Лабораторная работа №6.

Решения должны находиться в файле с названием, соответствующем вашей лиге внутри методов с соответствующими *номерами и названиями*. Решение номеров следует писать в блоке соответствующего номера с использованием входных и выходных параметров. Помимо методов номеров будут проверяться требуемые в задании методы с той сигнатурой, которая указана в задании. Если в номере отсутствует переменная для возврата, то проверяться будет передаваемый исходный массив. На вход **гарантируется** передача данных (не null и количество строк и столбцов ≥ 1). Для тестирования необходимо реализовать все методы, которые требуются по заданию или закомментировать лишние. Тестовые файлы других лиг удалить или закомментировать для запуска тестов.

Общие граничные условия для всех заданий:

1. **Проверка входных данных:**

   Если по условию нужно изменять переданный массив – выполняйте изменения на этом самом массиве и возвращайте его (или ничего, если метод void). Если по условию возвращается новый объект, возвращать новый экземпляр (не менять входной массив).

   Если выполнение невозможно (входные параметры не позволяют – например, требуется квадратная матрица, а она не квадратная; или переданный индекс выходит за границы массива) – вернуть значение по умолчанию:

   - для значимых типов (int, double и т.д.) – 0 (или -1, если по условию явно указано -1);
   - для ссылочных типов – null, если в условии не требуется пустой массив;

   Если по условию ожидается массив результатов, но подходящих для него элементов не обнаружено – вернуть пустой массив (не null).

2. **Равные и краевые значения матрицы:**

   При нормализации или поиске возвращать первые встречаемые значения при прямом (слева направо) проходе, кроме случаев, когда по условию требуется обратный проход.

   Если встречаются несколько одинаковых значений – всегда выбирать первое по порядку (слева направо, сверху вниз), если не указано иное.

   При подсчёте среднего (average): если в знаменателе будет ноль, среднее считать равным нулю.

3. **Возврат корректных пустых массивов:**

   Если ожидается массив, содержащий некий набор значений, но подходящие значения не были найдены в исходном массиве или же после выполнения задания все элементы массива были удалены, вернуть пустой массив, а не null (кроме случаев, когда по условию требуется null).

   При удалении строк/столбцов – возвращать новую матрицу (уменьшенную на 1 строку/столбец) либо менять входную матрицу, если задача этого требует. Если после удаления нет строк/столбцов – вернуть пустую матрицу соответствующих размеров (0×m или n×0) или null, если это явно оговорено в задаче.

Если после удаления остались строки, но в них стало 0 столбцов — вернуть матрицу n×0, а не null.

4. **Сохранение порядка:**

   При необходимости сдвига/перемещения/сортировки элементов сохранять порядок элементов относительно друг друга. Если массив пуст или содержит один элемент, перестановка элементов не требуется.

   При необходимости сортировки элементов следует использовать любую **устойчивую** сортировку.

Методы, указанные в номерах, будут тестироваться отдельно, поэтому не стоит решение всего номера помещать в этот номер. Разбивайте задачу на подзадачи. При желании можно создавать дополнительные методы.

Если все автоматические тесты пройдены успешно, Вы можете отправить лабораторную на заключительную проверку на GitHub. Более подробная инструкция описана в задании на Moodle. Если Ваша работа принята, мы рекомендуем прорешать номера из других лиг в качестве подготовки к контрольной. Особенно те, которые отличаются от заданий из Вашей лиге.

В работе разрешены методы классов: *Console, Math. Random*, *Convert, Array*, а также запросы и методы расширений библиотеки *LINQ*.

Не использовать больше двух уровней вложенности циклов.

Задания белой лиги.

1. В метод передаются одномерные массивы **A** и **B**. Заменить максимальные элементы на среднее арифметическое значение элементов, расположенных после максимального, в том массиве, для которого максимальный элемент расположен дальше от конца массива. Для поиска максимального создать и использовать метод **public int FindMaxIndex(double[] array)**. Если расстояния равны, изменения производить в массиве **A**. Если максимальный элемент последний, то после него нет элементов – в этом случае замену не производить.
2. В метод передаются матрицы (двумерные массивы) **A** и **B**. Поменять местами строки матриц, содержащие максимальные элементы в 1-м столбце. Для поиска индекса строки максимального элемента в заданном столбце создать и использовать метод **public int FindMaxRowIndexInColumn(int[,] matrix, int col)**. Если матрицы разного размера – не выполнять обмен.
3. В метод передается матрица (двумерный массив) **matrix**. Найти индекс строки, содержащей максимальное количество отрицательных элементов. Подсчет количества отрицательных элементов в строках оформить методом **public int[] GetNegativeCountPerRow(int[,] matrix)**.
4. В метод передаются матрицы (двумерные массивы) **A** и **B**. Найти максимальные элементы матриц и поменять их местами. Поиск максимального элемента матрицы оформить в виде метода **public int FindMax(int[,] matrix, out int row, out int col)**.
5. В метод передаются матрицы (двумерные массивы) **A** и **B**. Столбец матрицы **A**, содержащий максимальный элемент матрицы, поменять местами со столбцом матрицы **B**, содержащим максимальный элемент. Для поиска столбца, содержащего максимальный элемент матрицы использовать метод из предыдущего задания: **public int FindMax(int[,] matrix, out int row, out int col)**. Обмен столбцами оформить методом **public void SwapColumns(int[,] A, int colIndexA, int[,] B, int colIndexB)**. Если количество строк в **A** и **B** различается – не выполнять обмен.
6. В метод передается матрица (двумерный массив) **matrix** и экземпляр делегата **Sorting sort**. Упорядочить элементы главной диагонали *квадратной* матрицы. Для упорядочения элементов создать делегат **Sorting**. Упорядочение оформить в виде методов:
    a. по возрастанию **public void SortDiagonalAscending(int[,] matrix)**;
    b. по убыванию **public void SortDiagonalDescending(int[,] matrix)**;
7. В метод передаются целые числа **n** и **k**. Определить, сколькими способами можно отобрать команду в составе k человек из n кандидатов. Использовать метод для подсчета количества способов отбора по формуле: $C_n^k = \frac{n!}{k!\,(n-k)!}$. Создать и использовать рекурсивный метод для расчета факториала **public long Factorial(n)**.
8. В метод передаются вещественные числа **v** и **a** и экземпляр делегата **BikeRide ride**. Велосипедист начинает движение из одной точки с начальной скоростью V км/ч и ускорением A км/ч^2 после каждого часа езды. Создать делегат **BikeRide** и соответствующие его сигнатуре методы для соответствующего заезда:
    a. **public double GetDistance(double v, double a)**, который рассчитывает, какое расстояние проедет велосипедист за 10 часов.
    b. **public double GetTime(double v, double a)**, который рассчитывает, за сколько часов велосипедист проедет 100 км.
9. В метод передается зубчатый массив (массив массивов) **array**. Произвести попарную перестановку соседних элементов, начиная с первого элемента массива, если количество массивов четное, либо с последнего элемента (в противном случае). Вычислить сумму всех элементов заданного массива, стоящих на нечетных индексах. Создать и использовать делегат **Swapper**. Для нахождения суммы элементов с четными индексами массива использовать метод **public double Sum(double[] array)**. Для попарной перестановки элементов использовать методы:
    a. с начала массива **public void SwapFromLeft(double[] array)**;
    b. с конца массива **public void SwapFromRight(double[] array)**;
10. В метод передается зубчатый массив (массив массивов) **array** и экземпляр делегата **Func func**, который принимает в себя зубчатый массив и возвращает информацию о массиве. Делегат **Func**

должен получать зубчатый массив и возвращать одно из значений по выбору вызывающего метода. Для получения информации использовать методы:

a. количество положительных элементов во всех массивах **public int CountPositive(int[][] array)**;
b. максимальный элемент среди всех массивов **public int FindMax(int[][] array)**;
c. максимальная длина массива среди всех массивов **public int FindMaxRowLength(int[][] array)**.

Задания зеленой лиги.

1. В метод передаются одномерные массивы **A** и **B**. Объединить массивы и поместить в массив **A**, предварительно удалив максимальные элементы этих массивов.
    a. удаление элемента массива с заданным индексом осуществить в методе **public void DeleteMaxElement(ref int[] array)**;
    b. объединение массивов осуществлять с помощью метода **public int[] CombineArrays(int[] A, int[] B)**.
2. В метод передаются матрица (двумерный массив) **matrix** и массив **array**. Заменить в каждой строке матрицы максимальный элемент на элемент массива, индекс которой соответствует индексу строки матрицы если найденный элемент меньше элемента массива. Для поиска максимального элемента создать и использовать метод **public int FindMaxInRow(int[,] matrix, int row, out int col)**.
3. В метод передается матрица (двумерный массив) **matrix**. Столбец, содержащий максимальный элемент *квадратной* матрицы, поменять местами с главной диагональю.
    a. Поиск максимального элемента оформить методом **public void FindMax(int[,] matrix, out int row, out int col)**;
    b. Преобразование матрицы оформить методом **public void SwapColWithDiagonal(int[,] matrix, int col)**;
4. В метод передается матрица (двумерный массив) **matrix**. Удалить все строки, содержащие нулевые элементы. Удаление строки оформить в виде метода **public void RemoveRow(ref int[,] matrix, int row)**.
5. В метод передается матрица (двумерный массив) **matrix**. Составить одномерный массив из минимальных элементов строк, расположенных правее элементов главной диагонали (включая диагональ) *квадратной* матрицы. Формирование одномерного массива оформить в виде метода **public int[] GetRowsMinElements(int[,] matrix)**.
6. В метод передаются матрицы (двумерные массивы) **A** и **B**. Объединить массивы, сформированные из сумм положительных элементов столбцов матриц. Если столбец содержит только отрицательные элементы, в массиве-сумме записать 0.
    a. суммирование положительных элементов столбцов с получением массива сумм осуществить в методе **public int[] SumPositiveElementsInColumns(int[,] matrix)**;
    b. объединение массивов осуществлять с помощью метода **public int[] CombineArrays(int[] A, int[] B)** из номера 1.
7. В метод передается матрица (двумерный массив) **matrix** и экземпляр делегата **Sorting sort**. Упорядочить элементы строк матрицы, расположенные после максимального элемента в строке. Для упорядочения элементов создать делегат **Sorting**. Создать методы с соответствующими сигнатурами для сортировки:
    a. по возрастанию **public void SortEndAscending(int[,] matrix)**;
    b. по убыванию **public void SortEndDescending(int[,] matrix)**;
8. В метод передаются одномерные массивы **A** и **B**. Определить треугольник большей площади, стороны которого заданы массивом, вычисляя площади треугольников по формуле Герона: $S = \sqrt{p\,(p-a)(p-b)(p-c)}$, где p = $\frac{a+b+c}{2}$. Создать и использовать метод для расчета площади треугольника **public double GeronArea(a, b, c)**. Если стороны не образуют треугольник (нарушено неравенство треугольника), метод должен вернуть 0.
9. В метод передается матрица (двумерный массив) **matrix** и экземпляр делегата **Action sorter**, который принимает в себя одномерный массив. Расположить элементы четных строк в порядке возрастания или убывания, используя передаваемый делегат.
    a. для получения ряда матрицы использовать метод **public void SortMatrixRow(int[,] matrix, int row, Action<int[]> sorter)**;
    b. для замены элементов матрицы отсортированным массивом использовать метод **public void ReplaceRow(int[,] matrix, int row, int[] array)**;
    c. метод для сортировки по возрастанию **public void SortAscending(int[] array)**;
    d. метод для сортировки по убыванию **public void SortDescending(int[] array)**.

10. В метод передается зубчатый массив (массив массивов) **array** и экземпляр делегата **Func**, который принимает в себя зубчатый массив и возвращает информацию о массиве. Для получения информации использовать методы:
    a. количество скомпенсированных массивов (сумма элементов равна нулю) **public double CountZeroSum(int[][] array)**;
    b. медианное значение среди элементов всех массивов **public double FindMedian(int[][] array)**; (возвращает среднее двух средних элементов при чётном количестве элементов)
    c. общее количество элементов, которые превышают среднее значение в своем массиве среди всех массивов **public double CountLargeElements(int[][] array)**.

Задания синей лиги.

1. В метод передается матрица (двумерный массив) **matrix**. Удалить строку, содержащую максимальный элемент на главной диагонали *квадратной* матрицы.
    a. для поиска индекса максимального элемента диагонали создать и использовать метод **public int FindDiagonalMaxIndex(int[,] matrix)**;
    b. для удаления строки использовать метод **public void RemoveRow(ref int[,] matrix, int rowIndex)**.
2. В метод передаются матрицы (двумерные массивы) **A, B** и **C**. Для каждой из матриц найти среднее значение ее элементов без учета максимального и минимального элементов. Полученные значения занести в одномерный массив. Определить, образовали ли полученные значения убывающую или возрастающую последовательность. Нахождение среднего значения элементов матрицы оформить в вид метода **public double GetAverageExceptEdges(int[,] matrix)**.
3. В метод передается матрица (двумерный массив) **matrix** и делегат **Func method** для поиска индекса столбца. Найти индекс максимального элемента среди элементов, расположенных или ниже главной диагонали (включая диагональ) или выше главной диагонали *квадратной* матрицы. Удалить столбец, в которых он находятся.
    a. для поиска индекса максимального элемента создать методы
        i. выше главной диагонали **public int FindUpperColIndex (int[,] matrix)**;
        ii. ниже главной диагонали **public int FindLowerColIndex(int[,] matrix)**;
    b. удаление столбца оформить методом **public void RemoveCol(ref int[,] matrix, int col)**.
4. В метод передается матрица (двумерный массив) **matrix**. Удалить все столбцы, не содержащие нулевых элементов.
    a. для определения, есть ли в столбце нулевой элемент, использовать метод **public bool CheckZerosInColumn(int[,] matrix, int col)**;
    b. для удаления столбца использовать метод метод **public void RemoveCol(ref int[,] matrix, int col)** из предыдущего номера.
5. В метод передается матрица (двумерный массив) **matrix** и экземпляр делегата **Finder find**. С помощью делегата найти максимальное или минимальное значение в матрице. Удалить все строки, содержащие это значение.
    a. для удаления использовать метод **public void RemoveRow(ref int[,] matrix)** из первого номера.
    b. для поиска элемента матрицы создать делегат **FindElement** и методы с соответствующими сигнатурами, которые находят:
        i. максимальный элемент: **public int FindMax(int[,] matrix, out int row, out int col)**;
        ii. минимальный элемент: **public int FindMin(int[,] matrix, out int row, out int col)**.
6. В метод передается матрица (двумерный массив) **matrix** и экземпляр делегата **SortRowsStyle sort**. Элементы каждой третьей строки упорядочить в заданном порядке. Для упорядочения создать делегат **SortRowsStyle**. Упорядочение элементов строки оформить в виде методов:
    a. по возрастанию **public void SortRowAscending(int[,] matrix, int row)**;
    b. по убыванию **public void SortRowDescending(int[,] matrix, int row)**;
7. В метод передается матрица (двумерный массив) **matrix** и экземпляр делегата типа **ReplaceMaxElements transform**. Найти и преобразовать все максимальные элементы каждой строки.
    a. для поиска максимального элемента использовать метод **public int FindMaxInRow(int[,] matrix, int row)**.
    b. для замены максимальных элементов создать делегат **ReplaceMaxEleme**nts. Максимальные элементы строк преобразовать с учетом методов:
        i. заменить нулем **public void ReplaceByZero(int[,] matrix, int row, int maxValue)**;
        ii. умножить на номер столбца **public void MultiplyByColumn(int[,] matrix, int row, int maxValue)**;
8. В метод передаются вещественные числа **a, b, h** и экземпляр делегата **Func func**. Вычислить сумму **S** для **x** изменяющегося от **a** до **b** с шагом **h** и аналитическое выражение ряда **y** при том

же **x**. Результаты поместить в массив, где первая строка – найденные суммы, вторая строка – значение функции. Для вычисления суммы создать и использовать методы:

a. метод для формирования рядов и расчета их сумм **public double[,] GetSumAndY(double a, double b, double h, Func<double, double> sum, Func<double, double> y)**;

$$S = 1 + \sum_{i=1} \frac{cos(i \cdot x)}{i!}, y = e^{cos(x)} \cdot cos(sin(x)); \; a = 0.1, b = 1, h = 0.1$$

b. для **public double SumA(double x) и public double YA(double x)**;

$$S = \sum_{i=1} (-1)^i \frac{cos(i \cdot x)}{i^2}, y = \frac{x^2}{4} - \frac{\pi^2}{12}; \; a = \frac{\pi}{5}, b = \pi, h = \frac{\pi}{25}$$

c. для **public double SumB(double x) и public double YB(double x)**.

9. В метод передается матрица (двумерный массив) **matrix** и экземпляр делегата типа **GetTriangle triangle**. Вычислить сумму квадратов элементов вектора, полученного пересылкой в него либо верхнего, либо нижнего треугольника заданной *квадратной* матрицы (в обоих случаях включая главную диагональ построчно слева-направо). Создать и использовать делегат **GetTriangle** для выбора треугольника. Создать методы
   a. для суммирования квадратов элементов массива **public int Sum(int[] array)**.
   b. для получения массива элементов треугольника матрицы **public int GetSum(GetTriangle transformer, int[,] matrix)**.
   c. для выделения в массив верхнего треугольника **public int[] GetUpperTriangle(int[,] matrix)**;
   d. для выделения в массив нижнего треугольника **public int[] GetLowerTriangle(int[,] matrix)**;
10. В метод передается зубчатый массив (массив массивов) array и экземпляр делегата **Predicate**, который принимает в себя зубчатый массив и возвращает информацию о массиве. Для получения информации использовать методы:
    a. возможность преобразования в обычную матрицу путем передачи элементов из более длинных массивов в более короткие без потери данных (количество строк должно соответствовать количеству строк исходного массива) **public bool CheckTransformAbility(int[][] array)**;
    b. наличие строгой упорядоченности (по возрастанию или убыванию) последовательности сумм всех массивов **public bool CheckSumOrder(int[][] array)**;
    c. наличие хотя бы одного упорядоченного (по возрастанию или убыванию) массива среди всех массивов **public bool CheckArraysOrder(int[][] array)**.

Задания фиолетовой лиги.

1. В метод передаются матрицы (двумерные массивы) **A** и **B**. Поменять местами строку *квадратной* матрицы **A** и столбец *квадратной* матрицы **B**, содержащие максимальные элементы на диагоналях. Обмен должен быть поэлементным и изменения происходят в обоих массивах.
    a. для поиска индекса максимального элемента диагонали создать и использовать метод **public int FindDiagonalMaxIndex(int[,] matrix)**;
    b. для обмена строки и столбца использовать метод **public void SwapRowColumn(int[,] matrix, int rowIndex, int[,] B, int columnIndex)**.
2. В метод передаются матрицы (двумерные массивы) **A** и **B**. В матрицу **A** вставить столбец матрицы **B** в качестве новой строки. Позиция для вставки: после строки, содержащей максимальное количество положительных элементов. Выбор столбца: должен содержать максимальное количество положительных элементов. Если нет положительных чисел в матрице **B**, ничего не вставлять. Вставку столбца в матрицу реализовать в методе **public void InsertColumn(ref int[,] A, int rowIndex, int columnIndex, int[,] B)**. Создать и использовать методы для поиска количества положительных элементов:
    a. в заданной строке **public int CountPositiveElementsInRow(int[,] matrix, int row)**;
    b. в заданной строке **public int CountPositiveElementsInColumn(int[,] matrix, int col)**.
3. В метод передается матрица (двумерный массив) **matrix**. Пять наибольших элементов увеличить вдвое по абсолютной величине, остальные уменьшить вдвое. Если в матрице меньше 5 элементов, увеличить все. Если есть повторяющиеся значения, попадающие и выходящие за лимит в 5 элементов, увеличивать те, которые расположены раньше при проходи сверху-вниз слева-направо по матрице. Преобразование матрицы оформить в виде метода **public void ChangeMatrixValues(int[,] matrix)**.
4. В метод передаются матрицы (двумерные массивы) **A** и **B**. Поменять строки матриц, содержащие максимальное количество отрицательных элементов. Если в строках какой-либо из матриц одинаковое число отрицательных элементов – брать первую. Если в какой-либо из матриц нет отрицательных элементов, не производить обмен. Использовать методы для:
    a. формирование массива из отрицательных элементов матрицы **public int[] CountNegativesPerRow(int[,] matrix)**;
    b. нахождения максимального значения в массиве **public int FindMaxIndex(int[] array)**.
5. В метод передается одномерный массив **array** и экземпляр делегата **Sorting sort**. Упорядочить отрицательные элементы массива, оставив положительные элементы на прежних местах. Для упорядочения отрицательных элементов создать делегат **Sorting**. Упорядочение оформить в виде методов:
    a. по возрастанию **public void SortNegativeAscending(int[] matrix)**;
    b. по убыванию **public void SortNegativeDescending(int[] matrix)**;
6. В метод передается матрица (двумерный массив) **matrix** и экземпляр делегата **SortRowsByMax sort**. Расположить строки в порядке убывания или возрастания их максимальных элементов. Строки переставляются целиком, а не сортируются поэлементно. Для упорядочения создать делегат **SortRowsByMax**. Сортировку строк оформить в виде методов:
    a. по возрастанию **public void SortRowsByMaxAscending(int[,] matrix)**;
    b. по убыванию **public void SortRowsByMaxDescending(int[,] matrix)**;
    c. для получения максимального элемента строки использовать метод **public int GetRowMax(int[,] matrix, int row)**.
7. В метод передается матрица (двумерный массив) **matrix** и экземпляр делегата **FindNegatives find**. Сформировать одномерный массив, состоящих из количества отрицательных элементов строк или максимальных отрицательных элементов столбцов. Если в столбце нет отрицательных элементов, считать, что максимальное число в данном столбце равно 0. Создать делегат **FindNegatives** для сбора отрицательных элементов. Для создания массивов использовать методы с соответствующими делегату сигнатурами:
    a. для массива, состоящего из количества отрицательных элементов строк **public int[] FindNegativeCountPerRow(int[,] matrix)**;

   b. для массива, состоящего из максимальных отрицательных элементов столбцов **public int[] FindMaxNegativePerColumn(int[,] matrix)**;
8. В метод передается матрица (двумерный массив) **matrix** и экземпляр делегата **MathInfo info**. Функция y = f(x) задана таблично на отрезке [A, B] (т.е. 1-ая строка – x, 2-ая строка – y). Если все элементы равны, возвращать пустой массив. Функция f(x) называется монотонно возрастающей, если для любых двух точек в ее области определения x1 < x2, выполняется f(x1) ≤ f(x2). И наоборот, функция является монотонно убывающей, если f (x1) ≥ f (x2) при тех же условиях. Определить:
   a. является ли функция монотонно убывающей или монотонно возрастающей на заданном отрезке. Для этого создать и использовать метод **public int[,] DefineSeq(int[,] matrix)**; Метод должен возвращать матрицу, состоящую из 1 элемента со значением: 1 для возрастающей последовательности, -1 - для убывающей, иначе 0;
   b. все интервалы монотонности **public int[,] FindAllSeq(int[,] matrix)**. Каждая строка матрицы должна состоять из двух элементов: начало и конец интервала монотонности (по первой строке – x). Интервалы должны быть отсортированы по возрастанию сперва начального, а потом конечного элемента интервала;
   c. самый длинный интервал монотонности **public int[,] FindLongestSeq(int[,] matrix)**; Матрица должна состоять из одной строки и двух столбцов: начало и конец интервала монотонности (по первой строке – x).
9. В метод передаются вещественные числа **a, b, h** и экземпляр делегата **Func func**. Разработать метод определения количества интервалов смены знака функции, заданной на отрезке с постоянным шагом аргумента.
   a. Для подсчета использовать метод **public int CountSignFlips(double a, double b, double h, Func<double, double> func)**.
   b. Для вычисления очередного значения функции использовать:
      i. для функции $y = x^2 - \sin x$ метод **public double FuncA(double x)**;
      ii. для функции $y = e^x - 1$ метод **public double FuncB(double x)**;
10. В метод передается зубчатый массив (массив массивов) **array** и экземпляр делегата **Action**, который принимает в себя зубчатый массив и выполняет работу с массивом. Для обработки массива использовать методы:
   a. упорядочение по возрастанию каждого четного и по убыванию каждого нечетного массива среди всех массивов **public void SortInCheckersOrder(int[][] array)**;
   b. упорядочение массивов по убыванию их сумм среди всех массивов **public void SortBySumDesc(int[][] array)**;
   c. разворот каждого массива, а также массива массивов в обратном порядке **public void TotalReverse(int[][] array)**.