже (второй по счету).

Отобразить окно «Стека вызовов» в Visual Studio можно через "Отладка" > "Окна" > "Стек вызовов":



**Примечание:** Вы должны находиться в режиме отладки — используйте для этого команду «Шаг с заходом».

Рассмотрим пример:

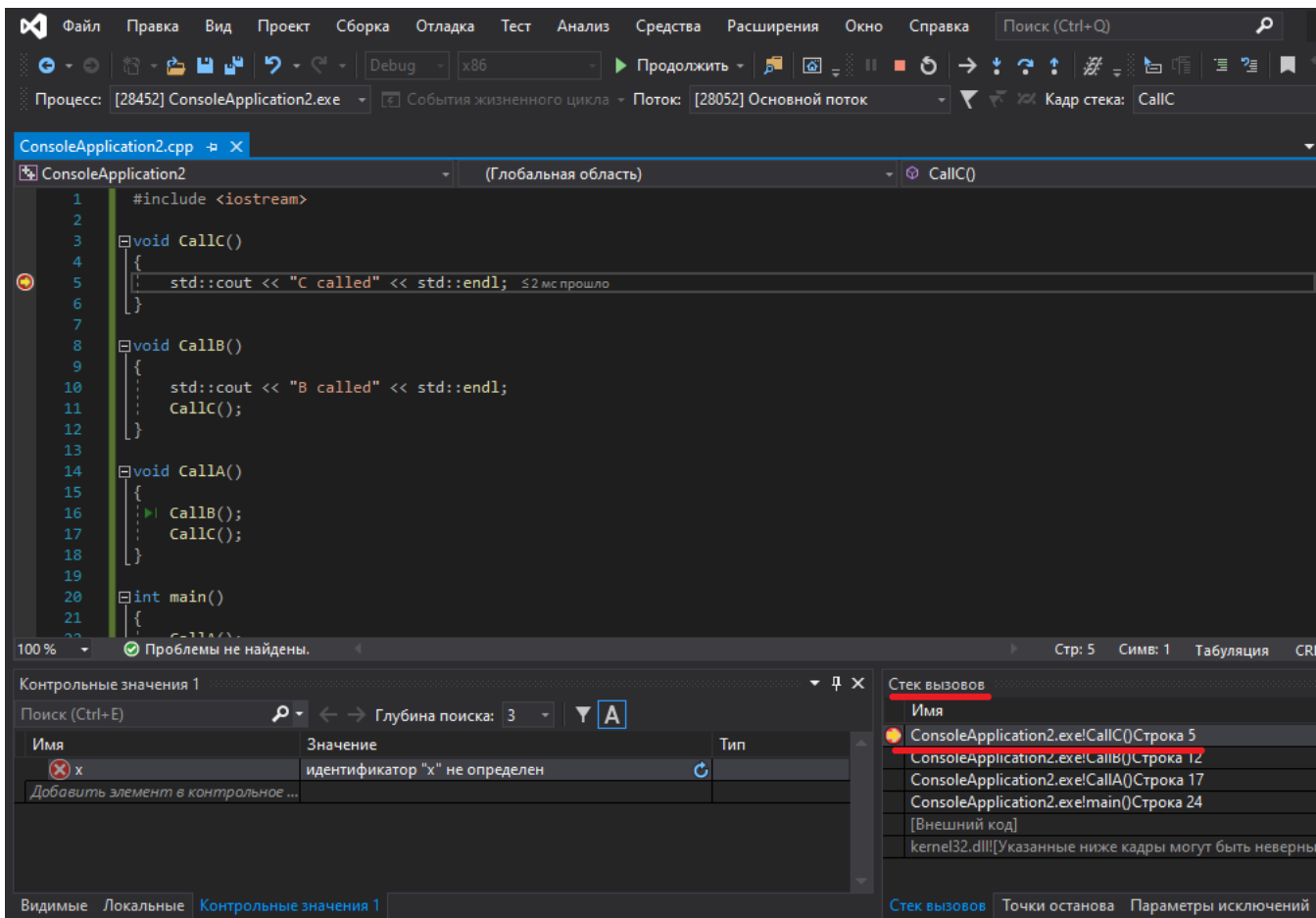
```

1  #include <iostream>
2
3  void CallC()
4  {
5      std::cout << "C called" << std::endl;
6  }
7
8  void CallB()
9  {
10     std::cout << "B called" << std::endl;
11     CallC();
12 }
13
14 void CallA()
15 {
16     CallB();
17     CallC();
18 }
19
20 int main()
21 {
22     CallA();
23
24     return 0;
25 }

```

Укажите точку останова в функции CallC(), а затем запустите отладку. Программа выполнится до точки останова.

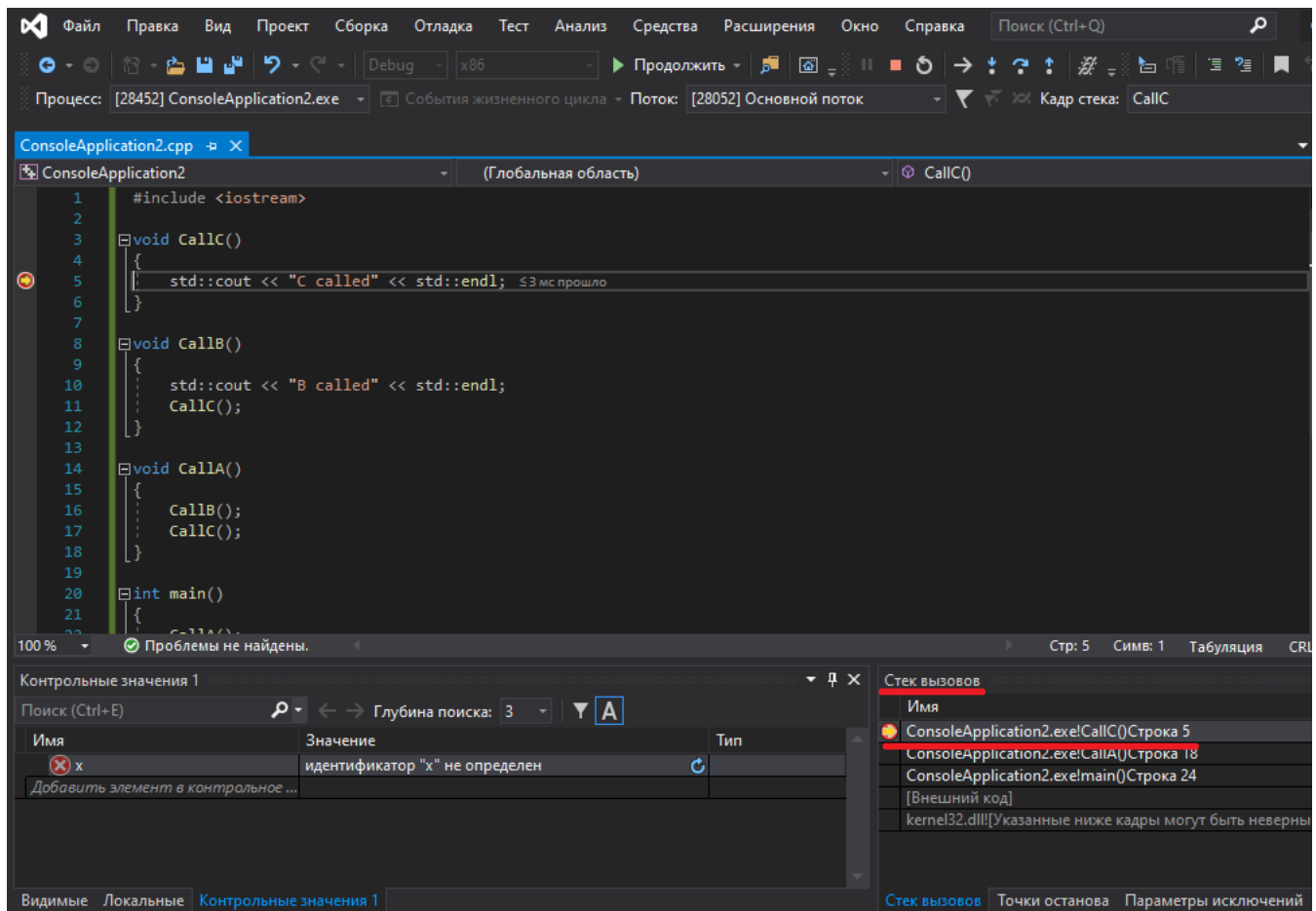
Несмотря на то, что вы знаете, что сейчас выполняется CallC(), в программе есть два вызова CallC(): в функции CallB() и в функции CallA(). Какая функция ответственна за вызов CallC() в данный момент? Стек вызовов нам это покажет:



Сначала выполняется `main()`. Затем `main()` вызывает `CallA()`, которая, в свою очередь, вызывает `CallB()`. Функция `CallB()` вызывает `CallC()`. Вы можете щелкнуть дважды по разным строкам в окне «Стек вызовов», чтобы увидеть больше информации о вызываемых функциях. Некоторые IDE переносят курсор непосредственно к вызову указанной функции. Visual Studio переносит курсор к следующей строке, которая находится после вызова функции. Попробуйте! Для того, чтобы возобновить степпинг, щелкните дважды по самой верхней (первой) строке в окне «Стек вызовов» и вы вернетесь к текущей точке выполнения.

Выберите команду «Продолжить». Точка останова должна сработать во второй раз, когда будет повторный вызов функции `CallC()` (на этот раз из функции `CallA()`). Всё происходящее вы должны увидеть в окне «Стек вызовов»:





## Заключение

Теперь вы знаете об основных возможностях встроенных отладчиков! Используя степпинг, точки останова, отслеживание переменных и окно «Стек вызовов» вы можете успешно проводить отладку программ.

Оценить статью:

★★★★★ (318 оценок, среднее: 4,91 из 5)



← [Урок №26. Отладка программ: степпинг и точки останова](#)

[Глава №1. Итоговый тест](#) →

## Комментариев: 12



1. [Александр:](#)  
[5 сентября 2020 в 13:38](#)

На самом деле в gdb все проще (как и в любой командной строке). Нужен стек вызовов — bt, подняться — up, нужна точка останова — b, посмотреть переменную — p и т.д. Т.е. десяток команд выписать, выучить, и хоть из серийной консоли удалённое приложение отлаживай ... зря всё так поверхностно в этом плане.

[Ответить](#)



2. [Юра:](#)  
[26 февраля 2020 в 16:44](#)

Автор похоже больше занимается разработкой, чем преподаёт т.к. в начале курса уделяет много внимания проблемам, которые появятся при разработке крупных проектов. Однако объясняет хорошо, по крайней мере мне всё понятно.

[Ответить](#)



3. *Сергей:*  
[20 марта 2019 в 19:08](#)

функция CallC() должна быть подписана в call stack 15-ой строкой, когда вызывается во второй раз?

[Ответить](#)



4. *Oleksiy:*  
[10 августа 2018 в 15:59](#)

"Поставьте breakpoint в функции CallC(), а затем перейдите в режим отладки."

Не понятно, где именно надо стабить breakpoint. Функция CallC() упоминается несколько раз в программе.

[Ответить](#)



1. *Юрий:*  
[21 августа 2018 в 21:40](#)

В этой функции:

```
1 void CallC()
2 {
3     std::cout << "C called" << std::endl;
4 }
```

[Ответить](#)



5. *Георгий:*  
[12 июля 2018 в 15:23](#)

при нажатии клавиши "F11" (step into) меня перебрасывает куда то совершенно в другой файл (не мой), называется "ostream", в чём проблема? (с "F10" (step over) такой проблемы нет)

[Ответить](#)



1. *Александр:*  
[30 января 2019 в 09:58](#)

Вас перебрасывает на выполнение оператора вывода в поток (<<)... видимо это следующая команда Вашего кода

[Ответить](#)



6. *painkiller:*  
[21 мая 2018 в 14:03](#)

Прочитал два урока об отладке, но, к сожалению, использую сейчас IDE DEV C++, а в ней проблемы с отладчиком. Подумал, что это у меня какие-то проблемы, но зашел на специализированные форумы и выяснилось, что такая проблема у многих. В этой среде отладчик очень слабый и нужно к ней подключать отладчики сторонних разработчиков.

Решил для себя пользоваться этой средой до той поры, когда без отладчика будет очень еще терпимо, но позже обязательно перейду на более мощную среду.

Написал это для тех, кто имеет эту же среду.

[Ответить](#)



1. *Александр:*  
[30 января 2019 в 10:00](#)

При всем удобстве дебаггеров, очень часто (особенно несложный код) удобней отлаживать выводом данных... просто расставьте в своем коде вывод (например в лог-файл) интересующих Вас в данном куске кода параметров и вуаля...

[Ответить](#)



1. *Константин:*  
[9 марта 2019 в 10:48](#)

Александр, а можешь по-шагово расписать "просто расставьте в своем коде вывод (например в лог-файл) интересующих Вас в данном куске кода параметров и вуаля..." как это сделать?

[Ответить](#)



7. *Виталий:*  
[10 апреля 2018 в 06:53](#)

"Обратите внимание, что переменные, которые выходят из области видимости (например, объявленные в функции, когда функция не выполняется) будут находиться в окне просмотра."

Наверное, имелось в виду, что переменные НЕ будут находиться в окне просмотра.

[Ответить](#)



1. Юрий:

[10 апреля 2018 в 16:15](#)

Переменные, которые выходят из области видимости будут находиться в окне просмотра. Если переменные возвращаются в область видимости (например, функция вызывается повторно), в окне появятся ЗНАЧЕНИЯ этих переменных.

То есть переменные остаются, только их значения могут либо пропадать, либо появляться.

[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

Электронная почта



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