

Ravesli [Ravesli](#)


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

Урок №94. Введение в std::array

 [Юрий](#) |

- [Уроки C++](#)

|

 Обновл. 16 Авг 2020 |

 44658

[1](#)  14

На предыдущих уроках мы подробно говорили о [фиксированных](#) и [динамических](#) массивах. Хотя они очень полезны и активно используются в языке C++, у них также есть свои недостатки: фиксированные массивы [распадаются в указатели](#), теряя информацию о своей длине; в динамических массивах проблемы могут возникнуть с освобождением памяти и с попытками изменить их длину после выделения.

Поэтому в Стандартную библиотеку C++ добавили функционал, который упрощает процесс управления массивами: std::array и std::vector. На этом уроке мы рассмотрим std::array, а на следующем — std::vector.

Оглавление:

1. [Введение в std::array](#)
2. [Размер и сортировка](#)
3. [Заключение](#)

Введение в std::array

Представленный в C++11, **std::array** — это фиксированный массив, который не распадается в указатель при передаче в функцию. std::array определяется в [заголовочном файле](#) array, внутри [пространства имен](#) std. Объявление переменной std::array следующее:

```
1 #include <array>
2
3 std::array<int, 4> myarray; // объявляем массив типа int длиной 4
```

Подобно обычным фиксированным массивам, длина `std::array` должна быть установлена во время компиляции. `std::array` можно инициализировать с использованием списка инициализаторов или [uniform-инициализации](#):

```
1 std::array<int, 4> myarray = { 8, 6, 4, 1 }; // список инициализаторов
2 std::array<int, 4> myarray2 { 8, 6, 4, 1 }; // uniform-инициализация
```

В отличие от стандартных фиксированных массивов, в `std::array` вы не можете пропустить (не указывать) длину массива:

```
1 std::array<int, > myarray = { 8, 6, 4, 1 }; // нельзя, должна быть указана длина массива
```

Также можно присваивать значения массиву с помощью списка инициализаторов:

```
1 std::array<int, 4> myarray;
2 myarray = { 0, 1, 2, 3 }; // ок
3 myarray = { 8, 6 }; // ок, элементам 2 и 3 присвоен ноль!
4 myarray = { 0, 1, 3, 5, 7, 9 }; // нельзя, слишком много элементов в списке инициализаторов
```

Доступ к значениям массива через оператор индекса осуществляется как обычно:

```
1 std::cout << myarray[1];
2 myarray[2] = 7;
```

Так же, как и в стандартных фиксированных массивах, оператор индекса не выполняет никаких проверок на диапазон. Если указан недопустимый индекс, то произойдут плохие вещи.

`std::array` поддерживает вторую форму доступа к элементам массива — **функция `at()`**, которая осуществляет проверку диапазона:

```
1 std::array<int, 4> myarray { 8, 6, 4, 1 };
2 myarray.at(1) = 7; // элемент массива под номером 1 - корректный, присваиваем ему значение 7
3 myarray.at(8) = 15; // элемент массива под номером 8 - некорректный, получим ошибку
```

В примере, приведенном выше, вызов `myarray.at(1)` проверяет, есть ли элемент массива под номером 1, и, поскольку он есть, возвращается [ссылка](#) на этот элемент. Затем мы присваиваем ему значение 7. Однако, вызов `myarray.at(8)` не срабатывает, так как элемента под номером 8 в массиве нет. Вместо возвращения ссылки, функция `at()` выдает ошибку, которая завершает работу программы (на самом деле выбрасывается исключение типа `std::out_of_range`. Об исключениях мы поговорим на соответствующих уроках). Поскольку проверка диапазона выполняется, то функция `at()` работает медленнее (но безопаснее), чем оператор `[]`.

`std::array` автоматически делает все очистки после себя, когда выходит из области видимости, поэтому нет необходимости прописывать это вручную.

Размер и сортировка

С помощью функции **size()** можно узнать длину массива:

```
1 #include <iostream>
2 #include <array>
3
4 int main()
5 {
6     std::array<double, 4> myarray{ 8.0, 6.4, 4.3, 1.9 };
7     std::cout << "length: " << myarray.size();
8
9     return 0;
10 }
```

Результат:

length: 4

Поскольку std::array не распадается в указатель при передаче в функцию, то функция size() будет работать, даже если её вызвать из другой функции:

```
1 #include <iostream>
2 #include <array>
3
4 void printLength(const std::array<double, 4> &myarray)
5 {
6     std::cout << "length: " << myarray.size();
7 }
8
9 int main()
10 {
11     std::array<double, 4> myarray { 8.0, 6.4, 4.3, 1.9 };
12
13     printLength(myarray);
14
15     return 0;
16 }
```

Результат тот же:

length: 4

Обратите внимание, Стандартная библиотека C++ использует термин «размер» для обозначения длины массива — не путайте это с результатами выполнения [оператора sizeof](#) с обычным фиксированным массивом, когда возвращается фактический размер массива в памяти (размер элемента * длина массива).

Также обратите внимание на то, что мы передаем std::array по ссылке (**константной**). Это делается по соображениям производительности для того, чтобы компилятор не выполнял копирование массива при передаче в функцию.

Правило: Всегда передавайте std::array в функции по обычной или по константной ссылке.

Поскольку длина массива всегда известна, то **циклы foreach** также можно использовать с std::array:

```
1 std::array<int, 4> myarray { 8, 6, 4, 1 };
2
3 for (auto &element : myarray)
4     std::cout << element << ' ';
```

Вы можете отсортировать std::array, используя **функцию std::sort()**, которая находится в заголовочном файле algorithm:

```
1 #include <iostream>
2 #include <array>
3 #include <algorithm> // для std::sort
4
5 int main()
6 {
7     std::array<int, 5> myarray { 8, 4, 2, 7, 1 };
8     std::sort(myarray.begin(), myarray.end()); // сортировка массива по возрастанию
9     // std::sort(myarray.rbegin(), myarray.rend()); // сортировка массива по убыванию
10
11     for (const auto &element : myarray)
12         std::cout << element << ' ';
13
14     return 0;
15 }
```

Результат:

1 2 4 7 8

Функция сортировки использует итераторы, которые мы еще не рассматривали. О них мы поговорим несколько позже.

Заключение

std::array — это отличная замена стандартных фиксированных массивов. Массивы, созданные с помощью std::array, более эффективны, так как используют меньше памяти. Единственными недостатками std::array по сравнению со стандартными фиксированными массивами являются немного неудобный синтаксис и то, что нужно явно указывать длину массива (компилятор не будет вычислять её за нас). Но это сравнительно незначительные нюансы. Рекомендуется использовать std::array вместо стандартных фиксированных массивов в любых нетривиальных задачах.

Оценить статью:

[← Урок №93. Указатели на указатели](#)[Урок №95. Введение в std::vector →](#)

Комментариев: 14



1. *Vlad:*
[12 октября 2020 в 13:27](#)

Мгер, Кто вам такое сказал ?
Ваш код с обычным двумерным массивом легко справляется.
Если вы не знаете как это сделать, то это не значит что невозможно.
Не вводите людей в заблуждение

[Ответить](#)

2. *Георгий:*
[4 августа 2020 в 14:43](#)

Кстати, начиная с C++17 компилятор может вывести не только длину, но и тип массива:

```
1 | std::array a = {1,2,3,4,5}; //абсолютно корректно
```

[Ответить](#)

3. *Снова я:*
[21 января 2020 в 12:02](#)

Можете объяснить этот момент?

```
1 | std::array<int, 4> myarray;  
2 | myarray = { 8, 6 }; // хорошо, элементам 2 и 3 присвоено ноль!
```

Почему элементам 2 и 3, а не 1 и 2 ?

[Ответить](#)

1. *Мгер:*
[22 января 2020 в 01:55](#)

тут все что не заполнено, получает значение по умолчанию.
так же как и с обычным массивом:

```
1 int m[10] = {}; // все значения равны 0
2 int m[5] = {1, 2, 3, 4} // m[4] = 0
```

[Ответить](#)



4. Антон:

[8 января 2020 в 10:39](#)

"std::array — это отличная замена стандартных фиксированных массивов. Они более эффективны, так как используют меньше памяти." Это вдруг почему?

[Ответить](#)



1. Георгий:

[4 августа 2020 в 14:41](#)

Не по чему. На деле, std::array и обычный массив подуцируют один и тот же ассемблерный код (проверял в VS). С std::array удобнее присваивание целого массива, передача в функции, копирование и т. д. (с обычными массивами тут большие проблемы). По эффективности они абсолютно идентичны

[Ответить](#)



5. Арман:

[8 ноября 2018 в 13:46](#)

Ну конечно array будет знать свой размер , если указывать его в параметрах :

```
1 void printLength(const std::array<double, 4> &myarray)
```

А если функция должна работать с разными array. Передавать размер как аргумент или я что то не понял?

[Ответить](#)



1. Мгер:

[23 января 2019 в 15:17](#)

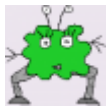
Не совсем то что ты хочешь, но вот еще как можно сделать:

```
1 #include <iostream>
2 #include <array>
3
4 template <int size>
5 void printLength(const std::array<double, size> &myarray)
```

```

6  {
7      std::cout << "length: " << myarray.size();
8  }
9
10 int main()
11 {
12     const int SIZE = 4;
13     std::array<double, SIZE> myarray { 8.0, 6.4, 4.3, 1.9 };
14
15     printLength<SIZE>(myarray);
16
17     return 0;
18 }

```

[Ответить](#)

6. Андрей:

[13 января 2018 в 22:51](#)

А как на счет двумерного массива?

[Ответить](#)**R**

1. Юрий:

[15 января 2018 в 15:57](#)

Двумерный массив объявляется как:

```

1 | std::array<std::array<int, 9>, 10> myarray;

```

Здесь myarray — это название вашего двумерного массива, а сам двумерный массив создается с помощью вложенного массива (т.е. один внутри другого).

[Ответить](#)

1. Vlad:

[30 ноября 2019 в 19:11](#)

Это очень неудобно. Тут лучше использовать обычные квадратные скобки

[Ответить](#)

1. Мгер:

[1 декабря 2019 в 17:15](#)

с обычным массивом много что нельзя сделать. Например это:

```

1 | std::array<std::array<int, 9>, 10> myarray;

```

```
2 for (const auto& row: myarray) {  
3     for (const auto& item: row) {  
4         cout << item << " ";  
5     }  
6     cout << endl;  
7 }
```



2. alex_1988:

[13 февраля 2020 в 17:13](#)

Покажите синтаксис использования функции `at()` для двумерного массива и почему `size()` не показывает суммарную длину всех ячеек двумерного массива?

[Ответить](#)



1. Сергей:

[5 марта 2020 в 16:23](#)

Спасибо alex_1988 за вопросы.

И конечно автору сайта за качественный перевод.

Синтаксис использования функции `at()` в двумерных массивах выглядит так:

`myarray2.at().at()`

Но есть особенность, например:

```
1 std::array<std::array<int, 10>,5> myarray2{};  
2 myarray2.at(4).at(9) = 1;
```

Здесь сначала идет проверка последнего (второго) индекса массива `myarray2` (объявлено 5, проверяем от 0 до 4), а затем предпоследнего (первого) индекса массива `myarray2` (объявлено 10, проверяем от 0 до 9).

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.

Отправить комментарий

[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