ФГАОУ ВО «КФУ им. В.И. Вернадского»
Физико-технический институт (структурное подразделение)

Кафедра компьютерной инженерии и моделирования

Скибинский Дмитрий Константинович

отчет по практической работе №2
по дисциплине **«ОБЪЕКТНО-ОРИЕНТИРОВАННОЕ ПРОГРАММИРОВАНИЕ»**

Направление подготовки:
09.03.04 "Программная инженерия"

Оценка 90

Симферополь, 2024

Практическая работа №2.
Тема: Использование программных конструкций C#

Цель работы: Изучить на практике использование перегрузки и переопределения (Override) методов, статические и виртуальные методы, научиться передавать в методы простые типы по ссылке, передавать и возвращать из методов несколько значений, в том числе и неопределенное значение параметров. Научиться использовать компоненты Grid или DataGridView в Windows Forms или WPF приложениях.

Описание ключевых понятий:

Перегрузка методов — это возможность определения нескольких методов с одинаковым именем, но с разными параметрами (количество, тип или порядок параметров) в одном классе. Компилятор определяет, какой метод вызывать, основываясь на переданных аргументах.

Переопределение методов — это возможность изменить реализацию метода, унаследованного от базового класса, в производном классе. Для переопределения метода в базовом классе должен быть указан модификатор virtual, а в производном классе — override.

Скрытие методов — это возможность определить метод в производном классе с тем же именем и сигнатурой, что и метод в базовом классе, но без использования ключевого слова override. В этом случае метод в производном классе скрывает метод в базовом классе.

Закрытые методы (private): Методы, которые доступны только внутри класса, в котором они определены. Не могут быть вызваны извне класса.

Открытые методы (public): Методы, которые доступны из любого места в программе, где есть доступ к классу, в котором они определены.

Статические методы (static): Методы, которые принадлежат классу, а не экземпляру класса. Они вызываются через имя класса, а не через экземпляр.

Виртуальные методы (virtual): Методы, которые могут быть переопределены в производном классе. Они предназначены для реализации полиморфизма.

Кортежи — это упрощенные структуры данных, которые позволяют группировать несколько значений в один объект. Кортежи могут содержать элементы разных типов и часто используются для возврата нескольких значений из метода.

Params — это ключевое слово, которое позволяет методу принимать переменное количество аргументов одного типа. Аргументы передаются в виде массива.

Перед выполнением лабораторной работы изучена следующая литература:

1. Прослушаны лекции
2. Прочитаны 2 презентации по лекциям по дисциплине «Объектно-ориентированное программирование»
3. Изучены материалы metanit.com

Выполнены 4 задания, описанных в методических указания к выполнению лабораторных работ.

Задание 1. Вычислить наибольший общий делитель двух целых чисел с помощью алгоритма Евклида.

Для выполнения задания необходимо создать 2 числа, которые будут принимать значения с TextBox`ов. Эти числа будут внесены в массив данных и переданы в функцию FindGCD.

```
Ссылок: 1
private void FindGCDButton_Click(object sender, RoutedEventArgs e)
{
    try
    {
        int a = int.Parse(FirstNumberTextBox.Text);
        int b = int.Parse(SecondNumberTextBox.Text);
        //int c = int.TryParse(ThirdNumberTextBox.Text, out int cValue) ? cValue : 0;
        //int d = int.TryParse(FourthNumberTextBox.Text, out int dValue) ? dValue : 0;
        //int g = int.TryParse(FifthNumberTextBox.Text, out int eValue) ? eValue : 0;

        var numbers = new[] { a, b/*, c, d, g*/ }.Where(x => x != 0).ToArray();

        if (numbers.Length < 2)
        {
            ResultLabel.Content = "Введите как минимум два числа.";
            return;
        }

        int gcd = FindGCD(numbers);
        ResultLabel.Content = $"НОД({string.Join(", ", numbers)}) = {gcd}";
    }
    catch (FormatException)
    {
        ResultLabel.Content = "Введите корректные числа.";
    }
}
```

Эта функция будет принимать два числа и находить наибольший делитель методом Евклида.

```
public static int FindGCD(int a, int b)
{
    while (b != 0)
    {
        int temp = b;
        b = a % b;
        a = temp;
    }
    return a;
}
```

Задание 2. Вычислить наибольший общий делитель 3,4 и 5 чисел с помощью перегрузки методов

Для нахождения НОД для нескольких чисел будет использовать перегрузку методов. Для этого в массив чисел будем добавлять, только те числа, которые не будут равны 0. Для этого при методе Parse добавим присваивание 0 числу, где этот метод не смог совершить парсинг значения.

```
public MainWindow()
{
    InitializeComponent();
}

Ссылок: 1
private void FindGCDButton_Click(object sender, RoutedEventArgs e)
{
    try
    {
        int a = int.Parse(FirstNumberTextBox.Text);
        int b = int.Parse(SecondNumberTextBox.Text);
        int c = int.TryParse(ThirdNumberTextBox.Text, out int cValue) ? cValue : 0;
        int d = int.TryParse(FourthNumberTextBox.Text, out int dValue) ? dValue : 0;
        int g = int.TryParse(FifthNumberTextBox.Text, out int eValue) ? eValue : 0;

        var numbers = new[] { a, b, c, d, g }.Where(x => x != 0).ToArray();

        if (numbers.Length < 2)
        {
            ResultLabel.Content = "Введите как минимум два числа.";
            return;
        }

        int gcd = FindGCD(numbers);
        ResultLabel.Content = $"НОД({string.Join(", ", numbers)}) = {gcd}";
    }
    catch (FormatException)
    {
        ResultLabel.Content = "Введите корректные числа.";
    }
}
```

После того как мы добавили в массив числа отличные от 0, мы проверим что элементов в массиве не меньше 2.

После этого воспользуемся технологией перегрузки в котором каждая последующая перегрузка будет использовать предыдущую и перегрузку двух элементов.

```
Ссылок: 1
public static class GCDAlg
{
    // Метод для двух чисел
    Ссылок: 5
    public static int FindGCD(int a, int b)
    {
        while (b != 0)
        {
            int temp = b;
            b = a % b;
            a = temp;
        }
        return a;
    }

    // Метод для трех чисел
    Ссылок: 1
    public static int FindGCD(int a, int b, int c)
    {
        return FindGCD(FindGCD(a, b), c);
    }

    // Метод для четырех чисел
    Ссылок: 1
    public static int FindGCD(int a, int b, int c, int d)
    {
        return FindGCD(FindGCD(a, b, c), d);
    }

    // Метод для пяти чисел
    Ссылок: 0
    public static int FindGCD(int a, int b, int c, int d, int e)
    {
        return FindGCD(FindGCD(a, b, c, d), e);
    }
```

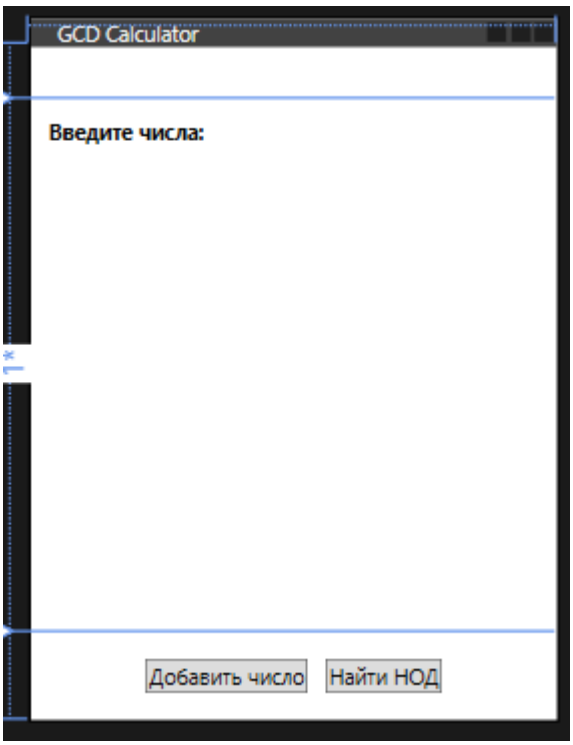
Однако, для решения этой задачи мы может не прибегать к методам перегрузки, а использовать метод Aggregate

```
            }

            int gcd = numbers.Aggregate(GCDAlg.FindGCD);
            ResultLabel.Content = $"НОД({string.Join(", ", numbers)}) = {gcd}";
        }
        catch (FormatException)
        {
            ResultLabel.Content = "Введите корректные числа.";
        }
    }
}

Ссылок: 1
public static class GCDAlg
{
    Ссылок: 1
    public static int FindGCD(int a, int b)
    {
        while (b != 0)
        {
            int temp = b;
            b = a % b;
            a = temp;
        }
        return a;
    }
```

По умолчанию он будет передавать по два элемента из массива в метод нахождения наибольшего общего делителя.

Дополнение. Найти наименьший общий делитель

Для этого нарисуем понятный интерфейс и добавим цикл для нахождения делителя.



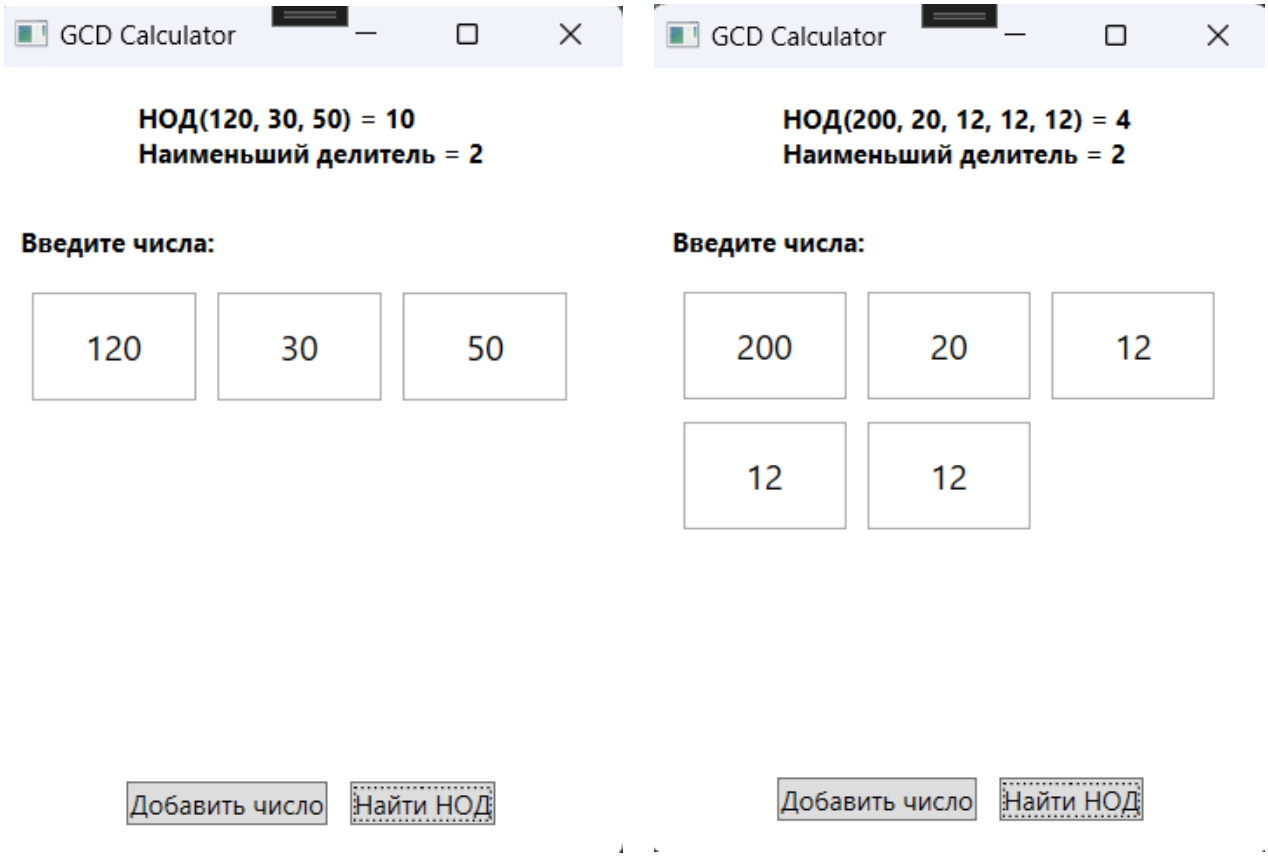
Интерфейс будет добавлять то количество окошек, сколько необходимо пользователю. По умолчанию их будет 3.

```
private void AddDefaultTextBoxes(int count)
{
    for (int i = 0; i < count; i++)
    {
        AddNumberTextBox();
    }
}

Ссылок: 2
private void AddNumberTextBox()
{
    var textBox = new TextBox
    {
        Width = 75,
        Height = 50,
        Margin = new Thickness(5),
        HorizontalAlignment = HorizontalAlignment.Center,
        FontSize = 16,
        TextAlignment = TextAlignment.Center,
        VerticalContentAlignment = VerticalAlignment.Center
    };
    NumbersWrapPanel.Children.Add(textBox);
}
```

Также будем присваивать значение 0, если окно осталось пустым и не добавлять число в массив. Напишем цикл for, который будет бежать от 2 до НОД. Если число gcd будет нацело делиться на i, то результат деления и будет наименьшим общим делителем этих чисел.

```
private void FindGCDButton_Click(object sender, RoutedEventArgs e)
{
    try
    {
        var numbers = NumbersWrapPanel.Children.OfType<TextBox>()
            .Select(tb => int.TryParse(tb.Text, out int value) ? value : 0)
            .Where(x => x != 0)
            .ToArray();

        if (numbers.Length < 2)
        {
            ResultLabel.Content = "Введите как минимум два числа.";
            return;
        }

        int gcd = FindGCD(numbers);
        int mincd = -1; // Инициализируем по умолчанию

        for (int i = 2; i < gcd; i++) // Изменено на i < gcd, чтобы исключить gcd
        {
            if (gcd % i == 0) // Проверяем, делит ли i на gcd
            {
                mincd = i;
                break;
            }
        }

        // Если mincd не найден, сообщаем об этом
        if (mincd == -1)
        {
            ResultLabel.Content = $"НОД({string.Join(", ", numbers)}) = {gcd} \n min CD не найден (равен 1).";
        }
        else
        {
            ResultLabel.Content = $"НОД({string.Join(", ", numbers)}) = {gcd} \n min CD = {mincd}";
        }
    }
```

Пример работы программы:



Задание 3. Сравнить эффективность двух алгоритмов (Евклида и Штейна)

Определим два числа a и b, для которых будем находить НОД. Также определим количество итераций для формирования статистики и нахождения среднего время работы алгоритмов. Для этого нам понадобиться два счетчика – totalEvklidTicks и totalSteinTicks.

```
static void Main()
{
    int a = 200;
    int b = 500;

    long totalEvklidTicks = 0;
    long totalSteinTicks = 0;
    int iterations = 100;
```

Циклам for будет производить 100 итераций, каждый раз меняя значения чисел для разных подаваемых значений, которые будут подаваться на вход алгоритмам нахождения НОД

```
for (int i = 0; i < iterations; i++)
{
    // Убедимся, что a и b остаются положительными
    int currentA = Math.Max(a - i, 1); // минимальное значение 1
    int currentB = Math.Max(b - i, 1);

    Stopwatch sw1 = Stopwatch.StartNew();
    int evklid_gcd = EuclideanAlgorithm(currentA, currentB);
    sw1.Stop();
    totalEvklidTicks += sw1.ElapsedTicks;

    Stopwatch sw2 = Stopwatch.StartNew();
    int steinn_gcd = SteinAlgorithm(currentA, currentB);
    sw2.Stop();
    totalSteinTicks += sw2.ElapsedTicks;
}
```

При каждой итерации запускаем таймер и прекращаем его работу, как только закончил выполнения алгоритм нахождения НОД. Полученные значения добавляем в счетчик времени, который мы определили ранее.

Далее, встроенной командой переводим это значение в секунды. Для удобства руками переведем в миллисекунды и разделим на количество итераций, чтобы получить среднее время выполнения.

```
double evklidTime = totalEvklidTicks / (double)Stopwatch.Frequency; // переводим тики в секунды
double steinTime = totalSteinTicks / (double)Stopwatch.Frequency;

Console.WriteLine($"Время, затраченное методом Евклида: {evklidTime * 1000 / iterations} миллисекунд");
Console.WriteLine($"Время, затраченное методом Штейна: {steinTime * 1000 / iterations} миллисекунд");

if (evklidTime > steinTime)
{
    Console.WriteLine("Эффективнее метод Штейна");
}
else
{
    Console.WriteLine("Эффективнее метод Евклида");
}
```

Теперь рассмотрим сами методы нахождения НОД. В отличие от классического алгоритма Евклида, основанного на делении с остатком, метод Штейна опирается на операции деления на два и вычитания, что делает его более эффективным для компьютеров.

Алгоритм начинает с проверки, являются ли оба числа четными. Если это так, они делятся на два, а множитель два временно «удаляется» и учитывается в конце вычислений. Если одно из чисел четное, его продолжают делить на два до тех пор, пока оно не станет нечетным. Если же оба числа нечетные, большее из них уменьшается на меньшее. Этот процесс продолжается до тех пор, пока одно из чисел не станет нулем, после чего оставшееся число является НОД.

В завершение алгоритм восстанавливает удалённые множители двойки, умножая результат на соответствующую степень двойки.

```
static int SteinAlgorithm(int a, int b)
{
    if (a == 0) return b;
    if (b == 0) return a;

    int shift;
    for (shift = 0; ((a | b) & 1) == 0; shift++)
    {
        a >>= 1;
        b >>= 1;
    }

    while ((a & 1) == 0) a >>= 1; // a нечетное
    do
    {
        while ((b & 1) == 0) b >>= 1; // b нечетное
        if (a > b)
        {
            int temp = a;
            a = b;
            b = temp; // Обмен a и b
        }
        b -= a; // b - a
    } while (b != 0);

    return a << shift; // Возвращаем результат с учетом общего множителя 2
}
```

Алгоритм Евклида оставим таким же как и в прошлых заданиях. Его основная идея заключается в том, что НОД двух чисел не изменится, если большее число заменить на остаток от деления этого числа на меньшее. Алгоритм повторяет эту операцию (деление с остатком) до тех пор, пока остаток не станет равен нулю. Когда это происходит, текущее меньшее число является НОД.

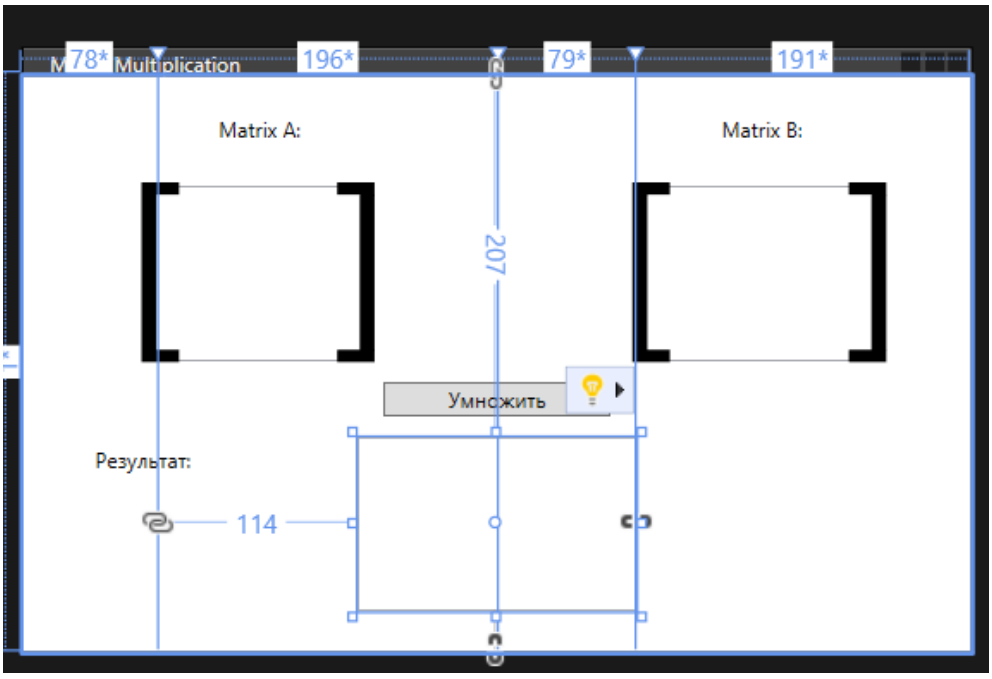
```
static int EuclideanAlgorithm(int a, int b)
{
    while (b != 0)
    {
        int temp = b;
        b = a % b;
        a = temp;
    }
    return a; // НОД
}
```

Вывод программы:

```
Время, затраченное методом Евклида: 0.001052 миллисекунд
Время, затраченное методом Штейна: 0.00126 миллисекунд
Эффективнее метод Евклида
```

Задание 4: Умножение матриц

Напишем интерфейс нашей программы. Здесь будут два окна куда можно будет руками ввести матрицу с соблюдением абзацев и пробелом. Ниже будет окно, куда будет выводиться результат



В методе Main мы просто будем проверять значение строк и столбцов матриц, чтобы сообщить пользователю, если такие матрицы нельзя умножать.

```
Ссылок: 0
public MainWindow()
{
    InitializeComponent();
}

Ссылок: 1
private void MultiplyMatrices(object sender, RoutedEventArgs e)
{
    try
    {
        var matrixA = ParseMatrix(MatrixA.Text);
        var matrixB = ParseMatrix(MatrixB.Text);

        if (matrixA[0].Length != matrixB.Length)
        {
            Result.Text = "Матрицы нельзя \nумножить!";
            return;
        }

        var result = Multiply(matrixA, matrixB);
        Result.Text = MatrixToString(result);
    }
    catch (Exception ex)
    {
        Result.Text = "Ошибка: " + ex.Message;
    }
}
```

В этом методе будут вызываться такие функции как ParseMatrix для преображения вводимых данных в двумерный массив, функция Multiply, которая будет умножать матрицы, а также небольшая функция MatrixToString, ее задачей будет корректно выводить полученную матрицу.

Опишем функцию ParseMatrix. Строка разбивается на отдельные строки (строки матрицы) с помощью метода Split, используя разделители \n и \r (символы новой строки и возврата каретки). Каждая строка разбивается на элементы (числа) по пробелам. Эти элементы преобразуются из строк в числа с помощью double.Parse. В итоге, каждая строка превращается в массив чисел, и все эти массивы собираются в общий массив массивов, представляющий матрицу.

```
Ссылок: 2
private double[][] ParseMatrix(string input)
{
    return input.Split(new[] { '\n', '\r' }, StringSplitOptions.RemoveEmptyEntries)
                .Select(line => line.Split(new[] { ' ' }, StringSplitOptions.RemoveEmptyEntries)
                                    .Select(double.Parse).ToArray())
                .ToArray();
}
```

Функция Multiply. Определяются размеры первой матрицы a (число строк aRows и столбцов aCols) и количество столбцов второй матрицы b (bCols).

Создается новая матрица result для хранения результата умножения, которая будет иметь столько строк, сколько у матрицы a, и столько столбцов, сколько у матрицы b.

Далее идут три вложенных цикла:

- Внешний цикл проходит по строкам первой матрицы a.
- Средний цикл проходит по столбцам второй матрицы b.
- Внутренний цикл отвечает за суммирование произведений элементов строки из первой матрицы и столбца из второй.

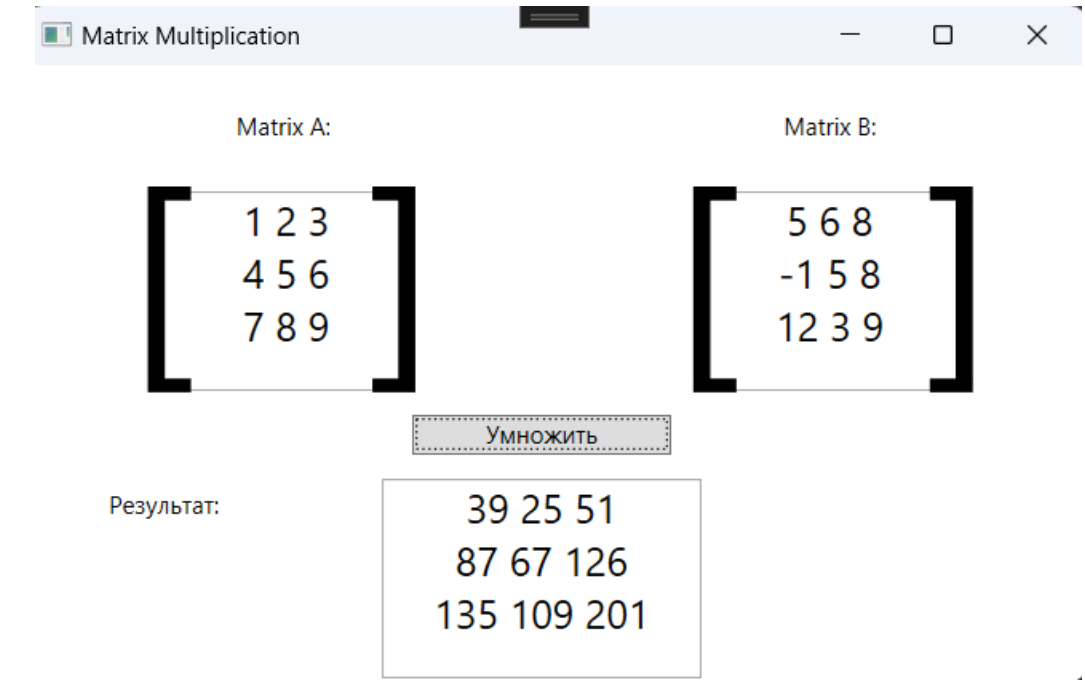
Произведение элементов суммируется для каждой ячейки результирующей матрицы, и итог записывается в соответствующую ячейку result[i][j].

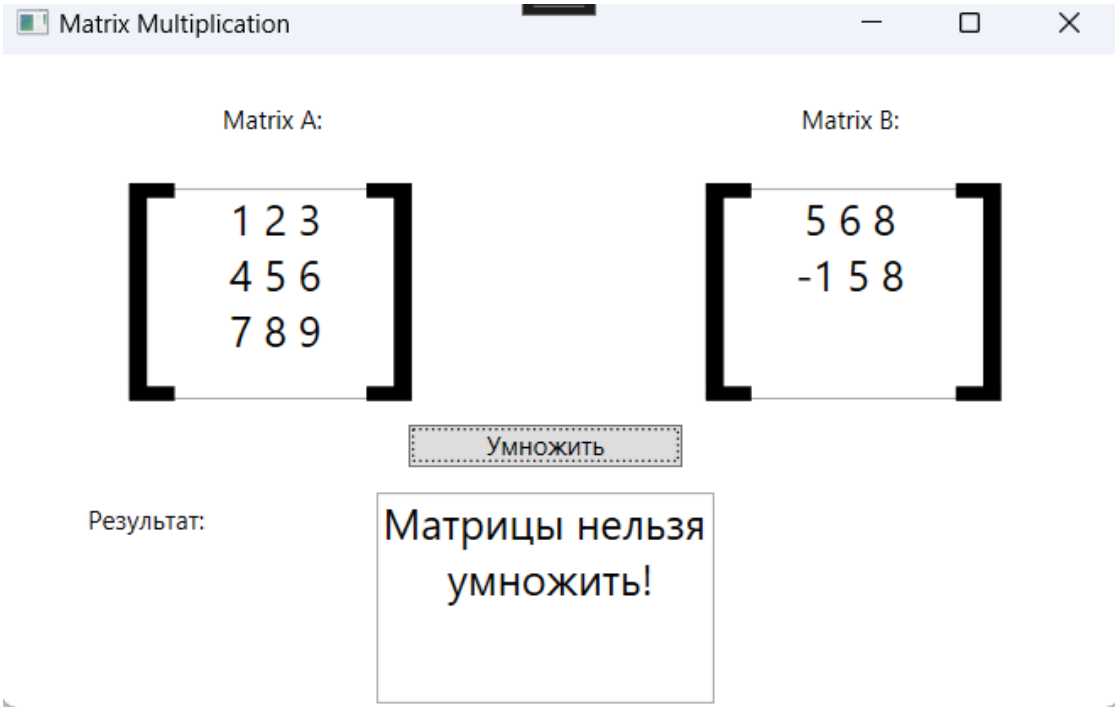
```
Ссылок: 1
private double[][] Multiply(double[][] a, double[][] b)
{
    int aRows = a.Length;
    int aCols = a[0].Length;
    int bCols = b[0].Length;

    var result = new double[aRows][];
    for (int i = 0; i < aRows; i++)
    {
        result[i] = new double[bCols];
        for (int j = 0; j < bCols; j++)
        {
            for (int k = 0; k < aCols; k++)
            {

                result[i][j] += a[i][k] * b[k][j];
            }
        }
    }
    return result;
}
```

Функция MatrixToString. Объединяет строки и столбцы в один элемент string. Результат функции будет передаваться в 3е окно в нашем приложении.

```
Ссылок: 1
private string MatrixToString(double[][] matrix)
{
    return string.Join(Environment.NewLine, matrix.Select(row => string.Join(" ", row)));
}
}
```

Пример работы:





Ответы на вопросы:

1. Что такое перегрузка функций? Какое служебное слово используется для перегрузки?

Перегрузка функций (или методов) — это возможность создания нескольких методов с одним и тем же именем, но с разными параметрами

(типами, количеством или порядком аргументов) в одном классе. При вызове метода компилятор определяет, какой вариант использовать, исходя из переданных аргументов.

2. Что такое переопределение (Override) методов?

Переопределение методов — это процесс, при котором производный класс предоставляет свою реализацию метода, который был объявлен в базовом классе. Для этого в базовом классе метод должен быть объявлен с ключевым словом virtual (или abstract), а в производном классе метод переопределяется с использованием ключевого слова override.

3. Что такое статические и виртуальные методы?

Статические методы (static) принадлежат классу, а не конкретному экземпляру класса. Это означает, что статический метод можно вызывать без создания объекта класса. Статические методы не могут быть виртуальными, их нельзя переопределять в производных классах, и они не имеют доступа к нестатическим членам класса напрямую.

Виртуальные методы (virtual) — это методы, которые могут быть переопределены в производных классах. Виртуальный метод определяет базовое поведение, которое может быть изменено в наследуемых классах с помощью override. Они позволяют создать полиморфизм в объектно-ориентированном программировании.

4. Как можно передавать и возвращать из методов несколько значений?

Использование кортежей (Tuple или ValueTuple): Это один из самых простых способов вернуть несколько значений. Кортеж позволяет вернуть сразу несколько значений разного типа.

5. Как можно передать в метод простой тип по ссылке?

Для передачи простого типа по ссылке используется ключевое слово ref. Это позволяет методу изменять переданное значение напрямую. Важно отметить, что переменная должна быть инициализирована до передачи в метод, если используется ref.

6. Как можно передать в метод неопределенное число параметров?

В C# можно передать неопределенное количество параметров с помощью ключевого слова params. Параметры передаются в виде массива.

7. Для чего служит метод Split?

Метод Split разделяет строку на части (подстроки) по заданным разделителям и возвращает массив этих подстрок. Его можно использовать, например, для разделения строки по пробелам, запятым или другим символам.

8. Почему метод Штейна для вычисления наибольшего общего делителя работает быстрее, чем метод Евклида?

Метод Штейна для вычисления наибольшего общего делителя (НОД) быстрее, чем классический метод Евклида, потому что он использует более простые операции — такие как побитовые сдвиги и вычитания, вместо операций деления, которые более затратны с точки зрения производительности.

Представлены 4 проекта, реализованных в Visual Studio Community 2022. Проекты представлены преподавателю в электронной форме, продемонстрирована их работоспособность, разъяснены детали программного кода.