

Массивы

Михаил Марков
C++ - Разработчик



Михаил Марков

О спикере:

С++-разработчик, фрилансер

- Разработка алгоритма для релевантной выдачи объявлений.
- Разработка эмуляторов оборудования



Вспоминаем прошлое занятие

Вопрос: какие виды циклов вы знаете?

Вспоминаем прошлое занятие

Вопрос: какие виды циклов вы знаете?

Ответ: циклы while, do...while, for

Вспоминаем прошлое занятие

Вопрос: какие ключевые слова для управления работой циклов вы знаете?

Вспоминаем прошлое занятие

Вопрос: какие ключевые слова для управления работой циклов вы знаете?

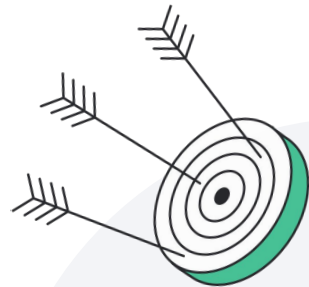
Ответ: continue, break

Цели занятия

1 Научимся создавать и инициализировать массивы, взаимодействовать с ними

2 Научимся создавать и инициализировать двумерные массивы, взаимодействовать с ними

3 Разберём алгоритм сортировки пузырьком



План занятия

- 1 [Массивы](#)
- 2 [Двумерные массивы](#)
- 3 [Сортировка](#)
- 4 [Итоги](#)
- 5 [Домашнее задание](#)

Массивы

3

Массив

Структура данных фиксированного размера, которая содержит элементы одного типа

В C++ индексация элементов массива начинается с 0



0	1	2	3	4	5
И	Н	Д	Е	К	С

Массив

Пример массива целых чисел длиной 10 элементов:

Первый индекс



Элемент под индексом 5



Индексы	0	1	2	3	4	5	6	7	8	9
Элементы	5	7	3	3	9	44	9	1	0	2

← Массив длиной 10 элементов →

Объявление и создание

Чтобы объявить и создать массив, нужно обязательно указать:

- **Тип** элементов массива
- **Название** (имя) массива
- **Размер** массива — количество элементов в этом массиве

Пример создания массива, содержащего целые числа (**тип int**), который называется **arr**, а его размер составляет **8 элементов**:

```
int arr[8];
```

Так массив будет создан в памяти, но ещё не инициализирован, то есть будет заполнен случайными значениями

Объявление и создание

Примеры объявления и создания массивов разного типа и размера:

```
int arr_int[3];  
char arr_char[6];  
double arr_double[9];
```


Для указания размера массива можно использовать константу, но не переменную:

```
const int size = 10;  
int arr_int[size];  
char arr_char[size];  
double arr_double[size];
```

Инициализация

Инициализация массива — это заполнение его конкретными значениями. Массив можно инициализировать сразу при создании:

```
int arr_int[6] = {1, 2, 3, 4, 5, 6};  
char arr_char[5] = {'H', 'e', 'l', 'l', 'o'};
```




Набор элементов, указанный в фигурных скобках, носит название «**список инициализации**»

Инициализация

Инициализация массива — это заполнение его конкретными значениями. Массив можно инициализировать сразу при создании:

```
int arr_int[6] = {1, 2, 3, 4, 5, 6};  
char arr_char[5] = {'H', 'e', 'l', 'l', 'o'};
```



Набор элементов, указанный в фигурных скобках, носит название «**список инициализации**».

При создании массива с использованием списка инициализации можно не указывать размер массива: компилятор вычислит его сам на основе размера списка инициализации.

```
int arr_int [ ] = {1, 2, 3, 4, 5, 6};  
char arr_char [ ] = {'H', 'e', 'l', 'l', 'o'};
```

Но нельзя указывать массиву размер, не соответствующий размеру списка инициализации. Тогда программа не скомпилируется

Особенности инициализации символьных массивов

Символьный массив (массив элементов типа `char`) можно инициализировать как отдельными элементами, так и целым строковым литералом. У этих двух способов есть существенное различие

```
// отдельными символами
char arr1[] = {'H', 'e', 'l', 'l', 'o'};
// целой строкой
char arr2[] = "Hello";
```

В первом случае массив будет содержать 5 элементов, а во втором случае — 6.

Так происходит потому, что второй массив содержит дополнительный невидимый символ, который находится в самом конце и обозначает конец строки.

Подробнее мы познакомимся с ним позже

Доступ к элементам

Чтобы получить значение элемента массива, нужно обратиться к нему по его индексу в массиве. В C++ индексация массива начинается с 0

```
int arr[] = { 5, 7, 3, 3, 9, 44, 9, 1, 0, 2 };
```

```
int arr0 = arr[0]; // 5
```

```
int arr1 = arr[1]; // 7
```

```
int arr5 = arr[5]; // ???
```

```
int arr9 = arr[9]; // ???
```

```
int arr10 = arr[10]; // ???
```

Какие значения появятся?



Доступ к элементам

Чтобы получить значение элемента массива, нужно обратиться к нему по его индексу в массиве. В C++ индексация массива начинается с 0

```
int arr[] = { 5, 7, 3, 3, 9, 44, 9, 1, 0, 2 };
```

```
int arr0 = arr[0]; // 5
```

```
int arr1 = arr[1]; // 7
```

```
int arr5 = arr[5]; // 44
```

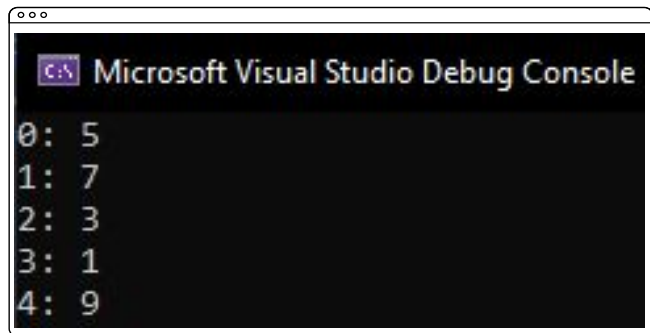
```
int arr9 = arr[9]; // 2
```

```
int arr10 = arr[10]; // Ошибка!
```

Доступ к элементам. Цикл for

Для перебора элементов массива удобно использовать цикл `for`:

```
const int size = 5;  
int arr[size] = { 5, 7, 3, 1, 9 };  
for (int i = 0; i < size; i++)  
{  
    std::cout << i << ": " << arr[i] << std::endl;  
}
```



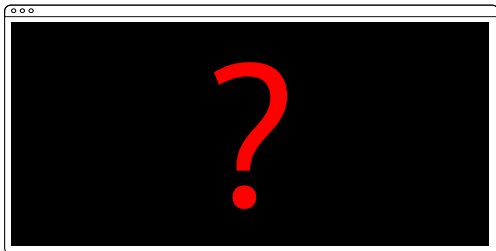
The screenshot shows a window titled "C:\> Microsoft Visual Studio Debug Console". The output displayed is:

```
0: 5  
1: 7  
2: 3  
3: 1  
4: 9
```

Доступ к элементам. Цикл for

Что выведет этот код на экран?

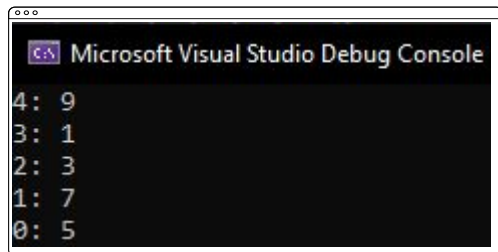
```
const int size = 5;  
int arr [size] = { 5, 7, 3, 1, 9 };  
for (int i = size - 1; i >= 0; i--)  
{  
    std::cout << i << ". " << arr [i] << std::endl;  
}
```



Доступ к элементам. Цикл for

Значения массива будут выведены в обратном порядке

```
const int size = 5;  
int arr[size] = { 5, 7, 3, 1, 9 };  
for (int i = size - 1; i >= 0; i--)  
{  
    std::cout << i << ": " << arr[i] << std::endl;  
}
```



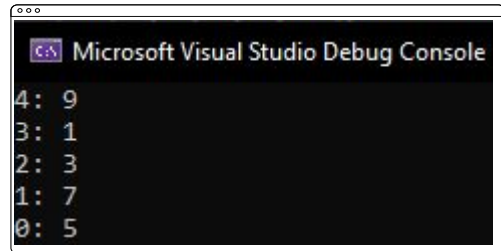
```
Microsoft Visual Studio Debug Console  
4: 9  
3: 1  
2: 3  
1: 7  
0: 5
```

Доступ к элементам. Цикл for

Как найти длину массива, созданного с помощью списка инициализации без явного указания его длины.

Для этого используется оператор **sizeof** — он возвращает размер операнда, к которому применяется, в байтах. Чтобы найти количество элементов в массиве, нужно размер всего массива разделить на размер одного элемента этого массива

```
int arr [ ] = { 5, 7, 3, 1, 9 };  
// делим размер всего массива на размер одного элемента  
int size = sizeof(arr)/sizeof(arr [ 0 ]);  
for (int i = size - 1; i >= 0; i--)  
{  
    std::cout << i << ": " << arr[i] << std::endl;  
}
```

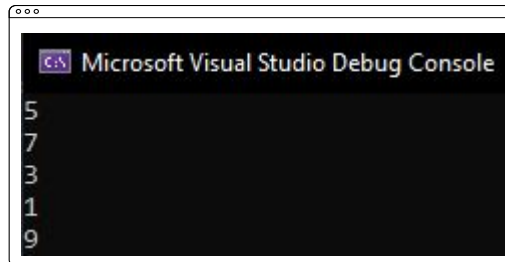


```
Microsoft Visual Studio Debug Console  
4: 9  
3: 1  
2: 3  
1: 7  
0: 5
```

Специальная форма цикла for

Специальная форма цикла `for` обеспечивает простой и понятный способ итерации по элементам массива:

```
int arr[] = { 5, 7, 3, 1, 9 };  
for (int number : arr)  
{  
    std::cout << number << std::endl;  
}
```



Однако при всей простоте и удобстве такого способа, у него есть один недостаток: он не позволяет изменять значения элементов в исходном массиве

Двумерные массивы

4

A decorative graphic consisting of two overlapping circles. The circle on the left is white with a thin black outline and contains the number '4' in a bold, black, sans-serif font. The circle on the right is partially visible, showing a white outline and a light blue fill.

Двумерный массив

Массивы могут иметь несколько измерений. Элементы такого массива сами являются массивами.

Пример двумерного массива целых чисел размером 3x3 элементов:

```
int arr[3][3] =  
{  
  {5, 3, 4},  
  {6, 7, 2},  
  {1, 9, 8}  
};
```

	0	1	2
0	5	3	4
1	6	7	2
2	1	9	8

Этот двумерный массив состоит из трёх одномерных массивов, каждый из которых содержит 3 целых числа

Двумерный массив

При объявлении многомерного массива и использовании списка инициализации можно не указывать явно самую первую размерность массива, но остальные указать придётся.

Пример **некорректного** объявления двумерного массива целых чисел размером 3x3 элемента:

```
int arr[] [] =  
{  
  {5, 3, 4},  
  {6, 7, 2},  
  {1, 9, 8}  
};
```

	0	1	2
0	5	3	4
1	6	7	2
2	1	9	8

Ошибка!

Двумерный массив

При объявлении многомерного массива и использовании списка инициализации можно не указывать явно самую первую размерность массива, но остальные указать придётся.

Пример корректного объявления двумерного массива целых чисел размером 3x3 элемента:

```
int arr [ ] [3] =  
{  
  {5, 3, 4},  
  {6, 7, 2},  
  {1, 9, 8}  
};
```

	0	1	2
0	5	3	4
1	6	7	2
2	1	9	8

Доступ к элементам

Каждый элемент в двумерном массиве индексируется с помощью двух чисел: одно число определяет строку, второе определяет столбец. Для доступа к элементу двумерного массива необходимо указать индексы элемента в квадратных скобках:

```
int arr[3][3] = { {5, 3, 4}, {6, 7, 2}, {1, 9, 8} };
```

```
int arr_0_0 = a[0][0]; // 5
```

```
int arr_1_2 = a[1][2]; // 2
```

```
int arr_2_2 = a[2][2]; // 8
```

```
int arr_2_1 = a[2][1]; // ?
```

```
int arr_1_1 = a[1][1]; // ?
```

```
int arr_0_1 = a[0][1]; // ?
```

Какие значения появятся?



Доступ к элементам

Каждый элемент в двумерном массиве индексируется с помощью двух чисел: одно число определяет строку, второе определяет столбец. Для доступа к элементу двумерного массива необходимо указать индексы элемента в квадратных скобках:

```
int arr[3][3] = {{5, 3, 4}, {6, 7, 2}, {1, 9, 8}};
```

```
int arr_0_0 = a[0][0];           // 5
```

```
int arr_1_2 = a[1][2];           // 2
```

```
int arr_2_2 = a[2][2];           // 8
```

```
int arr_2_1 = a[2][1];           // 9
```

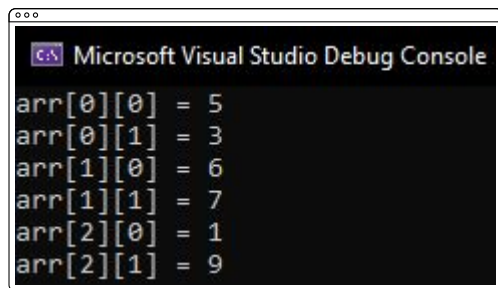
```
int arr_1_1 = a[1][1];           // 7
```

```
int arr_0_1 = a[0][1];           // 3
```

Доступ к элементам

Для перебора элементов двумерного массива удобно использовать вложенный цикл `for`

```
const int rows = 3, columns = 2;
int arr[rows][columns] = { {5, 3}, {6, 7}, {1, 9} };
for (int i = 0; i < rows; i++)
{
    for (int j = 0; j < columns; j++)
    {
        std::cout << "arr[" << i << "][" << j << "] = " << arr[i][j] << std::endl;
    }
}
```

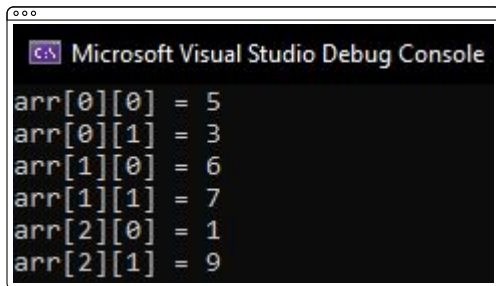


```
Microsoft Visual Studio Debug Console
arr[0][0] = 5
arr[0][1] = 3
arr[1][0] = 6
arr[1][1] = 7
arr[2][0] = 1
arr[2][1] = 9
```

Доступ к элементам

Длину двумерного массива можно также определить с помощью оператора `sizeof`

```
int arr [ ] [ 2 ] = { { 5, 3 }, { 6, 7 }, { 1, 9 } };
int rows = sizeof(arr)/sizeof(arr [ 0 ]);
int columns = sizeof(arr [ 0 ])/sizeof(arr [ 0 ] [ 0 ]);
for (int i = 0; i < rows; i++)
{
    for (int j = 0; j < columns; j++)
    {
        std::cout << "arr [ " << i << " ] [ " << j << " ] = " << arr [ i ] [ j ] << std::endl;
    }
}
```

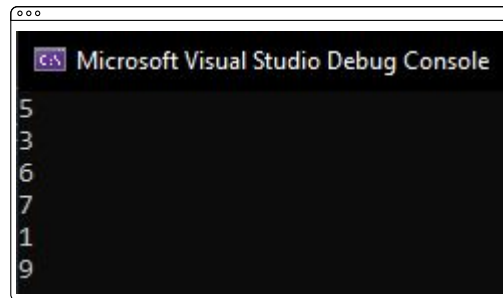


```
Microsoft Visual Studio Debug Console
arr[0][0] = 5
arr[0][1] = 3
arr[1][0] = 6
arr[1][1] = 7
arr[2][0] = 1
arr[2][1] = 9
```

Доступ к элементам

Для перебора элементов двумерного массива также удобно использовать специальную форму цикла `for`:

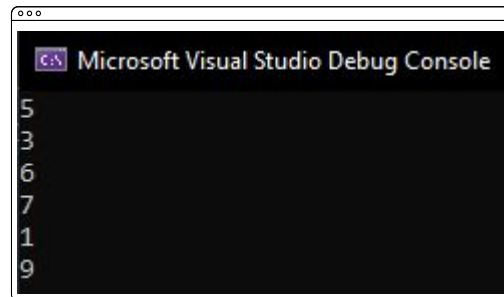
```
int arr [ ] [2] = { {5, 3}, {6, 7}, {1, 9} };  
for (auto &sub_arr : arr)  
{  
    for (int number : sub_arr)  
    {  
        std::cout << number << std::endl;  
    }  
}
```



Доступ к элементам

В этом примере в верхнем цикле `for` мы итерируемся по нижестоящим массивам двумерного массива, поэтому тип переменной `sub_arr` не `int`. Мы указываем ключевое слово `auto`, которое говорит, что компилятор сам определит тип переменной.

```
int arr [ ] [ 2 ] = { { 5, 3 }, { 6, 7 }, { 1, 9 } };  
for (auto &sub_arr : arr)  
{  
    for (int number : sub_arr)  
    {  
        std::cout << number << std::endl;  
    }  
}
```



Также обратите внимание на специальный символ `&` (амперсанд) — без него код не будет работать. Всё это мы разберём позже

Сортировка



5

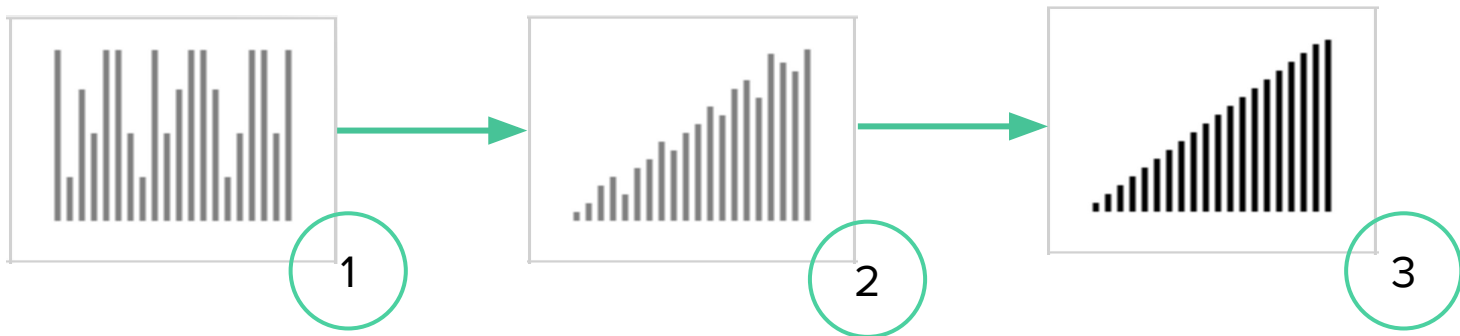
The diagram consists of two overlapping circles on a teal background. The left circle is white and contains the number '5' in teal. The right circle is teal with a white outline and is partially cut off by the right edge of the frame.

Сортировка

Сортировка — это процесс размещения элементов множества в определённом порядке.

Когда данные отсортированы, с ними проще работать и выполнять различные действия:

- поиск конкретного элемента
- поиск пропущенных элементов
- сравнение массивов



Алгоритмы сортировки

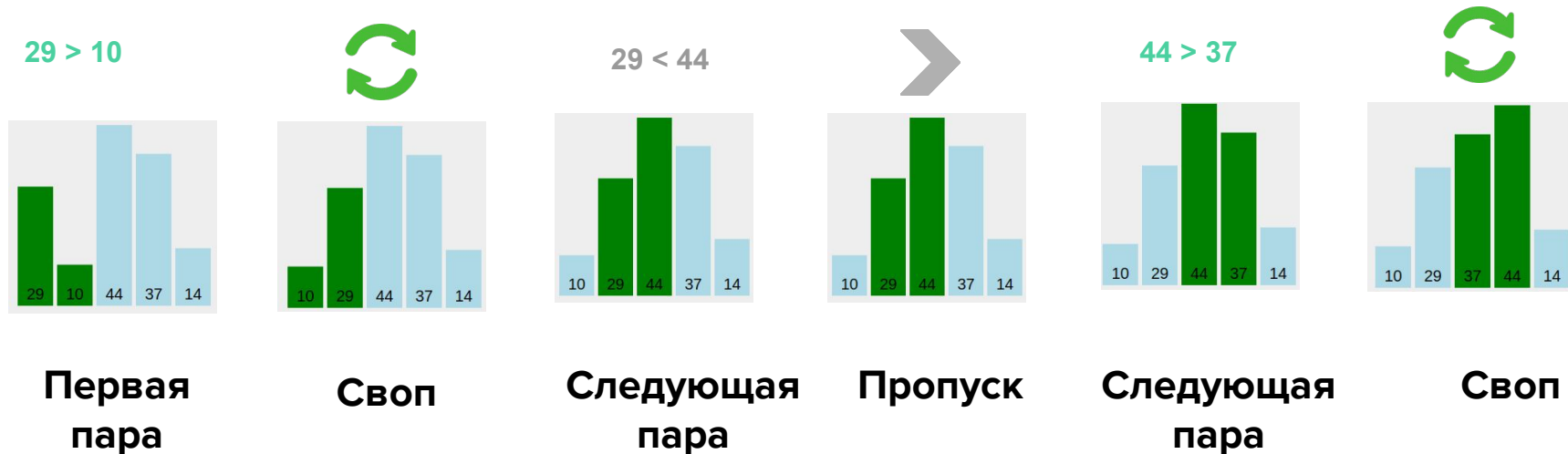
Существует множество различных алгоритмов сортировки, некоторые из них:

- 1 Сортировка пузырьком**
Простой алгоритм сортировки сравнивает все элементы массива попарно и при необходимости производит обмен. Отличается низкой эффективностью
- 2 Сортировка вставкой**
По очереди анализирует элементы массива и помещает элемент в подходящее для него место среди ранее упорядоченных элементов
- 3 Сортировка слиянием**
Отдельно сортирует части массива, после чего выполняет слияние отсортированных частей. Обычно реализуется при помощи рекурсии
- 4 Быстрая сортировка**
Один из самых эффективных универсальных алгоритмов сортировки
- 5 Сортировка подсчётом**
Применима для сортировки чисел, только если они имеют ограниченный диапазон возможных значений

Сортировка пузырьком

Алгоритм состоит из нескольких проходов по массиву от начала до конца.

При каждом проходе, если соседние элементы не отсортированы, то они меняются местами. Таким образом после первого прохода на последнее место всплывёт максимальный элемент. После второго прохода на предпоследнем месте окажется следующий



Псевдокод

процедура сортировка(A : массив)

 повторять

 запомнить, что перестановок нет

 для i от 1 до длины A повторять

 если $A[i-1] > A[i]$ тогда

 поменять местами $A[i-1]$ и $A[i]$

 запомнить, что была перестановка

 end if

end for

пока были перестановки

конец процедуры



**Напишем программу, которая
реализует сортировку
пузырьком**

[Готовый пример кода](#)

Итоги

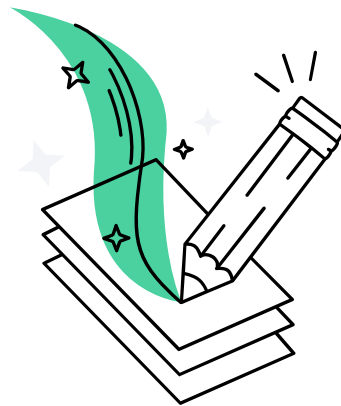
Сегодня мы:

- 1 Узнали, как создавать массивы и работать с ними
- 2 Узнали, как создавать двумерные массивы и работать с ними
- 3 Разобрали алгоритм сортировки пузырьком

Домашнее задание

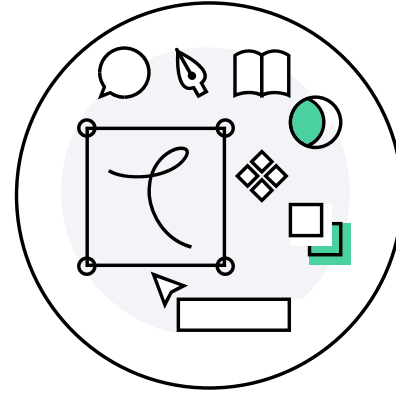
Давайте посмотрим ваше домашнее задание.

- 1 Вопросы по домашней работе задавайте в чате группы
- 2 Задачи можно сдавать по частям
- 3 Зачёт по домашней работе ставят после того, как приняты все задачи



Дополнительные материалы

- [Массивы, многомерные массивы](#)
- [Массивы, заполнение с клавиатуры](#)



Задавайте вопросы и пишите отзыв о лекции

Михаил Марков
C++ - разработчик

