

Ravesli [Ravesli](#)


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


## Урок №63. Операторы управления потоком выполнения программ

 [Юрий](#) |

- [Уроки C++](#)

|

 Обновл. 15 Сен 2020 |

 27842

[13](#)

При запуске программы, центральный процессор (сокр. «ЦП») начинает выполнение кода с первой строки функции `main()`, выполняя определенное количество стеитментов, а затем завершает выполнение при завершении блока `main()`. Последовательность стеитментов, которые выполняет ЦП, называется **порядком выполнения программы** (или «*потоком выполнения программы*»).

Оглавление:

1. [Порядок выполнения программ](#)
2. [Остановка](#)
3. [Прыжок](#)
4. [Условные ветвления](#)
5. [Циклы](#)
6. [Исключения](#)
7. [Заключение](#)

## Порядок выполнения программ

Большинство программ, которые мы рассматривали до этого момента, были **линейными с последовательным выполнением**, то есть порядок выполнения у них один и тот же каждый раз: выполняются одни и те же стеитменты, даже если значения, которые вводит пользователь, — меняются.

Но на практике это не всегда может быть то, что нам нужно. Например, если мы попросим пользователя сделать выбор между + и /, а пользователь введет некорректный символ (например, \*), то в идеале нам

нужно было бы попросить его ввести символ еще раз. Но это невозможно в линейной программе. Кроме того, бывают случаи, когда нужно выполнить что-то несколько раз, но количество этих повторений наперед неизвестно. Например, если бы мы хотели вывести все целые числа от 0 до числа, которое введет пользователь, то в линейной программе мы бы не смогли это сделать, не зная наперед число, которое введет пользователь.

К счастью, в языке C++ есть **операторы управления порядком выполнения программы**, которые позволяют программисту изменить поток выполнения программы центральным процессором.

## Остановка

Самый простой оператор управления порядком выполнения программы — это **остановка**, которая сообщает программе немедленно прекратить свое выполнение. В языке C++ остановка осуществляется с помощью **функции `exit()`**, которая определена в заголовочном файле `cstdlib`. Функция `exit()` принимает целочисленный параметр, который затем возвращает обратно в операционную систему в качестве кода выхода. Например:

```
1 #include <iostream>
2 #include <cstdlib> // для функции exit()
3
4 int main()
5 {
6     std::cout << 5;
7     exit(0); // завершаем выполнение программы и возвращаем 0 обратно в операционную с
8
9     // Следующие стейтменты никогда не выполнятся
10    std::cout << 3;
11    return 0;
12 }
```

## Прыжок

Следующим оператором управления порядком выполнения программы является **прыжок** (или «*переход*»). Он безоговорочно сообщает компилятору во время выполнения перейти от одного стейтмента к другому, т.е. выполнить прыжок. **Ключевые слова `goto`, `break` и `continue`** являются разными типами прыжков, об их различиях мы поговорим несколько позже.

Вызовы функций — это тоже, в некоторой степени, прыжки. При выполнении вызова функции, ЦП переходит к началу вызываемой функции. Когда вызываемая функция заканчивается, выполнение возвращается к следующему стейтменту, который находится после вызова этой функции.

## Условные ветвления

**Условное ветвление** заставляет программу изменить свой порядок выполнения, основываясь на значении результата выражения. Одним из основных операторов условного ветвления является `if`, который вы уже могли видеть в программах раньше. Например:

```
1 int main()
2 {
3     // Делаем A
4     if (expression)
5         // Делаем B
6     else
7         // Делаем C
8
9     // Делаем D
10 }
```

Здесь есть два возможных пути выполнения программы. Если результатом выражения будет `true`, то программа выполнит A, B и D. Если же результатом выражения будет `false`, то программа выполнит A, C и D. Таким образом, выполнится либо B, либо C, оба варианта выполняться вместе не будут. Это уже не линейная программа.

**Ключевое слово `switch`** также предоставляет механизм для выполнения условного ветвления. Более подробно об операторах `if` и `switch` мы поговорим на соответствующих уроках.

## Циклы

**Цикл** заставляет программу многократно выполнять определенное количество стейтментов до тех пор, пока заданное условие не станет ложным. Например:

```
1 int main()
2 {
3     // Делаем A
4     // B делается в цикле 0 или больше раз
5     // Делаем C
6 }
```

Эта программа может выполняться как ABC, ABBC, ABVBC, ABVVBC или даже AC. Опять же, она больше не является линейной, её порядок выполнения зависит от того, сколько раз выполнится цикл (если вообще выполнится).

**В языке C++ есть 4 типа циклов:**

- ➔ цикл `while`;
- ➔ цикл `do while`;
- ➔ цикл `for`;
- ➔ цикл `foreach` (добавили в C++11).

Мы подробно рассмотрим каждый из них в этой главе, кроме `foreach` (о нем немного позже).

# Исключения

**Исключения** предлагают механизм обработки ошибок, возникающих в функции. Если в функции возникает ошибка, с которой она не может справиться, то она может выбросить исключение. Это заставит ЦП перейти к ближайшему блоку кода, который обрабатывает исключения данного типа.

Обработка исключений — это довольно сложная тема и это единственный тип оператора управления порядком выполнения, который мы не будем рассматривать в этой главе.

## Заключение

Используя операторы управления порядком выполнения программы, вы можете повлиять на поток выполнения программы центральным процессором, а также на то, из-за чего он может быть прерван. До этого момента функционал наших программ был очень ограничен. Теперь же с операторами управления порядком выполнения программ мы сможем осуществить огромное количество разных интересных вещей, например, отображение меню до тех пор, пока пользователь не сделает правильный выбор; вывод каждого числа между  $x$  и  $y$ , и много-много чего еще.

Как только вы разберетесь с этой темой, вы перейдете на новый, более качественный уровень. Больше вы не будете ограничены игрушечными программами или простенькими упражнениями — вы сможете писать полноценные программы. Вот именно здесь и начинается всё самое интересное.

Погнали!

Оценить статью:

★★★★★ (284 оценок, среднее: 4,96 из 5)



← [Глава №4. Итоговый тест](#)

[Урок №64. Операторы условного ветвления if/else](#) →



## Комментариев: 13



1. Ангелина:

[14 октября 2020 в 16:27](#)

Юрий, большущее спасибо за всю эту огромную работу по переводу и адаптации уроков! Учусь по этому курсу полтора месяца с нуля, прям совсем с нуля, и очень ждала, когда же наконец дойду до циклов — и вот он, этот момент) Итоговое задание по главе 4 выполнено, и впереди действительно все самое интересное 😊

[Ответить](#)

1.  Юрий:  
[15 октября 2020 в 01:13](#)

Пожалуйста 😊 Мне очень приятно)

[Ответить](#)

2.  Ахмадиёр:  
[28 марта 2020 в 19:07](#)

Юрий спасибо вам огромное этот сайт ,эти уроки мне очень помогают спасибо огромное

[Ответить](#)

3.  Алексей:  
[17 июля 2019 в 12:26](#)

Юрий, елки-палки, как я этого ждал.

Я пусть большее время писал скрипты на bash, но без for || while просто не обходиться ничего.

Тут пойдет веселье. Тяжкое, сложное, но веселье.

[Ответить](#)

4.  Николай:  
[19 марта 2018 в 01:08](#)

Огромное спасибо человеку, который сделал перевод этого прекрасного учебника! Чувак, скинь данные — я перечислю тебе деньги, ты невероятно крутой!)))))) На англ версии еще куча глав, будешь ли заниматься их переводом?

[Ответить](#)

1.  Юрий:  
[19 марта 2018 в 16:20](#)

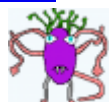
Привет, реально приятно читать такие отзывы, спасибо 😊 Да, постараюсь перевести все уроки/главы до конца.

[Ответить](#)5. *Человек:*[16 марта 2018 в 15:34](#)

Еще один вопрос: можно ли вместо `exit(0)` использовать `return 0`? Или с функцией `main` такой прием не работает?

[Ответить](#)1. *Юрий:*[17 марта 2018 в 01:03](#)

Функция `exit()` используется для возврата кода ошибки обратно в операционную систему и немедленного завершения выполнения программы (детальнее [здесь](#)). Использовать `exit(0)` вместо `return`-а в некоторых случаях можно, но нужно понимать, для чего используется `return`, а для чего `exit`. Зачем использовать вилку, если есть ложка — это аналогия вашего вопроса. Делать можете, что хотите, но есть неписанные правила, и лично я, еще нигде ни разу не видел, чтобы в `main`-е использовали `exit` вместо `return`-а.

[Ответить](#)1. *Человек:*[17 марта 2018 в 14:20](#)

И снова спасибо)

[Ответить](#)1. *Юрий:*[17 марта 2018 в 21:19](#)

Пожалуйста 😊

6. *Александр:*[30 сентября 2017 в 16:51](#)

Самый лучший учебник, который пока я встречал. Спасибо

[Ответить](#)1. *Юрий:*[1 октября 2017 в 15:04](#)

Пожалуйста 😊

[Ответить](#)7. *Sasha:*[3 июня 2017 в 03:04](#)

Еще уроки!!

[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