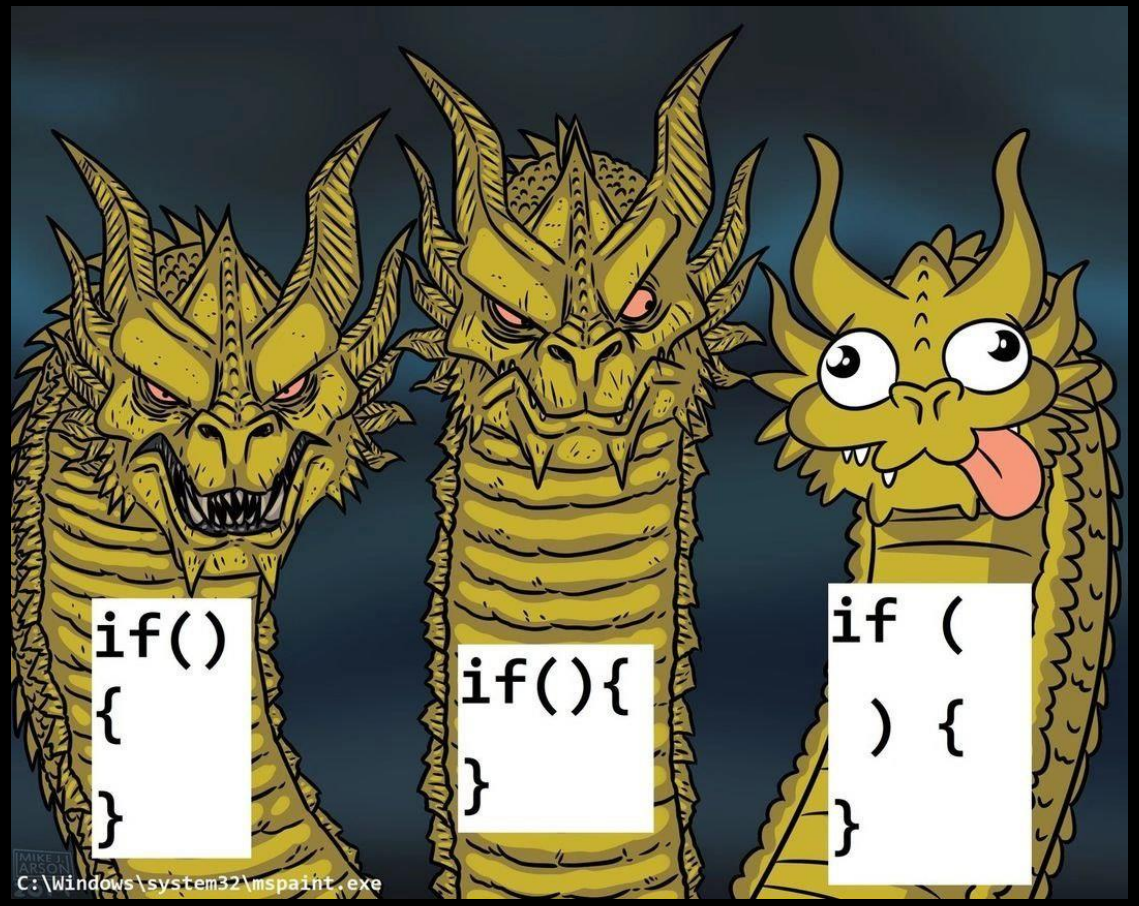
ЛЕКЦИЯ 7

- 27.09.2023
- Массивы в C#

ЦЕЛИ ЛЕКЦИИ

- Обобщить представления о структуре данных - массив
- Познакомится с реализацией одномерных массивов языка C#
- Поговорить о передаче массивов в методы

РЕЗУЛЬТАТЫ САМОСТОЯТЕЛЬНОЙ



- Ищем в SmartLMS, по датам, где писали работу
- Вопросы можно задать преподавателю, ведущему семинары

Все молодцы, видно, что старались

ВСПОМИНАЕМ

Что мы там напроходили…

ПАРАМЕТРЫ МЕТОДОВ

```
static void SetXY(ref int x, ref int y)
{
            x = ++x * 2 - 1;
            y = y-- * 2;
}
static void Main(string[] args)
{
            int x = 6;
            int y = 3;
            SetXY(ref x, ref y);
            Console.WriteLine(x + y);
}
```

Что выведет на экран фрагмент кода?

ВОПРОС

Верно, что выходной параметр (с модификатором out) метода:
(Выберите все верные ответы)

1. может быть только один
2. должен получить значение в теле метода
3. передается в метод по ссылке
4. должен быть инициализирован до передачи в метод
5. автоматически инициализируется нулём при передаче в метод

ВОПРОС

Верно, что параметр, передаваемый в метод по ссылке (с модификатором ref)
(Выберите все верные ответы)

1. может быть не инициализирован до передачи в метод
2. должен быть инициализирован перед передачей в метод
3. должен быть последним в списке параметров метода
4. должен получить значение в теле метода
5. передается в метод по ссылке

ВОПРОС

В результате выполнения следующего фрагмента кода:

```
int i = 20;
for ( ; ; )  {
   Console.Write(i);
   if (i >= -8) i -= 4;
   else break;
}
```

на экран будет выведено:

МАССИВЫ

Определения
Операция индексации
Типы массивов в C#

МАССИВЫ

- **Массив** [array] – последовательность однотипных элементов, к каждому из которых можно получить доступ по его индексу за константное время (примерно одинаковое время, не зависящее от значения индекса)
- **Индекс** [index] – порядковый номер элемента. Индекс сам по себе не является информационным элементом!

ЧТО НУЖНО ЗНАТЬ О МАССИВАХ?

- Массив – базовая структура данных с индексацией
- Массив представляет собой непрерывный участок памяти
- Массив идентифицируется адресом первого элемента

ОПЕРАЦИЯ ИНДЕКСАЦИИ

[ ] Операция для доступа к элементам массивов, индексаторов и указателей

Индексация массивов C# начинается с 0

ССылкаНаМассив[ ]

Попытка обратиться по индексу, выходящему за границы массива приводит к возникновению `IndexOutOfRangeException`

```
// Описание ссылки на массив.
int[] ar = new int[2]; // Инициализация массива.

ar[0] = 15; // Присваивание значения.
ar[1] = 22;

Console.Write(ar[1]); // Получение значения по индексу.
```

МАССИВЫ C#

массив массивов

многомерный

одномерный

- Тип массива ссылочный
- Наследники Array
- Массивы реализуют IList и IEnumerable
- Одномерный массив реализует дополнительно IList<> и IEnumerable<>
- Для итераций допустимо использовать foreach

ЭТАПЫ РАБОТЫ С МАССИВОМ

Объявление ссылки на массив

Создание экземпляра массива

Назначение значений элементам массива

Работа с элементами массива

ОДНОМЕРНЫЕ МАССИВЫ

Общие сведения

Создание

Размещение в памяти

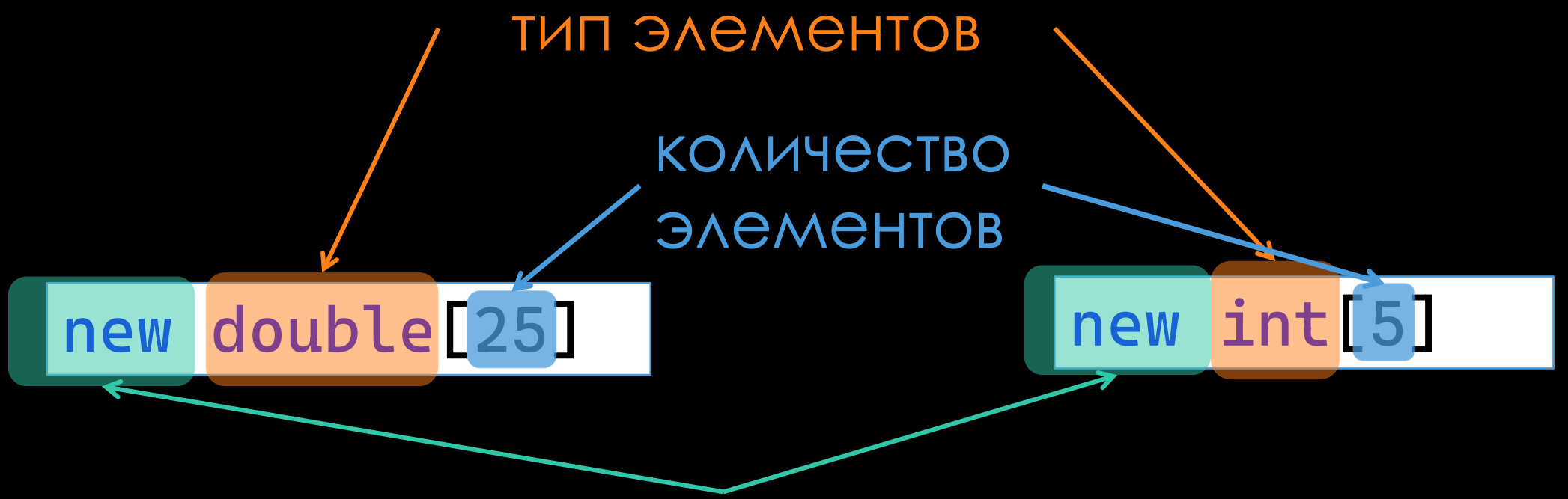
МАССИВЫ

Одномерный массив – набор однотипных элементов, доступ к которым осуществляется с помощью выражения с **операцией индексирования**

Элементы массива [Elements, items] – отдельные единицы данных массива

Все элементы массива должны иметь один тип или тип, приводимый к типу элементов массива

СОЗДАНИЕ ОДНОМЕРНОГО МАССИВА



ПРИМЕР. СОЗДАНИЕ МАССИВА

Описание ссылки **ar** на одномерный массив элементов **int**

Создание одномерного массива из пяти элементов типа **int**

Это назначение ссылки!

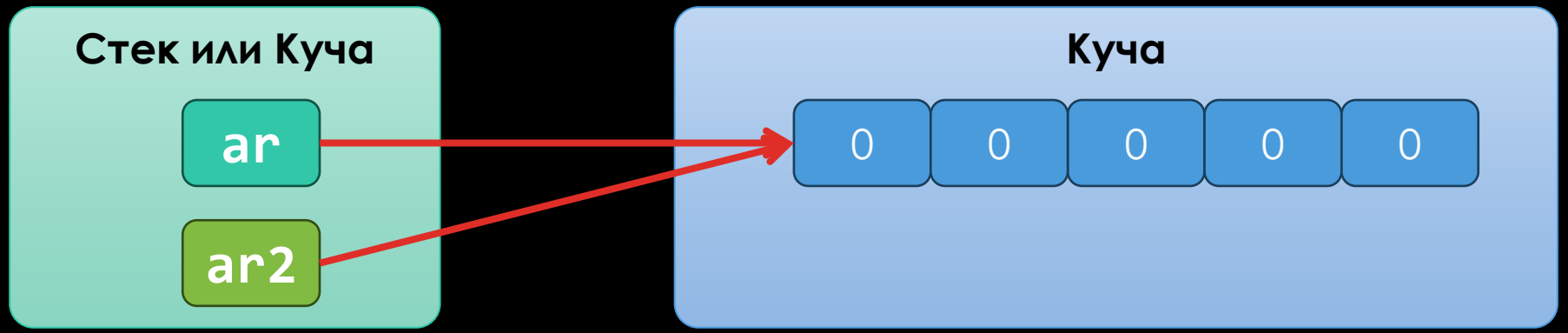
```
int[] ar = new int[5];

for (int i = 0; i < ar.Length; i++)
{
    Console.Write($"{ar[i]} ");
}
```

Элементы уже проинициализированы нулём типа **int**

Обращение к элементу массива через ссылку **ar** по индексу **i**

РАЗМЕЩЕНИЕ МАССИВА В ПАМЯТИ

```
int[] ar = new int[5];
int[] ar2 = ar;
for (int i = 0; i < ar.Length; i++)
{
    Console.WriteLine($"ar[{i}] = {ar[i]}:: ar[{i}] = {ar2[i]}");
}
```

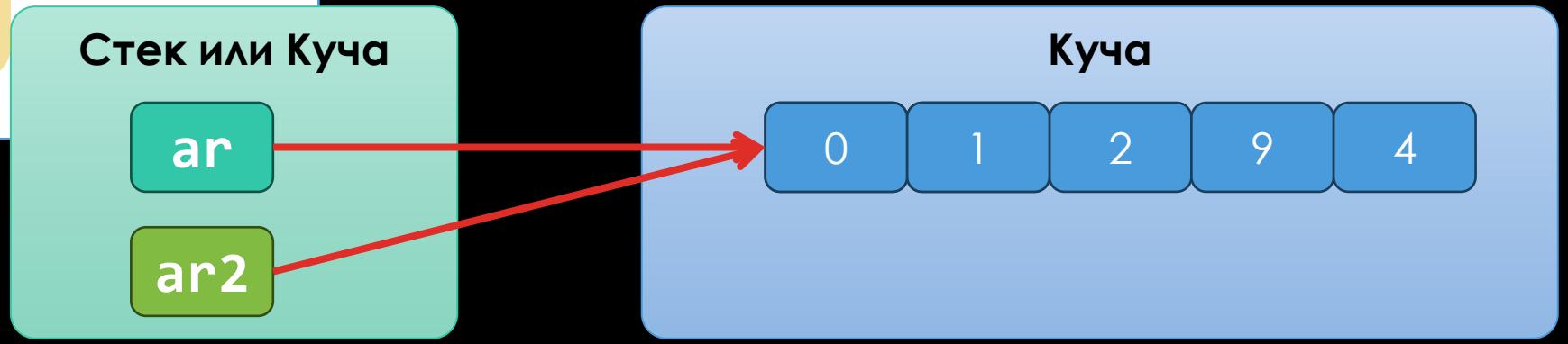


РАЗМЕЩЕНИЕ МАССИВА В ПАМЯТИ

```
int[] ar = new int[5];
int[] ar2 = ar;
for (int i = 0; i < ar.Length; i++)
{
    if (i % 2 == 0)
    {
        ar[i] = i;
    }
    else
    {
        ar2[i] = i * i;
    }
}
```

Изменение значений элементов массива

имя массива – ошибочный термин: имена даются ссылкам на массивы, а не самим массивам



ИНИЦИАЛИЗАЦИЯ ЭЛЕМЕНТОВ МАССИВА

ИНИЦИАЛИЗАЦИЯ

Описание с созданием

```
int[] ar = new int[5];
```

Описание с инициализацией списком (с new и выводом типа)

```
var arr5 = new[] { 8, 9, 10 };
```

Описание с инициализацией списком (без new)

```
string[] test = { "t1", "t2" };
```

```
int[] ar2 = { -5, 0, 4, 3, 2 };
```

```
int[] arr6 = { (int)Math.Round(3.14), 20 + 5 };
```

Описание с инициализацией списком (с new)

```
int[] ar2 = new int[] { -5, 0, 4, 3, 2 };
```

```
int[] ar3 = new int [3] {1, 2, 3};
```

```
int[] ar3 = new []{1, 2, 3};
```

НЕКОТОРЫЕ ОШИБКИ ИНИЦИАЛИЗАЦИИ

```
int[] ar2;
ar2 = new int[] { -5, 0, 4, 3, 2 };
```

```
int[] ar2;
ar2 = { -5, 0, 4, 3, 2 };
```

```
int[] ar3 = new[] { 1, 2, 3};
```

```
int[] ar3 = new int [3] { 1, 2 };
```

```
int[] ar3 = { };
```

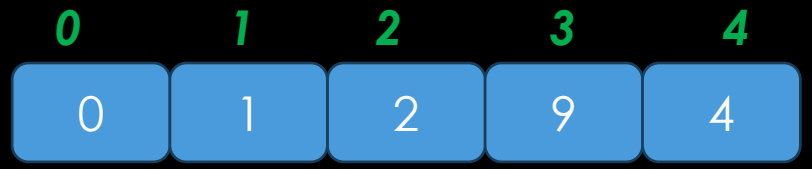
```
int[] ar3 = new[] { };
```

```
int[] ar3 = new int[]{ };
```

```
var ar3 = new { 1,2 };
```

РАЗМЕРЫ МАССИВА

- **Размерность** (Rank) – число измерений массива
  - количество индексов для обращения к элементу массива (положительное число)
- **Длина измерения** (Dimension Length) – число элементов в данном измерении массива
  - Каждое измерение массива имеет длину
- **Размер массива** (Array Length) – общее количество элементов, содержащееся во всех измерениях массива




```
Rank = 1

GetLength(0) = 5

Length = 5
```

ЧЛЕНЫ КЛАССА ARRAY

Член	Тип Члена	static	Краткое описание
Length	свойство	нет	Количество элементов во всех измерениях массива.
Rank	свойство	нет	Количество измерений массива.
GetLength	метод	нет	Длина указанного измерения массива.
Clear	метод	да	Заменяет диапазон значений массива значениями типа элемента по умолчанию.
Sort	метод	да	Выполняет сортировку одномерного массива.
BinarySearch	метод	да	Выполняет поиск значения в одномерном массиве.
Clone	метод	нет	Выполняет поверхностное копирование массива – копирует только лежащие в массиве значения/ссылки.

МЕТОДЫ КЛАССА ARRAY

Член	static	Краткое описание
Reverse	да	Разворачивает диапазон элементов массива.
Resize	да	Пересоздаёт массив большего/меньшего размера, содержащий элементы исходного.
IndexOf	да	Возвращает индекс первого вхождения элемента в массиве или -1 при его отсутствии.
LastIndexOf	да	Возвращает индекс последнего вхождения элемента в массиве или -1 при его отсутствии.
GetLowerBound	нет	Возвращает индекс первого элемента данного измерения массива.
GetUpperBound	нет	Возвращает индекс последнего элемента данного измерения массива.
CreateInstance	да	По указанному типу создаёт массив с заданной нижней границей и размерами.

ПЕРЕБОР ЭЛЕМЕНТОВ МАССИВА

Создаём массив

```
int[] ar = new int[5];
```

Свойство **Length** возвращает **Int32** общее количество элементов во всех измерениях массива

```
for (int i = 0; i < ar.Length; i++)
{
    // Обращаемся по индексу к элементам ar[i].
}
```

Метод **GetLength()** возвращает **Int32** общее количество элементов в указанном в параметре измерении массива

```
for (int i = 0; i < ar.GetLength(0); i++)
{
    // Обращаемся по индексу к элементам ar[i].
}
```

Метод **GetUpperBound()**возвращает **Int32** индекс последнего элемента заданного в параметре измерения

```
for (int i = 0; i <= ar.GetUpperBound(0); i++)
{
    // Обращаемся по индексу к элементам ar[i].
}
```

ВЫВОД НА ЭКРАН

Создаём массив

```
int[] ar = new int[5];
```

Блок операторов в теле **foreach** выполнится для всех элементов коллекции

```
foreach(int i in ar)
{
    Console.Write(i);
}
```

Ссылка на массив

Элемент массива доступен по значению, то есть скопирован в **i**

Выполняет действие **action** с каждым элементом массива **array**

```
Array.ForEach(array: ar, action: Console.Write);
```

Анонимный тип для **cur**

```
foreach (var cur in ar)
{
    Console.Write(cur);
}
```

МАССИВ КАК ССЫЛОЧНЫЙ ТИП ДАННЫХ

Общая идея

Проблемы и ошибки

ССЫЛОЧНЫЕ ТИПЫ

- Переменные ссылочных типов содержат ссылки на данные
- Две переменные ссылочного типа могут ссылаться на одни и те же данные

`class` `interface` `delegate` `record`

Встроенные ссылочные типы: `dynamic` `object` `string`

`null` - значение по умолчанию для переменных ссылочного типа

ПРОВЕРЯЕМ СЕБЯ

Что выведет на экран этот код?

```
int[] A = { 1, 2, 3, 4 };
int[] B;
B = A; // присваивание ссылки
foreach (int a in A)
    Console.Write(a + " ");
B[1] = 13;
Console.WriteLine();
foreach (int a in A)
    Console.Write(a + " ");
```

```
int[] ar = new int[] { 1, 2, 3 };
Console.Write(ar);
```

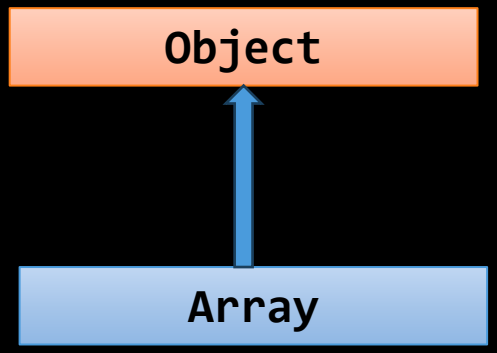
```
int[] ar = { 1, 2, 3, 4 };
double[] ar2;
ar2 = ar;
Console.Write(ar2);
```

```
int[] ar;
if (ar is null)
    Console.Write("1");
Console.Write("2");
```

```
int[] ar = new int[10];
if (ar is null)
    Console.Write("1");
else
    Console.Write("2");
```

ОШИБКИ И ИСКЛЮЧЕНИЯ

- Массивы – ссылочные типы
  - перед обращением по индексу возникает необходимость проверки на `null` (или использования операции `?[]`)
  - чтобы получить эту ошибку нужно, чтобы ссылка была проинициализирована хотя бы значением `null`




```
int[] ar = new int[5];
int[] ar2; // Эта ссылка не проиниализирована.
Console.Write(ar is null ? null : ar); // Можно.
Console.Write(ar2 is null ? null : ar2); // Нельзя.
```

ОПЕРАЦИИ ОБЪЕДИНЕНИЯ С NULL

Операция	Результат выполнения
x ?? y	Возвращает значение **x**, если он не **null** или вычисляет значение **y** и возвращает его
x ??= y	Вычисляет значение **y** и присваивает его **x** только в случае, когда **x** оказался равен **null**

- Операции объединения с **null** предоставляют лаконичный синтаксис для выполнения вычислений в случае, когда левый операнд оказывается равен **null**
- **С версии C# 8.0 левый операнд не может быть типом значения, не допускающим значения null**
- Операция может быть крайне удобна в случаях, когда нужна отложенная инициализация

ПРИМЕР

```
int[] ar2 = null;
int[] tmp = ar2 ?? new int[15];
```

Ссылке будет назначено значение, т.к. она **null**

Ссылка уже связана с массивом (не **null**), ей **не будет** назначено значение

```
int[] ar = { 1, 2, 3 };
int[] ar2 = null;
int[] ar3 = { 5, 6, 7 };

Console.WriteLine($"ar::");
Array.ForEach(ar, Console.Write);
Console.WriteLine($"{Environment.NewLine}ar3::");
Array.ForEach(ar3, Console.Write);

ar2 ??= ar;
ar3 ??= ar;

Console.WriteLine($"{Environment.NewLine}ar2::");
Array.ForEach(ar2, Console.Write);
Console.WriteLine($"{Environment.NewLine}ar3::");
Array.ForEach(ar3, Console.Write);
```

МАССИВЫ И МЕТОДЫ

Передача массивов в методы

Возврат массива из метода

ВОЗВРАТ МАССИВА ИЗ МЕТОДА

```
public static class Methods
{
    static Random rnd = new Random();

    public static int[] GetIntArray(int n)
    {
        int[] array = n < 0 ? null : new int[n];
        for(int i = 0;i < array?.Length; i++)
        {
            array[i] = rnd.Next();
        }
        return array;
    }
}
```

1. Возвращаем ссылку на массив целых, для некорректных данных – назначаем ссылке значение null
2. L-value – ссылка на массив целых

```
int n = -15;
int[] ar = Methods.GetIntArray(n);
Console.Write(ar is null);
```

ПЕРЕДАЧА МАССИВОВ В МЕТОДЫ: ССЫЛКА ПО ЗНАЧЕНИЮ

```
public static class Methods
{
    public static double Mean(double[] ar)
    {
        double mean = 0;
        foreach(double d in ar) {  mean += d; }
        return mean / ar.Length;
    }
}
```

```
double[] ar = { 1.9, 0.75, 2.55, 15, -8 };

Console.WriteLine(Methods.Mean(ar));
```

При вызове, ссылка будет скопирована, а массив нет

Какие проблемы есть в описании этого метода?

Внесите модификации сами, в качестве разминки дома

ПРИМЕР

```
static bool TryFindMaxElement(int[] array, out int maxValue)
{
    // Для пустых массивов и null максимума нет.
    if (array == null || array.Length == 0) {
        // Значение типа int по умолчанию, т. е. 0.
        maxValue = default;
        return false;
    }
    maxValue = array[0];
    for (int i = 1; i < array.Length; i++) {
        // Сравниваем элемент при обращении по индексу.
        if (array[i] > maxValue) {
            maxValue = array[i];
        }
    }
    return true;
}
```

Как можно модифицировать этот код?

ПЕРЕДАЧА МАССИВОВ В МЕТОДЫ: ССЫЛКА ПО ССЫЛКЕ

```
public static class Methods
{
    static Random rnd = new Random();

    public static void GetIntArray(int n, int[] ar = null)
    {
        if (n > 0)
        {
            ar = new int[n];
            for (int i = 0; i < ar?.Length; i++)
            {
                ar[i] = rnd.Next();
            }
        }
    }
}
```

ref

3 Модифицируем, ссылку передадим по ссылке - **ref**, значение по умолчанию уберём

1. Произошло назначение ссылки, т.е. установлена связь с новым массивом
2. Ссылка в вызывающем коде связана с прежним массивом, в метод она передана по значению

```
int[] ar = { 1, 2 };
Methods.GetIntArray(5, ar);
for (int i = 0;i < ar.Length;i++)
{
    Console.Write(ar[i]);
}
```

ref

МОДИФИКАТОР PARAMS

params – модификатор параметров метода, позволяющий передать переменное число аргументов через запятую или в виде одномерного массива

```
static int Sum(params int[] elems) {
    int sum = 0;
    foreach (int elem in elems) {
        sum += elem;
    }
    return sum;
}
```

```
int[] arr = { 1, 2, 3, 4, 5 };
int sum3 = Sum(3, 4, 5);  // Массив формируется неявно.
int sum5 = Sum(arr);      // Массив явно передаётся в метод.
Console.WriteLine($"sum3 = {sum3}; sum5 = {sum5}");
```

ПРИМЕР

```
double one = 0.1, two = 0.01, three = 0.001;
Console.WriteLine($"\nInverse sum = {Inverse(one, two, three)}");
Console.WriteLine($"one = {one}; two = {two}; three = {three}");
```

```
static double Inverse(params double[] elems) {
    double res = 0;
    for (int k = 0; k < elems.Length; k++) {
        res += (elems[k] = 1 / elems[k]);
        Console.Write($"elems[{k}] = {elems[k]} ");
    }
    return res;
}
```

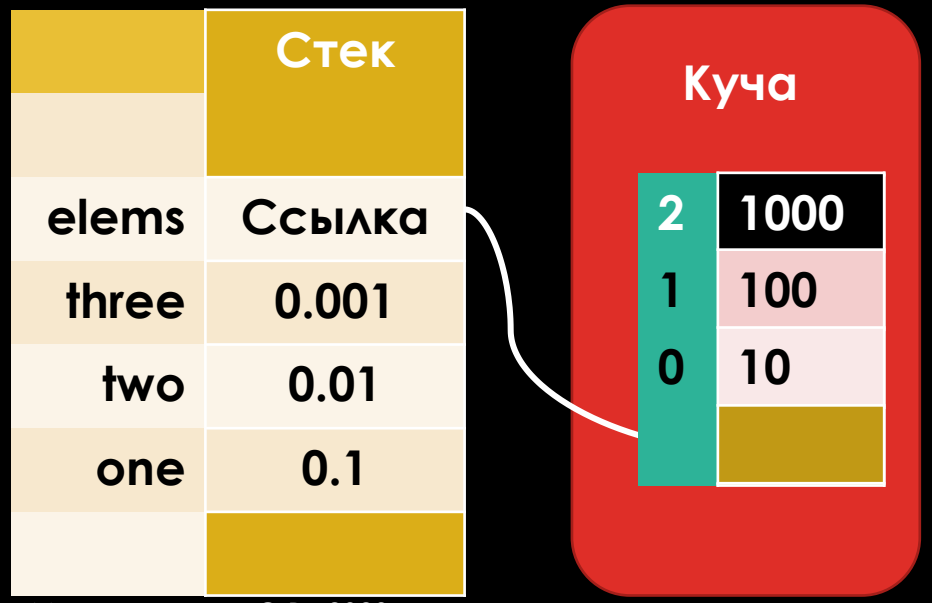
Результат выполнения:

elems[0] = 10 elems[1] = 100 elems[2] = 1000
Inversed sum = 1110
one = 0,1; two = 0,01; three = 0,001

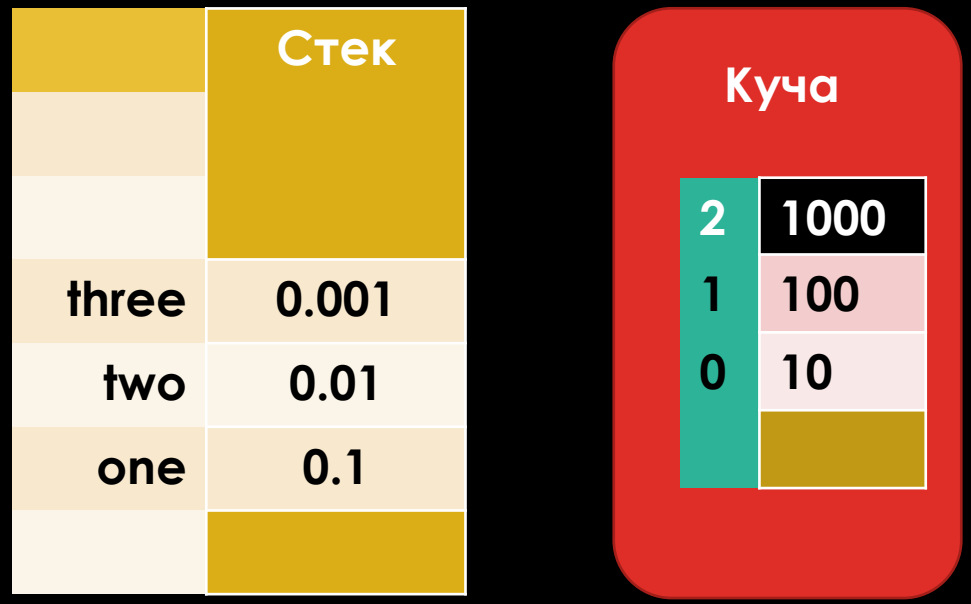
```
static double Inverse(params double[] elems) {
    double res = 0;
    for (int k = 0; k < elems.Length; k++) {
        res += (elems[k] = 1 / elems[k]);
        Console.Write($"elems[{k}] = {elems[k]} ");
    }
    return res;
}
```

```
double one = 0.1, two = 0.01, three = 0.001;
Console.WriteLine($"\nInverse sum = {Inverse(one, two, three)}");
Console.WriteLine($"one = {one}; two = {two}; three = {three}");
```

Вызов (начало выполнения)



Возврат (после завершения метода)



ПРИМЕР. ДРУГИЕ ПАРАМЕТРЫ

```
double[] array = { 1, 10, 100 };
Console.WriteLine(Inverse(array));
foreach (double value in array)
{
    Console.WriteLine(value);
}
```

```
static double Inverse(params double[] elems) {
    double res = 0;
    for (int k = 0; k < elems.Length; k++) {
        res += (elems[k] = 1 / elems[k]);
        Console.Write($"elems[{k}] = {elems[k]} ");
    }
    return res;
}
```

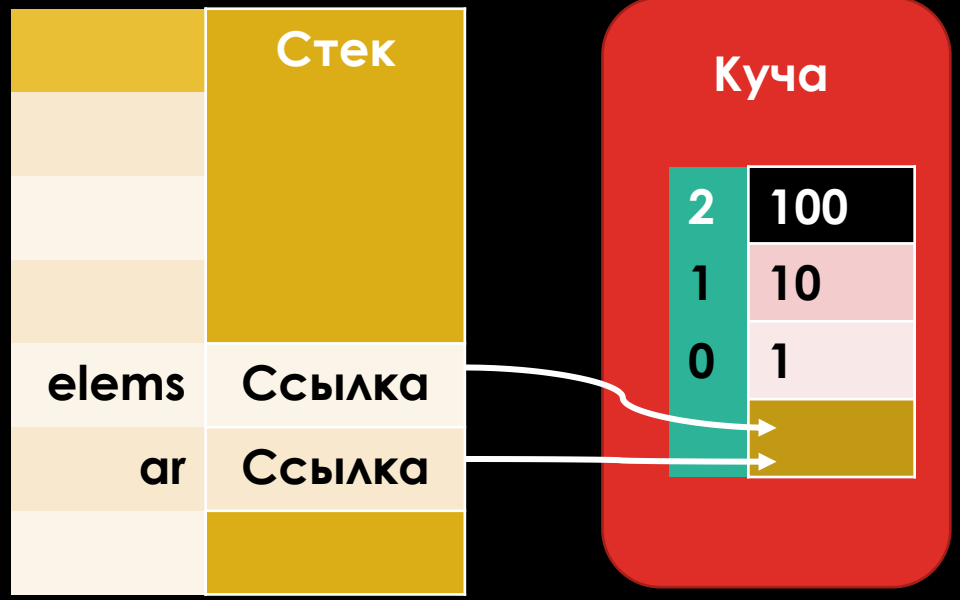
Результат выполнения:

elems[0] = 1 elems[1] = 0,1 elems[2] = 0,01
1,11
1
0,1
0,01

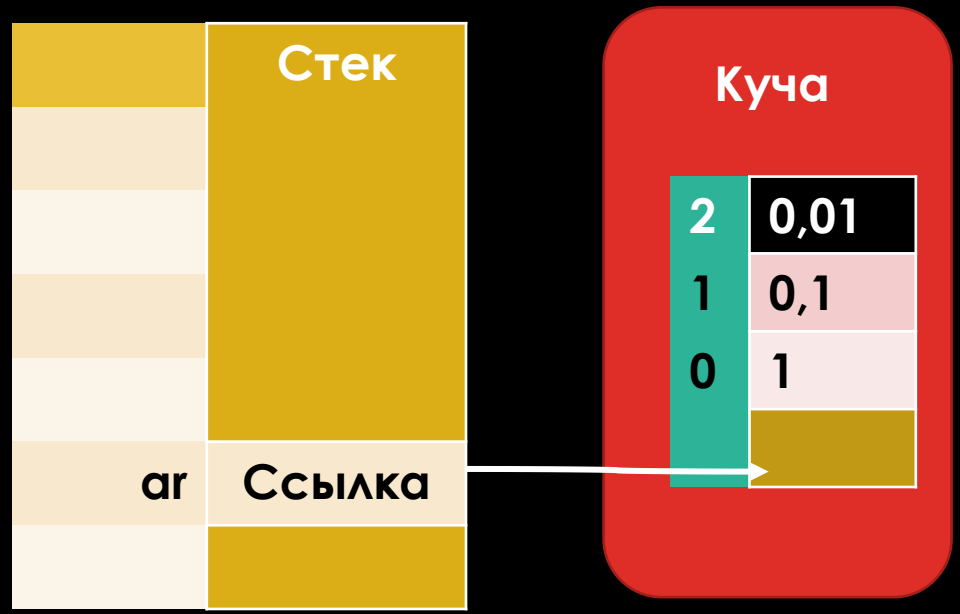
```
double[] array = { 1, 10, 100 };
Console.WriteLine(Inverse(array));
foreach (double value in array)
{
    Console.WriteLine(value);
}
```

```
static double Inverse(params double[] elems) {
    double res = 0;
    for (int k = 0; k < elems.Length; k++) {
        res += (elems[k] = 1 / elems[k]);
        Console.Write($"elems[{k}] = {elems[k]} ");
    }
    return res;
}
```

Вызов (начало выполнения)



Возврат (после завершения метода)



ИСПОЛЬЗОВАННАЯ ЛИТЕРАТУРА

- Richter, J. CLR via C#. Fourth edition
- https://learn.microsoft.com/ru-ru/dotnet/api/system.array.foreach?view=net-6.0
- https://learn.microsoft.com/ru-ru/dotnet/csharp/language-reference/statements/iteration-statements#the-foreach-statement
- https://learn.microsoft.com/ru-ru/dotnet/api/system.array.getupperbound?view=net-6.0
- https://learn.microsoft.com/en-us/dotnet/api/system.array?view=net-6.0