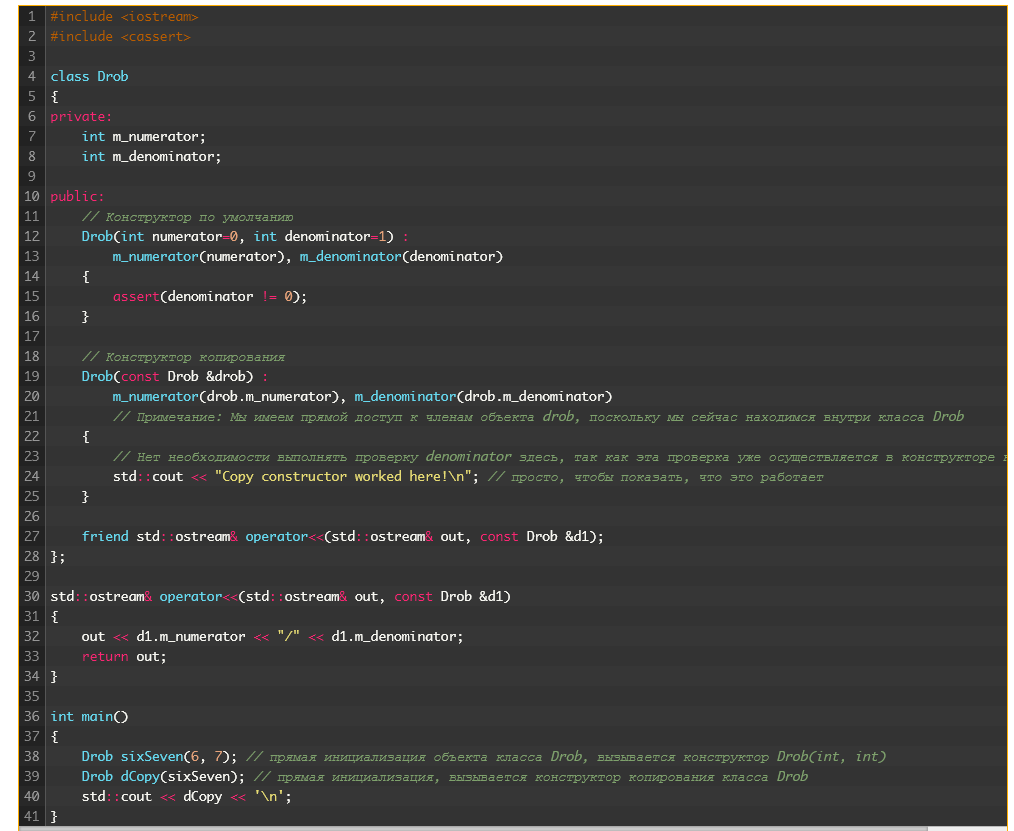
**Конструктор копирования**

Конструктор копирования — это особый тип конструктора, который используется для создания нового объекта через копирование существующего объекта. И, как в случае с конструктором по умолчанию, если вы не предоставите конструктор копирования для своих классов самостоятельно, то язык C++ создаст [public](https://ravesli.com/urok-114-spetsifikatory-dostupa-public-i-private/)-конструктор копирования автоматически. Поскольку компилятор мало знает о вашем классе, то по умолчанию созданный конструктор копирования будет использовать почленную инициализацию. Почленная инициализация означает, что каждый член объекта-копии инициализируется непосредственно из члена объекта-оригинала. Т.е. в примере, приведенном выше, dCopy.m\_numerator будет иметь значение sixSeven.m\_numerator (6), а dCopy.m\_denominator будет равен sixSeven.m\_ denominator (7).

Так же, как мы можем явно определить конструктор по умолчанию, так же мы можем явно определить и конструктор копирования. Конструктор копирования выглядит следующим образом:



## Предотвращение создания копий объектов

Мы можем предотвратить создание копий объектов наших классов, сделав конструктор копирования закрытым:



## Конструктор копирования может быть проигнорирован

Рассмотрим следующий код:



Дело в том, что инициализация анонимного объекта, а затем использование этого объекта для прямой инициализации уже не анонимного объекта выполняется в два этапа (первый этап — это создание анонимного объекта, второй этап — это вызов конструктора копирования). Однако, конечный результат по сути идентичен простому выполнению прямой инициализации, которая занимает всего лишь один шаг.

По этой причине в таких случаях компилятору разрешается отказаться от вызова конструктора копирования и просто выполнить прямую инициализацию. Этот процесс называется элизией.

Поэтому, даже если вы напишите:

Drob sixSeven(Drob(6, 7);

Компилятор может изменить это на:

Drob sixSeven(6, 7);

## **Поверхностное копирование**

Поскольку язык C++ не может знать наперед всё о вашем [классе](https://ravesli.com/urok-113-klassy-obekty-i-metody-klassov/), то [конструктор копирования](https://ravesli.com/urok-141-konstruktor-kopirovaniya/) и оператор присваивания, которые C++ предоставляет по умолчанию, используют почленный метод копирования — поверхностное копирование. Это означает, что C++ выполняет копирование для каждого члена класса индивидуально (используя оператор присваивания по умолчанию вместо [перегрузки оператора присваивания](https://ravesli.com/urok-144-peregruzka-operatora-prisvaivaniya/) и прямую инициализацию вместо конструктора копирования). Когда классы простые (например, в них нет членов с [динамически выделенной памятью](https://ravesli.com/urok-85-dinamicheskoe-vydelenie-pamyati-operatory-new-i-delete/)), то никаких проблем с этим не должно возникать.

Однако при работе с классами, в которых динамически выделяется память, почленное (поверхностное) копирование может вызывать проблемы! Это связано с тем, что при поверхностном копировании [указателя](https://ravesli.com/urok-80-ukazateli-vvedenie/) копируется только адрес указателя — никаких действий по содержимому адреса указателя не предпринимается.

## **Глубокое копирование**

Одним из решений этой проблемы является выполнение глубокого копирования. При глубоком копировании память сначала выделяется для копирования адреса, который содержит исходный указатель, а затем для копирования фактического значения. Таким образом копия находится в отдельной, от исходного значения, памяти и они никак не влияют друг на друга. Для выполнения глубокого копирования нам необходимо написать свой собственный конструктор копирования и перегрузку оператора присваивания.

