

Ravesli [Ravesli](#)

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





Урок №66. Оператор goto

 [Юрий](#) |

- [Уроки C++](#)

|

 Обновл. 1 Авг 2020 |

 75876

[1](#)  [7](#)

Оператор `goto` — это оператор управления потоком выполнения программ, который заставляет центральный процессор выполнить переход из одного участка кода в другой (осуществить прыжок). Другой участок кода идентифицируется с помощью **лейбла**. Например:

```
1  #include <iostream>
2  #include <cmath> // для функции sqrt()
3
4  int main()
5  {
6      double z;
7      tryAgain: // это лейбл
8          std::cout << "Enter a non-negative number: ";
9          std::cin >> z;
10
11         if (z < 0.0)
12             goto tryAgain; // а это оператор goto
13
14         std::cout << "The sqrt of " << z << " is " << sqrt(z) << std::endl;
15         return 0;
16 }
```

В этой программе пользователю предлагается ввести неотрицательное число. Однако, если пользователь введет отрицательное число, программа, используя оператор `goto`, выполнит переход обратно к лейблу `tryAgain`. Затем пользователю снова нужно будет ввести число. Таким образом, мы можем постоянно запрашивать у пользователя ввод числа, пока он не введет корректное число.

Ранее мы рассматривали два типа области видимости: **локальная** (или «блочная») и **глобальная** (или «файловая»). Лейблы используют третий тип области видимости: **область видимости функции**. Оператор `goto` и соответствующий лейбл должны находиться в одной и той же функции.

Существуют некоторые ограничения на использование операторов `goto`. Например, вы не сможете перепрыгнуть вперед через переменную, которая инициализирована в том же блоке, что и `goto`:

```
1 int main()
2 {
3     goto skip; // прыжок вперед недопустим
4     int z = 7;
5 skip: // лейбл
6     z += 4; // какое значение будет в этой переменной?
7     return 0;
8 }
```

В целом, программисты избегают использования оператора `goto` в языке C++ (и в большинстве других высокоуровневых языков программирования). Основная проблема с ним заключается в том, что он позволяет программисту управлять выполнением кода так, что точка выполнения может прыгать по коду произвольно. А это, в свою очередь, создает то, что опытные программисты называют «спагетти-кодом». **Спагетти-код** — это код, порядок выполнения которого напоминает тарелку со спагетти (всё запутано и закручено), что крайне затрудняет следование порядку и понимание логики выполнения такого кода.

Как говорил один известный специалист в информатике и программировании, Эдсгер Дейкстра: «Качество программистов — это уменьшающаяся функция плотности использования операторов `goto` в программах, которые они пишут».

Оператор `goto` часто используется в некоторых старых языках, таких как Basic или Fortran, или даже в языке Си. Однако в C++ `goto` почти никогда не используется, поскольку любой код, написанный с ним, можно более эффективно переписать с использованием других объектов в языке C++, таких как циклы, обработчики исключений или деструкторы (всё перечисленное мы рассмотрим чуть позже).

Правило: Избегайте использования операторов `goto`, если на это нет веских причин.

Оценить статью:

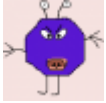
★★★★★ (258 оценок, среднее: 4,95 из 5)



← [Урок №65. Оператор switch](#)



Комментариев: 7



1. Артём:

[4 июля 2020 в 17:36](#)

Хоть использование `goto` не приветствуется, а порой даже проклиняется, есть случаи, когда без него оч трудно обойтись.

Например, при написании своей программы мне необходимо было сделать так, чтобы из любой точки выполнения программы пользователь мог при нажатии на Esc вернуться в самое главное меню. Да, альтернативно можно было весь код запихнуть в бесконечный цикл на отлов кода клавиши Esc. Однако, там был не безусловный переход в главное меню, а вызывалось подменю "Хотите выйти?" и в случае отказа, программа должна была продолжить выполнение, убрав окно с предложением выйти. Я не знаю на сегодняшний момент более изящного решения этой проблемы, чем `goto`.

[Ответить](#)



2. Максим:

[28 мая 2020 в 15:12](#)

Из рубрики "вдруг кому-то пригодится".

Раз не рекомендуется использовать оператор `goto`, прикладываю замену кода из лекции с помощью цикла `while()`:

```
1 #include <iostream>
2 #include <cmath> // для функции sqrt()
3
4 int main()
5 {
6     double z;
7
8     std::cout << "Enter a non-negative number: ";
9     std::cin >> z;
10
11     while (z < 0.0)
12     {
13         std::cout << "Enter a non-negative number: ";
14         std::cin >> z;
15     }
16
17     std::cout << "The sqrt of " << z << " is " << sqrt(z) << std::endl;
18     return 0;
```

19 | }

[Ответить](#)1. *Максим:*[30 мая 2020 в 16:25](#)

Ха, сначала следовало бы пройти все следующие уроки про циклы))
 Зато получилось показать разные варианты решения одной и той же "задачи".
 С циклом do while код получается меньше и лаконичнее:

```

1  #include <iostream>
2  #include <cmath> // для функции sqrt()
3
4  int main()
5  {
6      double z;
7
8      do
9      {
10         std::cout << "Enter a non-negative number: ";
11         std::cin >> z;
12     }
13     while (z < 0.0);
14
15     std::cout << "The sqrt of " << z << " is " << sqrt(z) << std::endl;
16     return 0;
17 }
```

[Ответить](#)3. *SuRprizZe:*[14 марта 2019 в 21:33](#)

Привет , вот я решал тест с главы номер 5, я как понял оператор goto лучше не использовать , но у меня возникла идея с этим оператором и я решил использовать , как можно заменить его?

```

1  bool moreLessGuesses(int guesses, int number)
2  {
3
4      for (int count = 1; count <= guesses; ++count)
5      {
6          std::cout << "Попытка #" << count << ":";
7          int guess;
8          start:
9          std::cin >> guess;
10         if (std::cin.fail()) // если предыдущее извлечение не выполнилось или про
11         {
```

```

12         std::cin.clear(); // возвращаем cin в 'обычный' режим работы
13         std::cin.ignore(32767, '\n'); // и удаляем значения предыдущего ввода
14         std::cout << "Нельзя вводить другие символы кроме чисел! Попробуй еще
15         std::cout << "Попытка #" << count << ":";
16         goto start;
17     }
18     if (guess < number)
19         std::cout << "Загаданное число - больше!\n";
20     else if (guess > number)
21         std::cout << "Загаданное число - меньше!\n";
22     else
23         return true;
24 }
25
26 return false;
27 }

```

[Ответить](#)



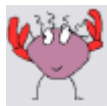
1. *Тукан:*

[20 марта 2019 в 14:17](#)

Например, использовать бесконечный for(;;) и использовать break, когда условие выполняется верно.

Сам goto, кстати, выгодно использовать для выхода из нескольких вложенных циклов. Либо использовать bool как флаг и делать несколько проверок через if.

[Ответить](#)



1. *игорь:*

[4 сентября 2019 в 12:37](#)

```

1  try
2  {
3      while(...)
4      {
5          while(...)
6          {
7              .....
8              throw exc;
9              .....
10         }
11     }
12 }
13 catch(... exc){}

```

и ни какого goto

[Ответить](#)

2. Moonilis:

[18 июля 2019 в 16:08](#)

Думаю уже не актуально, но кому-то может пригодится.
Можно заменить на цикл while:

```
1 ...
2 std::cin >> guess;
3 while (std::cin.fail()) // если предыдущее извлечение не выполнилось или про
4 {
5     std::cin.clear(); // возвращаем cin в 'обычный' режим работы
6     std::cin.ignore(32767, '\n'); // и удаляем значения предыдущего ввода из
7     std::cout << "Нельзя вводить другие символы кроме чисел! Попробуй еще ра
8     std::cout << "Попытка #" << count << ":";
9     std::cin >> guess; // Снова вводим
10 }
11 ...
```

[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