

Ravesli [Ravesli](#)


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

Урок №60. Псевдонимы типов: typedef и type alias

 [Юрий](#) |

- [Уроки C++](#)

|

 Обновл. 23 Сен 2020 |

 46654

[14](#)

На этом уроке мы рассмотрим псевдонимы типов typedef и type alias в языке C++.

Оглавление:

1. [typedef](#)
2. [typedef и читабельность кода](#)
3. [typedef и поддержка кода](#)
4. [typedef и кроссплатформенность](#)
5. [typedef и упрощение сложного](#)
6. [type alias](#)
7. [Тест](#)

typedef

Ключевое слово typedef позволяет программисту создать псевдоним для любого типа данных и использовать его вместо фактического имени типа. Чтобы объявить typedef (использовать псевдоним типа) — используйте ключевое слово typedef вместе с типом данных, для которого создается псевдоним, а затем, собственно, сам псевдоним. Например:

```
1 typedef double time_t; // используем time_t в качестве псевдонима для типа double
2
3 // Следующие два стейтмента эквивалентны
4 double howMuch;
5 time_t howMuch;
```

Обычно к псевдонимам typedef добавляют окончание `_t`, указывая, таким образом, что идентификатором является тип, а не переменная.

typedef не определяет новый тип данных. Это просто псевдоним (другое имя) для уже существующего типа. Его можно использовать везде, где используется обычный тип.

Даже если следующее не имеет смысла, оно все равно разрешено в языке C++:

```
1 typedef long miles_t;
2 typedef long speed_t;
3
4 miles_t distance = 8;
5 speed_t phr = 2100;
6
7 // Следующее разрешено, поскольку обе переменные distance и phr являются типа long
8 distance = phr;
```

typedef и читабельность кода

typedef используется в улучшении документации и разборчивости кода. Имена таких типов, как `char`, `int`, `long`, `double` и `bool` хороши для описания того, какой тип возвращает функция, но чаще всего мы хотим знать, с какой целью возвращается значение. Например, рассмотрим следующую функцию:

```
1 int GradeTest();
```

Мы видим, что возвращаемым значением является целое число, но что оно означает? Количество пропущенных вопросов? Идентификационный номер учащегося? Код ошибки? Сам `int` ни о чем нам не говорит. Исправим ситуацию:

```
1 typedef int testScore_t;
2 testScore_t GradeTest();
```

С использованием возвращаемого типа `testScore_t` становится очевидным, что функция возвращает тип, значением которого является результат теста.

typedef и поддержка кода

typedef также позволяет изменить базовый тип объекта без внесения изменений в большое количество кода. Например, если вы использовали тип `short` для хранения идентификационного номера учащегося, но потом решили, что лучше использовать тип `long`, то вам придется прошерстить кучу кода для замены `short` на `long`. И, вероятно, было бы трудно определить, какой из типов `short` используется для хранения идентификационных номеров, а какой — для других целей.

С typedef же всё, что вам нужно сделать, — это изменить объявление `typedef short studentID_t` на `typedef long studentID_t`. Тем не менее, не стоит забывать об осторожности при изменении типа typedef на тип из другого семейства (например, из `int` на `float` или наоборот)! Новый тип данных может

иметь проблемы со сравнением или делением целых чисел/чисел типа с плавающей точкой, которых старый тип не имел — об этом следует помнить.

typedef и кроссплатформенность

Еще одним большим преимуществом typedef является возможность скрывать специфические для определенных платформ (операционных систем) детали. На некоторых платформах тип `int` занимает 2 байта, на других — 4 байта. Таким образом, использование типа `int` для хранения более 2 байтов информации может быть потенциально опасным при написании кроссплатформенного кода.

Поскольку `char`, `short`, `int` и `long` не указывают свой размер, то для кроссплатформенных программ довольно часто используется typedef для определения псевдонимов, которые включают размер типа данных в битах. Например, `int8_t` — это 8-битный signed int, `int16_t` — это 16-битный signed int, а `int32_t` — это 32-битный signed int.

typedef и упрощение сложного

Хотя мы до сих пор рассматривали только простые типы данных, в языке C++ вы можете увидеть и следующие переменные/функции:

```
1 std::vector<std::pair<std::string, int>> pairlist;
2
3 boolean hasAttribute(std::vector<std::pair<std::string, int>> pairlist)
4 {
5     // Что-то делаем
6 }
```

Писать `std::vector<std::pair<std::string, int>>` всякий раз, когда нужно использовать этот тип — не очень эффективно и затратно как по времени, так и по приложенным усилиям. Гораздо проще использовать typedef:

```
1 typedef std::vector<std::pair<std::string, int>> pairlist_t; // используем pairlist_t в
2
3 pairlist_t pairlist; // объявляем pairlist_t
4
5 boolean hasAttribute(pairlist_t pairlist) // используем pairlist_t в качестве типа пара
6 {
7     // Что-то делаем
8 }
```

Вот! Другое дело! Ведь проще использовать `pairlist_t` вместо `std::vector<std::pair<std::string, int>>`, не так ли?

Не переживайте, если вы еще не знаете, что такое `std::vector`, `std::pair` и прочее. Гораздо важнее сейчас усвоить, что с помощью typedef вы можете давать простые имена сложным типам данных, что делает их проще как для использования, так и для понимания.

type alias

У typedef есть также свои нюансы. Во-первых, легко забыть, что пишется первым: псевдоним типа или имя типа:

```
1 typedef time_t double; // неправильно
2 typedef double time_t; // правильно
```

Во-вторых, синтаксис typedef становится уже менее привлекательным в связке со сложными типами данных (об этом мы поговорим детально, когда будем рассматривать указатели на функции).

Для решения этих проблем, в C++11 ввели новый улучшенный синтаксис для typedef, который имитирует способ объявления переменных. Этот синтаксис называется **type alias**. С помощью **type alias** мы пишем имя, которое затем используется как синоним конкретного типа данных (т.е. принцип тот же, но синтаксис более удобен).

Следующий typedef:

```
1 typedef double time_t; // используем time_t в качестве псевдонима для типа double
```

В C++11 можно объявить как:

```
1 using time_t = double; // используем time_t в качестве псевдонима для типа double
```

Эти два способа функционально эквивалентны.

Обратите внимание, что хоть мы и используем ключевое слово **using**, оно не имеет ничего общего с [using-стейтментами](#). Это ключевое слово имеет различный функционал в зависимости от контекста.

Новый синтаксис создания псевдонимов создает меньше проблем при использовании в сложных ситуациях, и его рекомендуется применять вместо обычного typedef, если ваш компилятор поддерживает C++11.

Правило: Используйте type alias вместо typedef, если ваш компилятор поддерживает C++11.

Тест

Задание №1

Используя следующий [прототип функции](#):

```
1 int editData();
```

Преобразуйте тип возвращаемого значения `int` в `status_t`, используя ключевое слово `typedef`. В ответе к этому заданию укажите стейтмент `typedef` и обновленный прототип функции.

Ответ №1

```
1 typedef int status_t;  
2 status_t editData();
```

Задание №2

Используя прототип функции из задания №1, преобразуйте тип возвращаемого значения `int` в `status_t`, используя ключевое слово `using` (C++11). В ответе к этому заданию укажите стейтмент создания псевдонима типа и обновленный прототип функции.

Ответ №2

```
1 using status_t = int;  
2 status_t editData();
```

Оценить статью:

★★★★★ (295 оценок, среднее: 4,94 из 5)



← [Урок №59. Классы enum](#)

[Урок №61. Структуры](#) →



Комментариев: 14



1. Руслан:

[15 июля 2020 в 21:34](#)

Здравствуйте! Вопрос про typedef и кроссплатформенность. В уроке №32 «Фиксированный размер целочисленных типов данных», говорилось чтобы решить вопрос кроссплатформенности, в язык C++ добавили набор целочисленных типов фиксированного размера, которые гарантированно имеют один и тот же размер на любой архитектуре (`#include <cstdint>`). В этом уроке говорится что можно использовать псевдоним, например, `int8_t` — это 8-битный signed int. Как видится мне в этом случае за размером типа данных, вернее за его наполнением, придется следить самому программисту. При этом псевдоним `int8_t` и т.п. совпадает с названием целочисленного типа фиксированного размера (что при подключенной библиотеке `<cstdint>` потенциально может привести к конфликту имён).

Что целесообразнее (и (или) безопаснее) использовать для обеспечения кроссплатформенности псевдоним типа или целочисленный тип фиксированного размера?

[Ответить](#)



2. *Ya:*

[25 июля 2019 в 10:03](#)

```
1 | std::vector<std::pair<std::string, int> > pairlist;
```

Поясните пожалуйста мне, недалекому, что в этой строке происходит.

[Ответить](#)



1. *Алексей:*

[16 января 2020 в 19:56](#)

Это вектор пар вида (строка, число)

[Ответить](#)



2. *Демьян:*

[21 октября 2020 в 14:33](#)

Раскройте скобки постепенно, начните с правой части. Мы хотим создать пару строка-число, затем вектор (этакий продвинутый список) получает уже эту пару в качестве элементов

[Ответить](#)



3. *Алексей:*

[15 июля 2019 в 17:56](#)

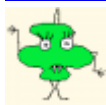
Тесты на ура пошли.

Елки-палки, наворотить можно, но с опытом с этим научись работать и будет тебе счастье.

Оптимизация и работоспособность — цель всех активностей в создании чего либо.

Спасибо за курс, хочу до конца изучить, for sure!

[Ответить](#)



4. *Алексей:*

[15 июля 2019 в 16:37](#)

А что тут переживать. Я знаю, что много не знаю по c++.

Я вижу, что простые вещи понять и принять — буду через пол года писать, то что в снах не присниться и это будет оптимизировано, работать, работать быстро.

К чему всегда и стремлюсь.

[Ответить](#)



5. *deb:*

[18 мая 2019 в 12:05](#)

Очень крутой понятный урок.

[Ответить](#)



1. *Юрий:*

[6 июня 2019 в 15:07](#)

Пасиб 😊

[Ответить](#)



6. *Геннадий:*

[27 февраля 2019 в 14:24](#)

Цель программиста- выразить вычисления, причем это должно быть сделано:

- правильно;
- просто;
- эффективно.

(Бьярне Страуструп. "Программирование". Второе издание. стр.156)

[Ответить](#)



7. *benya:*

[1 февраля 2019 в 09:26](#)

Зачем нужен typedef, когда мы можем того же достичь с помощью define?

[Ответить](#)



1. *Tosha:*

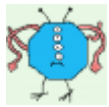
[4 февраля 2019 в 17:47](#)

typedef имеет контекст и не конфликтует с другими именами в коде. Например `typedef int a;` `a a = 1;` — вполне корректная запись, хоть и извращенная. В объявлении переменной первый идентификатор является ее типом, а второй — именем. Используя макросы, мы бы переписали и тип переменной, и ее имя, что завершилось бы ошибкой компиляции. Разумеется, подобное использовать не стоит, я просто попытался показать, что typedef дает больше возможностей. Другой пример — ты можешь использовать в разных пространствах имен одно и то же имя для псевдонима разных типов, более точно отображая смысл своего кода.

Если возникает идея, что так можно себя больше запутать — стоит вспомнить, что никто не обязует эти возможности использовать. Идеология языка — предоставить инструменты, а

правильное и безопасное их использование — это уже забота программиста (Хотя, предупреждения компилятора зачастую помогают избавиться от значительной доли потенциальных проблем).

[Ответить](#)



8. *kmish:*

[31 января 2019 в 13:39](#)

Добрый день, все понятно и прозрачно.

Но есть небольшое субъективное замечание, которое мне видится.

```
1 | int GradeTest();
```

Разве имя функции не должно объяснять, что она возвращает (как Вы ранее писали в одном из уроков)?

В данном случае, делая псевдоним еще и для типа, получается избыточность. По функции итак отлично понятно, что она результат теста дает и `int` тут вполне уместен.

[Ответить](#)



1. *Руслан:*

[11 июля 2019 в 12:53](#)

`int`, да уместен, а вот "`std::vector<std::pair<std::string, int>`" ну триндец как не уместен и порой возможно не понятен. И слава `typedef`, что можно просто написать вместо:

```
1 | std::vector<std::pair<std::string, int> > pairlist;
```

и вместо гемороя выше, подкорректировать в:

```
1 | typedef std::vector<std::pair<std::string, int> > pairlist_t;
```

что бы стало красивым и читабельным:

```
1 | pairlist_t pairlist;
```

[Ответить](#)



1. *Артеми:*

[25 февраля 2020 в 14:03](#)

Просто продолжай читать...

[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