

Ravesli [Ravesli](#)


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

Урок №88. Ссылки

 [Юрий](#) |

- [Уроки C++](#)

|

 Обновл. 16 Авг 2020 |

 64077

[1](#)  [5](#)

До этого момента мы успели рассмотреть 2 основных типа переменных:

- ➔ обычные переменные, которые хранят значения напрямую;
- ➔ **указатели**, которые хранят адрес другого значения (или [null](#)), для доступа к которым выполняется операция разыменования указателя.

Ссылки — это третий базовый тип переменных в языке C++.

Оглавление:

1. [Ссылки](#)
2. [Ссылки в качестве псевдонимов](#)
3. [Краткий обзор l-value и r-value](#)
4. [Инициализация ссылок](#)
5. [Ссылки в качестве параметров в функциях](#)
6. [Ссылки как более легкий способ доступа к данным](#)
7. [Ссылки vs. Указатели](#)
8. [Заключение](#)

Ссылки

Ссылка — это тип переменной в языке C++, который работает как псевдоним другого объекта или значения. **Язык C++ поддерживает 3 типа ссылок:**

- **Ссылки на неконстантные значения** (обычно их называют просто «ссылки» или «неконстантные ссылки»), которые мы обсудим на этом уроке.
- **Ссылки на константные значения** (обычно их называют «константные ссылки»), которые мы обсудим на следующем уроке.
- В C++11 добавлены **ссылки r-value**, о которых мы поговорим чуть позже.

Ссылка (на неконстантное значение) объявляется с использованием амперсанда (&) между типом данных и именем ссылки:

```
1 int value = 7; // обычная переменная
2 int &ref = value; // ссылка на переменную value
```

В этом контексте амперсанд не означает «оператор адреса», он означает «ссылка на».

Ссылки в качестве псевдонимов

Ссылки обычно ведут себя идентично значениям, на которые они ссылаются. В этом смысле ссылка работает как псевдоним объекта, на который она ссылается, например:

```
1 #include <iostream>
2
3 int main()
4 {
5     int value = 7; // обычная переменная
6     int &ref = value; // ссылка на переменную value
7
8     value = 8; // value теперь 8
9     ref = 9; // value теперь 9
10
11     std::cout << value << std::endl; // выведется 9
12     ++ref;
13     std::cout << value << std::endl; // выведется 10
14
15     return 0;
16 }
```

Результат выполнения программы:

```
9
10
```

В примере, приведенном выше, объекты `ref` и `value` обрабатываются как одно целое. Использование оператора адреса с ссылкой приведет к возврату адреса значения, на которое ссылается ссылка:

```
1 std::cout << &value; // выведется 0035FE58
2 std::cout << &ref; // выведется 0035FE58
```

Краткий обзор l-value и r-value

На [уроке №10](#) мы уже рассматривали, что такое l-value и r-value. l-value — это объект, который имеет определенный адрес памяти (например, переменная `x`) и сохраняется за пределами одного выражения. r-value — это временное значение без определенного адреса памяти и с областью видимости выражения (т.е. сохраняется в пределах одного выражения). В качестве r-values могут быть как результаты выражения (например, `2 + 3`), так и [литералы](#).

Инициализация ссылок

Ссылки должны быть инициализированы при создании:

```
1 int value = 7;
2 int &ref = value; // корректная ссылка: инициализирована переменной value
3
4 int &invalidRef; // некорректная ссылка: ссылка должна ссылаться на что-либо
```

В отличие от указателей, которые могут содержать нулевое значение, ссылки нулевыми быть не могут.

Ссылки на неконстантные значения могут быть инициализированы только неконстантными l-values. Они не могут быть инициализированы константными l-values или r-values:

```
1 int a = 7;
2 int &ref1 = a; // ок: a - это неконстантное l-value
3
4 const int b = 8;
5 int &ref2 = b; // не ок: b - это константное l-value
6
7 int &ref3 = 4; // не ок: 4 - это r-value
```

Обратите внимание, во втором случае вы не можете инициализировать неконстантную ссылку константным объектом. В противном случае, вы бы могли изменить значение константного объекта через ссылку, что уже является нарушением понятия «константа».

После инициализации изменить объект, на который указывает ссылка — нельзя. Рассмотрим следующий фрагмент кода:

```
1 int value1 = 7;
2 int value2 = 8;
3
4 int &ref = value1; // ок: ref - теперь псевдоним для value1
5 ref = value2; // присваиваем 8 (значение переменной value2) переменной value1. Здесь НЕ
```

Обратите внимание, во втором кейтменте (`ref = value2;`) выполняется не то, что вы могли бы ожидать! Вместо переприсваивания `ref` (ссылаться на переменную `value2`), значение из `value2` присваивается переменной `value1` (на которое и ссылается `ref`).

Ссылки в качестве параметров в функциях

Ссылки чаще всего используются в качестве **параметров** в функциях. В этом контексте ссылка-параметр работает как псевдоним аргумента, а сам аргумент не копируется при передаче в параметр. Это в свою очередь улучшает производительность, если аргумент слишком большой или затратный для копирования.

На [уроке №82](#) мы говорили о том, что передача аргумента-указателя в функцию позволяет функции при разыменовании этого указателя *напрямую* изменять значение аргумента.

Ссылки работают аналогично. Поскольку ссылка-параметр — это псевдоним аргумента, то функция, использующая ссылку-параметр, может изменять аргумент, переданный ей, также *напрямую*:

```
1  #include <iostream>
2
3  // ref - это ссылка на переданный аргумент, а не копия аргумента
4  void changeN(int &ref)
5  {
6      ref = 8;
7  }
8
9  int main()
10 {
11     int x = 7;
12
13     std::cout << x << '\n';
14
15     changeN(x); // обратите внимание, этот аргумент не обязательно должен быть ссылкой
16
17     std::cout << x << '\n';
18     return 0;
19 }
```

Результат выполнения программы:

```
7
8
```

Когда аргумент `x` передается в функцию, то параметр функции `ref` становится ссылкой на аргумент `x`. Это позволяет функции изменять значение `x` непосредственно через `ref`! Обратите внимание, переменная `x` не обязательно должна быть ссылкой.

Совет: Передавайте аргументы в функцию через неконстантные ссылки-параметры, если они должны быть изменены функцией в дальнейшем.

Основным недостатком использования неконстантных ссылок в качестве параметров в функциях является то, что аргумент должен быть неконстантным l-value (т.е. константой или литералом он быть не может). Мы поговорим об этом подробнее (и о том, как это обойти) на следующем уроке.

Ссылки как более легкий способ доступа к данным

Второе (гораздо менее используемое) применение ссылок заключается в более легком способе доступа к вложенным данным. Рассмотрим следующую [структуру](#):

```
1 struct Something
2 {
3     int value1;
4     float value2;
5 };
6
7 struct Other
8 {
9     Something something;
10    int otherValue;
11 };
12
13 Other other;
```

Предположим, что нам нужно работать с полем `value1` структуры `Something` переменной `other` структуры `Other` (звучит сложно, но такое также встречается на практике). Обычно, доступ к этому полю осуществлялся бы через `other.something.value1`. А что, если нам нужно неоднократно получать доступ к этому члену? В этом случае код становится громоздким и беспорядочным. Ссылки же предоставляют более легкий способ доступа к данным:

```
1 int &ref = other.something.value1;
2 // ref теперь может использоваться вместо other.something.value1
```

Таким образом, следующие два стейтмента идентичны:

```
1 other.something.value1 = 7;
2 ref = 7;
```

Ссылки позволяют сделать ваш код более чистым и понятным.

Ссылки vs. Указатели

Ссылка — это тот же указатель, который неявно разыменовывается при доступе к значению, на которое он указывает («под капотом» ссылки реализованы с помощью указателей). Таким образом, в следующем коде:

```
1 int value = 7;
2 int *const ptr = &value;
3 int &ref = value;
```

`*ptr` и `ref` обрабатываются одинаково. Т.е. это одно и то же:

```
1 *ptr = 7;
2 ref = 7;
```

Поскольку ссылки должны быть инициализированы корректными объектами (они не могут быть нулевыми) и не могут быть изменены позже, то они, как правило, безопаснее указателей (так как риск разыменования нулевого указателя отпадает). Однако они немного ограничены в функциональности по сравнению с указателями.

Если определенное задание может быть решено с помощью как ссылок, так и указателей, то лучше использовать ссылки. Указатели следует использовать только в тех ситуациях, когда ссылки являются недостаточно эффективными (например, при [динамическом выделении памяти](#)).

Заключение

Ссылки позволяют определять псевдонимы для других объектов или значений. Ссылки на неконстантные значения могут быть инициализированы только неконстантными l-values. Они не могут быть переприсвоены после инициализации. Ссылки чаще всего используются в качестве параметров в функциях, когда мы хотим изменить значение аргумента или хотим избежать его затратного копирования.

Оценить статью:

★★★★★ (309 оценок, среднее: 4,92 из 5)



[← Урок №87. Указатели и const](#)

[Урок №89. Ссылки и const →](#)



Комментариев: 5



1. *SASF:*

[4 ноября 2020 в 01:54](#)

Можно узнать более подробно о разнице ссылок и указателей, в особенности при оптимизации кода

[Ответить](#)



2. *Константин:*

[12 июня 2020 в 22:52](#)

А кстати, только у меня такая фишка что если сделать константную ссылку на неконстантную переменную, то при изменении оригинальной переменной будет меняться и ссылка на неё? Хотя по

идея это логично, но если бы это было не так, а ссылка хранила значение, записанное в неё во время создания, можно было бы это использовать довольно много где

[Ответить](#)



3. *Алексей:*

[20 августа 2019 в 14:37](#)

Самое забавное, что в игре Doom'16 похоже разработчики просто забыли освободить ячейки памяти.

20 мин работаешь в их SnapMap и у тебя на 1-2 гига памяти больше требуется, так же с видео памятью — увеличено потребление.

Вот уж не знаю если закончу C++ здесь, что будет далее.

bash тоже не любил, сейчас просто незаменим при обработке данных.

[Ответить](#)



1. *Sergey:*

[12 января 2020 в 13:29](#)

А bash как учили-полюбили? книгу или сайт посоветуйте

[Ответить](#)



2. *Ivan:*

[7 мая 2020 в 16:54](#)

bash незаменим для обработки данных? Вы серьёзно? Он уже морально и технически устарел. Даже для нужд администрирования Python вместо него с радостью применяют.

[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.

Отправить комментарий

[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