

Ravesli [Ravesli](#)


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


Урок №62. Вывод типов: ключевое слово auto

 [Юрий](#) |

- [Уроки C++](#)

|

 Обновл. 15 Сен 2020 |

 54633

[↑](#)  8

На этом уроке мы рассмотрим вывод типов с помощью ключевого слова auto в языке C++.

Оглавление:

1. [Вывод типов в C++11](#)
2. [Ключевое слово auto и параметры функций](#)
3. [Вывод типов в C++14](#)
4. [trailing-синтаксис в C++11](#)
5. [Заключение](#)

Вывод типов в C++11

До C++11 ключевое слово auto было наименее используемым ключевым словом в языке C++. Из [урока №48](#) мы уже знаем, что локальные переменные имеют автоматическую продолжительность жизни (создаются в точке определения и уничтожаются в конце блока, в котором определены).

Ключевое слово auto использовалось для явного указания, что переменная должна иметь автоматическую продолжительность жизни:

```
1 int main()
2 {
3     auto int boo(7); // явно указываем, что переменная boo типа int должна иметь автома
4
5     return 0;
6 }
```

Однако, поскольку все переменные в новых версиях языка C++ по умолчанию имеют автоматическую продолжительность жизни (если явно не указать другой [тип продолжительности жизни](#)), ключевое слово `auto` стало лишним и, следовательно, устаревшим.

В C++11 значение ключевого слова `auto` изменилось. Рассмотрим следующий кейс:

```
1 double x = 4.0;
```

Если C++ и так знает, что `4.0` является литералом [типа `double`](#), то зачем нам дополнительно указывать, что переменная `x` должна быть типа `double`? Правда, было бы неплохо, если бы мы могли указать переменной принять соответствующий тип данных, основываясь на инициализируемом значении?

Начиная с C++11, **ключевое слово `auto`** при инициализации переменной может использоваться вместо типа переменной, чтобы сообщить компилятору, что он должен присвоить тип переменной исходя из инициализируемого значения. Это называется **выводом типа** (или «*автоматическим определением типа данных компилятором*»). Например:

```
1 auto x = 4.0; // 4.0 - это литерал типа double, поэтому и x должен быть типа double
2 auto y = 3 + 4; // выражение 3 + 4 обрабатывается как целочисленное, поэтому и переменная y будет типа int
```

Это работает даже с возвращаемыми значениями функций:

```
1 int subtract(int a, int b)
2 {
3     return a - b;
4 }
5
6 int main()
7 {
8     auto result = subtract(4, 3); // функция subtract() возвращает значение типа int и
9     return 0;
10 }
```

Обратите внимание, это работает только с инициализированными переменными. Переменные, объявленные без инициализации, не могут использовать эту особенность (поскольку нет инициализируемого значения, и компилятор не может знать, какой тип данных присвоить переменной).

Используя ключевое слово `auto` вместо фундаментальных типов данных, мы не сэкономим много времени или усилий, но на следующих уроках, когда типы данных будут более сложными и длинными, ключевое слово `auto` может очень пригодиться.

Ключевое слово `auto` и параметры функций

Многие новички пытаются сделать что-то вроде следующего:

```
1 void mySwap(auto a, auto b)
```

```
2 {  
3     auto x = a;  
4     a = b;  
5     b = x;  
6 }
```

Это не сработает, так как компилятор не может определить типы данных для параметров функции `a` и `b` во время компиляции.

Если вы хотите создать функцию, которая будет работать с разными типами данных, то вам лучше воспользоваться шаблонами функций, а не выводом типа. Это ограничение, возможно, отменят в будущих версиях C++ (когда `auto` будет использоваться как сокращенный способ создания шаблонов функций), но в C++14 это не работает. Единственное исключение — [лямбда-выражения](#) (но это уже другая тема).

Вывод типов в C++14

В C++14 функционал ключевого слова `auto` был расширен до автоматического определения типа возвращаемого значения функции. Например:

```
1 auto subtract(int a, int b)  
2 {  
3     return a - b;  
4 }
```

Так как выражение `a - b` является типа `int`, то компилятор делает вывод, что и функция должна быть типа `int`.

Хотя это может показаться удобным, так делать не рекомендуется. Тип возвращаемого значения функции помогает понять caller-у, что именно функция должна возвращать. Если конкретный тип не указан, то caller может неверно интерпретировать тип возвращаемого значения, что может привести к непреднамеренным ошибкам.

Так почему же использование `auto` при инициализации переменных — это хорошо, а с функциями — плохо? Дело в том, что `auto` можно использовать при определении переменной, так как значение, из которого компилятор делает выводы о типе переменной, находится прямо там — в правой части стейтмента. Однако с функциями это не так — нет контекста, который бы указывал, какого типа данных будет возвращаемое значение. Фактически, пользователю придется лезть в тело функции, чтобы определить тип возвращаемого значения. Следовательно, такой способ не только не практичен, но и более подвержен ошибкам.

trailing-синтаксис в C++11

В C++11 появилась возможность использовать **синтаксис типа возвращаемого значения trailing** (или просто «*trailing-синтаксис*»), когда компилятор делает выводы о типе возвращаемого значения по конечной части [прототипа функции](#). Например, рассмотрим следующее объявление функции:

```
1 | int subtract(int a, int b);
```

В C++11 это можно записать как:

```
1 | auto subtract(int a, int b) -> int;
```

В этом случае auto не выполняет вывод типа — это всего лишь часть синтаксиса типа возвращаемого значения `trailing`. Зачем это стоит использовать? В основном, из-за удобства. Например, можно выстроить в колонку все названия ваших функций:

```
1 | auto subtract(int a, int b) -> int;  
2 | auto divide(double a, double b) -> double;  
3 | auto printThis() -> void;  
4 | auto calculateResult(int a, double x) -> std::string;
```

Но этот синтаксис более полезен в сочетании с некоторыми другими продвинутыми особенностями языка C++ такими, как классы и ключевое слово `decltype`.

На данный момент я рекомендую придерживаться традиционного (обычного) синтаксиса для определения типа возвращаемого значения функции (без использования ключевого слова `auto`).

Заключение

Начиная с C++11 ключевое слово `auto` может использоваться вместо типа переменной при инициализации для выполнения вывода типа. Во всех других случаях использования ключевого слова `auto` следует избегать, если на это нет веских причин.

Оценить статью:

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



← [Урок №61. Структуры](#)

[Глава №4. Итоговый тест](#) →



Комментариев: 8



1. *anonymus:*

[4 сентября 2020 в 11:39](#)

Таки в -std=c++20 можно параметры делать auto

[Ответить](#)2. *Антонида:*[6 августа 2020 в 21:09](#)

Большое спасибо за труд! Очень информативные, понятные и полезные статьи! В статье вы пишете: "Мы поговорим детально о других использованиях auto, когда будем рассматривать ключевое слово decltype." Можно, пожалуйста, ссылочку на статью, где это обсуждается?

[Ответить](#)1. *Юрий:*[11 августа 2020 в 22:49](#)

На этих уроках действительно не рассматривается использование decltype. Могу посоветовать эту статью — <https://habr.com/ru/post/206458/>, либо это объяснение — <https://en.cppreference.com/w/cpp/language/decltype>.

[Ответить](#)3. *Артёмий:*[28 февраля 2020 в 15:21](#)

Спасибо за очередной урок

[Ответить](#)1. *Юрий:*[28 февраля 2020 в 23:30](#)

Пожалуйста)

[Ответить](#)4. *Torgu:*[3 июля 2018 в 19:14](#)

за овер 60 уроков у меня сложилось несколько вопросу по плавающей точке:

1. Почему вы все время используете double? Ведь точности того же float будет достаточно для большинства дробей
2. В начале этого урока пишется, что "4.0" — литерал типа double. Почему double? Разве стандартный не float?
3. В уроке о плавающей точке при инициализации переменной типа float вы добавляли f сразу после значения. Зачем? Ведь тип уже указан — float, зачем еще дополнительно что-то писать? Думал позже пойму, но нет, вот уже какой десяток уроков, до сих пор не догоняю

[Ответить](#)

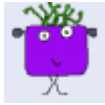


1. *Torgu:*

[20 июля 2018 в 18:18](#)

UPD: в итоге, сам ответил на свои вопросы 😊 Оказывается double стандартный тип для плавающей точки, а мне почему-то упорно казалось, что float. А фишу с постфиксом 'f' для чисел так и не понял

[Ответить](#)



1. *Владимир:*

[29 ноября 2018 в 14:54](#)

Добавление к константе постфикса f или F для float используется для повышения оптимизации, поскольку компилятор видит суффикс f и понимает, что это константа типа float, а не double. следовательно, не выполняется преобразование из double в float. А то, что тип double используется по умолчанию, говорилось ещё в уроке про литералы и магические числа (36 урок).

[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