**std::remove\_reference.**

Определено в заголовке <type\_traits>

template<class T> struct remove\_reference; // since c++11

Если тип T является ссылочным, то член структуры type будет также являться ссылочным. В противном случае, type будет являеться типом значения T.

Возможные специализации:

template< class T > struct remove\_reference {typedef T type;};

template< class T > struct remove\_reference<T&> {typedef T type;};

template< class T > struct remove\_reference<T&&> {typedef T type;};

Необходимость определения всех трех структур поясняется на форуме [stackoverflow](https://stackoverflow.com/questions/17000179/stdremove-reference-explained).

Зачем нужны специализации для lvalue- и rvalue- ссылок?

Если бы существовал только первый шаблон, то инстанс

std::remove\_reference<int&>::type

даст вам

int&

а инстанс

std::remove\_reference<int&&>::type

даст вам

int&&

не то, что вы хотели.

Специализации для lvalue- и rvalue- ссылок позволяют вам удалить ссылочность & и && соответственно из типа аргумента, который вы передаете. Например, инстанс с аргументом int&& будет соответствовать шаблону со специализацией T&&, для которой тип T будет int.

Почему мы должны выполнять приведение к типу std::remove\_reference<T>::type&& в move()?

Это необходимо потому, что move() определена как шаблон вида:

template<typename T>

T&& move(T&& t) { ... }

// ^^^

// Resolves to X& if T is X& (which is the case if the input has type X

// and is an lvalue)

NB: по правилу вывода типа №2 при передачи в функцию lvalue, тип аргумента и возвращаемого значения будет выведен как lvalue-ссылка.

Таким образом, если шаблон будет инстанцирован lvalue-ссылкой X&, то функция будет также возвращать lvalue-ссылку на тип X. Но мы хотим, чтобы возвращаемый тип всегда был rvalue-ссылкой. Ведь задача функции move() вернуть вам именно rvalue, независимо от того, что вы передадите ей на вход. Именно поэтому мы приводим к типу remove\_reference<T>::type&&, поскольку добавление двойного амперсанда && к не-ссылочному типу всегда даст тип rvalue-ссылки.

Можно ли каким-то образом напечатать тип?

Не существует универсального способа преобразования имени типа в строку. Если вашей целью является подтверждение факта передачи rvalue, вы можете использовать static assertion вида:

#include <type\_traits>

template<typename T>

void foo(T&&)

{

static\_assert(!std::is\_reference<T>::value, "Error: lvalue was passed!");

// ...

}

Это работает на основе того факта, что если вы передадите в foo lvalue X, то тип T будет выведен как X& (NB: в этом случае выражение в static\_assert будет являтсья ложью и произойдет ошибка компиляции).