ЛЕКЦИЯ 8

- 27.09.2023
- Индексы и диапазоны, обратная индексация
- Многомерные массивы
- Зубчатые массивы

ЦЕЛИ ЛЕКЦИИ

- Изучить дополнительные возможности индексации массива C#
- Разобраться с многомерными (прямоугольными массивами)
- Припомнить базовые понятия алгебры о матрицах
- Понять идею массива массивов (зубчатых массивов)
- Поговорить о размещении разных типов массивов в памяти

ИНДЕКСЫ И ДИАПАЗОНЫ

ИНДЕКСЫ И ДИАПАЗОНЫ

- **Диапазоны** и **индексы** обеспечивают лаконичный синтаксис для доступа к отдельным элементам или диапазонам в последовательности
  - а двух типах и операторах:
  - Тип System.Index представляет индекс в последовательности
  - Оператор ^ (индекс с конца), который указывает, что индекс указан относительно конца последовательности
  - Тип System.Range представляет вложенный диапазон последовательности.
  - Оператор диапазона .., который задает начало и конец диапазона в качестве своих операндов

ИНДЕКСАЦИЯ С КОНЦА. INDEX

В C# 8.0 была добавлена операция **^x** для **получения индекса с конца**

- индекс с конца **^x** вычисляется как **length – x**
- последний элемент имеет индекс **^1**
- первый – **^length**

```
int[] arr = { 1, 2, 3, 4, 5 };
Console.WriteLine(arr[^1]);   // 5.
Console.WriteLine(arr[^5]);   // 1.
```

Операция ^x возвращает элемент специального типа **System.Index**, который неявно преобразуется к типу int, благодаря чему Index применим везде, где ожидается int

ПРИМЕР

Стартуем с элемента с индексом **arr.Length - 1**, это **^1**

Завершимся на элементе с индексом **0**, это **^arr.Length**

```
int[] arr = { 35, 35, 555, 800, 8 };
// Обход массива в обратную сторону с помощью индексов с конца.
for (int i = 1; i <= arr.Length; i++)
{
    Console.Write(arr[^i] + " ");
}
```

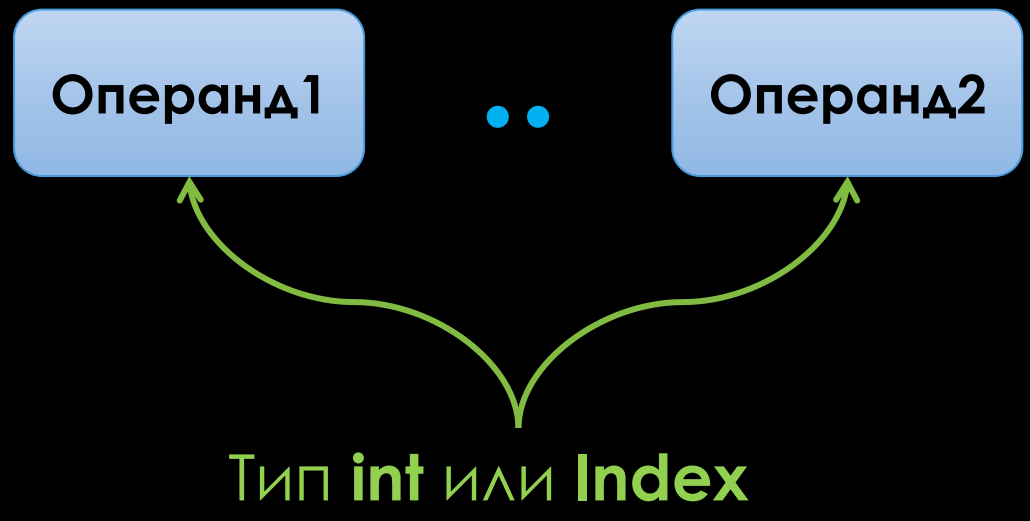
Обращение по индексу, идет преобразование из Index в Int32

Результат выполнения:
8 800 555 35 35

ДИАПАЗОНЫ. RANGE

C# 8.0 - диапазоны (System.Range)

Бинарная операция **диапазона ..**




```
Range rng1 = ..;  // Эквивалент диапазона [0;^0).
Range rng2 = x..; // Эквивалент диапазона [x;^0).
Range rng2 = ..y; // Эквивалент диапазона [0;y).
```

ПРИМЕР

```
using System;

int[] arr = { 0, 5, 10, 15, 20, 25, 30, 35, 40, 45, 50 };
int[] newArr1 = arr[5..11];

arr[^4] = 100000;
foreach (int val in newArr1)
{
    Console.Write(val + " ");
}
```

Изменили значение элемента старого массива

По этой ссылке изменений не будет, извлечённый диапазон - копия

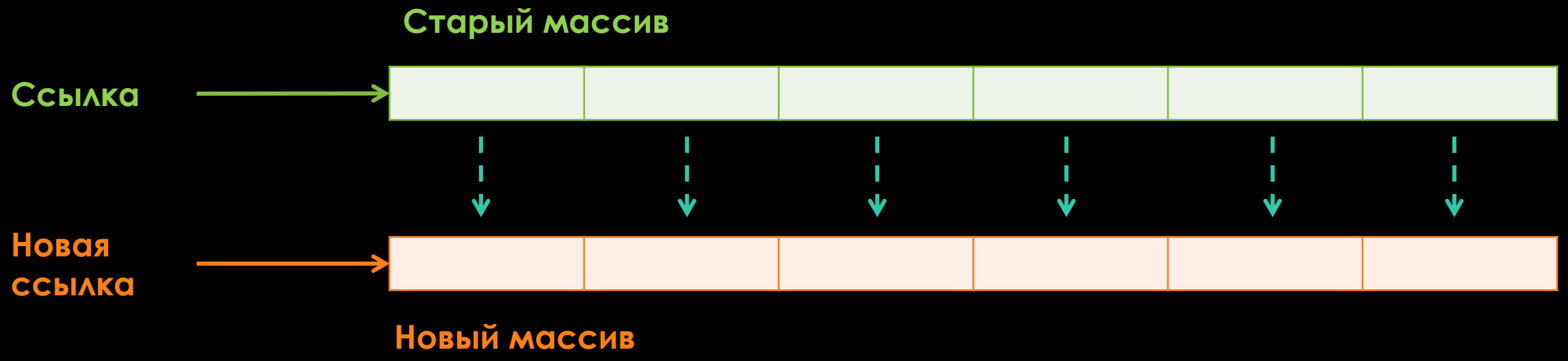
Результат выполнения:
25 30 35 40 45 50

КОПИРОВАНИЕ МАССИВОВ

Глубокое и мелкое (не глубокое) копирование

НЕ ГЛУБОКОЕ КОПИРОВАНИЕ

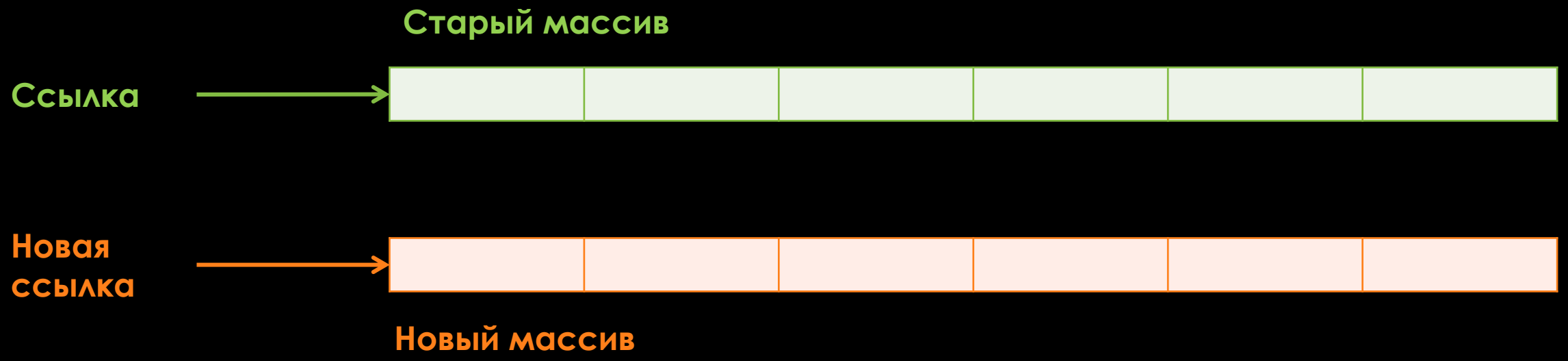
- **Не глубокое / поверхностное копирование** [shallow copy] – это создание нового объекта посредством копирования всех полей с типом значений другого объекта и копирование ссылок данных ссылочных типов, но не адресуемых ими объектов




УЧЕБНЫЙ ПЛАН
ЧТО МЫ ПРОХОДИЛИ
ЧТО Я ЗАКОНСПЕКТИРОВАЛ
ЧТО Я СМОГ ВСПОМНИТЬ НА ЭКЗАМЕНЕ

ГЛУБОКОЕ КОПИРОВАНИЕ

- **Глубокое копирование** [deep copy] – это создание нового объекта посредством копирования всех данных исходного объекта, включая доступные по внешним ссылкам



КОПИРОВАНИЕ МАССИВОВ

Не глубокое копирование

Название метода
Array.Clone()
Array.ConstrainedCopy()
Array.Copy()
Array.CopyTo)

Не глубокое копирование

МЕТОД CLONE(): ПРИМЕР – ТИПЫ ЗНАЧЕНИЙ

```
using System;

int[] intArr1 = { 1, 2, 3 };
// Требуется приведение - Clone возвращает ссылку типа Object.
int[] intArr2 = intArr1.Clone() as int[];
intArr2[0] = 100;
intArr2[1] = 200;
intArr2[2] = 300;
foreach (int item in intArr1) {
    Console.Write(item + " ");
}
Console.WriteLine();
foreach (int item in intArr2) {
    Console.Write(item + " ");
}
```

Операция as пробует выполнить приведение и возвращает результат в случае успеха, иначе - null.

Результат выполнения:
1 2 3
100 200 300

МЕТОД CLONE(): ПРИМЕР – ССЫЛОЧНЫЕ ТИПЫ

```
class A {
    public int Value = 5;
}

A[] AArray1 = { new A(), new A(), new A() };
A[] AArray2 = AArray1.Clone() as A[];
AArray2[0].Value = 100;
AArray2[1].Value = 200;
AArray2[2].Value = 300;
foreach (A item in AArray1) {
    System.Console.Write(item.Value + " ");
}
System.Console.WriteLine();
foreach (A item in AArray2) {
    System.Console.Write(item.Value + " ");
}
```

За счёт поверхностного копирования будут скопированы только ссылки на объекты-массивы, а сами вложенные массивы не будет скопированы (на них будут ссылаться дважды).

Результат выполнения:

100 200 300
100 200 300

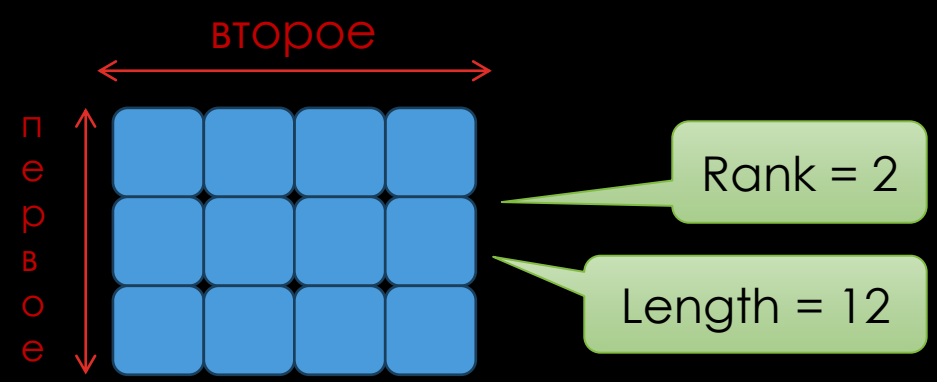
МНОГОМЕРНЫЕ МАССИВЫ

Прямоугольные массивы

ПРЯМОУГОЛЬНЫЕ МАССИВЫ

Прямоугольные (многомерные) массивы – массивы, которые имеют более одного измерения

Размерность (Rank) прямоугольного массива равна количеству измерений, а **длина** (Length) – количеству элементов массива во всех измерениях



ССЫЛКИ НА МНОГОМЕРНЫЕ МАССИВЫ

```
//  Ссылка на двумерный массив без инициализации.
int[,] arr0;

//  Ссылка на трёхмерный массив без инициализации.
int[,,] arr1;
```

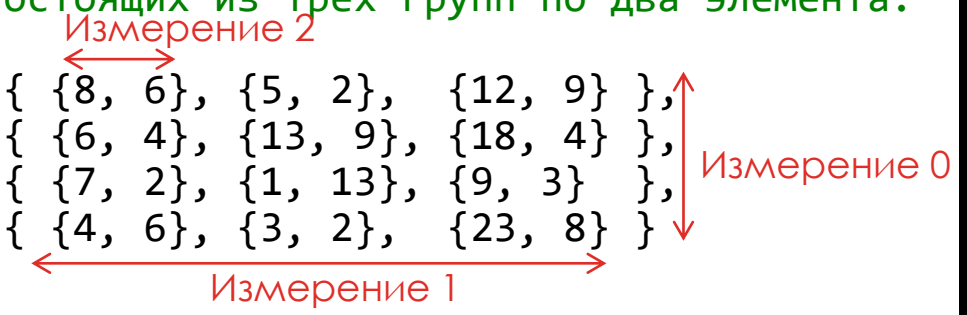
Обращение к элементу многомерного массива требует столько индексов, сколько измерений у массива

`arr0[i, j]`

`arr1[i, j, k]`

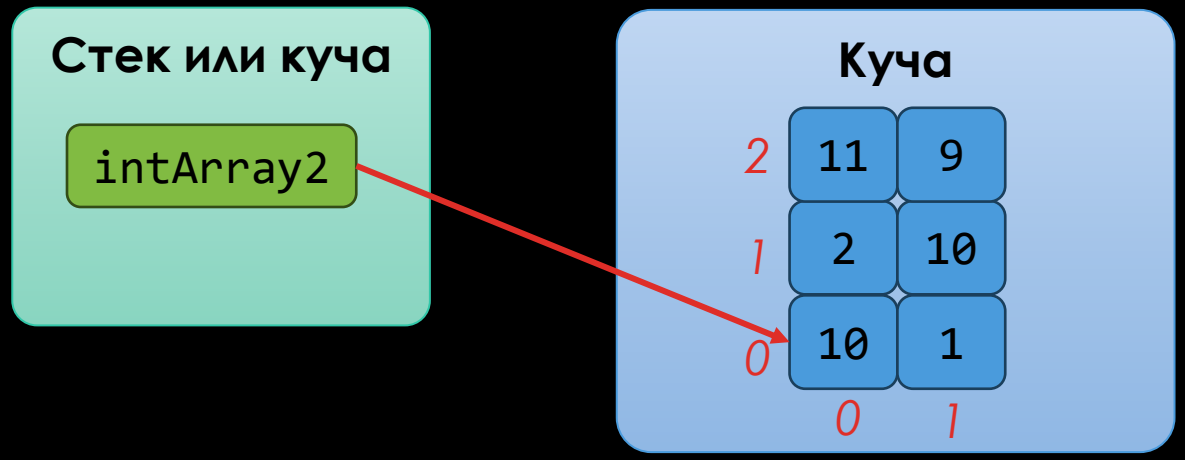
СОЗДАНИЕ МНОГОМЕРНОГО МАССИВА

```
// Ссылка на двумерный массив с инициализацией созданным массивом.
int[,] arr2 = new int[2, 2] { { 2, 3 }, { 2, 3 } };

// Ссылка на трёхмерный массив с инициализацией созданным массивом.
int[,,] arr1 = new int[2, 3, 5];

// Укороченный синтаксис инициализации (список инициализации).
int[,] arr3 = { { 0, 0 }, { 0, 1 }, { 1, 0 }, { 1, 1 } };

// Ссылка на трёхмерный массив с явной инициализацией.
// Все элементы всех измерений заданы явно.
// arr4 состоит из четырёх групп, состоящих из трёх групп по два элемента.
int[,,] arr4 = new int[4, 3, 2] {
                                   { {8, 6}, {5, 2},  {12, 9} },
                                   { {6, 4}, {13, 9}, {18, 4} },
                                   { {7, 2}, {1, 13}, {9, 3}  },
                                   { {4, 6}, {3, 2},  {23, 8} }
                               };
```



ЯВНАЯ ИНИЦИАЛИЗАЦИЯ ПРЯМОУГОЛЬНОГО МАССИВА

```
int[,] intArray2 = new int[,] { { 10, 1 }, { 2, 10 }, { 11, 9 } };
```



ОБРАЩЕНИЕ К ЭЛЕМЕНТАМ МАССИВОВ

intArr1					
0	0	**10**	0	0	0

intArr2				
0	0	0	0	0
0	0	0	0	0
0	0	0	**7**	0
0	0	0	0	0

Одномерные массивы

```
int[] intArr1 = new int[6]; // Объявление одномерного массива.
intArr1[2] = 10;            // Запись элемента с индексом 2.
int var1 = intArr1[2];      // Чтение элемента с индексом 2.
```

Многомерные массивы

```
int[,] intArr2 = new int[4, 5];  // Объявление двумерного м.
intArr2[2, 3] = 7;               // Запись элемента [2, 3].
int var2 = intArr2[2, 3];        // Чтение элемента [2, 3].
```

ПРЯМОУГОЛЬНЫЕ МАССИВЫ: ПРИМЕР

```
// Объявляем и инициализируем двумерный массив.
int[,] arr = new int[4, 4] {
    { 1, 2, 3, 4 },                 {5, 6, 7, 8},
    { 9, 10, 11, 12 },    { 13, 14, 15, 16 }
};

// Проходимся по каждому из элементов в цикле.
for (int i = 0; i < 4; ++i)
{
    for (int j = 0; j < 4; ++j)
    {
        // Выводим все элементы выше главной диагонали.
        if (j > i)
        {
            Console.Write(arr[i, j] + " ");
        }
    }
}
```

1	2	3	4
5	6	7	8
9	10	11	12
13	13	15	16

Результат выполнения:
2 3 4 7 8 12

ВСПОМИНАЕМ МАТЕМАТИКУ

Первый индекс -номер строки

$$\begin{Vmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \dots & \dots & \dots & \dots \\ a_{m1} & a_{m2} & \dots & a_{mn} \end{Vmatrix}$$

Второй индекс- номер столбца

Числа, составляющие матрицу, называются её **элементами** и характеризуются своим положением в таблице.

Пусть дано некоторое числовое поле ***K***. Прямоугольную таблицу чисел из поля ***K*** будем называть **матрицей**.

$$\begin{pmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \dots & \dots & \dots & \dots \\ a_{m1} & a_{m2} & \dots & a_{mn} \end{pmatrix}$$

В общем случае матрица называется **прямоугольной** (с размерами ***mxn***).

Под **числовым полем** понимают любую совокупность чисел, в пределах которой всегда выполнимы и однозначны четыре операции: сложение, вычитание, умножение и деление на число, отличное от нуля.

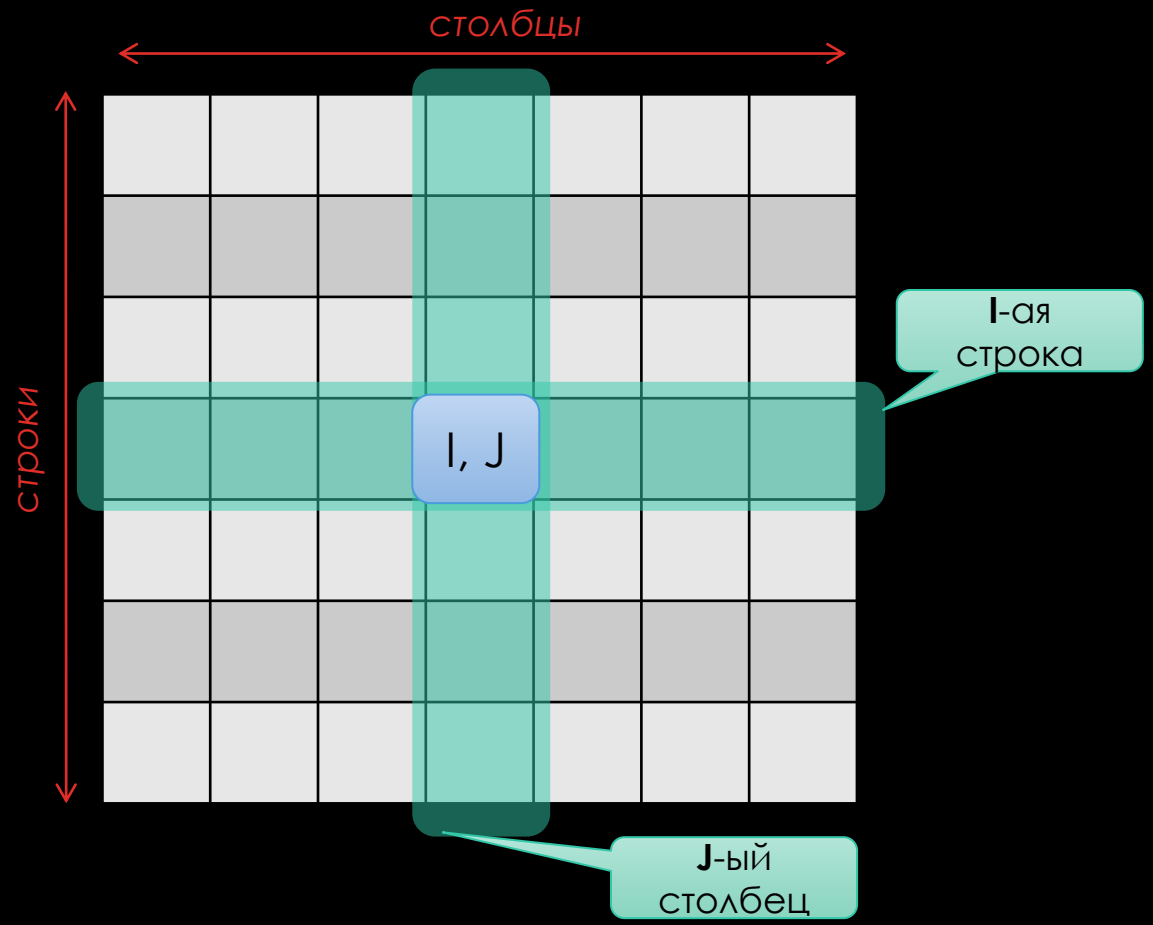
ВЕРНЁМСЯ К ПРОГРАММИРОВАНИЮ

Первый индекс -номер строки

$$\begin{Vmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \dots & \dots & \dots & \dots \\ a_{m1} & a_{m2} & \dots & a_{mn} \end{Vmatrix}$$

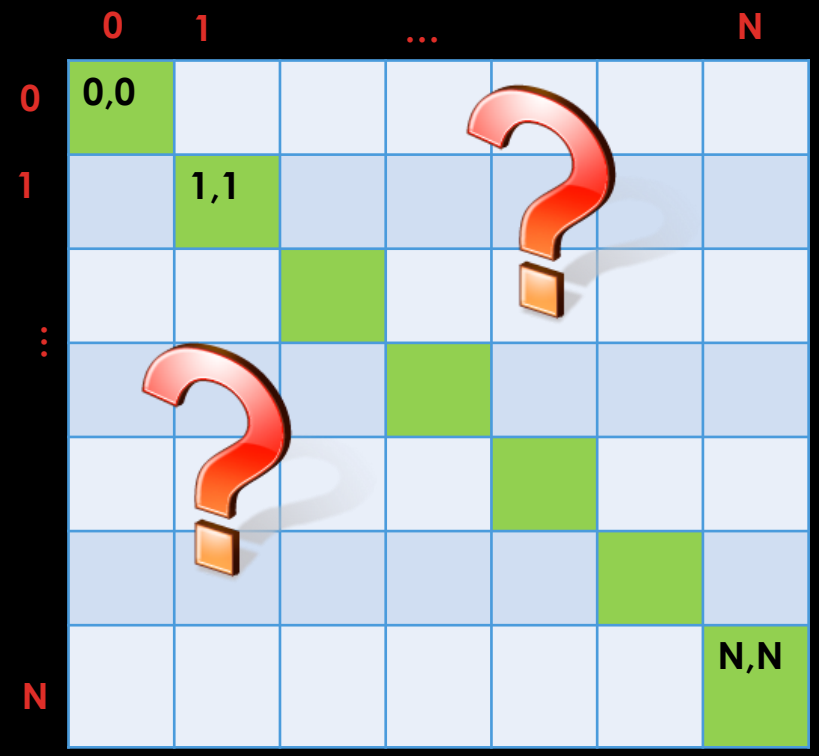
Второй индекс- номер столбца

Числа, составляющие матрицу, называются её **элементами** и характеризуются своим положением в таблице.

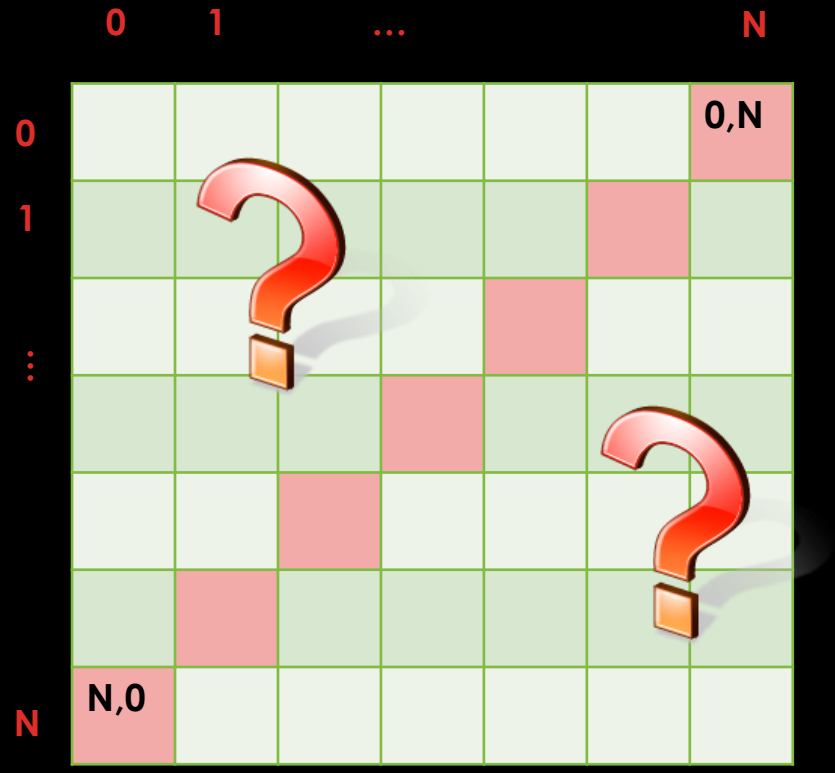


КВАДРАТНАЯ МАТРИЦА

Соотношение для индексов элементов главной диагонали I == J

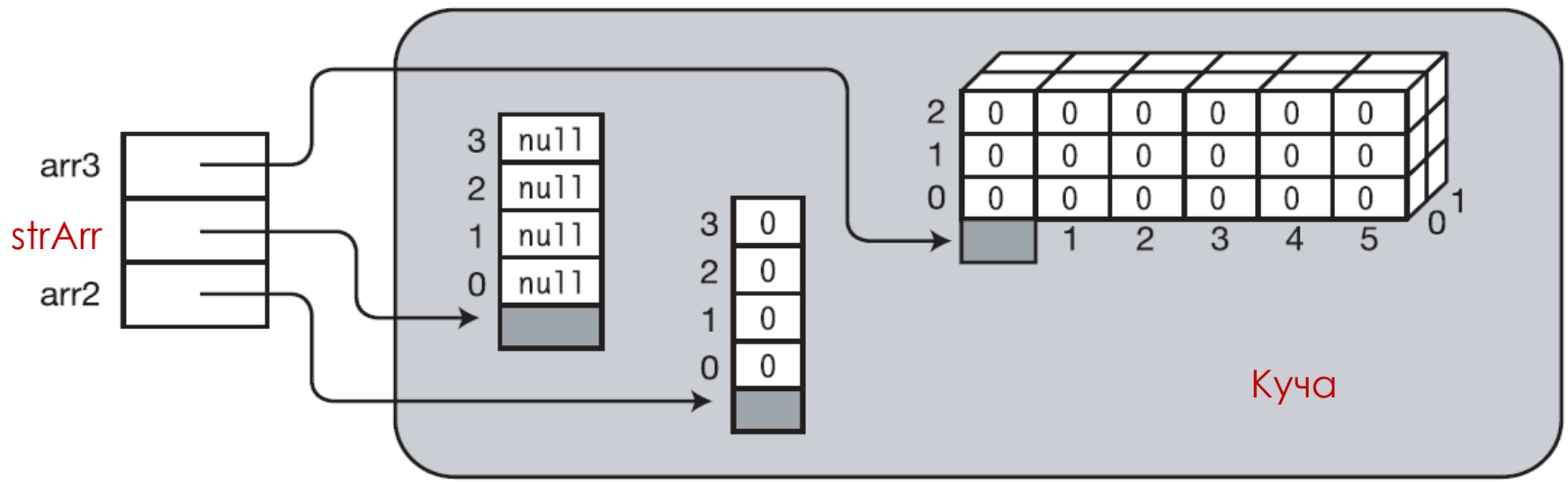


Соотношение для индексов элементов побочной диагонали J == N - I



ЗНАЧЕНИЯ ЭЛЕМЕНТОВ МАССИВОВ ПО УМОЛЧАНИЮ

```
int[] arr2 = new int[4];          // Одномерный массив целых чисел из 4 элементов.
string[] strArr = new string[4]; // Одномерный массив строк из 4 элементов.
int[,,] arr3 = new int[3, 6, 2]; // Трёхмерный массив с измерениями длиной 3, 6 и 2.
```



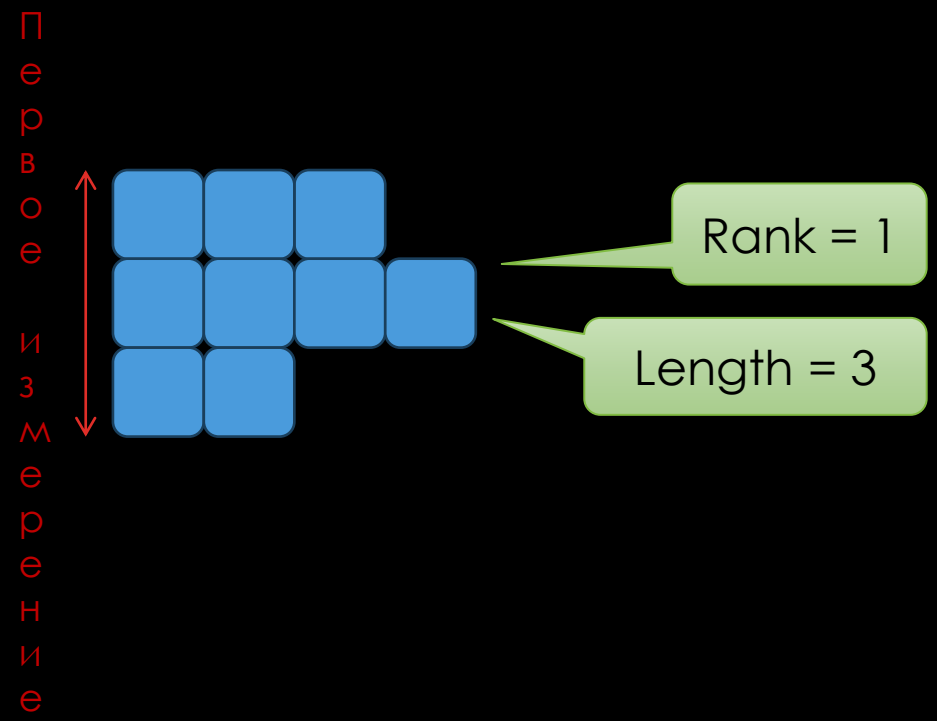
Элементы массива при неявной инициализации будут всегда иметь значения типа элемента по умолчанию

МАССИВЫ МАССИВОВ

Зубчатые массивы

ЗУБЧАТЫЕ МАССИВЫ

Зубчатые массивы [jagged arrays] – массивы, каждый элемент которых сам по себе является ссылкой на массив



ОБЪЯВЛЕНИЕ ЗУБЧАТОГО МАССИВА

Объявление зубчатого массива с инициализацией:

```
<Тип>[][] <Идентификатор> = new [<Длина>][]
{
    new [<Длина Вложенного Массива 1>] {<Значения…>}, …
};
```

Допустимо комбинировать многомерные и зубчатые массивы

```
// Массив arr состоит из 3 одномерных массивов.
int[][] arr = new int[3][];
// Инициализируем каждый вложенный одномерный массив отдельно:
arr[0] = new int[] { 1, 2, 3 };
arr[1] = new int[] { 4, 5, 6, 7 };
arr[2] = new int[] { 8, 9, 10, 11, 12 };
```

ИНИЦИАЛИЗАЦИЯ ЗУБЧАТОГО МАССИВА

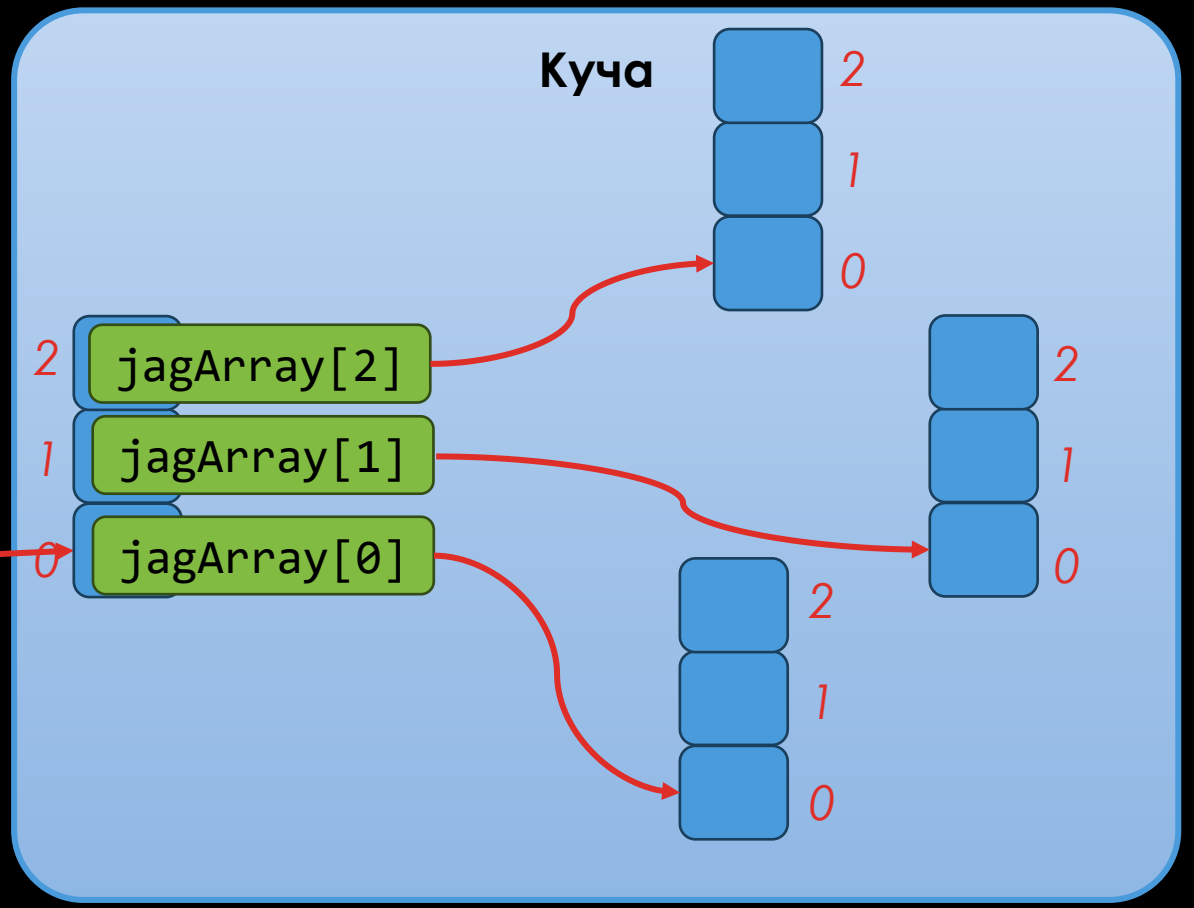
```
// Зубчатый массив одномерных зубчатых массивов без инициализации.
int[][][] arr0;
// Массив двумерных массивов с явной инициализацией.
int[][,] arr2 =
{
    new int[,] { { 10, 20 }, { 100, 200 } },
    new int[2, 3] { { 30, 40, 50 }, { 300, 400, 500 } },
    new int[2, 4] { { 60, 70, 80, 90 }, { 600, 700, 800, 900 } }
};
```

Такой вариант **НЕ компилируется**:

```
int[][] arrIncorrect = new int[2][3];
```

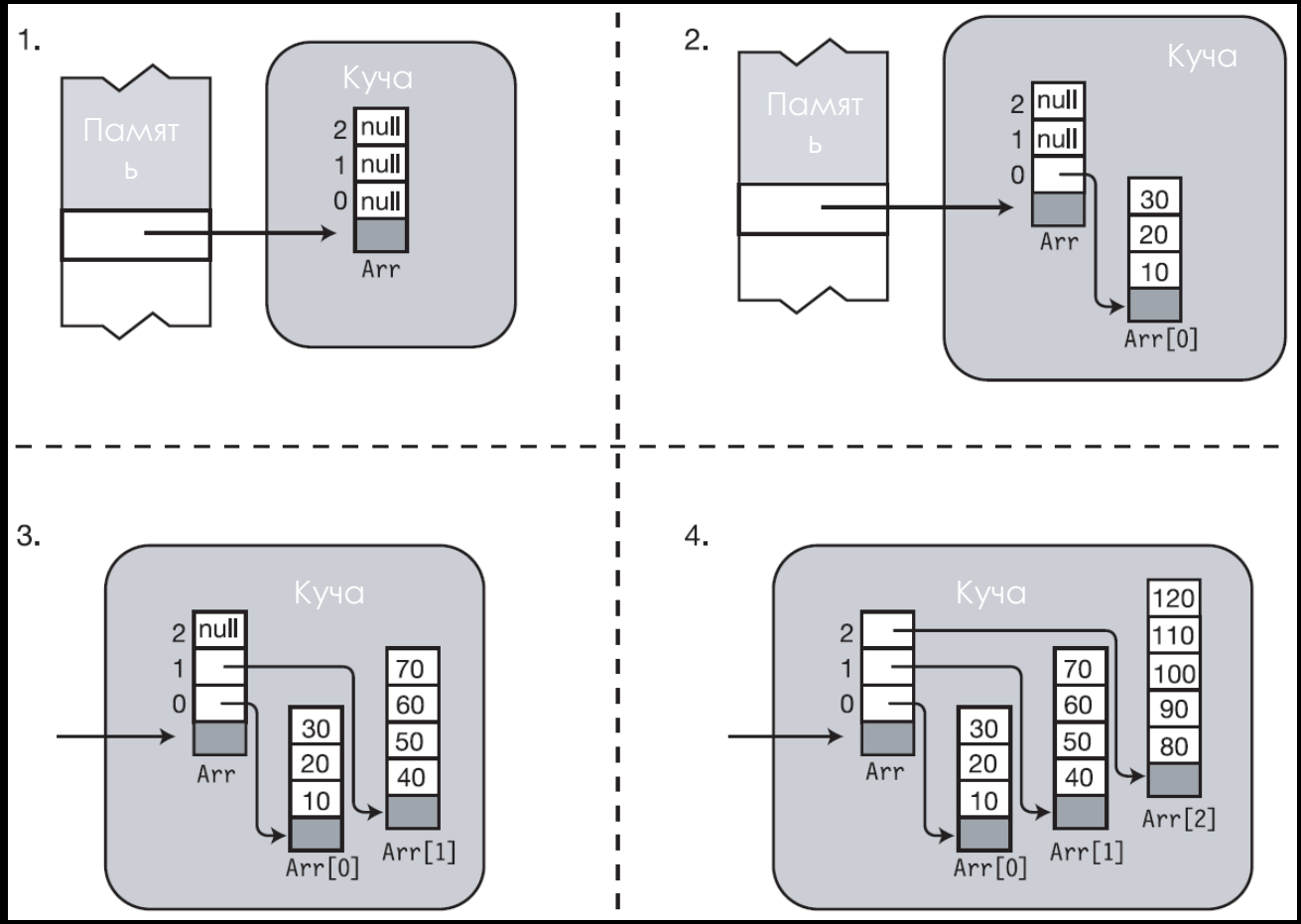
ЗУБЧАТЫЕ МАССИВЫ В ПАМЯТИ

```
// Создадим зубчатый массив:
int[][] jagArr = new int[3][];
jagArr[0] = new int[3];
jagArr[1] = new int[5];
jagArr[2] = new int[15];
```



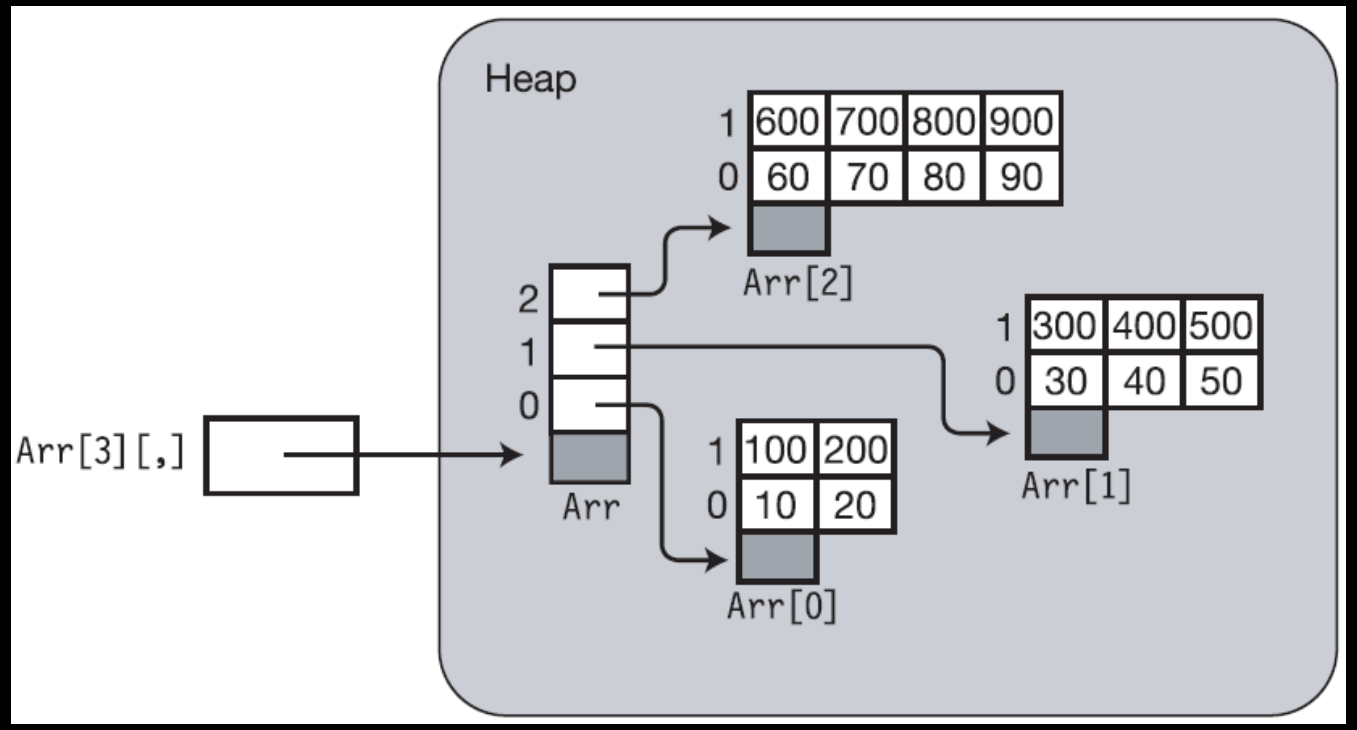
СОЗДАНИЕ ЗУБЧАТОГО МАССИВА: РАЗМЕЩЕНИЕ В ПАМЯТИ

```
// 1) Создать внешний массив:
int[][] Arr = new int[3][];

// 2) Создать связанные массивы:
Arr[0] = new int[]{ 10, 20, 30 };
Arr[1] = new int[]{ 40, 50, 60, 70 };
Arr[2] = new int[]{80, 90, 100, 110, 120};
```



СОЗДАНИЕ ЗУБЧАТОГО МАССИВА: РАЗМЕЩЕНИЕ В ПАМЯТИ

```
int[][,] Arr = new int[3][,];
Arr[0] = new int[,] { { 10, 20 }, { 100, 200 } };
Arr[1] = new int[,] { { 30, 40, 50 }, { 300, 400, 500 } };
Arr[2] = new int[,] { { 60, 70, 80, 90 }, { 600, 700, 800, 900 } };
```



ОБХОД ЭЛЕМЕНТОВ ЗУБЧАТЫХ МАССИВОВ – FOR

```
int[][,] arr = {          // Массив двумерных массивов.
    new[,] { { 10, 20 }, { 100, 200 } },
    new int[,] { { 30, 40, 50 }, { 300, 400, 500 } },
    new int[,] { { 60, 70, 80, 90 }, { 600, 700, 800, 900 } }
};
```

```
for (int i = 0; i < arr.GetLength(0); i++) {
    for (int j = 0; j < arr[i].GetLength(0); j++) {
        for (int k = 0; k < arr[i].GetLength(1); k++) {
            Console.Write($"[{i}][{j},{k}] = {arr[i][j, k]}\t");
        }
        Console.WriteLine();
    }
    Console.WriteLine();
}
```

УСЛОВНАЯ ОПЕРАЦИЯ ДОСТУПА К ЭЛЕМЕНТУ

```
int[][] test = { new int[] { 1, 3, 3 }, null, new int[] { -1, 2 } };
```

```
for(int i = 0; i < test.Length; i++)
{
    if (test is not null)
    {
        Array.ForEach(test[i], Console.Write);
    }
}
```

```
foreach (int[] arrays in test)
{
    if (arrays is not null)
    {
        Array.ForEach(arrays, Console.Write);
    }
}
```

```
for (int i = 0; i < test.Length; i++)
{
     Array.ForEach(test[i], Console.Write);
}
```

ПРИМЕР. ИСПОЛЬЗОВАНИЕ ДИАПАЗОНА В ЗУБЧАТОМ МАССИВЕ

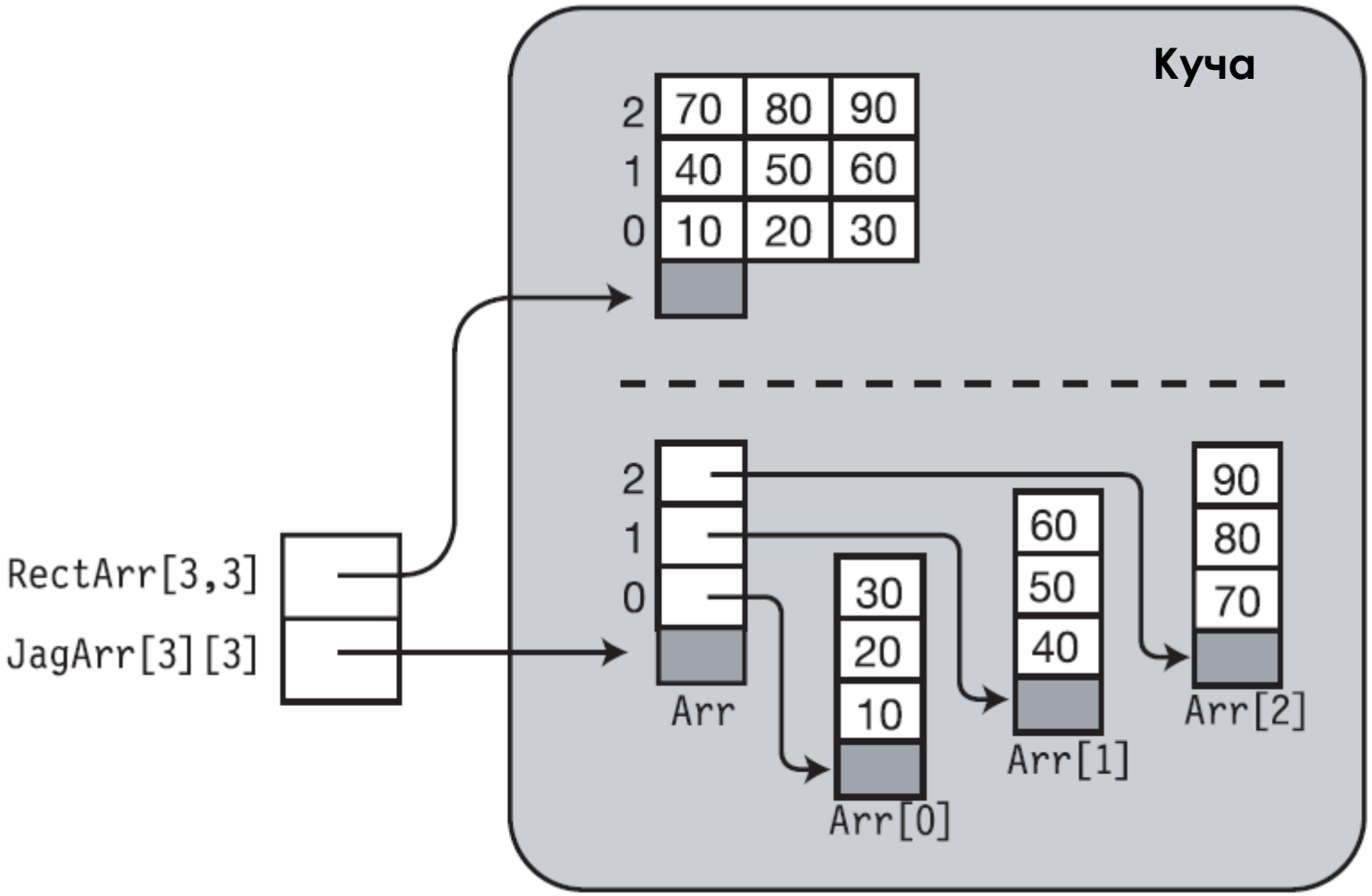
```
int[][] jaggedArr = {
    new int[]{ 1, 2, 3 }, new int[]{ 4, 5, 6 },
    new int[]{ 6, 5, 4 }, new int[]{ 3, 2, 1 }
};
```

Результат выполнения:

100 5 6
6 5 4

```
int[][] newJagged = jaggedArr[1..^1];
jaggedArr[1][0] = 100;
for (int i = 0; i < newJagged.Length; i++)
{
    for (int j = 0; j < newJagged[i].Length; j++)
    {
        Console.Write(newJagged[i][j] + " ");
    }
    Console.WriteLine();
}
```

СРАВНЕНИЕ ПРЯМОУГОЛЬНЫХ И ЗУБЧАТЫХ МАССИВОВ



Прямоугольный массив размера 3 на 3:
- Один объект-массив;
- Тип ссылки – *System.Int32[,]*;
- Прямоугольные массивы не оптимизируются в IL.

Зубчатый массив из 3 элементов:
- Четыре объекта-массива;
- Тип ссылки – *System.Int32[][]*;
- Более сложная структура;
- Одномерные массивы оптимизируются в IL.

СРАВНЕНИЕ РАЗНЫХ ВИДОВ МАССИВОВ

Тип Массива	Количество объектов-массивов	Синтаксис	Особенности
Одномерный	1	1 пара скобок	Оптимизируется в IL.
Прямоугольный	1	1 пара скобок и запятые	Многомерный, все «вложенные массивы» должны быть одинаковой длины.
Зубчатый	Несколько	Несколько пар скобок.	Многомерный, вложенные массивы могут быть разной длины

ПРИМЕР FOREACH ДЛЯ ЗУБЧАТЫХ МАССИВОВ

```
int sum = 0;
int[][] arr1 = {
    new int[] { 10, 11 },
    new int[] { 12, 13, 14 }
};
// Цикл foreach отдельно проходит по
// каждому из вложенных в arr1 массивов.
foreach (int[] array in arr1) {
    System.Console.WriteLine("Starting new array:");
    // Обход по каждому из элементов вложенных массивов.
    foreach (int item in array) {
        sum += item;
        System.Console.WriteLine($"Item: {item}, Current sum: {sum}");
    }
}
```

Результат выполнения:

Starting new array:
Item: 10, Current sum: 10
Item: 11, Current sum: 21
Starting new array:
Item: 12, Current sum: 33
Item: 13, Current sum: 46
Item: 14, Current sum: 60

ОТВЛЕЧЁМСЯ

ПРОВЕРЯЕМ СЕБЯ (1)

```
int[] ar = { 1, 2, 3, 4, 5, 6 };
int[] subAr = ar[..];
for(int i=0;i < ar.Length;i++)
{
    Console.Write(ar[i] + " ");
}
Console.WriteLine();
for (int i = 0; i < subAr.Length; i++)
{
    Console.Write(subAr[i] + " ");
}
```

Что будет выведено на экран?

ПРОВЕРЯЕМ СЕБЯ (2)

Что будет выведено на экран?

```
int[] ar = { 1, 2, 3, 4, 5, 6 };
int[] subAr = ar[3..];
for(int i=0;i < ar.Length;i++)
{
    Console.Write(ar[i] + " ");
}
Console.WriteLine();
for (int i = 0; i < subAr.Length; i++)
{
    Console.Write(subAr[i] + " ");
}
```

```
int[] ar = { 1, 2, 3, 4, 5, 6 };
int[] subAr = ar[^2..];
for(int i=0;i<ar.Length;i++)
{
    Console.Write(ar[i] + " ");
}
Console.WriteLine();
for (int i = 0; i < subAr.Length; i++)
{
    Console.Write(subAr[i] + " ");
}
```

ПРОВЕРЯЕМ СЕБЯ (3)

Что будет выведено на экран?

```
int[] ar = { 1, 2, 3, 4, 5, 6 };
int[] subAr = ar[..3];
for(int i = 0; i < ar.Length; i++)
{
    Console.Write(ar[i] + " ");
}
Console.WriteLine();
for (int i = 0; i < subAr.Length; i++)
{
    Console.Write(subAr[i] + " ");
}
```

```
int[] ar = { 1, 2, 3, 4, 5, 6 };
int[] subAr = ar[..^2];
for(int i = 0; I < ar.Length; i++)
{
    Console.Write(ar[i] + " ");
}
Console.WriteLine();
for (int i = 0; i < subAr.Length; i++)
{
    Console.Write(subAr[i] + " ");
}
```

ИСПОЛЬЗОВАННАЯ ЛИТЕРАТУРА

- Richter, J. CLR via C#. Fourth edition
- https://learn.microsoft.com/ru-ru/dotnet/csharp/whats-new/tutorials/ranges-indexes
- https://www.geeksforgeeks.org/shallow-copy-and-deep-copy-in-c-sharp/
- https://www.c-sharpcorner.com/article/how-to-copy-an-array-in-c-sharp/