

Ravesli [Ravesli](#)

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



Урок №26. Отладка программ: stepping и точки останова

👤 [Юрий](#) |

- [Уроки C++](#)

|

🕒 Обновл. 2 Сен 2020 |

👁 40779

[16](#)

Как ни странно, программирование может быть сложным и ошибок может быть очень много. Ошибки, как правило, попадают в одну из двух категорий: синтаксические или семантические/смысловые.

Оглавление:

1. [Типы ошибок](#)
2. [Отладчик](#)
3. [Stepping](#)
 4. [Команда «Шаг с заходом»](#)
 5. [Команда «Шаг с обходом»](#)
 6. [Команда «Шаг с выходом»](#)
7. [Команда «Выполнить до текущей позиции»](#)
8. [Команда «Продолжить»](#)
9. [Точки останова](#)

Типы ошибок

Синтаксическая ошибка возникает, когда вы пишете код, который не соответствует правилам грамматики языка C++. Например, пропущенные точки с запятой, необъявленные переменные, непарные крупные или фигурные скобки и т.д. В следующей программе есть несколько синтаксических ошибок:

```
1 #include <iostream>; // директивы препроцессора не заканчиваются точкой с запятой
2
3 int main()
4 {
5     std::cout < "Hi there; << x; // недействительный оператор (:), незаконченное предложение (пропущено ") и необъявленная переменная x
6     return 0 // пропущена точка с запятой в конце стейтмента
7 }
```

К счастью, компилятор ловит подобные ошибки и сообщает о них в виде предупреждений или ошибок.

Семантическая ошибка возникает, когда код является синтаксически правильным, но делает не то, что задумал программист.

Иногда это может привести к сбою в программе, например, если делить на ноль:

```
1 #include <iostream>
2
3 int main()
4 {
5     int a = 10;
6     int b = 0;
7     std::cout << a << " / " << b << " = " << a / b; // делить на 0 нельзя
8     return 0;
9 }
```

Иногда это может привести к неверным результатам:

```
1 #include <iostream>
2
3 int main()
4 {
5     std::cout << "Hello, word!"; // орфографическая ошибка
```

```
6 |     return 0;  
7 | }
```

Либо делать вообще не то, что нужно:

```
1 | #include <iostream>  
2 |  
3 | int add(int x, int y)  
4 | {  
5 |     return x - y; // функция должна выполнять сложение, но выполняет вычитание  
6 | }  
7 |  
8 | int main()  
9 | {  
10 |     std::cout << add(5, 3); // должно быть 8, но результат - 2  
11 |     return 0;  
12 | }
```

К сожалению, компилятор не ловит подобные ошибки, так как он проверяет только то, что вы написали, а не то, что вы хотели этим сделать.

В примерах, приведенных выше, ошибки довольно легко обнаружить. Но в большинстве программ (в которых больше 40 строк кода), семантические ошибки увидеть с помощью простого просмотра кода будет не так-то и легко.

И здесь нам на помощь приходит отладчик.

Отладчик

Отладчик (или «*дебаггер*», от англ. «*debugger*») — это компьютерная программа, которая позволяет программисту контролировать выполнение кода. Например, программист может использовать отладчик для выполнения программы пошагово, последовательно изучая значения переменных в программе.

Более ранние дебаггеры, такие как [GDB](#), имели интерфейс командной строки, где программисту приходилось вводить специальные команды для старта работы. Более современные дебаггеры уже имеют «графический» интерфейс, что значительно упрощает работу с ними. Сейчас почти все современные [IDE](#) имеют встроенные отладчики. То есть, вы можете использовать одну среду разработки как для написания кода, так и для его отладки (вместо постоянного переключения между разными программами).

Базовый функционал у всех отладчиков один и тот же. Отличаются они, как правило, тем, как этот функционал и доступ к нему организованы, горячими клавишами и дополнительными возможностями.

Примечание: Перед тем как продолжить, убедитесь, что вы находитесь в [режиме конфигурации «Debug»](#). Все скриншоты данного урока выполнены в Visual Studio 2019.

Стейпинг

Стейпинг (англ. «*stepping*») — это возможность отладчика выполнять код пошагово (строка за строкой). Есть три команды стейпинга:

- Команда "Шаг с заходом"
- Команда "Шаг с обходом"
- Команда "Шаг с выходом"

Мы сейчас рассмотрим каждую из этих команд в индивидуальном порядке.

Команда «Шаг с заходом»

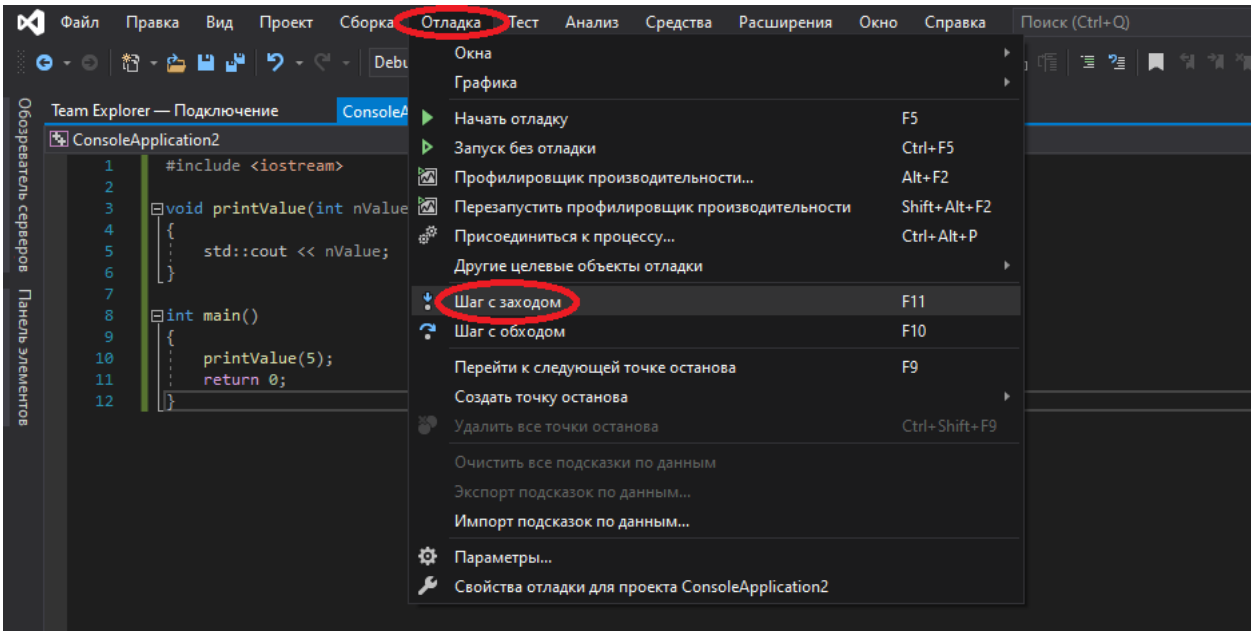
Команда «**Шаг с заходом**» (англ. «*Step into*») выполняет следующую строку кода. Если этой строкой является вызов функции, то «Шаг с заходом» открывает функцию и выполнение переносится в начало этой функции.

Давайте рассмотрим очень простую программу:

```
1 | #include <iostream>  
2 |  
3 | void printValue(int nValue)  
4 | {  
5 |     std::cout << nValue;  
6 | }  
7 |  
8 | int main()  
9 | {  
10 |     printValue(5);  
11 |     return 0;  
12 | }
```

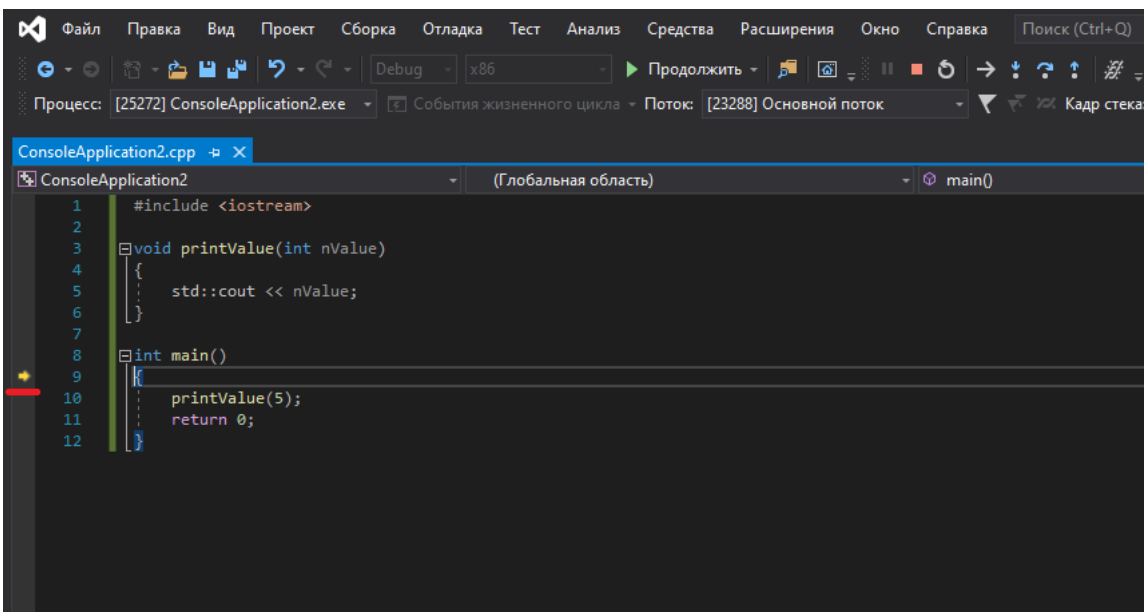
Как вы уже знаете, при запуске программы выполнение начинается с вызова главной функции `main()`. Так как мы хотим выполнить отладку внутри функции `main()`, то давайте начнем с использования команды «Шаг с заходом».

В Visual Studio, перейдите в меню "Отладка" > "Шаг с заходом" (либо нажмите F11):

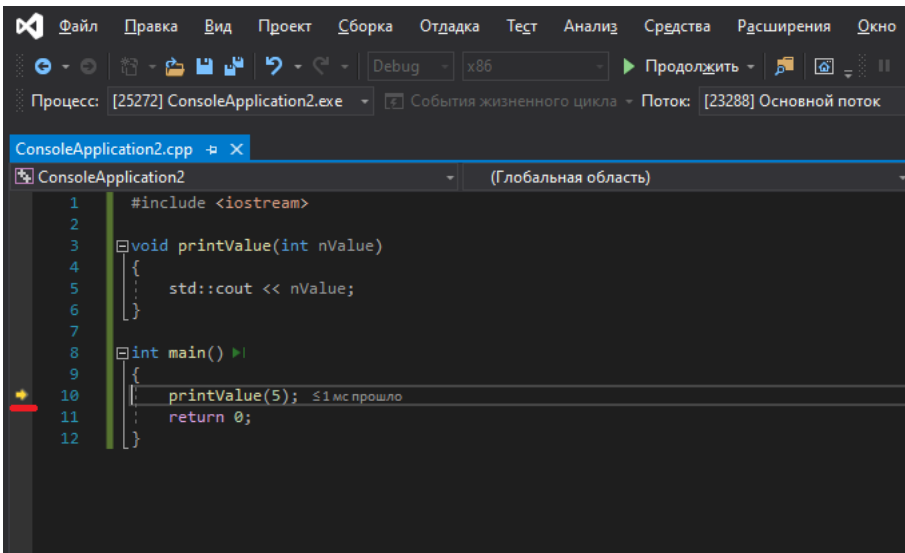


Если вы используете другую IDE, то найдите в меню команду "Step Into/Шаг с заходом" и выберите её.

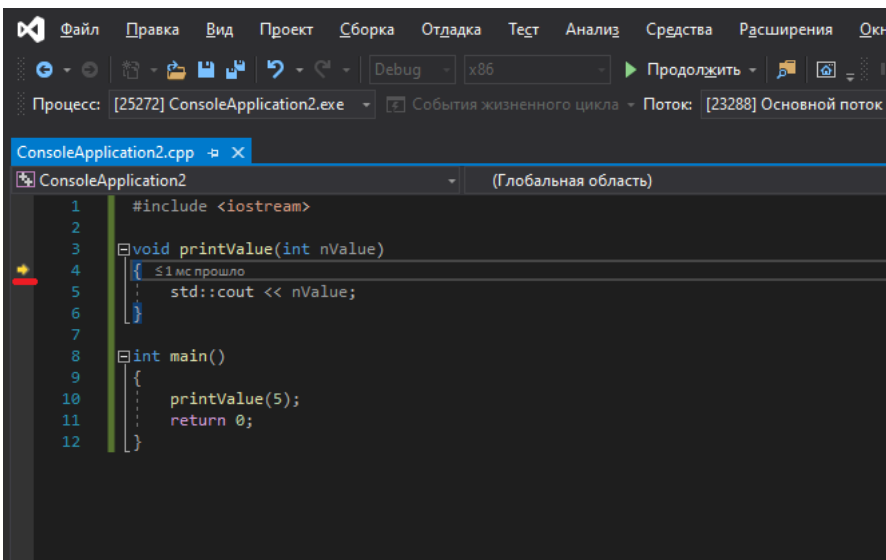
Когда вы это сделаете, должны произойти две вещи. Во-первых, так как наше приложение является консольной программой, то должно открыться консольное окно. Оно будет пустым, так как мы еще ничего не выводили. Во-вторых, вы должны увидеть специальный маркер слева возле открывающей скобки функции main(). В Visual Studio этим маркером является жёлтая стрелочка (если вы используете другую IDE, то должно появиться что-нибудь похожее):



Стрелка-маркер указывает на следующую строку, которая будет выполняться. В этом случае отладчик говорит нам, что следующей строкой, которая будет выполняться, — будет открывающая фигурная скобка функции main(). Выберите «Шаг с заходом» еще раз — стрелка переместится на следующую строку:



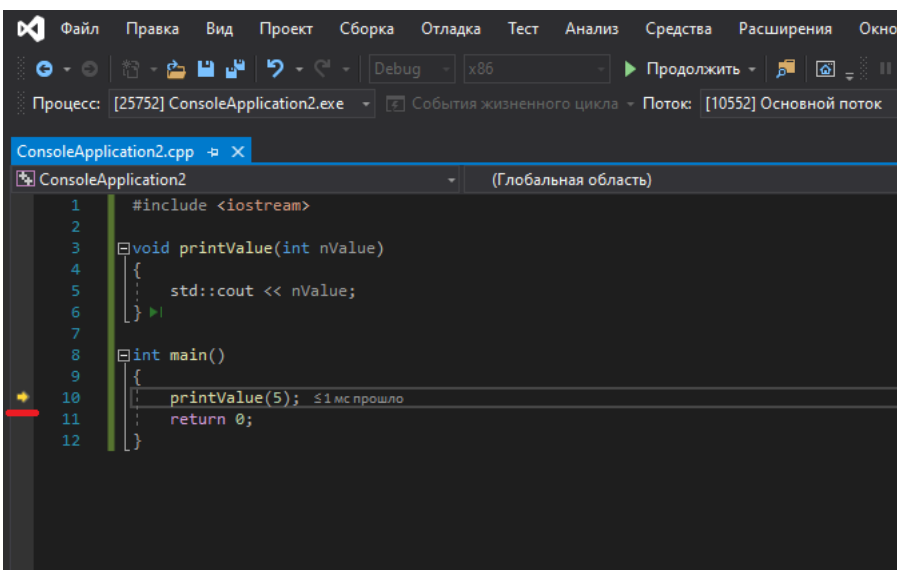
Это значит, что следующей строкой, которая будет выполняться, — будет вызов функции `printValue()`. Выберите «Шаг с заходом» еще раз. Поскольку `printValue()` — это вызов функции, то мы переместимся в начало функции `printValue()`:



Выберите еще раз «Шаг с заходом» для выполнения открывающей фигурной скобки `printValue()`. Стрелка будет указывать на `std::cout << nValue;`.

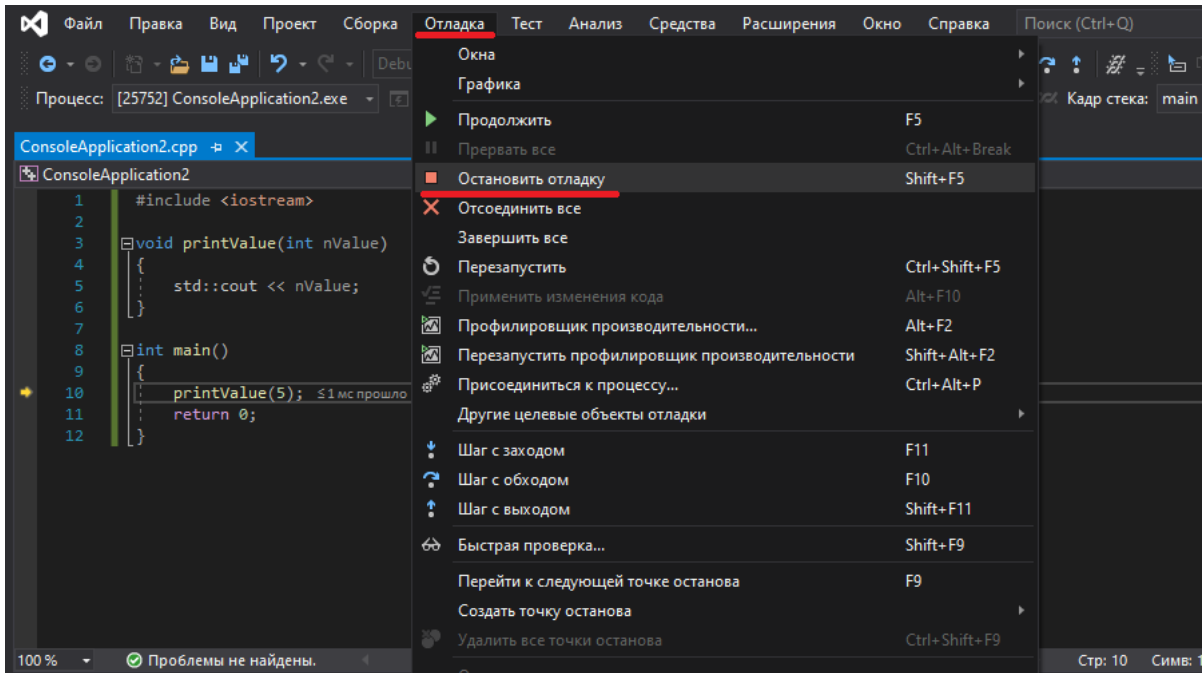
Теперь выберите «Шаг с обходом» (F10). Вы увидите число 5 в консольном окне.

Выберите «Шаг с заходом» еще раз для выполнения закрывающей фигурной скобки `printValue()`. Функция `printValue()` завершит свое выполнение и стрелка переместится в функцию `main()`. Обратите внимание, в `main()` стрелка снова будет указывать на вызов `printValue()`:



Может показаться, будто отладчик намеревается еще раз повторить цикл с функцией `printValue()`, но в действительности он нам просто сообщает, что он только что вернулся из этой функции.

Выберите «Шаг с заходом» два раза. Готово, все строки кода выполнены. Некоторые дебаггеры автоматически прекращают сеанс отладки в этой точке. Но Visual Studio так не делает, так что если вы используете Visual Studio, то выберите "Отладка" > "Остановить отладку" (или Shift+F5):



Таким образом мы полностью остановили сеанс отладки нашей программы.

Команда «Шаг с обходом»

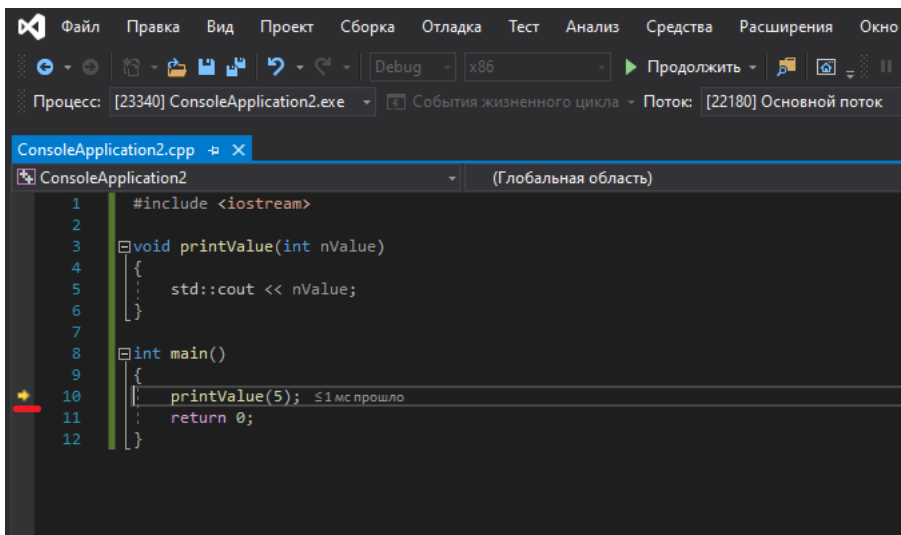
Как и команда «Шаг с заходом», команда «Шаг с обходом» (англ. «*Step over*») позволяет выполнить следующую строку кода. Только если этой строкой является вызов функции, то «Шаг с обходом» выполнит весь код функции в одно нажатие и возвратит нам контроль после того, как функция будет выполнена.

Примечание для пользователей Code::Blocks: Команда «Step over» называется «Next Line».

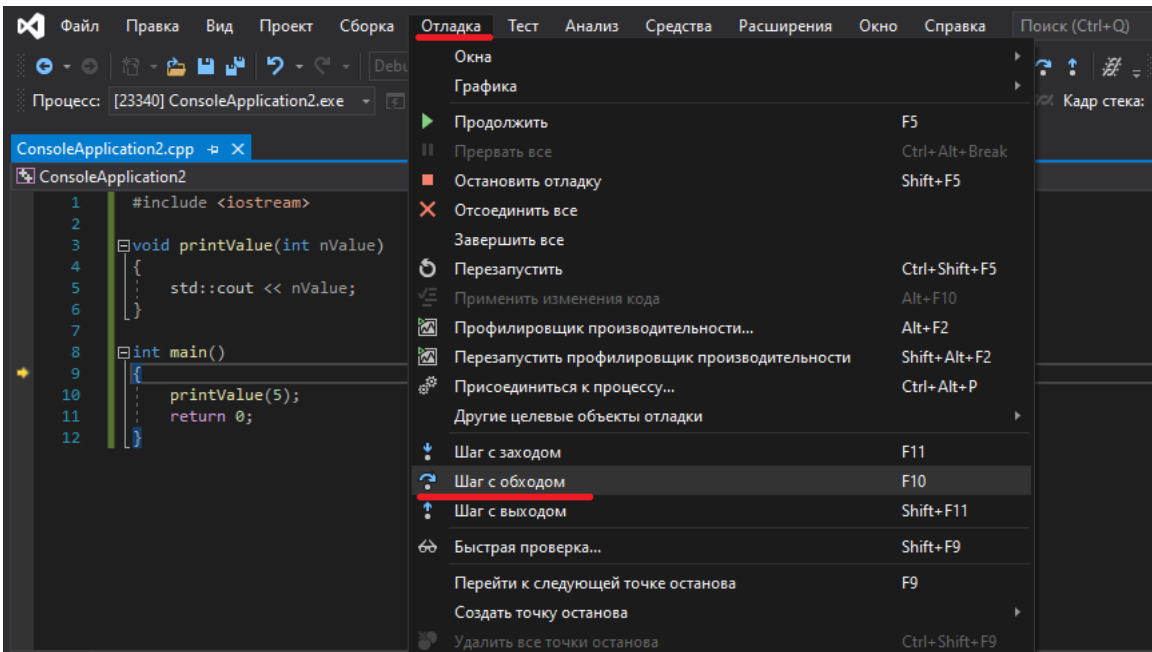
Рассмотрим пример, используя следующую программу:

```
1 #include <iostream>
2
3 void printValue(int nValue)
4 {
5     std::cout << nValue;
6 }
7
8 int main()
9 {
10    printValue(5);
11    return 0;
12 }
```

Нажмите «Шаг с заходом», чтобы дойти до вызова функции printValue():



Теперь вместо команды «Шаг с заходом» выберите «Шаг с обходом» (или F10):



Отладчик выполнит функцию (которая выведет значение 5 в консоль), а затем возвратит нам управление на строке `return 0;`. И это всё за одно нажатие.

Команда «Шаг с обходом» позволяет быстро пропустить код функций, когда мы уверены, что они работают корректно и их не нужно отлаживать.

Команда «Шаг с выходом»

В отличие от двух предыдущих команд, команда «**Шаг с выходом**» (англ. «*Step out*») не просто выполняет следующую строку кода. Она выполняет весь оставшийся код функции, в которой вы сейчас находитесь, и возвращает контроль только после того, когда функция завершит свое выполнение. Проще говоря, «Шаг с выходом» позволяет выйти из функции.

Обратите внимание, команда «Шаг с выходом» появится в меню «Отладка» только после начала сеанса отладки (что делается путем использования одной из двух вышеприведенных команд).

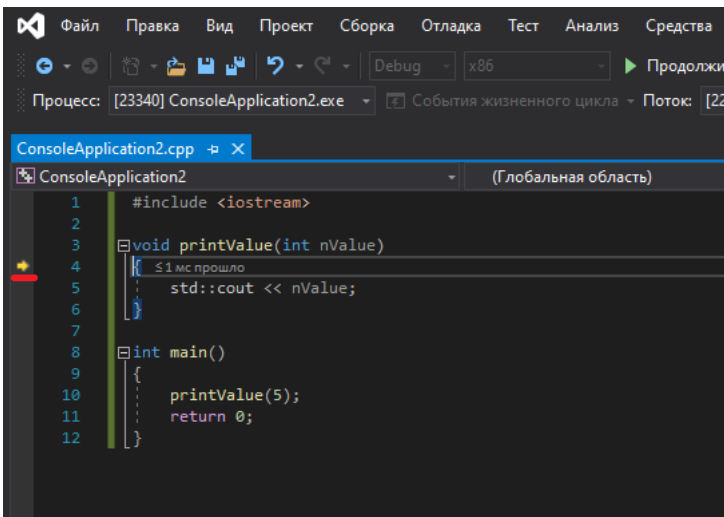
Рассмотрим все тот же пример:

```

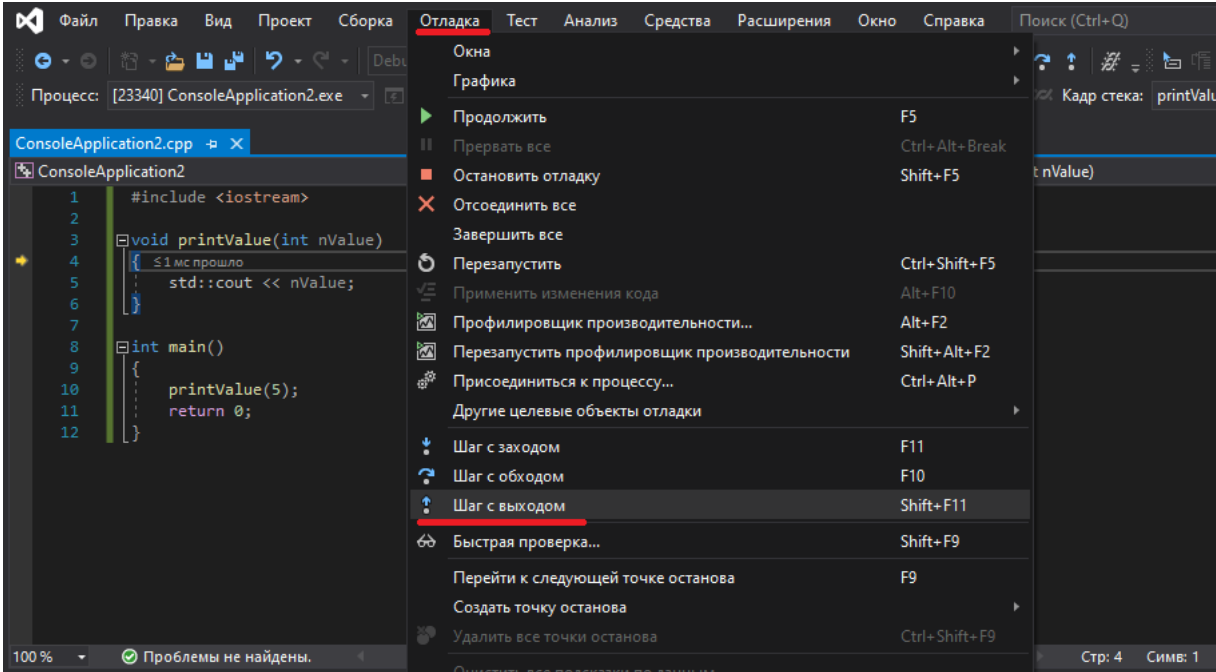
1 #include <iostream>
2
3 void printValue(int nValue)
4 {
5     std::cout << nValue;
6 }
7
8 int main()
9 {
10     printValue(5);
11     return 0;
12 }

```

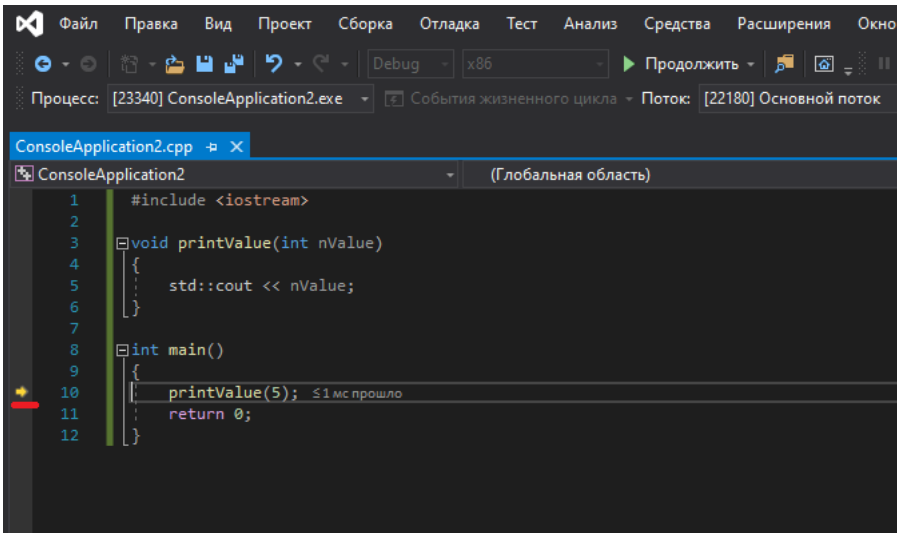
Нажимайте «Шаг с заходом» до тех пор, пока не перейдете к открывающей фигурной скобке функции `printValue()`:



Затем выберите "Отладка" > "Шаг с выходом" (либо Shift+F11):



Вы заметите, что значение 5 отобразилось в консольном окне, а отладчик перешел к вызову функции printValue() в main():



Команда «Выполнить до текущей позиции»

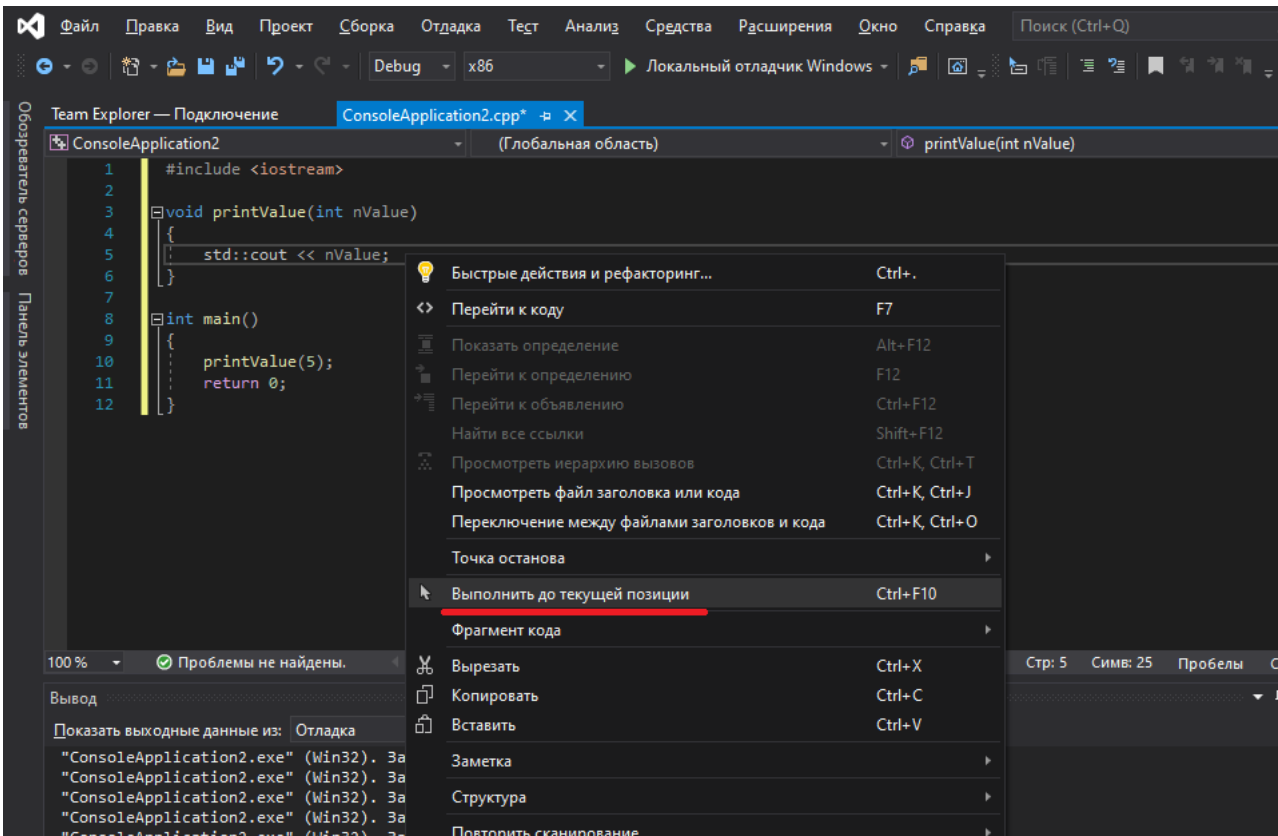
В то время как стейпинг полезен для изучения каждой строки кода по отдельности, в большой программе перемещаться по коду с помощью этих команд будет не очень удобно.

Но и здесь современные отладчики предлагают еще несколько инструментов для эффективной отладки программ.

Команда «Выполнить до текущей позиции» позволяет в одно нажатие выполнить весь код до строки, обозначенной курсором. Затем контроль обратно возвращается к вам, и вы можете проводить отладку с указанной точки уже более детально. Давайте попробуем, используя уже знакомую нам программу:



Поместите курсор на строку `std::cout << nValue;` внутри функции `printValue()`, затем щелкните правой кнопкой мыши и выберите "Выполнить до текущей позиции" (либо Ctrl+F10):

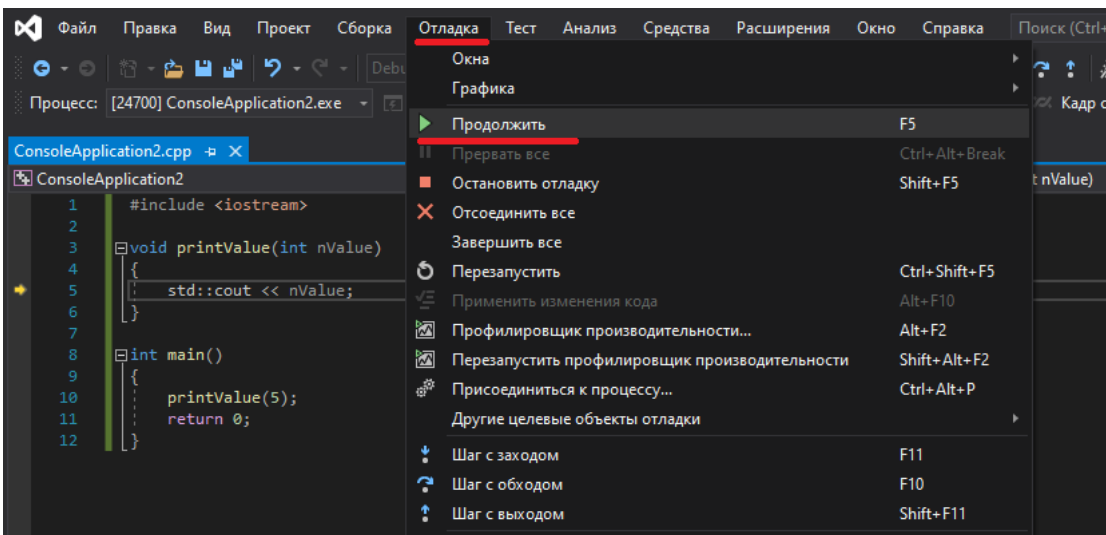


Вы заметите, что жёлтая стрелочка переместится на указанную вами строку. Выполнение программы остановится в этой точке, и программа будет ждать ваших дальнейших команд.

Команда «Продолжить»

Если вы находитесь в середине сеанса отладки вашей программы, то вы можете сообщить отладчику продолжать выполнение кода до тех пор, пока он не дойдет до конца программы (или до следующей контрольной точки). В Visual Studio эта команда называется «**Продолжить**» (англ. «*Continue*»). В других дебаггерах она может иметь название «Run» или «Go».

Возвращаясь к вышеприведенному примеру, мы находимся как раз внутри функции `printValue()`. Выберите "Отладка" > "Продолжить" (или F5):



Программа завершит свое выполнение и выйдет из сеанса отладки.

Точки останова

Точки останова (англ. «*breakpoints*») — это специальные маркеры, на которых отладчик останавливает процесс выполнения программы.

Чтобы задать точку останова в Visual Studio, щелкните правой кнопкой мыши по выбранной строке > "Точка останова" > "Вставить точку останова":

Появится кружочек возле строки:

В программе, приведенной выше, создайте точку останова на строке `std::cout << nValue;`. Затем выберите «Шаг с заходом» для старта сеанса отладки, а затем «Продолжить». Вы увидите, что вместо завершения выполнения программы и остановки сеанса отладки, отладчик остановится в указанной вами точке:

Точки останова чрезвычайно полезны, если вы хотите изучить только определенную часть кода. Просто задайте точку останова в выбранном участке кода, выберите команду «Продолжить» и отладчик автоматически остановится возле указанной строки. Затем вы сможете использовать команды stepping для более детального просмотра/изучения кода.

Оценить статью:

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



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

[Урок №27. Отладка программ: стек вызовов и отслеживание переменных](#)



Комментариев: 16



1. *Данил:*
[3 декабря 2019 в 18:44](#)

Помогите! я работаю в Dev C++ там отладчик вызывается по другому (а как? я не знаю) и он не работает только консоль появляется и все!
И почему во многих уроках вы практически не объясняете по Dev c++,
ну например урок #4 почему там есть все а Дес c++ нет?
А так все очень доступно и понятно объясняете в остальных уроках)

[Ответить](#)



2. *Владимир:*
[14 ноября 2019 в 18:12](#)

Еще раз поблагодарю. Классный материал и перевод! Тоже склоняюсь к тому что скорее всего куплю PDF — ваш труд более чем стоит этого.

[Ответить](#)



1. *Юрий:*
[14 ноября 2019 в 22:52](#)

Спасибо за отзыв, мне приятно 😊

[Ответить](#)



3. *Константин:*
[16 октября 2019 в 19:10](#)

Лютый респект Вам, Юрий! Изучаю кресты по вашим урокам, обязательно дойду до финала. (Задумался о покупке PDF версии)

[Ответить](#)



1. *Юрий:*
[16 октября 2019 в 21:31](#)

Огромное пожалуйста 😊 Мне приятно!

[Ответить](#)



4. *Кекс:*
[8 августа 2019 в 10:36](#)

Привет! Почему порой шаги с заходом и обходом не работают, а просто компилируется вся программа? Стоит режим debug

[Ответить](#)



5. *Анастасия:*
[10 мая 2019 в 17:22](#)

Хотелось бы узнать, как выполнить эти действия в других средах. У меня DEV C++ и там ничего подобного не наблюдаю.
P.S. VS у меня тоже есть, но не могу найти иконку к ней, приходится заходить через установщик, это очень долго. Поэтому использую DEV C++.
Буду очень признательна, если кто-то подскажет, как найти иконку VS, чтобы вынести её на рабочий стол.

[Ответить](#)



1. *Илья:*

[22 июля 2019 в 12:22](#)

1. В нижней левой части экрана жмёшь на иконку "ПУСК" (значок винды)
- 2.1. Если у тебя windows 10:
пишешь фразу "Vis" и выбираешь из того, что появится Visual Studio
- 2.2. Если у тебя не 10-ка:
Ищешь вкладку поиск и там пишешь фразу "Vis" и выбираешь из того, что появится Visual Studio
3. Когда запустишь VS, внизу на панели задач будет её иконка.
Кликаешь на эту иконку правой кнопкой мыши
Выбираешь пункт что-то типа "Закрепить на панели задач"

[Ответить](#)



6. *Виталий:*

[25 ноября 2018 в 14:00](#)

Возможно ли, и как осуществлять отладку dll-проектов?

[Ответить](#)



7. *Николай:*

[13 июля 2018 в 22:40](#)

Не плохо было бы добавить как подключать отладчик, у меня почему то не прописан был путь к нему Code::Blocks . Пришлось гуглить. В Setting-> debugger -> Default->Executable path... указать путь нужного нам отладчика лежит в папке компилятора CodeBlocks\MinGW\bin\gdb32.exe

[Ответить](#)



1. *Илья:*

[22 июля 2019 в 12:24](#)

Спасибо, я использую и VS и Code::Blocks для разных задач и дебаггера там серьёзно не хватало.

[Ответить](#)



8. *Георгий:*

[12 июля 2018 в 15:06](#)

Мне кажется, стоило показать пример на более масштабной программе, а то, я думаю, многим не понятно, как это может пригодиться.

[Ответить](#)



1. *shaslataushang:*

[11 июня 2019 в 21:35](#)

Мне пригодилось. Там дальше будет урок по вложенным циклам. И у меня одно задание никак не хотело выполняться правильно, и я не мог понять в чём причина. Начал отладку, шаг за шагом следил что и как происходит. Я выявил все причины, программа начала работать именно так, как я и задумывал. Уже третий раз прохожу курс, ибо жизненные обстоятельства то и дело принуждают его забросить... Но я его закончу!

[Ответить](#)



1. *Никита:*

[17 июня 2019 в 18:06](#)

Заканчиваешь?

[Ответить](#)



9. *Александр:*

[4 июля 2018 в 15:36](#)

Ни одного комментария. А тема в программировании самая важная.

Без отладчика — как без рук.

joxi.ru/zAN47oWUBwgkPm

Вот пример нашего отладчика. Вроде горячие клавиши совпадают
=))

[Ответить](#)



1. *Юрий:*

[10 июля 2018 в 00:26](#)

Это 1С? Жестко 😊

[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