

Ravesli [Ravesli](#)


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


## Урок №12. Функции и оператор возврата return

 [Юрий](#) |

- [Уроки C++](#)

|

 Обновл. 2 Сен 2020 |

 129497

[↑](#)  36

Вы уже знаете, что каждая программа должна содержать функцию `main()` (с которой она и начинает свое выполнение). Тем не менее, большинство программ используют и много других функций.

Оглавление:

1. [Функции](#)
2. [Возвращаемые значения](#)
3. [Тип возврата void](#)
4. [Возврат значений обратно в функцию main\(\)](#)
5. [Еще о возвращаемых значениях](#)
6. [Повторное использование функций](#)
7. [Вложенные функции](#)
8. [Тест](#)
9. [Ответы](#)

## Функции

**Функция** — это последовательность действий для выполнения определенного задания. Часто ваши программы будут прерывать выполнение одних функций ради выполнения других. Вы делаете аналогичные вещи в реальной жизни постоянно. Например, вы читаете книгу и вспомнили, что должны были сделать телефонный звонок. Вы оставляете закладку в своей книге, берете телефон и набираете номер. После того, как вы уже поговорили, вы возвращаетесь к чтению: к той странице, на которой остановились.

Программы на языке C++ работают похожим образом. Иногда, когда программа выполняет код, она может столкнуться с вызовом функции. **Вызов функции** — это выражение, которое указывает процессору прервать выполнение текущей функции и приступить к выполнению другой функции. Процессор «оставляет закладку» в текущей точке выполнения, а затем выполняет вызываемую

функцию. Когда выполнение вызываемой функции завершено, процессор возвращается к *закладке* и возобновляет выполнение прерванной функции.

Функция, в которой находится вызов, называется **caller**, а функция, которую вызывают — **вызываемая функция**, например:

```
1 #include <iostream> // для std::cout и std::endl
2
3 // Объявление функции doPrint(), которую мы будем вызывать
4 void doPrint() {
5     std::cout << "In doPrint()" << std::endl;
6 }
7
8 // Объявление функции main()
9 int main()
10 {
11     std::cout << "Starting main()" << std::endl;
12     doPrint(); // прерываем выполнение функции main() вызовом функции doPrint(). Функ
13     std::cout << "Ending main()" << std::endl;
14     return 0;
15 }
```

Результат выполнения программы:

```
Starting main()
In doPrint()
Ending main()
```

Эта программа начинает выполнение с первой строки функции `main()`, где выводится на экран следующая строка: `Starting main()`. Вторая строка функции `main()` вызывает функцию `doPrint()`. На этом этапе выполнение стейтментов в функции `main()` приостанавливается и процессор переходит к выполнению стейтментов внутри функции `doPrint()`. Первая (и единственная) строка в `doPrint()` выводит текст `In doPrint()`. Когда процессор завершает выполнение `doPrint()`, он возвращается обратно в `main()` к той точке, на которой остановился. Следовательно, следующим стейтментом является вывод строки `Ending main()`.

Обратите внимание, для вызова функции нужно указать её имя и список параметров в круглых скобках `()`. В примере, приведенном выше, параметры не используются, поэтому круглые скобки пусты. Мы детально поговорим о параметрах функций на следующем уроке.

**Правило: Не забывайте указывать круглые скобки `()` при вызове функций.**

## Возвращаемые значения

Когда функция `main()` завершает свое выполнение, она возвращает целочисленное значение обратно в операционную систему, используя **оператор `return`**.

Функции, которые мы пишем, также могут возвращать значения. Для этого нужно указать **тип возвращаемого значения** (или «*тип возврата*»). Он указывается при объявлении функции, перед её именем. Обратите внимание, тип возврата не указывает, какое именно значение будет возвращаться. Он указывает только тип этого значения.

Затем, внутри вызываемой функции, мы используем оператор `return`, чтобы указать **возвращаемое значение** — какое именно значение будет возвращаться обратно в `caller`.

Рассмотрим простую функцию, которая возвращает целочисленное значение:

```
1  #include <iostream>
2
3  // int означает, что функция возвращает целочисленное значение обратно в caller
4  int return7()
5  {
6      // Эта функция возвращает целочисленное значение, поэтому мы должны использовать
7      return 7; // возвращаем число 7 обратно в caller
8  }
9
10 int main()
11 {
12     std::cout << return7() << std::endl; // выведется 7
13     std::cout << return7() + 3 << std::endl; // выведется 10
14
15     return7(); // возвращаемое значение 7 игнорируется, так как функция main() ничего
16
17     return 0;
18 }
```

Результат выполнения программы:

```
7
10
```

Разберемся детально:

- Первый вызов функции `return7()` возвращает 7 обратно в `caller`, которое затем передается в [`std::cout`](#) для вывода.
- Второй вызов функции `return7()` опять возвращает 7 обратно в `caller`. Выражение `7 + 3` имеет результат 10, который затем выводится на экран.
- Третий вызов функции `return7()` опять возвращает 7 обратно в `caller`. Однако функция `main()` ничего с ним не делает, поэтому ничего и не происходит (возвращаемое значение игнорируется).

**Примечание:** Возвращаемые значения не выводятся на экран, если их не передать объекту `std::cout`. В последнем вызове функции `return7()` значение не отправляется в `std::cout`, поэтому ничего и не происходит.

## Тип возврата void

Функции могут и не возвращать значения. Чтобы сообщить компилятору, что функция не возвращает значение, нужно использовать **тип возврата void**. Взглянем еще раз на функцию `doPrint()` из вышеприведенного примера:

```
1  void doPrint() // void - это тип возврата
2  {
3      std::cout << "In doPrint()" << std::endl;
```

```
4 | // Эта функция не возвращает никакого значения, поэтому оператор return здесь не  
5 | }
```

Эта функция имеет тип возврата `void`, который означает, что функция не возвращает значения. Поскольку значение не возвращается, то и оператор `return` не требуется.

Вот еще один пример использования функции типа `void`:

```
1 | #include <iostream>  
2 |  
3 | // void означает, что функция не возвращает значения  
4 | void returnNothing()  
5 | {  
6 |     std::cout << "Hi!" << std::endl;  
7 |     // Эта функция не возвращает никакого значения, поэтому оператор return здесь не  
8 | }  
9 |  
10 | int main()  
11 | {  
12 |     returnNothing(); // функция returnNothing() вызывается, но обратно в main() ниче  
13 |  
14 |     std::cout << returnNothing(); // ошибка, эта строка не скомпилируется. Вам нуж  
15 |     return 0;  
16 | }
```

В первом вызове функции `returnNothing()` выводится `Hi !`, но ничего не возвращается обратно в `caller`. Точка выполнения возвращается обратно в функцию `main()`, где программа продолжает свое выполнение.

Второй вызов функции `returnNothing()` даже не скомпилируется. Функция `returnNothing()` имеет тип возврата `void`, который означает, что эта функция не возвращает значения. Однако функция `main()` пытается отправить это значение (которое не возвращается) в `std::cout` для вывода. `std::cout` не может обработать этот случай, так как значения на вывод не предоставлено. Следовательно, компилятор выдаст ошибку. Вам нужно будет закомментировать эту строку, чтобы компиляция прошла успешно.

## Возврат значений обратно в функцию `main()`

Теперь у вас есть понимание того, как работает функция `main()`. Когда программа выполняется, операционная система делает вызов функции `main()` и начинается её выполнение. Стейтменты в `main()` выполняются последовательно. В конце функция `main()` возвращает целочисленное значение (обычно `0`) обратно в операционную систему. Поэтому `main()` объявляется как `int main()`.

Почему нужно возвращать значения обратно в операционную систему? Дело в том, что возвращаемое значение функции `main()` является **кодом состояния**, который сообщает операционной системе об успешном или неудачном выполнении программы. Обычно, возвращаемое значение `0` (ноль) означает что всё прошло успешно, тогда как любое другое значение означает неудачу/ошибку.

Обратите внимание, по стандартам языка C++ функция `main()` должна возвращать целочисленное значение. Однако, если вы не укажете `return` в конце функции `main()`, компилятор возвратит `0`

автоматически, если никаких ошибок не будет. Но рекомендуется указывать return в конце функции main() и использовать тип возврата int для функции main().

## Еще о возвращаемых значениях

Во-первых, если тип возврата функции не void, то она должна возвращать значение указанного типа (использовать оператор return). Единственно исключение — функция main(), которая возвращает 0, если не предоставлено другое значение.

Во-вторых, когда процессор встречает в функции оператор return, он немедленно выполняет возврат значения обратно в caller и точка выполнения также переходит в caller. Любой код, который находится за return-ом в функции — игнорируется.

Функция может возвращать только одно значение через return обратно в caller. Это может быть либо число (например, 7), либо значение переменной, либо выражение (у которого есть результат), либо определенное значение из набора возможных значений.

Но есть способы обойти правило возврата одного значения, возвращая сразу несколько значений, но об этом детально мы поговорим на соответствующем уроке.

Наконец, автор функции решает, что означает её возвращаемое значение. Некоторые функции используют возвращаемые значения в качестве кодов состояния для указания результата выполнения функции (успешно ли выполнение или нет). Другие функции возвращают определенное значение из набора возможных значений. Кроме того, существуют функции, которые вообще ничего не возвращают.

## Повторное использование функций

Одну и ту же функцию можно вызывать несколько раз, даже в разных программах, что очень полезно:

```
1  #include <iostream>
2
3  // Функция getValueFromUser() получает значение от пользователя, а затем возвращает
4  int getValueFromUser()
5  {
6      std::cout << "Enter an integer: ";
7      int x;
8      std::cin >> x;
9      return x;
10 }
11
12 int main()
13 {
14     int a = getValueFromUser(); // первый вызов функции getValueFromUser()
15     int b = getValueFromUser(); // второй вызов функции getValueFromUser()
16
17     std::cout << a << " + " << b << " = " << a + b << std::endl;
18
19     return 0;
20 }
```

Результат выполнения программы:

```
Enter an integer: 4
Enter an integer: 9
4 + 9 = 13
```

Здесь `main()` прерывается 2 раза. Обратите внимание, в обоих случаях, полученное пользовательское значение сохраняется в переменной `x`, а затем передается обратно в `main()` с помощью `return`, где присваивается переменной `a` или `b`!

Также `main()` не является единственной функцией, которая может вызывать другие функции. Любая функция может вызывать любую другую функцию!

```
1  #include <iostream>
2
3  void printO()
4  {
5      std::cout << "O" << std::endl;
6  }
7
8  void printK()
9  {
10     std::cout << "K" << std::endl;
11 }
12
13 // Функция printOK() вызывает как printO(), так и printK()
14 void printOK()
15 {
16     printO();
17     printK();
18 }
19
20 // Объявление main()
21 int main()
22 {
23     std::cout << "Starting main()" << std::endl;
24     printOK();
25     std::cout << "Ending main()" << std::endl;
26     return 0;
27 }
```

Результат выполнения программы:

```
Starting main()
O
K
Ending main()
```

## Вложенные функции

В языке C++ одни функции не могут быть объявлены внутри других функций (т.е. быть вложенными). Следующий код вызовет ошибку компиляции:

```
1 #include <iostream>
2
3 int main()
4 {
5     int boo() // эта функция находится внутри функции main(), что запрещено
6     {
7         std::cout << "boo!";
8         return 0;
9     }
10
11     boo();
12     return 0;
13 }
```

Правильно вот так:

```
1 #include <iostream>
2
3 int boo() // теперь уже не в main()
4 {
5     std::cout << "boo!";
6     return 0;
7 }
8
9 int main()
10 {
11     boo();
12     return 0;
13 }
```

## Тест

Какие из следующих программ не скомпилируются (и почему), а какие скомпилируются (и какой у них результат)?

Программа №1:

```
1 #include <iostream>
2
3 int return5()
4 {
5     return 5;
6 }
7
8 int return8()
9 {
10     return 8;
11 }
12
13 int main()
```

```
14 {  
15     std::cout << return5() + return8() << std::endl;  
16  
17     return 0;  
18 }
```

Программа №2:

```
1 #include <iostream>  
2  
3 int return5()  
4 {  
5     return 5;  
6  
7     int return8()  
8     {  
9         return 8;  
10    }  
11 }  
12  
13 int main()  
14 {  
15     std::cout << return5() + return8() << std::endl;  
16  
17     return 0;  
18 }
```

Программа №3:

```
1 #include <iostream>  
2  
3 int return5()  
4 {  
5     return 5;  
6 }  
7  
8 int return8()  
9 {  
10    return 8;  
11 }  
12  
13 int main()  
14 {  
15     return5();  
16     return8();  
17  
18     return 0;  
19 }
```

Программа №4:

```
1 #include <iostream>  
2
```



```
3 void print0()
4 {
5     std::cout << "0" << std::endl;
6 }
7
8 int main()
9 {
10     std::cout << print0() << std::endl;
11
12     return 0;
13 }
```

Программа №5:

```
1 #include <iostream>
2
3 int getNumbers()
4 {
5     return 6;
6     return 8;
7 }
8
9 int main()
10 {
11     std::cout << getNumbers() << std::endl;
12     std::cout << getNumbers() << std::endl;
13
14     return 0;
15 }
```

Программа №6:

```
1 #include <iostream>
2
3 int return 6()
4 {
5     return 6;
6 }
7
8 int main()
9 {
10     std::cout << return 6() << std::endl;
11
12     return 0;
13 }
```

Программа №7:

```
1 #include <iostream>
2
3 int return6()
4 {
5     return 6;
```

```
6 | }
7 |
8 | int main()
9 | {
10 |     std::cout << return6 << std::endl;
11 |
12 |     return 0;
13 | }
```

## Ответы

Чтобы просмотреть ответ, кликните на него мышкой.

### Ответ №1

Скомпилируется, результатом выполнения программы будет значение 13.

### Ответ №2

Эта программа не скомпилируется. Вложенные функции запрещены.

### Ответ №3

Эта программа скомпилируется, но не будет никакого вывода. Возвращаемые значения из функций не используются в `main()` и, таким образом, игнорируются.

### Ответ №4

Эта программа не скомпилируется, так как тип возврата функции `printO()` — `void`, а мы отправляем несуществующее возвращаемое значение на вывод. Результат — ошибка компиляции.

### Ответ №5

Результатом выполнения этой программы будет:

6  
6

Оба раза, когда вызывается функция `getNumbers()`, возвращается значение 6. Компилятор, встречая первый `return`, сразу же выполняет возврат этого значения, и всё, что находится за первым `return`-ом, — игнорируется. Строка `return 8;` никогда не выполнится.

### Ответ №6

Эта программа не скомпилируется из-за недопустимого имени функции.

### Ответ №7

Эта программа скомпилируется, но функция не будет вызвана, так как в её вызове отсутствуют круглые скобки. Результат вывода зависит от компилятора.

Оценить статью:

 (769 оценок, среднее: 4,86 из 5)



← [Урок №11. cout, cin и endl](#)



[Урок №13. Параметры и аргументы функций](#) →

## Комментариев: 36



1. *Менфис:*

[28 сентября 2020 в 02:15](#)

Очень хорошие уроки, все понятно. Удобно и просто воспринимается информация. Уже несколько недель по степенно ознакомливаюсь. Спасибо за полезные и понятные уроки!!!

[Ответить](#)

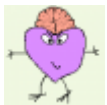


1. *Юрий:*

[28 сентября 2020 в 16:19](#)

Пожалуйста 😊

[Ответить](#)



2. *Денис:*

[26 сентября 2020 в 21:06](#)

Информация заходит легко и понятно. Респект автору!

[Ответить](#)



3. *Rejer:*

[2 июля 2020 в 23:40](#)

thank you !

Спасибо братан большое это очень круто:)

[Ответить](#)



4. *Fray:*

[17 июня 2020 в 14:00](#)

Юрий, большое спасибо за ваши труды и разложенный по полочкам материал! Удачи во всем!

[Ответить](#)



1. *Юрий:*

[17 июня 2020 в 16:22](#)

Пожалуйста 😊

[Ответить](#)



5. *АРТЕМ:*

[22 июня 2019 в 21:46](#)

Привет Автору статьи, спасибо что так всё расписал.

[Ответить](#)

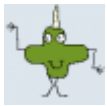


1. *Юрий:*

[23 июня 2019 в 12:06](#)

Привет, пожалуйста)))

[Ответить](#)



6. *Александр:*

[5 апреля 2019 в 04:54](#)

При много благодарности!

[Ответить](#)



7. *Артем:*

[24 марта 2019 в 22:43](#)

Здравствуй! Написал у себя код по вашему примеру, но почему-то у меня visual studio ругается на не объявленные идентификаторы cout и endl

```
1 #include "pch.h"
2 #include <iostream>
3
4 using namespace std;
5
6 void print0()
7 {
8     cout << "0" << endl;
9 }
10
11 void printK()
12 {
13     cout << "K" << endl;
14 }
15
16 void printOK()
17 {
18     print0();
```

```
19     printK();
20 }
21
22 int main()
23 {
24     cout << "Starting main()" << endl;
25     printOK();
26     cout << "Ending main()" << endl;
27     return 0;
28 }
```

[Ответить](#)

1. *Михаил:*  
[31 марта 2019 в 19:47](#)

Попробуй напиши не cout, а std::cout( std::endl)

[Ответить](#)

8. *Александр:*  
[9 марта 2019 в 05:51](#)

В языке C++ понятия "процедура" нет? Я раньше немного знакомился с дельфи, так там, на сколько я помню, функция должна возвращать какое либо значение, о процедуре нет.

[Ответить](#)

1. *Анастасия:*  
[9 мая 2019 в 21:50](#)

Процедура в C++ — это как функция с типом возвращаемого значения void

[Ответить](#)

9. *Александр:*  
[9 марта 2019 в 05:38](#)

Важный момент при написании текста программы, я возможно где то упустил, но функция main должна быть записана всегда внизу? А вызываемые функции вверх? И более поздняя вызываемая функция выше предыдущей? Или это не принципиально?

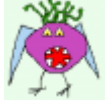
[Ответить](#)

1. *Анастасия:*  
[9 мая 2019 в 21:54](#)

Ответ на Ваш вопрос есть дальше в уроках. Я сначала тоже им задавалась. Если коротко, то используемые функции могут определяться и ниже функции, в которой используются, но

тогда их нужно обязательно хотя бы "объявить" перед функцией, в которой они используются.

[Ответить](#)



10. *Сергей:*

[6 марта 2019 в 11:32](#)

Большое спасибо за сайт!

[Ответить](#)



11. *Денис:*

[23 января 2019 в 22:39](#)

А для чего это вообще нужно? Тот же Void, который не возвращает. Можно же просто не прописывать столько строк, а написать то, что нужно, присвоив то или иное к чему-то и выполнив действие.

[Ответить](#)



1. *Константин:*

[24 февраля 2019 в 06:30](#)

Я использовал void() для обработки большого числа данных по одному какому-то предмету и в ходе выполнения получал ответ за ответом. А другой void() у меня работает с другим предметом.

[Ответить](#)



12. *Захар:*

[29 сентября 2018 в 19:28](#)

```
1  #include "pch.h"
2  #include <iostream>
3
4  int doubleNumber(int a)
5  {
6      return a+a;
7  }
8
9  int main()
10 {
11     std::cout << "Entered number! ";
12     int a = 0;
13     std::cin >> a;
14     std::cout << "You entered - " << doubleNumber(a) << std::endl;
15     return 0;
16 }
```

// здравствуйте я правильно понял вопрос и выполнил задание ?

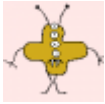
[Ответить](#)

13. Константин:

[4 сентября 2018 в 04:18](#)

Юра, в примере с printOK(), main(), выполняя код строку за строкой, видит printOK(), вызывает его и не дожидаясь ответа (т.к. он же void), продолжает свою работу, а тот в свою очередь также поступает с printO() и с printK() и все они параллельно работают или прерывают свое выполнение (а если так, то как они понимают где должна оказаться точка выполнения) и как возобновляют выполнение — ведь return-а у void-а нет

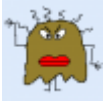
Поясни, пожалуйста.

[Ответить](#)

1. Данила:

[27 ноября 2018 в 13:27](#)

вот как я понял там функций возврата не нужна, так как функция вызывает другую функцию, и так каждая функция по очереди. Это как раз пример как любая функция может дать вызов. Майн вызывает ПринтОК, функция ПринтОК, в свою очередь (по запросу майн) вызывает уже две функции ПринтО и ПринтК. А Майн спокойно ждет, когда эти две функции придут к ней по её вызовам. :D

[Ответить](#)

14. Ray:

[29 мая 2018 в 20:47](#)

Доброго времени суток!

Сайт наилучший, все очень доходчиво. Большое спасибо!

Собрал примеры с нескольких уроков в одном примере и возникла пара вопросов:

```

1  #include <iostream>
2
3  void doPrint()
4  {
5      std::cout<<"Enter a number: ";
6      int a = 0;
7      std::cin>> a;
8      std::cout<<"Your number is: "<<a<<std::endl;
9  }
10 int getValueFromUsers()
11 {
12     std::cout<<"Enter an integer: ";
13     int x;
14     std::cin>>x;
15     return x;
16 }
17

```

```
18 int main()
19 {
20     std::cout<<"Starting main()"<<std::endl;
21     doPrint();
22     getValueFromUsers();
23
24     int a=getValueFromUsers();
25     int b=getValueFromUsers();
26     int c=getValueFromUsers();
27     int d=getValueFromUsers();
28     std::cout<<a<<"+"<<b<<"+"<<c<<"-"<<d<<std::endl;
29
30     std::cout<<"Ending main()"<<std::endl;
31     return 0;
32 }
```

Компиляция выдает следующее:

Starting main()

Enter a number: 7

Your number is: 7

Inter an integer: 1

Inter an integer: 2

Inter an integer: 3

Inter an integer: 4

Inter an integer: 5

2+3+4-5

Ending main()

Program ended with exit code: 0

Вопрос: почему при компиляции, при многократном применении функции, возвращенных значений получается 5 (хотя переменных было 4:a,b,c,d) ? и почему они начинаются со второго значения, т.е. 2+3+4-5, а не 1+2+3-4 ?

Заранее благодарен.

[ОТВЕТИТЬ](#)



1. Юрий:

[8 июня 2018 в 15:29](#)

Так посмотрите сами у себя в функции main() код: что и за чем выполняется.

Вы сначала вызываете doPrint — первое значение получаете (7). Затем вызываете getValueFromUsers — получаете второе значение (1). Затем инициализируете 4 переменные: a(2), b(3), c(4), d(5). Затем выполняете операции с 4 значениями: a + b + c — d. Второе значение getValueFromUsers (1) вы не присвоили никакой переменной в функции main(), только вывели в консоль, потому и значений 5. Почему вы добавляете 4 переменные и первое число 2, а не 1 — потому что отсчет начинается с переменной a(2).

[ОТВЕТИТЬ](#)



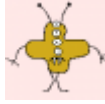
2. Максим:



[18 июля 2018 в 15:04](#)

Почему я такой не доходчивый?! Я понимаю 50 на 50, но написать ничего не могу(

[Ответить](#)



3. Данила:

[27 ноября 2018 в 13:07](#)

у вас ,после допринт идет лишняя строка ,она обозначается как 1 ,её надо убрать и тогда 1 = a.

```

1  int main()
2  {
3      std::cout << "Starting main()" << std::endl;
4      doPrint();
5      getValueFromUsers(); // <== это лишняя строка ,её убрать и тогда буде
6
7      int a = getValueFromUsers();
8      int b = getValueFromUsers();
9      int c = getValueFromUsers();
10     int d = getValueFromUsers();
11     std::cout << a << "+" << b << "+" << c << "-" << d << std::endl;
12
13     std::cout << "Ending main()" << std::endl;
14     return 0;
15 }
```

[Ответить](#)



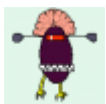
1. Fray:

[17 июня 2020 в 13:54](#)

Мне кажется автор комментария подразумевал, что для возврата какого-нибудь значения из функции `GetValueFromUsers ()` нужно иметь место переменную, куда можно вывести это значение. Но даже если она и есть, остается вопрос, почему после первого вызова `GetValueFromUsers ()` ее возвратное значение выводится на экран консоли, хотя команды `std::cout` не было?

Извиняюсь, если я что то неправильно пишу, я еще ламер

[Ответить](#)



15. Михаил:

[23 мая 2018 в 21:40](#)

Привет)). Скажи пожалуйста я правильно понял, что если функция к примеру эта:

```

1  using namespace std;
2
3  int ok(){
4      cout << "введите значение a:" << endl;
```

```
5 | int x;  
6 | cin >> x;  
7 | return x;
```

инструкция return должна быть обязательно тоже x? Я просто к тому, что у тебя в примерах я заметил, что ты в return указываешь число, то в caller оно и возвращается.

[Ответить](#)



1. Юрий:

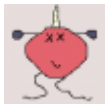
[26 мая 2018 в 00:06](#)

Привет. Во всех функциях, кроме типа void, return должен возвращать значение указанного типа данных. Какое именно значение — не важно, главное, чтобы совпадал тип данных. В вашей функции кроме x, вы также можете просто записать:

```
1 | return 7;
```

Ошибок от компилятора не будет. Но целесообразно возвращать x, так как вы используете функцию для получения значения.

[Ответить](#)



16. илья:

[30 марта 2018 в 10:56](#)

странно, мне 10 лет но я что то понимаю

[Ответить](#)



1. Юрий:

[30 марта 2018 в 11:40](#)

Ничего странного, если есть желание — будут возможности.

[Ответить](#)



17. beksheikh:

[14 марта 2018 в 14:20](#)

Здравствуйте. скажите в институте учился и мне не говорили о ретурне. вместо этого использую getch(); подскажите в чем разница? Сначала добавляю библиотеку <conio.h> и потом в конце пишу гетч. и Второй вопрос. Как вы пишете std::cout(endl, cin) вместо этого научили писать в начале using namespace std; и потом вместо std::cout и тд пишу без std. скажите как правильнее?) заранее Спасибо за ответ. С Уважением Beksheikh.

[Ответить](#)



1. Юрий:

[15 марта 2018 в 21:17](#)

Привет.

1. Функция `getch()` используется для захвата одного символа из консоли. Её вам говорили использовать в программах в самом конце, чтобы консольное окно не закрывалось сразу же после выполнения всех действий и чтобы вы успели увидеть результат. `return` же возвращает код состояния, в случае с `return` в функции `main()` — он возвращает 0, если всё произошло без ошибок, всё корректно и 1 (или любой другой символ ненулевой), если были обнаружены ошибки. Детальнее об этом говорится в этом же уроке.

Функция `main()` должна возвращать значения, так как она типа `int`, а не `void`. `return` и `getch` выполняют разные задачи. `return` должен быть во всех функциях не `void`. `getch()` в конце можно прописывать, можно нет. Аналог ему:

```
1 | system("pause");
```

Если у вас консольное окно закрывается сразу же и вы не успеваете увидеть результат выполнения вашей программы — используйте либо `getch()`;, либо `system(«pause»)`;

2. Подробный ответ на второй ваш вопрос находится в [уроке 24](#) и в [уроке 54](#).

[Ответить](#)



18. *nurdosramazan:*

[26 ноября 2017 в 09:31](#)

Отличный сайт. Автору спасибо  
(даже адблок отключил ;))

[Ответить](#)



1. *Юрий:*

[26 ноября 2017 в 11:34](#)

Ахахах спасибо 😊

[Ответить](#)



19. *Алексей:*

[27 июля 2017 в 12:00](#)

Тут не хватает «{»  
У вас не поставлена=)

```
1 | void doPrint()  
2 |     std::cout << "In doPrint()" << std::endl;  
3 | }
```

[Ответить](#)



1. *Юрий:*

[27 июля 2017 в 15:37](#)

Спасибо, исправил.

[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