

# Указатели. Массивы и параметры функций

Владислав Хорев  
Ведущий программист, Luxoft



# Владислав Хорев

О спикере:

- Ведущий программист в компании Luxoft
- Работает в IT с 2011 года
- Опыт разработки на C++ более 10 лет



# Вспоминаем прошрое занятие

**Вопрос:** какие бывают типы памяти?



# Вспоминаем прошрое занятие

**Вопрос:** какие бывают типы памяти?

**Ответ:** статическая, автоматическая,  
динамическая



# Вспоминаем прошрое занятие

**Вопрос:** что такое статическая память?



# Вспоминаем прошрое занятие

**Вопрос:** что такое статическая память?

**Ответ:** статическая память выделяется один раз в фиксированном размере перед стартом программы. В ней размещаются в том числе статические объекты, которые существуют в течение всего времени работы программы



# Вспоминаем прошрое занятие

**Вопрос:** что такое автоматическая память?



# Вспоминаем прошрое занятие

**Вопрос:** что такое автоматическая память?

**Ответ:** автоматическая память выделяется один раз в фиксированном размере при старте программы. В ней размещаются локальные переменные и информация о последовательности вызванных функций





# Вспоминаем прошрое занятие

**Вопрос:** что такое динамическая память?



# Вспоминаем прошрое занятие

**Вопрос:** что такое динамическая память?

**Ответ:** динамическая память выделяется операционной системой по требованию программиста. Её размер ограничен фактической оперативной памятью компьютера.

Программист должен не забыть очистить память (вернуть её операционной системе) после того, как она станет ему не нужна, иначе произойдёт утечка памяти



# Вспоминаем прошрое занятие

**Вопрос:** что такое пространство имён?



# Вспоминаем прошрое занятие

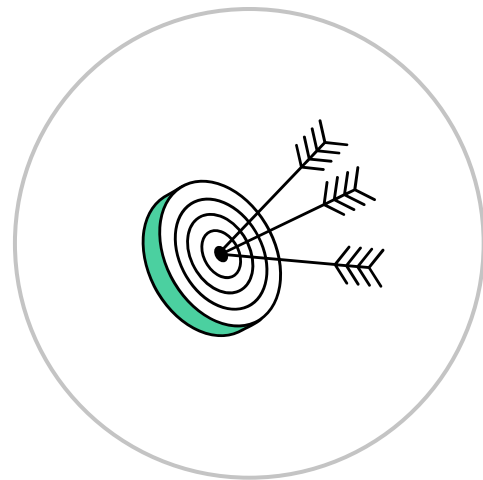
**Вопрос:** что такое пространство имён?

**Ответ:** пространство имён — это средство для обеспечения уникальности потенциально неуникальных имён глобальных объектов: функций и глобальных переменных



# Цели занятия

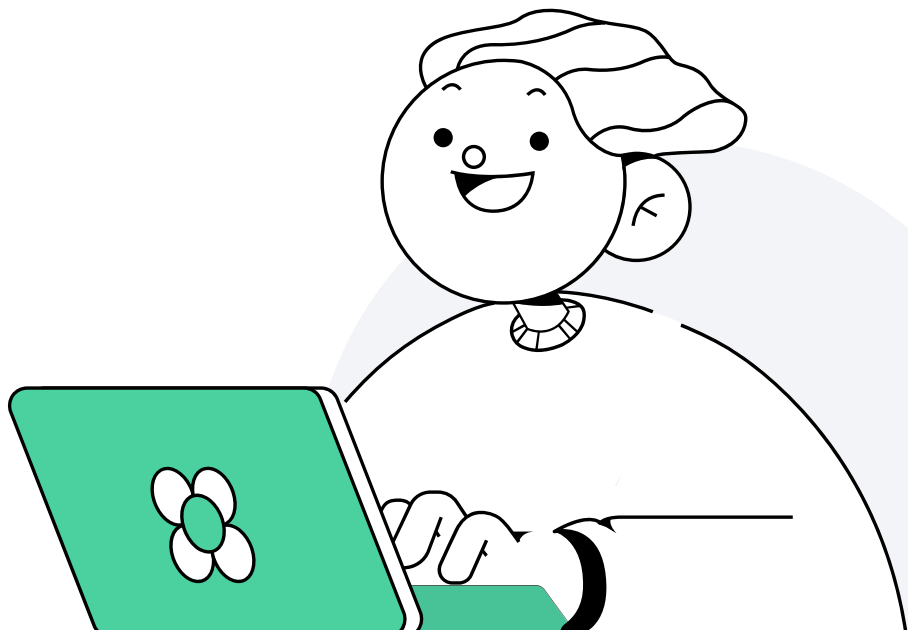
- Разберёмся с оператором typeid
- Познакомимся с указателями
- Узнаем, что такое указатели на указатели
- Выясним, как передавать массивы в функцию



# План занятия

- 1 Оператор typeid
- 2 Указатели
- 3 Массивы и параметры функций
- 4 Итоги
- 5 Домашнее задание

\*Нажми на нужный раздел для перехода



# Оператор typeid



1

# Узнаём тип значения

Прежде чем мы перейдём к указателям, нам надо познакомиться с инструментом, позволяющим узнать тип переменной или значения.

Чтобы узнать **тип** переменной или литерала, применяется оператор **typeid**.

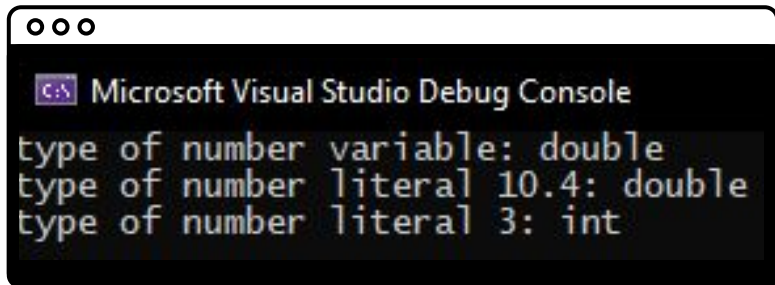
Использование оператора выглядит так:

```
typeid(<имя переменной или литерал>).name()
```



# Пример использования typeid

```
int main(int argc, char** argv)
{
    double number = 5.6;
    std::cout << "type of number variable: " << typeid(number).name() << std::endl;
    std::cout << "type of number literal 10.4: " << typeid(10.4).name() << std::endl;
    std::cout << "type of number literal 3: " << typeid(3).name() << std::endl;
    return 0;
}
```



ooo

Microsoft Visual Studio Debug Console

```
type of number variable: double
type of number literal 10.4: double
type of number literal 3: int
```

# Указатели



2

# Адреса переменных

Проведём эксперимент. **&** (амперсанд) — это оператор взятия адреса. Он позволяет нам узнать адрес ячейки памяти, на которую «смотрит» переменная.

Раньше мы всё время просто выводили адрес на консоль. Давайте попробуем **сохранить его** (т. е. результат работы оператора **&**) **в переменную**

# Вспоминаем

**Вопрос:** адрес переменной — это целое шестнадцатеричное число. Какого типа переменную нужно создать, чтобы хранить адрес переменной?



# Вспоминаем

**Вопрос:** адрес переменной — это целое шестнадцатеричное число. Какого типа переменную нужно создать, чтобы хранить адрес переменной?

**Ответ:** переменную типа **int**



# Адреса переменных

```
int main(int argc, char** argv)
{
    int number = 5;
    int address = &number; // Ошибка компиляции. Несоответствие типа значения типу
    переменной
    return 0;
}
```

Код выше не скомпилируется. Как думаете, почему?

Напишите ответ в чат

# Адреса переменных

```
int main(int argc, char** argv)
{
    int number = 5;
    int address = &number; // Ошибка компиляции. Несоответствие типа значения типу
    переменной
    return 0;
}
```

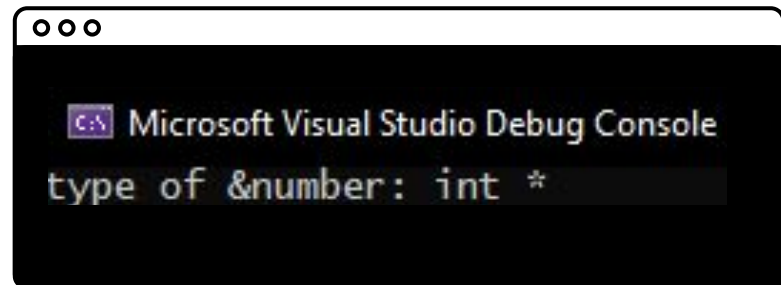
**Ответ:** потому что тип значения, возвращаемого оператором `&`, не соответствует типу `int`.

Какой же тогда у этого значения тип?

# Какой это тип?

Чтобы узнать тип значения адреса, используем только что изученный нами оператор **typeid**

```
int main(int argc, char** argv)
{
    int number = 5;
    std::cout << "type of &number: " << typeid(&number).name() << std::endl;
    return 0;
}
```





# Указатель

Оператор взятия адреса при применении к типу **int** возвращает значение типа **int \***.

**int \*** — это один из типов в C++, который используется для **указателей**.

Указатель — это переменная, значение которой — адрес ячейки памяти.

Объявляется указатель так: **<тип данных> \* <имя указателя>;**

# Указатель

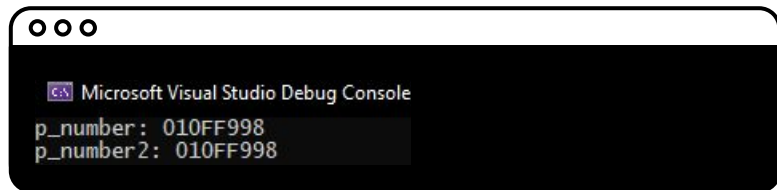
Тип указателя содержит информацию о том, на значение какого типа он указывает.

Указатель типа **int** \* будет содержать в себе адрес значения типа **int**, указатель типа **double** \* будет указывать на значение типа **double**

# Сохраним адрес в переменную

Теперь сохраним адрес нашей целочисленной переменной в другую переменную:

```
int main(int argc, char** argv)
{
    int number = 5;
    int * p_number = &number;    // Инициализируем указатель сразу при объявлении
    int * p_number2;              // Сначала объявляем
    p_number2 = &number;          // Потом инициализируем
    std::cout << "p_number: " << p_number << std::endl;
    std::cout << "p_number2: " << p_number2 << std::endl;
    return 0;
}
```



# Зачем это нужно

Зачем же нам нужны все эти указатели? Какой смысл в том, чтобы сохранять адрес в переменных?

С помощью указателей мы можем не только передавать адрес в функции, но и получать **доступ к содержимому ячейки**, в том числе **изменять это содержимое**

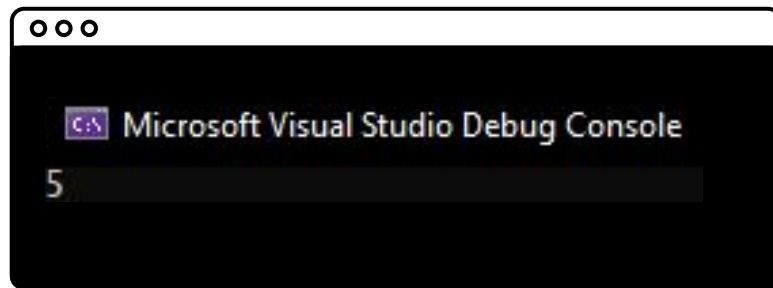
# Оператор разыменования

Получить доступ к ячейке, адрес которой хранится в указателе, можно с помощью **оператора разыменования** — \* (звёздочка).

Чтобы разыменовать указатель, необходимо написать оператор разыменования **перед** указателем: так же, как мы берём адрес переменной.

Взятие адреса (&) и разыменование (\*) — это обратные операции:

```
int main(int argc, char** argv)
{
    int number = 5;
    std::cout << *&number << std::endl;
    return 0;
}
```



# Разыменование: пример

Изменим содержимое ячейки памяти с помощью указателя:

```
int main(int argc, char** argv)
{
    int number = 5;
    int * p_number = &number;
    std::cout << "number: " << number << std::endl;           // что выведет?
    std::cout << "*p_number: " << *p_number << std::endl;      // что выведет?
    *p_number = 10;
    std::cout << "number: " << number << std::endl;           // что выведет?
    std::cout << "*p_number: " << *p_number << std::endl;      // что выведет?
    return 0;
}
```

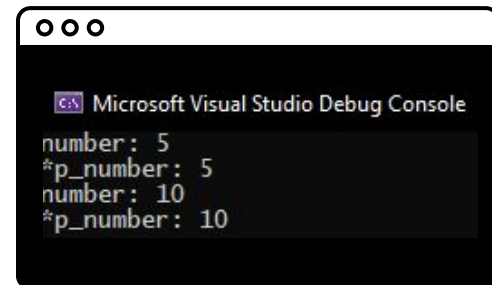
?

Напишите ответ в чат

# Разыменование: пример

Изменим содержимое ячейки памяти с помощью указателя:

```
int main(int argc, char** argv)
{
    int number = 5;
    int * p_number = &number;
    std::cout << "number: " << number << std::endl;
    std::cout << "*p_number: " << *p_number << std::endl;
    *p_number = 10; // изменяем значение
    std::cout << "number: " << number << std::endl;
    std::cout << "*p_number: " << *p_number << std::endl;
    return 0;
}
```



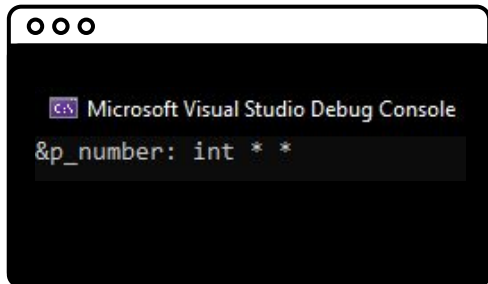
Мы изменили значение переменной без её участия. Такие возможности даёт нам указатель

# Указатели на указатели

Что будет, если мы попробуем получить адрес указателя? Ведь указатель — это всего лишь переменная, которая внутри себя хранит адрес другой ячейки памяти.

Посмотрим на тип значения адреса указателя:

```
int main(int argc, char** argv)
{
    int number = 5;
    int * p_number = &number;
    std::cout << "&p_number: " << typeid(&p_number).name() << std::endl;
    return 0;
}
```





# Указатели на указатели

Тип адреса указателя на **int** (переменной типа **int\***) — это **int \*\***.

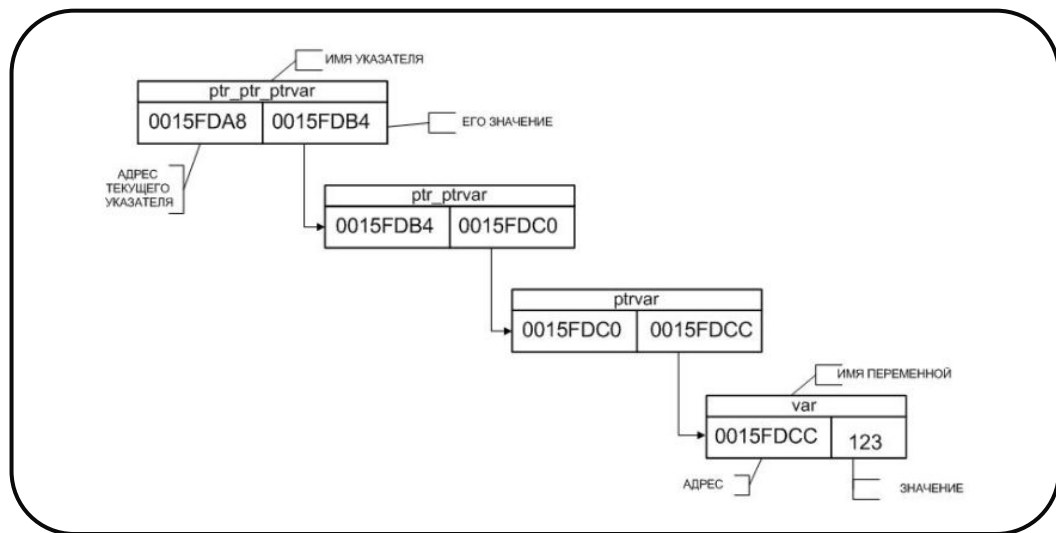
**int \*\*** — это **указатель на указатель**.

То есть переменная типа **int \*\*** содержит адрес ячейки, в которой хранит своё значение переменная типа **int \***. В свою очередь, это значение — адрес ячейки памяти, в которой хранит своё значение переменная типа **int**, и которое уже является каким-то целым числом

# Указатели на указатели

Значение типа **int \*\*** тоже хранится в какой-то ячейке памяти, у которой есть адрес. Если мы получим этот адрес, то сможем сохранить его в переменной, у которой будет тип **int \*\*\***. Это может продолжаться до бесконечности.

Так выглядит указатель третьего порядка (**int \*\*\***):



# Указатели на указатели

Операция разыменования, применённая к указателю типа `int ***`, возвращает значение типа `int **`.

То есть операция разыменования возвращает **значение, которое содержится в ячейке с адресом, хранящимся в указателе, который мы разыменовываем**

# Напишем программу с указателем третьего порядка

Готовый [пример кода](#)



# Массивы и параметры функций



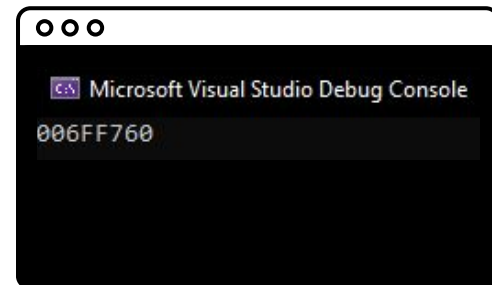
3

# Массив

Мы уже знакомы с тем, как создавать массивы. И мы знаем, как получать доступ к элементам массива — с помощью операции индексации [ ] (квадратные скобки). С помощью этой операции мы можем выводить на консоль элементы массива.

**Вопрос:** что будет, если мы попытаемся вывести на консоль **саму переменную массива**? Давайте проверим:

```
int main(int argc, char** argv)
{
    int arr[] = {5, 3, 4};
    std::cout << arr << std::endl;
    return 0;
}
```

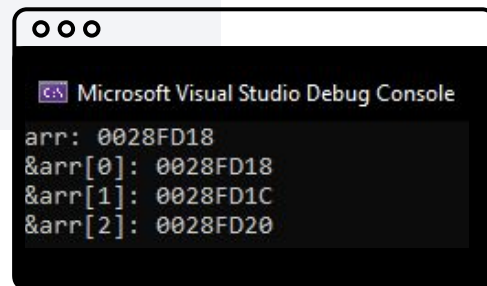


# Массив

Это очень похоже на **адрес ячейки**.

Не так много вариантов того, адрес какой ячейки хранится в переменной `arr`. Попробуем узнать адреса элементов массива:

```
int main(int argc, char** argv)
{
    int arr[] = {5, 3, 4};
    std::cout << "arr: " << arr << std::endl;
    for(int i = 0; i < 3; i++)
    {
        std::cout << "&arr[" << i << "]: " << &arr[i] << std::endl;
    }
    return 0;
}
```



# Массив

Получается, что **в переменной массива хранится адрес первого элемента массива**. То есть массив — это указатель.

Он необычный, потому что в локальном массиве (а мы работали пока только с такими) хранится информация не только о **типе** элементов массива, но и об их **количестве**. Это можно узнать с помощью оператора **sizeof** и несложного деления



# Массив

Тот факт, что массив — это указатель на его первый элемент, позволяет нам передавать массив в функцию и **изменять его** внутри функции.

Бывают ещё **динамические** массивы, которые работают иначе (и скоро мы с ними познакомимся)

# Передача массива в функцию

Теперь мы готовы узнать, как передать массив в функцию, чтобы потом с ним что-то сделать.

Есть три способа, которыми можно передать автоматический (локальный) массив в качестве аргумента функции:

с указанием размера

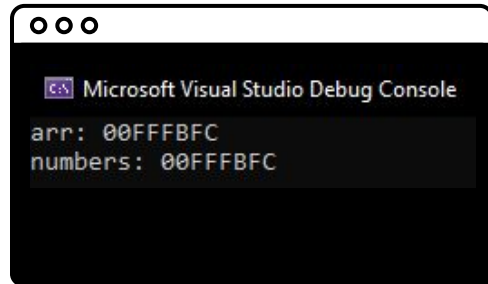
как безразмерный массив

как указатель

# Передача с указанием размера

Чтобы передать массив в функцию с указанием его размера, нужно в качестве типа аргумента функции указать **тип** элементов массива и его **размер в квадратных скобках**. Синтаксис такой же, как при объявлении, но не инициализации, массива:  
**<тип элементов массива> <имя>[<размер массива>]**

```
void fun(int numbers[3])
{
    std::cout << "numbers: " << numbers << std::endl;
}
int main(int argc, char** argv)
{
    int arr[] = {5, 3, 4};
    fun(arr);
    std::cout << "arr: " << arr << std::endl;
    return 0;
}
```

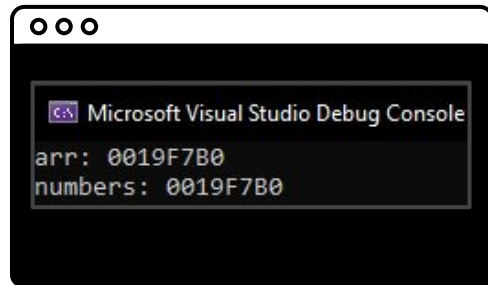


# Передача с указанием размера

Несмотря на то, что вы указываете размер массива, который ожидаете в функции, попытка передать туда массив другого размера увенчается успехом — то есть C++ не гарантирует, что вам в функцию придёт массив именно указанного размера

```
void fun(int numbers[3])
{
    std::cout << "numbers: " << numbers << std::endl;
}

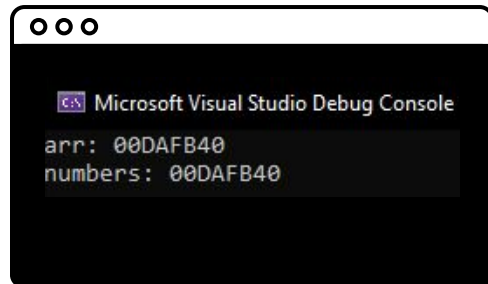
int main(int argc, char** argv)
{
    int arr[] = {5, 3, 4, 6};
    fun(arr);
    std::cout << "arr: " << arr << std::endl;
    return 0;
}
```



# Передача безразмерного массива

Безразмерная передача массива выглядит так же, как передача с размером, но квадратные скобки остаются пустыми: **<тип элементов массива> <имя>[ ]**

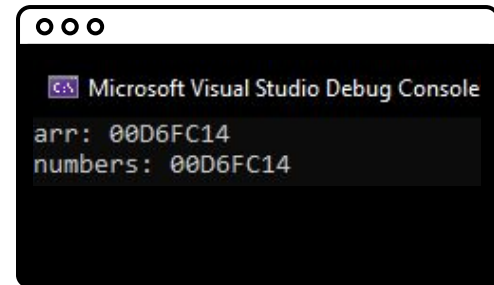
```
void fun(int numbers[])
{
    std::cout << "numbers: " << numbers << std::endl;
}
int main(int argc, char** argv)
{
    int arr[] = {5, 3, 4};
    fun(arr);
    std::cout << "arr: " << arr << std::endl;
    return 0;
}
```



# Передача массива с помощью указателя

Передача массива в виде указателя немного отличается по виду, но по сути остаётся точно такой же: **<тип элементов массива> \* <имя>**

```
void fun(int* numbers)
{
    std::cout << "numbers: " << numbers << std::endl;
}
int main(int argc, char** argv)
{
    int arr[] = {5, 3, 4};
    fun(arr);
    std::cout << "arr: " << arr << std::endl;
    // чему равно выражение &arr[1]?
    return 0;
}
```

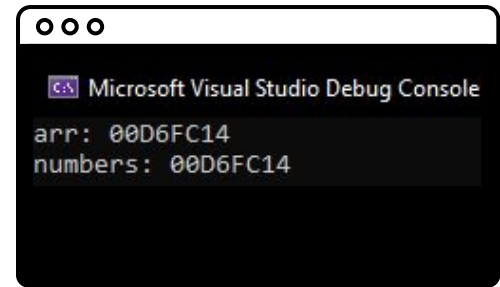


Напишите в чат

# Передача массива с помощью указателя

Передача массива в виде указателя немного отличается по виду, но по сути остаётся точно такой же: **<тип элементов массива> \* <имя>**

```
void fun(int* numbers)
{
    std::cout << "numbers: " << numbers << std::endl;
}
int main(int argc, char** argv)
{
    int arr[] = {5, 3, 4};
    fun(arr);
    std::cout << "arr: " << arr << std::endl;
    // &arr[1] == 00D6FC18
    return 0;
}
```



# Передача массива в функцию

Все три способа передачи похожи, однако наиболее предпочтительный и распространённый — третий, т. е. передача с помощью указателя.

Все три способа объединяет то, что в каждом из них для полноценной работы с массивом **недостаточно** передать только массив — вместе с ним нужно передавать его размер, чтобы функция случайно не вышла за границы переданного ей массива, когда будет с ним работать



# Передача массива в функцию: пример

```
void print_numbers(int* numbers, int size)
{
    for (int i = 0; i < size; i++)
    {
        std::cout << numbers[i] << " ";
    }
    std::cout << std::endl;
}

int main(int argc, char** argv)
{
    int arr1[] = {5, 3, 4};
    int arr2[] = {1, 2, 3, 4, 5, 6};
    print_numbers(arr1, 3);
    print_numbers(arr2, 6);
    return 0;
}
```



# Итоги



# Итоги

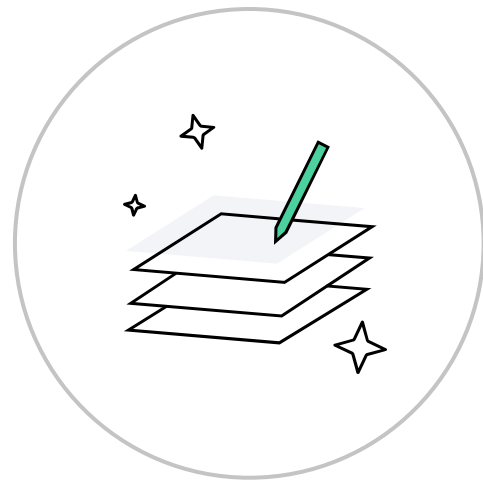
- 1 Разобрались с оператором typeid
- 2 Познакомились с указателями
- 3 Узнали, что такое указатели на указатели
- 4 Выяснили, как передавать массивы в функции



# 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

- [Статья](#) Что такое указатели
- [Статья](#) Передача массивов в функции



# Задавайте вопросы и пишите отзыв о лекции

Владислав Хорев  
Ведущий программист, Luxoft

