

Ravesli [Ravesli](#)


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

Урок №82. Указатели и массивы

 [Юрий](#) |

- [Уроки C++](#)

|

 Обновл. 12 Авг 2020 |

 44011

[↑](#)  6

В языке C++ указатели и массивы тесно связаны между собой.

Оглавление:

1. [Сходства между указателями и массивами](#)
2. [Различия между указателями и массивами](#)
3. [Передача массивов в функции](#)
4. [Передача по адресу](#)
5. [Массивы в структурах и классах](#)

Сходства между указателями и массивами

Фиксированный массив определяется следующим образом:

```
1 | int array[4] = { 5, 8, 6, 4 }; // определяем фиксированный массив из 4-х целых чисел
```

Для нас это массив из 4-х целых чисел, но для компилятора `array` является переменной типа `int[4]`. Мы знаем что `array[0] = 5`, `array[1] = 8`, `array[2] = 6` и `array[3] = 4`. Но какое значение имеет сам `array`?

Переменная `array` содержит адрес первого элемента массива, как если бы это был **указатель**!

Например:

```
1 | #include <iostream>
2 |
```

```
3 int main()
4 {
5     int array[4] = { 5, 8, 6, 4 };
6
7     // Выводим значение массива (переменной array)
8     std::cout << "The array has address: " << array << '\n';
9
10    // Выводим адрес элемента массива
11    std::cout << "Element 0 has address: " << &array[0] << '\n';
12
13    return 0;
14 }
```

Результат на моем компьютере:

```
The array has address: 004BF968
Element 0 has address: 004BF968
```

Обратите внимание, адрес, хранящийся в переменной `array`, является адресом первого элемента массива.

Распространенная ошибка думать, что переменная `array` и указатель на `array` являются одним и тем же объектом. Это не так. Хотя оба указывают на первый элемент массива, информация о типе данных у них разная. В вышеприведенном примере типом переменной `array` является `int[4]`, тогда как типом указателя на массив является `int *`.

Путаница вызвана тем, что во многих случаях, при вычислении, фиксированный массив **распадается (неявно преобразовывается)** в указатель на первый элемент массива. Доступ к элементам по-прежнему осуществляется через указатель, но информация, полученная из типа массива (например, его размер), не может быть доступна из типа указателя.

Однако и это не является столь весомым аргументом, чтобы рассматривать фиксированные массивы и указатели как разные значения. Например, мы можем разыменовывать массив, чтобы получить значение первого элемента:

```
1 int array[4] = { 5, 8, 6, 4 };
2
3 // Разыменование массива (переменной array) приведет к возврату первого элемента массива
4 std::cout << *array; // выведется 5!
5
6 char name[] = "John"; // строка C-style (также массив)
7 std::cout << *name; // выведется 'J'
```

Обратите внимание, мы не разыменовываем фактический массив. Массив (типа `int[4]`) неявно конвертируется в указатель (типа `int *`), и мы разыменовываем указатель, который указывает на значение первого элемента массива.

Также мы можем создать указатель и присвоить ему `array`:

```
1 #include <iostream>
```

```
2
3 int main()
4 {
5     int array[4] = { 5, 8, 6, 4 };
6     std::cout << *array; // выведется 5
7
8     int *ptr = array;
9     std::cout << *ptr; // выведется 5
10
11     return 0;
12 }
```

Это работает из-за того, что переменная `array` распадается в указатель типа `int *`, а тип нашего указателя такой же (т.е. `int *`).

Различия между указателями и массивами

Однако есть случаи, когда разница между фиксированными массивами и указателями имеет значение. Основное различие возникает при использовании [оператора sizeof](#). При использовании в фиксированном массиве, оператор `sizeof` возвращает размер всего массива (длина массива * размер элемента). При использовании с указателем, оператор `sizeof` возвращает размер адреса памяти (в байтах). Например:

```
1 #include <iostream>
2
3 int main()
4 {
5     int array[4] = { 5, 8, 6, 4 };
6
7     std::cout << sizeof(array) << '\n'; // выведется sizeof(int) * длина array
8
9     int *ptr = array;
10    std::cout << sizeof(ptr) << '\n'; // выведется размер указателя
11
12    return 0;
13 }
```

Результат выполнения программы:

```
16
4
```

Фиксированный массив знает свою длину, а указатель на массив — нет.

Второе различие возникает при использовании оператора адреса `&`. Используя адрес указателя, мы получаем адрес памяти переменной указателя. Используя адрес массива, возвращается указатель на целый

массив. Этот указатель также указывает на первый элемент массива, но информация о типе отличается. Вряд ли вам когда-нибудь понадобится это использовать.

Передача массивов в функции

На [уроке №75](#) мы говорили, что, из-за того, что копирование больших массивов при передаче в функцию является очень затратной операцией, C++ не копирует массив. При передаче массива в качестве аргумента в функцию, массив распадается в указатель на массив и этот указатель передается в функцию:

```
1  #include <iostream>
2
3  void printSize(int *array)
4  {
5      // Здесь массив рассматривается как указатель
6      std::cout << sizeof(array) << '\n'; // выведется размер указателя, а не длина массива
7  }
8
9  int main()
10 {
11     int array[] = { 1, 2, 3, 4, 4, 9, 15, 25 };
12     std::cout << sizeof(array) << '\n'; // выведется sizeof(int) * длина массива
13
14     printSize(array); // здесь аргумент array распадается на указатель
15
16     return 0;
17 }
```

Результат выполнения программы:

```
32
4
```

Обратите внимание, результат будет таким же, даже если параметром будет фиксированный массив:

```
1  #include <iostream>
2
3  // C++ неявно конвертирует параметр array[] в *array
4  void printSize(int array[])
5  {
6      // Здесь массив рассматривается как указатель, а не как фиксированный массив
7      std::cout << sizeof(array) << '\n'; // выведется размер указателя, а не размер массива
8  }
9
10 int main()
11 {
12     int array[] = { 1, 2, 3, 4, 4, 9, 15, 25 };
13     std::cout << sizeof(array) << '\n'; // выведется sizeof(int) * длина массива array
14 }
```

```
15 |     printSize(array); // здесь аргумент array распадается на указатель
16 |
17 |     return 0;
18 | }
```

Результат выполнения программы:

```
32
4
```

В примере, приведенном выше, C++ неявно конвертирует параметр из синтаксиса массива ([]) в синтаксис указателя (*). Это означает, что следующие два объявления функции идентичны:

```
1 | void printSize(int array[]);
2 | void printSize(int *array);
```

Некоторые программисты предпочитают использовать синтаксис [], так как он позволяет понять, что функция ожидает массив, а не указатель на значение. Однако, в большинстве случаев, поскольку указатель не знает, насколько велик массив, вам придется передавать размер массива в качестве отдельного параметра (строки являются исключением, так как они нуль-терминированные).

Рекомендуется использовать синтаксис указателя, поскольку он позволяет понять, что параметр будет обработан как указатель, а не как фиксированный массив, и определенные операции, такие как в случае с оператором sizeof, будут выполняться с параметром-указателем (а не с параметром-массивом).

Совет: Используйте синтаксис указателя (*) вместо синтаксиса массива ([]) при передаче массивов в качестве параметров в функции.

Передача по адресу

Тот факт, что массивы распадутся на указатели при передаче в функции, объясняет основную причину, по которой изменение массива в функции приведет к изменению фактического массива. Рассмотрим следующий пример:

```
1 | #include <iostream>
2 |
3 | // Параметр ptr содержит копию адреса массива
4 | void changeArray(int *ptr)
5 | {
6 |     *ptr = 5; // поэтому изменение элемента массива приведет к изменению фактического
7 | }
8 |
9 | int main()
10 | {
11 |     int array[] = { 1, 2, 3, 4, 4, 9, 15, 25 };
12 |     std::cout << "Element 0 has value: " << array[0] << '\n';
13 | }
```

```
14     changeArray(array);  
15  
16     std::cout << "Element 0 has value: " << array[0] << '\n';  
17  
18     return 0;  
19 }
```

Результат выполнения программы:

Element 0 has value: 1

Element 0 has value: 5

При вызове функции `changeArray()`, массив распадается на указатель, а значение этого указателя (адрес памяти первого элемента массива) копируется в параметр `ptr` функции `changeArray()`. Хотя значение `ptr` в функции является копией адреса массива, `ptr` всё равно указывает на фактический массив (а не на копию!). Следовательно, при разыменовании `ptr`, разыменовывается и фактический массив!

Этот феномен работает так же и с указателями на значения не из массива. Более детально эту тему мы рассмотрим на соответствующем уроке.

Массивы в структурах и классах

Стоит упомянуть, что массивы, которые являются частью **структур** или классов, не распадаются, когда вся структура или класс передается в функцию.

На следующем уроке мы рассмотрим адресную арифметику и поговорим о том, как на самом деле работает индексация массива.

Оценить статью:

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



← [Урок №81. Нулевые указатели](#)

[Урок №83. Адресная арифметика и индексация массивов](#) →



Комментариев: 6



1. *Spardoks:*

[30 ноября 2020 в 18:46](#)

Возможно, интересно будет, если упомянуть, что хоть передача массивов в функцию имеет такую специфику, тем не менее, массивы фиксированной длины можно передавать в функции, например, так:

```

1  #include <iostream>
2
3  void print_array(const int (&ar)[5]) {
4      for (const int& x : ar)
5          std::cout << x << ' ';
6  }
7
8  int main() {
9
10     int ar1[3] = {1, 2, 3};
11     int ar2[5] = {1, 2, 3, 4, 6};
12
13     print_array(ar1); // не скомпилируется
14     print_array(ar2); // всё хорошо
15
16     return 0;
17 }
```

[Ответить](#)



2. *Определения это здорово:*

[16 января 2020 в 18:44](#)

"int array[4] = { 5, 8, 6, 4 }; // объявляем фиксированный массив из 4-ёх целых чисел" — не объявляем, а определяем/инициализируем. Зачем так долго учить терминологии чтобы потом ее полностью игнорировать?

int array[4]; — вот объявление, правильно?

array[4] = { 5, 8, 6, 4 }; — вот инициализация, правильно?

int array[4] = { 5, 8, 6, 4 }; — это либо объявление с инициализацией, либо определение, что тоже самое, правильно?

[Ответить](#)



3. *Константин:*

[6 июля 2019 в 19:08](#)

Так ещё яснее:

```

1  #include <iostream>
2  #include <Windows.h>
3  #include <typeinfo>
4
5  int main()
6
```

```

7  { SetConsoleCP(1251); SetConsoleOutputCP(1251); using std::cout; using std::endl;
8  cout << "Трёхмерная массивная переменная f:\n";
9  const int x{4}, y{5}, z{3};
10 double f[z][y][x] =
11 {
12     {
13         {0.01, 0.02, 0.03, 0.04},
14         {0.11, 0.12, 0.13, 0.14},
15         {0.21, 0.22, 0.23, 0.24},
16         {0.31, 0.32, 0.33, 0.34},
17         {0.41, 0.42, 0.43, 0.44}
18     },
19     {
20         {1.01, 1.02, 1.03, 1.04},
21         {1.11, 1.12, 1.13, 1.14},
22         {1.21, 1.22, 1.23, 1.24},
23         {1.31, 1.32, 1.33, 1.34},
24         {1.41, 1.42, 1.43, 1.44}
25     },
26     {
27         {25.4, 36.6, 69.1, 22.2},
28         {11.7, 77.5, 12.2, 54.2},
29         {90.4, 85.2, 81.9, 38.6},
30         {13.5, 51.3, 19.8, 42.0},
31         {66.6, 77.7, 88.8, 99.9}
32     }
33 };
34 for(int i{}; i < z; ++i)
35 {
36     cout << "\nЭтаж " << i << "\n";
37     for(int k{}; k < y; ++k)
38     {
39         cout << "\n";
40         for(int l{}; l < x; ++l)
41             cout << f[i][k][l] << "|";
42     }
43 }
44 cout << endl; cout << endl;
45 cout << "          f....." << f << endl;
46 cout << "          *f....." << *f << endl;
47 cout << "          **f....." << **f << endl;
48 cout << "          ***f....." << ***f << " - значение начальной ячейки\n";
49 cout << endl;
50 cout << "          &f....." << &f << " - адрес массивной переменной\n";
51 cout << endl;
52
53 cout << "          *&f....." << *&f << endl;
54 cout << "          **&f....." << **&f << endl;
55 cout << "          ***&f....." << ***&f << endl;

```



```

56 cout << "      ****&f....." << ****&f << " - получение значения с адре
57 cout << endl;
58 cout << " Поименуем указатель т.е. выполним стэйтмент: double *ptrONf = ***&
59 double *ptrONf = ***&f;
60 cout << " ptrONf....." << ptrONf << endl;
61 cout << "Разыменуем указатель:\n";
62 cout << " *ptrONf....." << *ptrONf << endl;
63 cout << "Адрес указателя:\n";
64 cout << " &ptrONf....." << &ptrONf << endl;
65 cout << "Значение на адресе указателя:\n";
66 cout << "**&ptrONf....." << *&ptrONf << endl;
67 cout << "typeid( ptrONf).name().." << typeid( ptrONf).name() << endl;
68 cout << "typeid( *ptrONf).name().." << typeid( *ptrONf).name() << endl;
69 cout << "typeid( &ptrONf).name().." << typeid( &ptrONf).name() << endl;
70 cout << "typeid(*&ptrONf).name().." << typeid(*&ptrONf).name() << endl;
71 cout << endl;
72 cout << "typeid(      f).name()..." << typeid(      f).name() << endl;
73 cout << "typeid(     *f).name()..." << typeid(     *f).name() << endl;
74 cout << "typeid(    **f).name()..." << typeid(    **f).name() << endl;
75 cout << "typeid(   ***f).name()..." << typeid(   ***f).name() << " - тип значе
76 cout << endl;
77 cout << "typeid(    &f).name()..." << typeid(    &f).name() << " - тип указа
78 cout << endl;
79 cout << "typeid(   *&f).name()..." << typeid(   *&f).name() << " - A[z]этажи
80 cout << "typeid(  **&f).name()..." << typeid(  **&f).name() << endl;
81 cout << "typeid( ***&f).name()..." << typeid( ***&f).name() << endl;
82 cout << "typeid(****&f).name()..." << typeid(****&f).name() << " - тип указа
83 cout << endl;
84 cout << "      f[0][0][0]....." <<      f[0][0][0] << endl;
85 // cout << "     *f[0][0][0]....." <<     *f[0][0][0] << endl;
86 cout << "    &f[0][0][0]....." <<    &f[0][0][0] << endl;
87 cout << "   *&f[0][0][0]....." <<   *&f[0][0][0] << endl;
88 cout << "typeid(      f[0][0][0]).name()..." << typeid(      f[0][0][0]).name() <<
89 // cout << "typeid(     *f[0][0][0]).name()..." << typeid(     *f[0][0][0]).name() <<
90 cout << "typeid(    &f[0][0][0]).name()..." << typeid(    &f[0][0][0]).name() <<
91 cout << "typeid(   *&f[0][0][0]).name()..." << typeid(   *&f[0][0][0]).name() <<
92 cout << endl;
93 cout << "Пример из урока 82\n"; cout << endl;
94 int array[4] = { 5, 8, 6, 4 };
95 cout << "      array....." <<      array << endl;
96 cout << "typeid(  array).name()..." << typeid(  array).name() << endl;
97 // Разыменование массива (переменной array) приведёт к возврату первого элемента
98 cout << "      *array....." <<      *array << endl; // вы
99 cout << "typeid( *array).name()..." << typeid( *array).name() << endl;
100 cout << "      &array....." <<      &array << endl;
101 cout << "typeid( &array).name()..." << typeid( &array).name() << endl;
102 cout << "      *&array....." <<      *&array << endl;
103 cout << "typeid(*&array).name()..." << typeid(*&array).name() << " - A[4]сто
104

```

```

105     cout << endl;
106     int *ptrONarray = array;
107     cout << " ptrONarray....." << ptrONarray << endl;
108     cout << "typeid( ptrONarray).name()..." << typeid( ptrONarray).name() << endl;
109     cout << " *ptrONarray....." << *ptrONarray << endl;
110     cout << "typeid( *ptrONarray).name()..." << typeid( *ptrONarray).name() << endl;
111     cout << " &ptrONarray....." << &ptrONarray << endl;
112     cout << "typeid( &ptrONarray).name()..." << typeid( &ptrONarray).name() << endl;
113     cout << "&*ptrONarray....." << &*ptrONarray << endl;
        cout << "typeid(&*ptrONarray).name()..." << typeid(&*ptrONarray).name() << endl;

```

Ответить1. *Марат:*10 января 2020 в 07:27

Добрый день! У меня текст кириллицей не верно отображается.
Каракули русскими символами.

Ответить1. *Евгений:*13 февраля 2020 в 20:53

Марат, скорее всего, вы используете IDE с поддержкой UTF-8.
Посмотрите этот пример <https://ideone.com/LcNKOC>

Ответить4. *Константин:*6 июля 2019 в 17:01

Вот код для лучшего понимания темы:

```

1  #include <iostream>
2  #include <Windows.h>
3  #include <typeinfo>
4
5  int main()
6  {
7      SetConsoleCP(1251); SetConsoleOutputCP(1251); using std::cout; using std::endl;
8      cout << "Трёхмерная массивная переменная f:\n";
9      const int x{4}, y{5}, z{3};
10     double f[z][y][x] =
11     {
12         {
13             {0.01, 0.02, 0.03, 0.04},

```

```

14         {0.11, 0.12, 0.13, 0.14},
15         {0.21, 0.22, 0.23, 0.24},
16         {0.31, 0.32, 0.33, 0.34},
17         {0.41, 0.42, 0.43, 0.44}
18     },
19     {
20         {1.01, 1.02, 1.03, 1.04},
21         {1.11, 1.12, 1.13, 1.14},
22         {1.21, 1.22, 1.23, 1.24},
23         {1.31, 1.32, 1.33, 1.34},
24         {1.41, 1.42, 1.43, 1.44}
25     },
26     {
27         {25.4, 36.6, 69.1, 22.2},
28         {11.7, 77.5, 12.2, 54.2},
29         {90.4, 85.2, 81.9, 38.6},
30         {13.5, 51.3, 19.8, 42.0},
31         {66.6, 77.7, 88.8, 99.9}
32     }
33 };
34 for(int i{}; i < z; ++i)
35 {
36     cout << "\nЭтаж " << i << "\n";
37     for(int k{}; k < y; ++k)
38     {
39         cout << "\n";
40         for(int l{}; l < x; ++l)
41             cout << f[i][k][l] << "|";
42     }
43 }
44 cout << endl; cout << endl;
45 cout << "          f....." << f << endl;
46 cout << "          *f....." << *f << endl;
47 cout << "          **f....." << **f << endl;
48 cout << "          ***f....." << ***f << " - значение начальной ячейки!"
49 cout << endl;
50 cout << "          &f....." << &f << " - адрес массивной переменной"
51 cout << endl;
52 cout << "          *&f....." << *&f << endl;
53 cout << "          **&f....." << **&f << endl;
54 cout << "          ***&f....." << ***&f << endl;
55 cout << "          ****&f....." << ****&f << " - получение значения с адреса"
56 cout << endl;
57 cout << "typeid(      f).name()..." << typeid(      f).name() << endl;
58 cout << "typeid(     *f).name()..." << typeid(     *f).name() << endl;
59 cout << "typeid(    **f).name()..." << typeid(    **f).name() << endl;
60 cout << "typeid(   ***f).name()..." << typeid(   ***f).name() << " - тип значения"
61 cout << endl;
62 cout << "typeid(    &f).name()..." << typeid(    &f).name() << " - тип указателя"

```

```

63     cout << endl;
64     cout << "typeid(    *&f).name()..." << typeid(    *&f).name() << " - A[z]этажи_";
65     cout << "typeid(   **&f).name()..." << typeid(   **&f).name() << endl;
66     cout << "typeid(  ***&f).name()..." << typeid(  ***&f).name() << endl;
67     cout << "typeid(****&f).name()..." << typeid(****&f).name() << " - тип указате";
68     cout << endl;
69     cout << "          f[0][0][0]....." <<          f[0][0][0] << endl;
70 // cout << "          *f[0][0][0]....." <<          *f[0][0][0] << endl; // не
71     cout << "          &f[0][0][0]....." <<          &f[0][0][0] << endl;
72     cout << "          *&f[0][0][0]....." <<          *&f[0][0][0] << endl;
73     cout << "typeid(    f[0][0][0]).name()..." << typeid(    f[0][0][0]).name() <<
74 // cout << "typeid(   *f[0][0][0]).name()..." << typeid(   *f[0][0][0]).name() <<
75     cout << "typeid(    &f[0][0][0]).name()..." << typeid(    &f[0][0][0]).name() <<
76     cout << "typeid(   *&f[0][0][0]).name()..." << typeid(*&f[0][0][0]).name() <<
77     cout << endl;
78     cout << "Пример из урока 82\n"; cout << endl;
79     int array[4] = { 5, 8, 6, 4 };
80     cout << "          array....." <<          array << endl;
81     cout << "typeid(  array).name()..." << typeid(  array).name() << endl;
82 // Разыменование массива (переменной array) приведёт к возврату первого элемента и
83     cout << "          *array....." <<          *array << endl; // выв
84     cout << "typeid( *array).name()..." << typeid( *array).name() << endl;
85     cout << "          &array....." <<          &array << endl;
86     cout << "typeid( &array).name()..." << typeid( &array).name() << endl;
87     cout << "          *&array....." <<          *&array << endl;
88     cout << "typeid(*&array).name()..." << typeid(*&array).name() << "A[4]столбцы";

```

[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.

[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