

Ravesli [Ravesli](#)


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


## Урок №70. Операторы break и continue

 [Юрий](#) |

- [Уроки C++](#)

|

 Обновл. 15 Сен 2020 |

 61675

[↑](#)  8

Хотя вы уже видели оператор break в связке с [оператором switch](#), все же он заслуживает большего внимания, поскольку может использоваться и с циклами. **Оператор break** приводит к завершению выполнения циклов do, for или while.

Оглавление:

1. [break и switch](#)
2. [break и циклы](#)
3. [break и return](#)
4. [Оператор continue](#)
5. [break и continue](#)

### break и switch

В контексте оператора switch оператор break обычно используется в конце каждого кейса для его завершения (предотвращая [fall-through](#)):

```
1 switch (op)
2 {
3     case '+':
4         doAddition(a, b);
5         break;
6     case '-':
7         doSubtraction(a, b);
8         break;
```

```
9     case '*':
10         doMultiplication(a, b);
11         break;
12     case '/':
13         doDivision(a, b);
14         break;
15 }
```

## break и циклы

В контексте циклов оператор break используется для завершения работы цикла раньше времени:

```
1  #include <iostream>
2
3  int main()
4  {
5      int sum = 0;
6
7      // Разрешаем пользователю ввести до 10 чисел
8      for (int count=0; count < 10; ++count)
9      {
10         std::cout << "Enter a number to add, or 0 to exit: ";
11         int val;
12         std::cin >> val;
13
14         // Выходим из цикла, если пользователь введет 0
15         if (val == 0)
16             break;
17
18         // В противном случае, добавляем число к общей сумме
19         sum += val;
20     }
21
22     std::cout << "The sum of all the numbers you entered is " << sum << "\n";
23
24     return 0;
25 }
```

Эта программа позволяет пользователю ввести до 10 чисел и в конце подсчитывает их сумму. Если пользователь введет 0, то выполнится break и цикл завершится (не важно, сколько чисел в этот момент успел ввести пользователь).

Обратите внимание, оператор break может использоваться и для выхода из бесконечного цикла:

```
1  #include <iostream>
2
3  int main()
```

```
4 {
5     while (true) // бесконечный цикл
6     {
7         std::cout << "Enter 0 to exit or anything else to continue: ";
8         int val;
9         std::cin >> val;
10
11         // Выходим из цикла, если пользователь ввел 0
12         if (val == 0)
13             break;
14     }
15
16     std::cout << "We're out!\n";
17
18     return 0;
19 }
```

## break и return

Новички часто путают или не понимают разницы между операторами [break](#) и [return](#). Оператор `break` завершает работу `switch` или цикла, а выполнение кода продолжается с первого кейс-ментента, который находится сразу же после этого `switch` или цикла. Оператор `return` завершает выполнение всей функции, в которой находится цикл, а выполнение продолжается в точке после вызова функции:

```
1 #include <iostream>
2
3 int breakOrReturn()
4 {
5     while (true) // бесконечный цикл
6     {
7         std::cout << "Enter 'b' to break or 'r' to return: ";
8         char sm;
9         std::cin >> sm;
10
11         if (sm == 'b')
12             break; // выполнение кода продолжится с первого кейс-ментента после цикла
13
14         if (sm == 'r')
15             return 1; // выполнение return приведет к тому, что управление сразу возвратит
16     }
17
18     // Использование оператора break приведет к тому, что выполнение цикла продолжится
19
20     std::cout << "We broke out of the loop\n";
21
22     return 0;
23 }
24
```

```
25 int main()
26 {
27     int returnValue = breakOrReturn();
28     std::cout << "Function breakOrContinue returned " << returnValue << '\n';
29
30     return 0;
31 }
```

## Оператор continue

**Оператор continue** позволяет сразу перейти в конец тела цикла, пропуская весь код, который находится под ним. Это полезно в тех случаях, когда мы хотим завершить текущую итерацию раньше времени. Например:

```
1  #include <iostream>
2
3  int main()
4  {
5      for (int count = 0; count < 20; ++count)
6      {
7          // Если число делится нацело на 4, то пропускаем весь код в этой итерации после
8          if ((count % 4) == 0)
9              continue; // пропускаем всё и переходим в конец тела цикла
10
11         // Если число не делится нацело на 4, то выполнение кода продолжается
12         std::cout << count << std::endl;
13
14         // Точка выполнения после оператора continue перемещается сюда
15     }
16
17     return 0;
18 }
```

Эта программа выведет все числа от 0 до 19, которые не делятся нацело на 4.

В случае с циклом for часть инкремента/декремента счетчика по-прежнему выполняется даже после выполнения continue (так как инкремент/декремент происходит вне тела цикла).

Будьте осторожны при использовании оператора continue с циклами while или do while. Поскольку в этих циклах инкремент счетчиков выполняется непосредственно в теле цикла, то использование continue может привести к тому, что цикл станет бесконечным! Например:

```
1  #include <iostream>
2
3  int main()
4  {
```

```
5   int count(0);
6   while (count < 10)
7   {
8       if (count == 5)
9           continue; // переходим в конец тела цикла
10      std::cout << count << " ";
11      ++count;
12
13      // Точка выполнения после оператора continue перемещается сюда
14  }
15
16  return 0;
17 }
```

Предполагается, что программа выведет все числа от 0 до 9, за исключением 5. Но на самом деле:

0 1 2 3 4

А затем цикл станет бесконечным. Когда значением `count` становится 5, то условие оператора if станет true, затем выполнится `continue` и мы, минуя вывод числа и инкремент счетчика, перейдем к следующей итерации. Переменная `count` так и не увеличится. Как результат, в следующей итерации переменная `count` по-прежнему останется со значением 5, а оператор `if` по-прежнему останется true, и цикл станет бесконечным.

А вот правильное решение, но с использованием цикла `do while`:

```
1  #include <iostream>
2
3  int main()
4  {
5      int count(0);
6      do
7      {
8          if (count == 5)
9              continue; // переходим в конец тела цикла
10         std::cout << count << " ";
11
12         // Точка выполнения после оператора continue перемещается сюда
13     } while (++count < 10); // этот код выполняется, так как он находится вне тела цикла
14
15     return 0;
16 }
```

Результат выполнения программы:

0 1 2 3 4 6 7 8 9

## break и continue

Многие учебники рекомендуют не использовать операторы `break` и `continue`, поскольку они приводят к произвольному перемещению точки выполнения программы по всему коду, что усложняет понимание и следование логике выполнения такого кода.

Тем не менее, разумное использование операторов `break` и `continue` может улучшить читабельность циклов в программе, уменьшив при этом количество вложенных блоков и необходимость наличия сложной логики выполнения циклов. Например, рассмотрим следующую программу:

```
1  #include <iostream>
2
3  int main()
4  {
5      int count(0); // считаем количество итераций цикла
6      bool exitLoop(false); // контролируем завершение выполнения цикла
7      while (!exitLoop)
8      {
9          std::cout << "Enter 'e' to exit this loop or any other key to continue: ";
10         char sm;
11         std::cin >> sm;
12
13         if (sm == 'e')
14             exitLoop = true;
15         else
16         {
17             ++count;
18             std::cout << "We've iterated " << count << " times\n";
19         }
20     }
21
22     return 0;
23 }
```

Эта программа использует [логическую переменную](#) для выхода из цикла, а также вложенный блок, который запускается только в том случае, если пользователь не использует символ выхода.

А вот более читабельная версия, но с использованием оператора `break`:

```
1  #include <iostream>
2
3  int main()
4  {
5      int count(0); // считаем количество итераций цикла
6      while (true) // выполнение цикла продолжается, если его не завершит пользователь
7      {
8          std::cout << "Enter 'e' to exit this loop or any other key to continue: ";
9          char sm;
10         std::cin >> sm;
11
12         if (sm == 'e')
```

```
13         break;
14
15         ++count;
16         std::cout << "We've iterated " << count << " times\n";
17     }
18
19     return 0;
20 }
```

Здесь (с одним оператором break) мы избежали использования как логической переменной (а также понимания того, зачем она и где используется), так и оператора else с вложенным блоком.

Уменьшение количества используемых переменных и вложенных блоков улучшают читабельность и понимание кода намного больше, чем операторы break или continue могут нанести вред. По этой причине считается приемлемым их разумное использование.

Оценить статью:

★★★★★ (268 оценок, среднее: 4,97 из 5)



[← Урок №69. Цикл for](#)

[Урок №71. Генерация случайных чисел](#)



## Комментариев: 8



1. Яна:

[16 июня 2020 в 13:47](#)

Наконец могу ещё с начала темы о вводе переменных реализовать программу , позволяющая ввести имя и в случае ошибки начать всё заново. Маленькая победа длиной пятью глав 😊

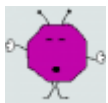
```
1 #include <iostream>
2
3 char name [10];
4 char uname()
5 {
6     int op;
7     while (true) // бесконечный цикл
8     {
9         std::cout << "Please enter your name: ";
```

```

10     std::cin >> ::name;
11
12     std::cout << "Your name is " << ::name << "? " << std::endl;
13     std::cout << "Please enter the answer (1-yes, 2-no): ";
14     std::cin >> op;
15
16     if (op == 1)
17         break;
18 }
19 std::cout << "Hello, " << ::name << " !" << std::endl;
20 return 0;
21 }
22 int main()
23 {
24     uname();
25 }

```

### [Ответить](#)



2. Александр:

[8 февраля 2019 в 12:21](#)

Вы очень хорошо оговорились о "разумном использовании". Если человек только начинает изучать программирование, то ни о каком "разумном использовании" речи идти не может 😊

В учебниках много дичи, но рекомендация не использовать goto и его производные хорошая. Во всяком случае на стадии обучения.

Я бы еще одну рекомендацию для новичков добавил — "одна функция — один return"...

### [Ответить](#)



3. Александр:

[29 сентября 2018 в 14:18](#)

Почему бы просто цикл while не исправить, а не писать do while. Достаточно же добавить 1 строку:

```

1  int count(0);
2  while (count < 10)
3  {
4      if (count == 5)
5      {
6          ++count; /// вот эту. Тем самым 5-рка проигнорируется, но счетчик увеличится,
7          continue;
8      }
9
10     cout << count << " ";
11     ++count;

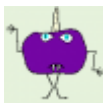
```



```
12 | }
```

Лично у меня всё отлично заработало (я просто работаю на Linux-Ubuntu, и особенности MSVisual не знаю, возможно на ней так не прокатит, но по идее должно).

[Ответить](#)



1. *Вадим:*

[29 января 2019 в 12:12](#)

Данный вариант будет работать везде, но это костыль, у вас в двух местах ++count. Зачем писать такой говнокод, если есть средства языка позволяющие делать лаконичные решения, например, с do while?

[Ответить](#)



1. *Борис:*

[30 апреля 2020 в 20:07](#)

Это тоже лаконично (ему там даже continue не нужно, он забыл его удалить). Это ещё далеко не говнокод, а вполне приемлемо.

[Ответить](#)



2. *Борис:*

[30 апреля 2020 в 20:05](#)

Можно и так конечно, и в вашем случае даже continue не нужен, сотрите его!

[Ответить](#)



4. *Герман:*

[2 февраля 2018 в 21:27](#)

Уважаемый автор, возможно ли использовать оператор return, без параметров, для того чтобы прекратить выполнение функции типа void

[Ответить](#)



1. *Юрий:*

[4 февраля 2018 в 22:02](#)

Да, иногда используется оператор return без каких-либо значений.

[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 «SameGame» на C++/MFC](#)
-  [Урок №4. Установка IDE \(Интегрированной Среды Разработки\)](#)

- [Ravesli](#)
- - [О проекте/Контакты](#) -
- - [Пользовательское Соглашение](#) -
- - [Все статьи](#) -
- Copyright © 2015 - 2020