

Ravesli [Ravesli](#)


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

Урок №74. Массивы

 [Юрий](#) |

- [Уроки C++](#)

|

 Обновл. 15 Сен 2020 |

 64323

[16](#)

На уроке о [структурах](#) мы узнали, что с их помощью можно объединять переменные разных типов под одним идентификатором. Это идеально, когда нужно смоделировать объект, который имеет много разных свойств. Однако удобство работы со структурами при наличии большого количества элементов оставляет желать лучшего.

Оглавление:

1. [Что такое массив?](#)
2. [Элементы массива](#)
3. [Пример программы с использованием массива](#)
4. [Типы данных и массивы](#)
5. [Индексы массивов](#)
6. [Объявление массивов фиксированного размера](#)
7. [Чуть-чуть о динамических массивах](#)
8. [Заключение](#)

Что такое массив?

К счастью, структуры не являются единственным агрегированным типом данных в языке C++. Есть еще **массив** — совокупный тип данных, который позволяет получить доступ ко всем переменным одного и того же типа данных через использование одного идентификатора.

Рассмотрим случай, когда нужно записать результаты тестов 30 студентов в классе. Без использования массива нам придется выделить почти 30 одинаковых переменных!

```
1 // Выделяем 30 целочисленных переменных (каждая с разным именем)
2 int testResultStudent1;
3 int testResultStudent2;
4 int testResultStudent3;
5 // ...
6 int testResultStudent30;
```

С использованием массива всё гораздо проще. Следующая строка эквивалентна коду, приведенному выше:

```
1 int testResult[30]; // выделяем 30 целочисленных переменных, используя фиксированный ма
```

В объявлении переменной массива мы используем квадратные скобки `[]`, чтобы сообщить компилятору, что это переменная массива (а не обычная переменная), а в скобках — количество выделяемых элементов (это называется **длиной** или **размером массива**).

В примере, приведенном выше, мы объявили фиксированный массив с именем `testResult` и длиной 30. **Фиксированный массив** (или «**массив фиксированной длины**») представляет собой массив, размер которого известен во время компиляции. При создании `testResult`, компилятор выделит 30 целочисленных переменных.

Элементы массива

Каждая из переменных в массиве называется **элементом**. Элементы не имеют своих собственных уникальных имен. Вместо этого для доступа к ним используется имя массива вместе с **оператором индекса** `[]` и параметром, который называется **индексом**, и который сообщает компилятору, какой элемент мы хотим выбрать. Этот процесс называется **индексированием массива**.

В вышеприведенном примере первым элементом в нашем массиве является `testResult[0]`, второй — `testResult[1]`, десятый — `testResult[9]`, последний — `testResult[29]`. Хорошо, что уже не нужно отслеживать и помнить кучу разных (хоть и похожих) имен переменных — для доступа к разным элементам нужно изменять только индекс.

Важно: В отличие от повседневной жизни, отсчет в программировании и в языке C++ всегда начинается с 0, а не с 1!

В массиве длиной `N` элементы массива будут пронумерованы от 0 до `N-1`! Это называется **диапазоном массива**.

Пример программы с использованием массива

Здесь мы можем наблюдать как определение, так и индексирование массива:

```
1 #include <iostream>
2
3 int main()
4 {
```

```
5   int array[5]; // массив из пяти чисел
6   array[0] = 3; // индекс первого элемента - 0 (нулевой элемент)
7   array[1] = 2;
8   array[2] = 4;
9   array[3] = 8;
10  array[4] = 12; // индекс последнего элемента - 4
11
12  std::cout << "The array element with the smallest index has the value " << array[0]
13  std::cout << "The sum of the first 5 numbers is " << array[0] + array[1] + array[2]
14
15  return 0;
16 }
```

Результат выполнения программы:

The array element with the smallest index has the value 3
The sum of the first 5 numbers is 29

Типы данных и массивы

Массив может быть любого типа данных. Например, объявляем массив типа double:

```
1  #include <iostream>
2
3  int main()
4  {
5      double array[3]; // выделяем 3 переменные типа double
6      array[0] = 3.5;
7      array[1] = 2.4;
8      array[2] = 3.4;
9
10     std::cout << "The average is " << (array[0] + array[1] + array[2]) / 3 << "\n";
11
12     return 0;
13 }
```

Результат выполнения программы:

The average is 3.1

Массивы также можно сделать из структур, например:

```
1  struct Rectangle
2  {
3      int length;
4      int width;
5  };
```

```
6 | Rectangle rects[4]; // объявляем массив с 4-мя прямоугольниками
```

Чтобы получить доступ к члену структуры из элемента массива, сначала нужно выбрать элемент массива, затем использовать оператор выбора члена структуры, а затем требуемый член структуры:

```
1 | rects[0].length = 15;
```

Индексы массивов

В языке C++ индексы массивов всегда должны быть интегрального типа данных (т.е. типа char, short, int, long, long long, bool и т.д.). Эти индексы могут быть либо константными значениями, либо неконстантными значениями. Например:

```
1 | int array[4]; // объявляем массив длиной 4
2 |
3 | // Используем литерал (константу) в качестве индекса
4 | array[2] = 8; // хорошо
5 |
6 | // Используем перечисление (константу) в качестве индекса
7 | enum Animals
8 | {
9 |     ANIMAL_CAT = 3
10 | };
11 | array[ANIMAL_CAT] = 5; // хорошо
12 |
13 | // Используем переменную (не константу) в качестве индекса
14 | short index = 4;
15 | array[index] = 8; // хорошо
```

Объявление массивов фиксированного размера

При объявлении массива фиксированного размера, его длина (между квадратными скобками) должна быть **константой типа compile-time** (которая определяется во время компиляции). Вот несколько разных способов объявления массивов с фиксированным размером:

```
1 | // Используем литерал
2 | int array[7]; // хорошо
3 |
4 | // Используем макрос-объект с текст_замена в качестве символьной константы
5 | #define ARRAY_WIDTH 4
6 | int array[ARRAY_WIDTH]; // синтаксически хорошо, но не делайте этого
7 |
8 | // Используем символьную константу
9 | const int arrayWidth = 7;
10 | int array[arrayWidth]; // хорошо
11 |
```

```
12 // Используем перечислитель
13 enum ArrayElements
14 {
15     MIN_ARRAY_WIDTH = 3
16 };
17 int array[MIN_ARRAY_WIDTH]; // хорошо
18
19 // Используем неконстантную переменную
20 int width;
21 std::cin >> width;
22 int array[width]; // плохо: width должна быть константой типа compile-time!
23
24 // Используем константную переменную типа runtime
25 int temp = 8;
26 const int width = temp;
27 int array[width]; // плохо: здесь width является константой типа runtime, но должна бы
```

Обратите внимание, в двух последних случаях мы должны получить ошибку, так как длина массива не является константой типа `compile-time`. Некоторые компиляторы могут разрешить использование таких массивов, но они являются некорректными в соответствии со стандартами языка C++ и не должны использоваться в программах, написанных на C++.

Чуть-чуть о динамических массивах

Поскольку массивам фиксированного размера память выделяется во время компиляции, то здесь мы имеем два ограничения:

- Массивы фиксированного размера не могут иметь длину, основанную на любом пользовательском вводе или другом значении, которое вычисляется во время выполнения программы (`runtime`).
- Фиксированные массивы имеют фиксированную длину, которую нельзя изменить.

Во многих случаях эти ограничения являются проблематичными. К счастью, C++ поддерживает еще один тип массивов, известный как **динамический массив**. Размер такого массива может быть установлен во время выполнения программы и его можно изменить. Однако создание динамических массивов несколько сложнее фиксированных, поэтому мы поговорим об этом несколько позже.

Заключение

Фиксированные массивы обеспечивают простой способ выделения и использования нескольких переменных одного типа данных до тех пор, пока размер массива известен во время компиляции.

На следующем уроке мы рассмотрим больше тем, связанных с фиксированными массивами.

Оценить статью:



(306 оценок, среднее: 4,85 из 5)

[←Глава №5. Итоговый тест](#)[Урок №75. Фиксированные массивы](#)

Комментариев: 16

1. *Алексей:*[26 июля 2019 в 15:49](#)

Изучаю и UE4, там массивы вообще не рекомендуют использовать)

Хотя недавно делал инициализацию структуры при случаи, это был выбор сложности для Hi-Lo.

С массивами немного легче, хотя кто знает)

[Ответить](#)1. *Александр:*[20 мая 2020 в 12:34](#)

Алексей, а где, если не секрет, вы изучаете UE4? Кстати, выходит UE5 =)

[Ответить](#)2. *Владислав:*[26 марта 2019 в 16:40](#)

Всем здравствуйте! Друзья может кто помочь с решением такой задачи. Знаю что базовое задание, но в программировании не силен, но решить нужно.

Задание: Массив $A = (15, 9, -6, 12, -9, 18, 0)$ преобразован к виду $A = (5, 3, 0, 4, 0, 6, 0)$. Размер массива A — 20 элементов из диапазона $[-25, 25]$. Вычислить сумму четных элементов преобразованного массива.

Вот что то попробовал но не уверен что работает правильно.

```
1 #include <iostream>
2 using namespace std;
3
4 int A[] = {5, 3, 1, 4, 0, 6, 0};
5 int sumEl;
6
7 int main(int argc, const char * argv[]) {
8
```

```

9   for (int i = 0; i < sizeof(A) / sizeof(int); i++) {
10  if (i%2 == 0) {
11  sumEl+=A[i];
12  }
13  }
14
15  std::cout << sumEl <<std::endl;
16  return 0;
17  }

```

[Ответить](#)



1. *Юшка:*

[18 мая 2020 в 16:10](#)

```

1  #include <iostream>
2  #include <ctime>
3
4  const int N = 20;
5
6  int converter(int *arr, int n);    // функция для конвертирования массива
7  void printArray(int *arr);        // функция для распечатки массива
8  int sumEvenElementsArray(int *arr); // функция для подсчёта суммы чётных
9
10 int main()
11 {
12     setlocale(LC_ALL, "Rus");
13
14     srand(time(0));
15
16     int array[N] = {0};
17
18     for(int i=0; i < N; i++)
19         array[i] = rand() % 50 + (-25) ;
20
21     printArray(array);
22
23     int arrConvert[N] = {0};
24
25     for(int i=0; i<N; i++)
26         arrConvert[i] = converter(array, i);
27
28     printArray(arrConvert);
29
30     std::cout << "Сумма чётных элементов, равна : " << sumEvenElementsArray(arrConvert);
31
32     return 0;
33 }

```

```

34
35
36 int converter(int *arr, int i)
37 {
38     int temp = 0;
39     if(arr[i] % 3 == 0 && arr[i] > 0)
40         temp = arr[i] / 3;
41     return temp;
42 }
43
44 void printArray(int *arr)
45 {
46     for(int i=0; i<N; i++)
47         std::cout << arr[i] << ' ';
48     std::cout << "\n\n";
49 }
50
51 int sumEvenElementsArray(int *arr)
52 {
53     int sum = 0;
54
55     for(int i=0; i < N; i++)
56         if(arr[i] % 2 == 0)
57             sum += arr[i];
58     return sum;
59 }

```

[Ответить](#)



3. *SuRprizZe:*

[16 марта 2019 в 00:04](#)

Почему элементу массива которого как бы не существует, если вначале объявляется элемент с 4 значениями, можно 5 элементу присвоить что-то хотя его нет и как вообще это объяснить?!

[Ответить](#)



1. *Роман:*

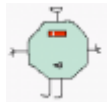
[26 июля 2019 в 14:35](#)

Как я понимаю, объявляя массив `Array[n]`, мы резервируем n -ое количество элементов памяти соответствующего типу данных размера, а `Array[i]` — это указатель на i -ый элемент памяти такого размера, начиная с нулевого (`Array[0]`). Следовательно, если мы напишем "`Array[n]=чему-то`", мы обратимся к элементу памяти, следующему за местом хранения последнего члена массива (`Array[n-1]`), — куда и будет помещено наше "что-то" и будет до поры до времени там спокойно лежать. Программу такая "просьба" ни капли не затруднит. Однако, поскольку мы договорились с программой, что хранит она наш массив только в n элементах, то, как только ей понадобится ближайшая к массиву ячейка памяти, программа

немедленно, без зазрения совести, выбросит наше "что-то" в забвение и мы уже никогда, никогда его не найдем...

К слову, тут, мне кажется, может быть еще хуже: программа зарезервирует кусок памяти, идущий вслед за массивом, под некую нашу переменную x , со значением, скажем Y — а мы возьми и помести туда наш $n+1$ -ый элемент!.. И будет, обратно, он там лежать, покуда иксу не присвоят новое значение. Или программа вовсе уйдет в глюк.

[Ответить](#)



4. *alien:*

[20 февраля 2019 в 07:46](#)

Почему вы используете `std::`:

Если вместо этого можно написать

`using namespace std;` ??

Думаю лучше добавить эту библиотеку и не писать каждый раз `std::`:
просто небольшой совет 😊

[Ответить](#)

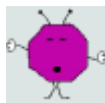


1. *Юрий:*

[21 февраля 2019 в 01:17](#)

Читайте уроки 24 и 54.

[Ответить](#)

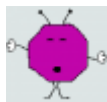


2. *Александр:*

[21 февраля 2019 в 21:03](#)

С ростом объемов кода этот "небольшой совет" превращается в "большую проблему"... А при попытке отладки этого кода может перерасти вообще в "катастрофу" :)))

[Ответить](#)



5. *Александр:*

[14 февраля 2019 в 23:55](#)

Вот тоже всегда считал и всем рассказывал, что размер массива должен задаваться константой `compile time`, а недавно в техностриме `mail.ru` (или `яндекса`) услышал, что в текущем стандарте (то ли с C++14, то ли с C++17) можно и переменными задавать и это считается корректным... причем считается это лучшей идеей, чем выделять динамический массив, вроде как потому, что он самоубивается корректно...

Хотя мне все равно кажется, что "безопасней" работать с константными размерами

[Ответить](#)



6. *Игорь:*

[21 сентября 2018 в 05:02](#)

Почему в конце инициализируется значением 8 массив с индексом 4. (array[index] = 8). ведь последний индекс у этого массива 3 (array[3])?

```
1 int array[4]; // объявляем массив с длиной 4
2
3 // используем литерал (константу) в качестве индекса
4 array[2] = 8; // хорошо
5
6 // используем перечисление (константу) в качестве индекса
7 enum Animals
8 {
9     ANIMAL_CAT = 3
10 };
11 array[ANIMAL_CAT] = 5; // хорошо
12
13 // используем переменную (не константу) в качестве индекса
14 short index = 4;
15 array[index] = 8; // хорошо
```

[Ответить](#)



1. *ffolax:*

[11 декабря 2018 в 19:55](#)

Ошибка допущена. Значение будет записано в ячейку следующую за последним элементом массива. Место под ту ячейку не зарезервировано массивом и может в любой момент быть изменено другим процессом.

[Ответить](#)



2. *SuRprizZe:*

[15 марта 2019 в 23:52](#)

Тут общее количество элементов в массиве , считая с 1

[Ответить](#)



7. *Oleksiy:*

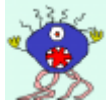
[24 августа 2018 в 13:52](#)

Пример программы с использованием массива...

"The lowest number is: " << array[0]

самое маленькое число находится в array[1]

[Ответить](#)



8. *KiFoR:*

[3 мая 2018 в 21:00](#)

О Даэдра... Как же я ненавижу массивы...

[Ответить](#)



1. *Юрий:*

[4 мая 2018 в 11:49](#)

Увы, но от этого ничего не поменяется 😊

[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