

Ravesli [Ravesli](#)


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


Урок №98. Передача по ссылке

 [Юрий](#) |

- [Уроки C++](#)

|

 Обновл. 21 Сен 2020 |

 45997

[1](#)  [3](#)

Хотя [передача по значению](#) является хорошим вариантом во многих случаях, она имеет несколько ограничений. Во-первых, при передаче по значению большой [структуры](#) или класса в функцию, создается копия аргумента и уже эта копия передается в параметр функции. В большинстве своем это бесполезная трата ресурсов, которая снижает производительность.

Во-вторых, при передаче аргументов по значению, единственный способ вернуть значение обратно в вызывающий объект — это использовать возвращаемое значение функции. Но иногда случаются ситуации, когда нужно, чтобы функция изменила значение переданного аргумента. Передача по ссылке решает все эти проблемы.

Оглавление:

1. [Передача по ссылке](#)
2. [Возврат сразу нескольких значений](#)
3. [Передача по константной ссылке](#)
4. [Плюсы и минусы передачи по ссылке](#)

Передача по ссылке

При **передаче переменной по ссылке** нужно просто объявить параметры функции как [ссылки](#), а не как обычные переменные:

```
1 void func(int &x) // x - это переменная-ссылка
2 {
3     x = x + 1;
```

4 | }

При вызове функции переменная `x` станет ссылкой на аргумент. Поскольку ссылка на переменную обрабатывается точно так же, как и сама переменная, то любые изменения, внесенные в ссылку, приведут к изменениям исходного значения аргумента! В следующем примере это хорошо проиллюстрировано:

```
1 #include <iostream>
2
3 void boo(int &value)
4 {
5     value = 7;
6 }
7
8 int main()
9 {
10     int value = 6;
11
12     std::cout << "value = " << value << '\n';
13     boo(value);
14     std::cout << "value = " << value << '\n';
15     return 0;
16 }
```

Эта программа точно такая же, как и программа из предыдущего урока, за исключением того, что параметром функции `boo()` теперь является ссылка вместо обычной переменной.

Результат выполнения программы:

```
value = 6
value = 7
```

Как вы можете видеть, функция изменила значение аргумента с 6 на 7!

Вот еще один пример:

```
1 #include <iostream>
2
3 void addOne(int &x) // x - это переменная-ссылка
4 {
5     x = x + 1;
6 } // x уничтожается здесь
7
8 int main()
9 {
10     int a = 7;
11     std::cout << "a = " << a << '\n';
12     addOne(a);
13     std::cout << "a = " << a << '\n';
14     return 0;
15 }
```

Результат выполнения программы:

a = 7

a = 8

Обратите внимание, значение аргумента **a** было изменено функцией.

Возврат сразу нескольких значений

Иногда нам может понадобиться, чтобы функция возвращала сразу несколько значений. Однако оператор `return` позволяет функции иметь только одно возвращаемое значение. Одним из способов возврата сразу нескольких значений является использование ссылок в качестве параметров:

```
1 #include <iostream>
2 #include <math.h> // для sin() и cos()
3
4 void getSinCos(double degrees, double &sinOut, double &cosOut)
5 {
6     // sin() и cos() принимают радианы, а не градусы, поэтому необходима конвертация
7     const double pi = 3.14159265358979323846; // значение Пи
8     double radians = degrees * pi / 180.0;
9     sinOut = sin(radians);
10    cosOut = cos(radians);
11 }
12
13 int main()
14 {
15     double sin(0.0);
16     double cos(0.0);
17
18     // функция getSinCos() возвратит sin и cos в переменные sin и cos
19     getSinCos(30.0, sin, cos);
20
21     std::cout << "The sin is " << sin << '\n';
22     std::cout << "The cos is " << cos << '\n';
23     return 0;
24 }
```

Эта функция принимает один параметр (передача по значению) в качестве входных данных и «возвращает» два параметра (передача по ссылке) в качестве выходных данных. Параметры, которые используются только для возврата значений обратно в caller, называются **параметрами вывода**. Они дают понять caller-у, что значения исходных переменных, переданных в функцию, не столь значительны, так как мы ожидаем, что эти переменные будут перезаписаны.

Давайте рассмотрим это детально. Во-первых, в функции `main()` мы создаем локальные переменные `sin` и `cos`. Они передаются в функцию `getSinCos()` по ссылке (а не по значению). Это означает, что функция `getSinCos()` имеет прямой доступ к исходным значениям переменных `sin` и `cos`, а не к их копиям.

Функция `getSinCos()`, соответственно, присваивает новые значения переменным `sin` и `cos` (через ссылки `sinOut` и `cosOut`), перезаписывая их старые значения. Затем `main()` выводит эти обновленные значения.

Если бы `sin` и `cos` были переданы по значению, а не по ссылке, то функция `getSinCos()` изменила бы копии `sin` и `cos`, а не исходные значения и эти изменения уничтожились бы в конце функции — переменные вышли бы из **локальной области видимости**. Но, поскольку `sin` и `cos` передавались по ссылке, любые изменения, внесенные в `sin` или `cos` (через ссылки), сохраняются и за пределами функции `getSinCos()`. Таким образом, мы можем использовать этот механизм для возврата сразу нескольких значений обратно в `caller`.

Хотя этот способ хорош, но он также имеет свои нюансы. Во-первых, синтаксис немного непривычен, так как параметры ввода и вывода указываются вместе с вызовом функции. Во-вторых, в `caller`-е не очевидно, что `sin` и `cos` являются параметрами вывода, и они будут изменены функцией. Это, вероятно, самая опасная часть данного способа передачи (так как может привести к ошибкам). Некоторые программисты считают, что это достаточно большая проблема, и не советуют передавать аргументы по ссылке, отдав предпочтение передаче по адресу, не смешивая при этом параметры ввода и вывода.

Лично я не рекомендую смешивать параметры ввода и вывода именно по этой причине, но если вы это делаете, то обязательно добавляйте **комментарии к коду**, описывая, что вы делаете и как это делаете.

Неконстантные ссылки могут ссылаться только на неконстантные ***l-values*** (например, на неконстантные переменные), поэтому параметр-ссылка не может принять аргумент, который является константным *l-value* или *r-value* (например, литералом или результатом выражения).

Передача по константной ссылке

Одним из самых главных недостатков передачи по значению является то, что все аргументы, переданные по значению, *копируются* в параметры функции. Когда аргументами являются большие структуры или классы, то этот процесс может занять много времени. В случае с передачей по ссылке эта проблема легко решается. Когда аргумент передается по ссылке, то создается ссылка на фактический аргумент (что занимает минимальное количество времени на выполнение), и никакого копирования значений не происходит. Это позволяет передавать большие структуры или классы с минимальной затратой ресурсов.

Однако здесь также могут возникнуть потенциальные проблемы. Ссылки позволяют функции изменять значения аргументов напрямую, что нежелательно, если мы хотим, чтобы аргумент был доступен только для чтения. Когда мы знаем, что функция не должна изменять значение аргумента, но не хотим использовать передачу по значению, то лучшим решением будет использовать **передачу по константной ссылке**.

Вы уже знаете, что **константная ссылка** — это ссылка на переменную, значение которой изменить через эту же ссылку не получится никак. Следовательно, если мы используем константную ссылку в качестве параметра, то получаем 100% гарантию того, что функция не изменит аргумент!

Запустив следующий фрагмент кода, мы получим ошибку компиляции:

```
1 void boo(const int &y) // y - это константная ссылка
2 {
3     y = 8; // ошибка компиляции: константная ссылка не может изменить свое же значение!
4 }
```

Использование `const` полезно по нескольким причинам:

- ➔ Мы получаем гарантию от компилятора, что значения, которые не должны быть изменены — не изменятся (компилятор выдаст ошибку, если мы попытаемся сделать нечто подобное тому, что было в вышеприведенном примере).
- ➔ Программист, видя `const`, понимает, что функция не изменит значение аргумента. Это может помочь при [отладке программы](#).
- ➔ Мы не можем передать константный аргумент в неконстантную ссылку-параметр. Использование константного параметра гарантирует, что мы сможем передавать как неконстантные, так и константные аргументы в функцию.
- ➔ Константные ссылки могут принимать любые типы аргументов, включая l-values, константные l-values и r-values.

Правило: При передаче аргументов по ссылке всегда используйте константные ссылки, если вам не нужно, чтобы функция изменяла значения аргументов.

Плюсы и минусы передачи по ссылке

Плюсы передачи по ссылке:

- ✚ Ссылки позволяют функции изменять значение аргумента, что иногда полезно. В противном случае, для гарантии того, что функция не изменит значение аргумента, нужно использовать константные ссылки.
- ✚ Поскольку при передаче по ссылке копирования аргументов не происходит, то этот способ гораздо эффективнее и быстрее передачи по значению, особенно при работе с большими структурами или классами.
- ✚ Ссылки могут использоваться для возврата сразу нескольких значений из функции (через параметры вывода).

Минусы передачи по ссылке:

- Трудно определить, является ли параметр, переданный по неконстантной ссылке, параметром ввода, вывода или того и другого одновременно. Разумное использование `const` и суффикса `Out` для внешних переменных решает эту проблему.
- По вызову функции невозможно определить, будет аргумент изменен функцией или нет. Аргумент, переданный по значению или по ссылке, выглядит одинаково. Мы можем определить способ передачи аргумента только просмотрев объявление функции. Это может привести к ситуации, когда программист не сразу поймет, что функция изменяет значение аргумента.

Когда использовать передачу по ссылке:

- ➔ при передаче структур или классов (используйте `const`, если нужно только для чтения);

→ когда нужно, чтобы функция изменяла значение аргумента.

Когда не использовать передачу по ссылке:

- при передаче фундаментальных типов данных (используйте передачу по значению);
- при передаче обычных **массивов** (используйте передачу по адресу).

Оценить статью:

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



← [Урок №97. Передача по значению](#)

[Урок №99. Передача по адресу](#) →



Комментариев: 3

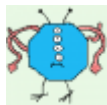


1. *smigles:*

[12 июля 2018 в 09:57](#)

Почему автор не советует передавать фундаментальные типы по ссылке? Ведь тогда тоже не произойдёт ненужного копирования значения. А чтобы нельзя было изменить значение аргумента, достаточно к параметру добавить const.

[Ответить](#)

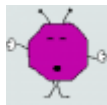


1. *kmish:*

[13 февраля 2019 в 13:29](#)

Вероятно потому, что памяти для создания ссылки требуется больше, чем для фундаментальных типов, ну или по крайней мере не меньше.

[Ответить](#)



1. *Александр:*

[24 февраля 2019 в 16:01](#)

вряд-ли...

все понятия, с которыми нам удобно работать при написании программ: ссылки, переменные, константы и тому подобное, существуют только до того, как отработает компилятор.

Когда мы что-то объявляем константой, то запрет на изменения этого чего-то нам ставит компилятор... соответственно как-то дополнительно отмечать константы или ссылки в итоговом коде смысла никакого нет

Скорее всего тут больше вопрос семантики, а не производительности. Причем есть совершенно разные рекомендации. В некоторых вариантах этих рекомендаций вообще императивно заявляется "передавать в функцию можно только ссылки или константные ссылки. Никаких копий и никаких исключений". Часто можно в коде увидеть константную ссылку в заголовке функции и в первой же строке создание копии переданной ссылки.

[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