

Ravesli [Ravesli](#)


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


Урок №95. Введение в std::vector

 [Юрий](#) |

- [Уроки C++](#)

|

 Обновл. 16 Авг 2020 |

 62600

[↑](#)  16

На предыдущем уроке мы рассматривали [std::array](#), который является более безопасной и удобной формой обычных **фиксированных массивов** в языке C++. Аналогично, в Стандартной библиотеке C++ есть и улучшенная версия **динамических массивов** (более безопасная и удобная) — `std::vector`.

В отличие от `std::array`, который недалеко отходит от базового функционала обычных фиксированных массивов, `std::vector` идет в комплекте с дополнительными возможностями, которые делают его одним из самых полезных и универсальных инструментов в языке C++.

Оглавление:

1. [Векторы](#)
2. [Нет утечкам памяти!](#)
3. [Длина векторов](#)
4. [Заключение](#)

Векторы

Представленный в C++03, **`std::vector`** (или просто «*вектор*») — это тот же динамический массив, но который может сам управлять выделенной себе памятью. Это означает, что вы можете создавать массивы, длина которых задается во время выполнения, без использования **[операторов new и delete](#)** (явного указания выделения и освобождения памяти). `std::vector` находится в **[заголовочном файле vector](#)**. Объявление `std::vector` следующее:

```
1 #include <vector>
2
```

```
3 // Нет необходимости указывать длину при инициализации
4 std::vector<int> array;
5 std::vector<int> array2 = { 10, 8, 6, 4, 2, 1 }; // используется список инициализаторов
6 std::vector<int> array3 { 10, 8, 6, 4, 2, 1 }; // используется uniform-инициализация дл
```

Обратите внимание, что в неинициализированном, что в инициализированном случаях вам не нужно явно указывать длину массивов. Это связано с тем, что `std::vector` динамически выделяет память для своего содержимого по запросу.

Подобно `std::array`, доступ к элементам массива может выполняться как через оператор `[]` (который не выполняет проверку диапазона), так и через функцию `at()` (которая выполняет проверку диапазона):

```
1 array[7] = 3; // без проверки диапазона
2 array.at(8) = 4; // с проверкой диапазона
```

В любом случае, если вы будете запрашивать элемент, который находится вне диапазона `array`, длина вектора автоматически изменяться не будет. Начиная с C++11, вы также можете присваивать значения для `std::vector`, используя список инициализаторов:

```
1 array = { 0, 2, 4, 5, 7 }; // ок, длина array теперь 5
2 array = { 11, 9, 5 }; // ок, длина array теперь 3
```

В таком случае вектор будет самостоятельно изменять свою длину, чтобы соответствовать количеству предоставленных элементов.

Нет утечкам памяти!

Когда переменная-вектор выходит из области видимости, то она автоматически освобождает память, которую контролировала (занимала). Это не только удобно (так как вам не нужно это делать вручную), но также помогает предотвратить утечки памяти. Рассмотрим следующий фрагмент:

```
1 void doSomething(bool value)
2 {
3     int *array = new int[7] { 12, 10, 8, 6, 4, 2, 1 };
4
5     if (value)
6         return;
7
8     // Делаем что-нибудь
9
10    delete[] array; // если value == true, то этот стейтмент никогда не выполнится
11 }
```

Если переменной `value` присвоить значение `true`, то `array` никогда не будет удален, память никогда не будет освобождена и произойдет утечка памяти.

Однако, если бы `array` был вектором, то подобное никогда бы и не произошло, так как память освобождалась бы автоматически при выходе `array` из области видимости (независимо от того, выйдет ли функция раньше из области видимости или нет). Именно из-за этого использование `std::vector` является более безопасным, чем динамическое выделение памяти через оператор `new`.

Длина векторов

В отличие от стандартных динамических массивов, которые не знают свою длину, `std::vector` свою длину запоминает. Чтобы её узнать, нужно использовать **функцию `size()`**:

```
1 #include <vector>
2 #include <iostream>
3
4 int main()
5 {
6     std::vector<int> array { 12, 10, 8, 6, 4, 2, 1 };
7     std::cout << "The length is: " << array.size() << '\n';
8
9     return 0;
10 }
```

Результат:

The length is: 7

Изменить длину стандартного динамически выделенного массива довольно проблематично и сложно. Изменить длину `std::vector` так же просто, как вызвать **функцию `resize()`**:

```
1 #include <vector>
2 #include <iostream>
3
4 int main()
5 {
6     std::vector<int> array { 0, 1, 2 };
7     array.resize(7); // изменяем длину array на 7
8
9     std::cout << "The length is: " << array.size() << '\n';
10
11     for (auto const &element: array)
12         std::cout << element << ' ';
13
14     return 0;
15 }
```

Результат:

The length is: 7
0 1 2 0 0 0 0

Здесь есть две вещи, на которые следует обратить внимание. Во-первых, когда мы изменили длину `array`, существующие значения элементов сохранились! Во-вторых, новые элементы были инициализированы значением по умолчанию в соответствии с определенным типом данных (значением `0` для типа `int`).

Длину вектора также можно изменить и в обратную сторону (обрезать):

```
1 #include <vector>
2 #include <iostream>
3
4 int main()
5 {
6     std::vector<int> array { 0, 1, 4, 7, 9, 11 };
7     array.resize(4); // изменяем длину array на 4
8
9     std::cout << "The length is: " << array.size() << '\n';
10
11     for (auto const &element: array)
12         std::cout << element << ' ';
13
14     return 0;
15 }
```

Результат:

```
The length is: 4
0 1 4 7
```

Изменение длины вектора является затратной операцией, поэтому вы должны стремиться минимизировать количество подобных выполняемых операций.

Заключение

Это вводная статья, предназначенная для ознакомления с основами `std::vector`. На следующих уроках мы детально рассмотрим `std::vector`, в том числе и разницу между длиной и ёмкостью вектора, и то, как в `std::vector` выполняется выделение памяти.

Поскольку переменные типа `std::vector` могут сами управлять выделенной себе памятью (что помогает предотвратить утечку памяти), отслеживают свою длину и легко её изменяют, то рекомендуется использовать `std::vector` вместо стандартных динамических массивов.

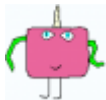
Оценить статью:

★★★★★ (287 оценок, среднее: 4,87 из 5)



[← Урок №94. Введение в std::array](#)[Введение в итераторы в C++](#)

Комментариев: 16



1. *PAPAUA:*

[10 сентября 2020 в 16:33](#)

Юрий, вы восхитительны! У вас не только классные уроки, но и абсолютно замечательная система кросс-навигации между ними — на сайт изначально попала в поисках информации про классы. Пойду читать ваши уроки с самого начала, у меня хоть какая-то минимальная база знаний и есть, но всё равно так много новых и полезных штук открываю 😊

[Ответить](#)



1. *Юрий:*

[11 сентября 2020 в 20:08](#)

Спасибо! 😊

[Ответить](#)



2. *Игорь:*

[28 сентября 2019 в 09:12](#)

Доброго времени суток!

В этой статье нет ни слова про дву-х, трё-х, *N — мерные векторы и массивы пространства имён std. Подозреваю что можно реализовать аналогично указателю на указатель (древовидная структура) массив в массиве, но надеюсь есть способ проще и удобней. Конечно сам доберусь до этого, но надеюсь получить ответ и возможно мой вопрос поможет улучшить/дополнить 95-й урок.

[Ответить](#)



1. *Юрий:*

[10 ноября 2019 в 17:51](#)

Это было вводное занятие, поверхностное обозрение темы, написано же в конце. Дальше, естественно, будет подробнее

[Ответить](#)



3. *Oleh:*

[3 июля 2019 в 15:14](#)

Что означает здесь оператор (:)?

```
1 | for (auto const &element: array)
```

[Ответить](#)



1. *Анастасия:*
[4 июля 2019 в 12:08](#)

это синтаксис цикла foreach, означает что element является элементом упорядоченной структуры (массива).

[Ответить](#)



4. *Alexey:*
[10 февраля 2019 в 13:44](#)

Отличные статьи, но очень плохо, что в этой не сказано про такую полезную вещь, как добавление элемента в конец вектора:

```
1 | std::vector<int> v ={1, 2, 3};  
2 | v.push_back(25);
```

После этого 25 добавится в конец вектора v.

[Ответить](#)



5. *Andrey:*
[6 октября 2018 в 13:39](#)

Отличный слог, легко читается. Спасибо.

P.S. Немного знаком с синтаксисами других языков, и когда читал про в Ваших статьях про стандартные массивы CPP, был озадачен. Оказалось что есть нормальные решения. Можно не только на счетах работать. 😊

[Ответить](#)



6. *Новичок:*
[20 ноября 2017 в 13:43](#)

Спасибо Вам огромное! Буду с нетерпением ждать!

[Ответить](#)



1. *Юрий:*
[20 ноября 2017 в 14:50](#)

Спасибо и Вам 😊

[Ответить](#)



7. *Новичок:*

[20 ноября 2017 в 04:20](#)

А когда будет продолжение? Просто очень хочется учиться дальше!

[Ответить](#)



1. *Юрий:*

[20 ноября 2017 в 12:53](#)

На этой неделе постараюсь две-три статьи опубликовать. Просто сейчас времени мало, сессия скоро.

[Ответить](#)



8. *Andrey:*

[18 ноября 2017 в 09:00](#)

Хочу пройти ваш курс по c++, после всех уроков какой уровень знаний у меня будет? Я выучу весь c++?

[Ответить](#)

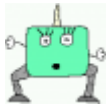


1. *Юрий:*

[19 ноября 2017 в 12:23](#)

Вы поймёте всё необходимое для программирования на языке C++.

[Ответить](#)



9. *Zufar:*

[17 ноября 2017 в 20:11](#)

Продолжай , всё доступно и понятно! Спасибо за уроки!

[Ответить](#)



1. *Юрий:*

[19 ноября 2017 в 12:22](#)

Спасибо, буду продолжать.

[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